Plataforma *cloud* para la integración espacial de geoinformación en estructuras de teselas multiescalares asimétricas, su análisis y su visualización

Juan Pedro Pérez Alcántara

Una tesis presentada para el grado de
Doctor en Geografía

Director:
Dr. José Ojeda Zújar

Departamento de Geografía Física y
Análisis Geográfico Regional

Universidad de Sevilla
Marzo 2021

A mi familia.

Prefacio

La presente tesis doctoral se posiciona, dentro del espectro de la investigación instrumental de la Geografía, en la parte más cercana a la ingeniería de *software*. En la última década se ha acuñado un término, las *Tecnologías de la Información Geográfica* (TIG), también llamada *geomática*, que hace referencia a este cruce interdisciplinar entre la ciencia Geográfica y las herramientas y capacidades que el *software* nos proporciona.

Podemos definir la geomática, por tanto, como una interdisciplina orientada al máximo aprovechamiento de las posibilidades que la ingeniería de *software* tiene como constructora de herramientas de alta capacidad analítica para la gestión y diseminación de datos geográficos. El campo de las herramientas del análisis de datos, que en su nivel instrumental están el ámbito de la ingeniería del *software*, ofrecen una serie de ventajas que ninguna disciplina científica puede obviar. Las últimas tendencias en este campo, como el *Machine Learning* o el *Big Data*, y cuyos avances en disciplinas como la Biomedicina o la Genética son ampliamente publicitadas por el periodismo científico generalista, ofrecen unas posibilidades analíticas cuyo tren la Geografía no puede permitirse el lujo de perder. La Geografía instrumental ya perdió el tren de las bases de datos relacionales hace 40 años, lo que sin duda ha tenido un efecto no cuantificado en el desarrollo de técnicas de procesamientos de datos geográficos que hoy ya empiezan a estar extendidas en la práctica académica y profesional de la disciplina. Está en el ánimo del autor y el Director el que esta tesis ponga su granito de arena en el fomento de estas últimas tendencias en nuestro campo porque sin duda los beneficios de su aplicación son muy grandes.

Además, el autor quiere destacar, desde este mismo prefacio, la importancia de los proyectos de *Software Libre* en el presente estado de la cuestión de las posibilidades que la ingeniería de *software* brinda al analista de datos geográficos. Nuestro reconocimiento y gratitud a todos los programadores, testeadores, escritores de documentación técnica, divulgadores o simplemente usuarios que, sea cual sea su nivel y forma de involucración en los distintos proyectos, que hacen posible librerías, *frameworks* y demás herramientas que constituyen la base tecnológica sobre la que se pueden asentar trabajos como el presentado en esta tesis. La abundancia de tecnología de libre acceso es de tal magnitud que el límite a lo que se puede hacer es la imaginación. La imaginación y el acceso a los datos, claro, esa otra gran barrera. El movimiento *Open Data*, con cada vez más instituciones de todo tipo volcadas en la publicación transparente y accesible de bancos de datos de gran riqueza temática, convierten los tiempos en los que este trabajo se escribe en un momento dulce para el análisis de datos geográficos.

También ha de tenerse en cuenta que la magnitud del presente trabajo no se puede

entender sin considerar la gran cantidad de material aportado como resultado en forma de repositorios de código de aplicaciones informáticas, de datos y de prototipos desplegados. Entrar hasta el más mínimo detalle de la implementación de dichos resultados, en forma de programas de ordenador, excede la lógica de descripción del presente documento. Remitimos al lector a dichos repositorios, publicados, como no podría ser de otra manera, con licencias de *Software Libre* con objeto de propiciar su estudio, uso, modificación e intercambio.

Juan Pedro Pérez Alcántara

Sumario

Esta tesis tiene como objetivo general el diseño y creación de una plataforma *cloud* para la gestión de estructuras de rejillas (teselas) multiescalares asimétricas. Las rejillas multiescalares abstraen y homogenizan la información geográfica en una estructura de información que facilita el análisis temático multicriterio con técnicas de procesamiento distribuido *Big Data* y *Machine Learning*, superando las dificultades que estas metodologías presentan sobre datos geográficos, así como proporcionan un medio para el almacenamiento eficiente de información y su tránsito entre servidores y clientes. El prototipo de plataforma de *software* implementado para la prueba de estos conceptos, que hemos denominado *Cell*, constituye un sistema integral basado en *Software Libre* para construir, gestionar, analizar y visualizar información geográfica almacenada en rejillas multiescalares asimétricas.

La integración de información espacial, de especial interés en aplicaciones ambientales y socio-demográficas, ha sido y es un problema clásico en Geografía y en las Tecnologías de la Información Geográfica. La enorme heterogeneidad de modelos de datos (vectorial y ráster), así como la propia disimilitud en los datos que la escala y la propia naturaleza geométrica, sobre todo en el modelo vectorial, imponen a la información geográfica ha dificultado históricamente los procesos de integración para llevar a cabo análisis multivariantes. En esta tesis se propone un marco metodológico e instrumental como vehículo para mejorar esta capacidad de integración espacial, proponiendo un nuevo modelo de datos: la *rejilla multiescalar asimétrica*. Esta estructura de datos hace uso de las capacidades de gestión de datos de estructuración laxa de los sistemas de bases de datos *NoSQL*, específicamente documentos *JSON*, para consignar la información temática.

Esta estructura es multiescalar, utilizando unidades geométricas de diferentes resoluciones, y asimétrica tanto en su dimensión geométrica como temática, en el sentido de que los elementos *NODATA* no están están contemplados. Esto se consigue definiendo matemáticamente la rejilla y utilizando estructuras de datos laxas para la información temática. Esta estructura favorece la computación en paralelo de la información geográfica adscrita y la obtención de vectores de datos.

Estos vectores son la entrada de datos natural a los algoritmos del campo del *Machine Learning* y el *Deep Learning*, permitiendo la exploración de la información integrada mediante estas potentes técnicas analíticas. Como muestra de esta capacidad, el prototipo implementa la aplicación de dos métodos de *Machine Learning*, el *K-Means* y el *Random forest*.

Para las pruebas de concepto se han seleccionado una serie de datos masivos, ya sea por

su tamaño o por el procesamiento que requieren (*Big Data Geográfico*). El prototipo de la plataforma se ha probado adscribiendo información de los Hábitats de Interés Comunitario, el Catastro, la población y un Modelo Digital del Terreno que cubren la totalidad de la Comunidad Autónoma Andaluza.

Esta plataforma está construida con una arquitectura de microservicios que permite su escalabilidad horizontal, pudiendo dimensionarse para asumir considerables volúmenes de trabajo. La arquitectura de estos microservicios y las librerías en las que se basan son lo suficientemente flexibles para permitir a un programador incorporar de una forma sencilla nuevos procedimientos de creación de información teselada.

Para la publicación de resultados, la plataforma aporta una *API* de servicios que permite a usuarios y programas cliente conectarse a la misma para solicitar trabajos de integración y acceso a la información geográfica procesada en la nueva estructura de datos propuesta. Esta *API* aprovecha la capacidad asimétrica y de definición matemática de la rejilla para no enviar, de forma explícita, los elementos geométricos en el tránsito de información servidor - cliente, ya que gracias a esta definición los clientes pueden recrear el elemento geométrico localmente.

No menos importante son los medios de visualización de la información final integrada. Se han creado una serie de soluciones de visualización web, centradas en el consumo de datos integrados y su representación y simbolización en el propio cliente. Esto permite crear aplicaciones de visualización web muy dinámicas e interactivas. Estos visores utilizan tecnologías en el marco de los nuevos estándares de desarrollo *web*, en los que se enfatiza un enfoque de las aplicaciones a datos (cartografía dinámica y *widgets*). Tecnologías como el ecosistema *HTML5* (*WebGL*, *canvas*, etc.) o librerías orientadas a datos como *D3* son revisadas para hacer la explotación cartográfica y de exploración de datos de los productos de la rejilla asimétrica desarrollada en la metodología.

Abstract

In this Thesis the design and implementation of a cloud platform for the management of asimetric multiscalar grid data structures is discussed. Asimetric multiscalar grid data structures is a mean of abstraction and homogeneization of geographic information aimed at making multicriteria thematic analysis easier, allowing for a more accessible use of advanced Big Data and Machine Learning analytics on geographic information, something not always straightforward thanks to the intrinsic topologic relationships present in geographic information. It also provides a mean for efficient storage and dissemination of adscribed multithematic geographic information on a server - client distributed architecture. The main product of this Thesis is not only the description of such a platform, but also a fully featured software prototype called *Cell*, able to build, manage, analyze, and visualize geographic information stored in such a data structure proposed in this work.

Geographic data integration has been a central topic of the discipline for decades, and has important applications in the fields of enviroment and socio-demographic studies. Heterogeneous data structures (mostly vector and raster), coupled with dissimetries in data induced by scale and the geometric nature of geographic information, specially in the case of vector data, has historically hindered multivariate analysis in Geography. This Thesis proposes a methodologic and instrumental framework to enhance spatial data integration by means of a new data model: the assimetric, multiscalar grid. This data model uses mixed instrumental approaches to SQL and NoSQL techniques, using JSON as the physical storage technology to achieve its objectives.

This new data model is multiscalar, using different resolution geometric units, and handles data asymmetrically both in the geometric and thematic aspects of geographic information. This is achieved by eliminating the necessity of NODATA values by mathematically defining the underlying grid. This structure, also, favours the practice of parallel computing on the integrated information and the retrieval of mathematical data vectors.

These mathematical data vectors are the traditional inputs for Machine and Deep Learning algorithms, allowing the use of this powerfull analysis techniques on the integrated data. As a test, the prototype implements two algorithms of Machine Learning, the K-Means and the Random forest.

For concept testing, an array of big datasets has been chosen to test parallel, massive integration of data. This datasets ranges from environmental ones (habitat data, DTM) to basic territorial or demographic ones (cadastrial parcels or population), all of them covering the total area of the Comunidad Autónoma of Andalusia, south Spain.

This platform has been built with a microservices architecture, allowing for enhanced horizontal scalability, making it possible to dimension it to accomodate heavy workloads. The architecture of the system and their libraries is flexible enough to allow a programmer to easily extend the range of spatial integration analytical functions.

For the dissemination of data, the prototype implement an API REST interface to allow users and client programs to connect and retrieve integrated data. This API takes advantage of the asymetry of the integrated data and the mathematical definition of the grid to ease the bulk of data packages to be sent to the client. Thanks to that, the clients are able to recreate the geometries locally.

This work also reviews options for rich visualization of the integrated data. A range of web data visualizations solutions has been explored, focusing on data-driven interactions and rich client-side processing of thematic and geometric data. This allows for highly dynamic and interactive visualization apps, with widgets and dashboard compositions being a possibility. The modern stack of web technologies (HTML5, WebGL) are leveraged in these solutions, as well as data visualization libraries like D3.

Agradecimientos

Por supuesto mi primer pensamiento es para mi familia, Rocío, María, Paula, Dani y Nacho, mis padres, Margarita y Juan Pedro, y mi hermana Alba y el Clan de los Irlandeses, Conor, Molly, Ronan y Martín, que han sido los primeros sufridores de las ausencias y el estrés acaecidos en momentos, afortunadamente, eso sí, puntuales, y sin cuyo apoyo esto habría sido del todo imposible de finalizar. Creo que el momento de la vida para abordar esta tarea es otro bien distinto al que elegí yo, y equilibrar trabajo, familia y tesis la verdad es que no ha sido fácil por momentos. En las últimas etapas, hacerlo en pandemia de COVID-19 ha sido la guinda del pastel a estos momentos puntuales de agobio. Recalco lo de puntual porque no quiero parecer dramático y hay que relativizarlo en lo que es, que tampoco nos va la vida en esto ni mucho menos.

Quiero agradecer, como no, el apoyo recibido por mi Director de tesis, José Ojeda Zújar, al que conozco desde hace décadas y con el que he colaborado en muchos proyectos y en muchas modalidades distintas durante la primera etapa de mi carrera en la empresa privada. Creo que los dos sabíamos donde nos metíamos y con quién lo estábamos haciendo, así que le agradezco mucho su paciencia y que me haya permitido trabajar a mi aire y a mi ritmo, sabiendo como sabe desde siempre lo que me cuesta darle la puntilla y cerrar las cosas, y que tengo tendencia a perderme en los detalles de la "tecnología por la tecnología", perdiendo de vista a veces el efecto práctico y temático inmediato ("investigación básica", la llaman en otras disciplinas). Han sido muchas las conversaciones sobre las Tecnologías de Información Geográfica mantenidas durante estos años, creo que siendo conscientes del momento dulce que vive la disciplina, lo que ha hecho esta relación estimulante a nivel académico y agradable a nivel personal.

Por supuesto ha sido un placer compartir estos años con compañeros, muchos de ellos profesores y maestros, como Juanma, Ismael, Pilar, Alfonso, María Fernanda, Jose, Joaquín, Esperanza, Belen, Mónica, Emilia, Víctor, Arsenio, Miguel, Lorenzo, Jose Antonio, Maika y Estrella, por citar a algunos. Ha sido una experiencia compartir vida universitaria durante estos años. He aprendido mucho de ellos en esas partes más académicas de este negociado que me eran, por mi trayectoria anterior, tan ajenas. Gracias a los compañeros con los que he publicado durante estos años por tener en cuenta mis aportaciones, y espero haber sido también útil por mi parte a tan fantástico colectivo. Agradezco los estimulantes almuerzos en calle San Fernando, y agradezco especialmente que me hayan ayudado a depurar, estoicamente, mi repertorio doctrinal acerca de las bondades del *Software Libre*. Creo que algo he conseguido, aunque sea por desgaste. Espero que podamos seguir colaborando en el futuro independientemente de mi vinculación con la Universidad de Sevilla.

Quisiera agradecer también a las instituciones que han financiado y han hecho posible la realización de este trabajo. Quiero agradecer a la Comisión de Doctorado de la Facultad de Geografía su apoyo y su trabajo, especialmente a su Directora, Rosa Jordá, que siempre ha estado presta a solucionarnos cualquier cuestión o trámite. Quisiera agradecer muy especialmente su ayuda con el papeleo y el enlace con el lejano Ministerio en Madrid a Esperanza Ramírez Torres, de Investigación, siempre atenta y amable ante todas mis torpezas administrativas y de plazos. Agradezco a la Universidad de Sevilla la posibilidad de acceder al Programa de Doctorado con mi antiguo título de licenciatura sin tener que pasar por el Máster habilitante y al Ministerio de Ciencia e Innovación la posibilidad de acceder a la beca de investigación. También quisiera agradecer a estas instituciones su respuesta ante la crisis de la COVID-19 y cómo nos han facilitado las cosas con las ampliaciones de plazos y demás. Agradecer también al Centro Informático Científico de Andalucía (CICA) y al Servicio de Informática y Comunicaciones (SIC) de la Universidad de Sevilla la oportunidad que me han brindado de acceder a sus recursos de computación y obtener, de primera mano, una valiosa experiencia en gestión de infraestructuras informáticas difícil de adquirir por otros medios.

También me gustaría aprovechar esta ocasión para agradecer a los alumnos que he tenido durante todos estos años en Másteres, Grado, cursos del CFP, Universidad de Salamanca y en la Escuela de Organización Industrial, así como a los responsables de estas actividades docentes, por su confianza. La docencia es algo que me apasiona y he aprendido mucho de todos ellos.

Y aunque ya los he nombrado en el prefacio, es de justicia agradecer a toda la comunidad de *Software Libre*, en especial a la que se dedica a las TIG de *Software Libre*, todo su esfuerzo. Sin ellos este trabajo no habría sido posible. Es muy difícil describir e imaginar el derroche de talento, desde los programadores a los divulgadores, que destila la comunidad para llevar a cabo proezas colaborativas que ponen a disposición del que quiera invertir tiempo en ello toda la tecnología que pueda necesitar. La gestión, en un ambiente de ciencia colaborativa, de proyectos de *software* como *QGIS* o *PostGIS*, por citar algunos, debería ser muy tenida en cuenta no ya por el producto que sacan adelante, sino por la forma de hacerlo. Son un modelo a seguir en muchas otras disciplinas y actividades. Anita Graser, Tim Sutton, Paul Ramsey, Even Rouault o Frank Warmerdam son algunas de estas personas con nombres y apellidos que hacen todo esto posible, pero no son mas que la parte visible de una gran comunidad cuyo impacto tecnológico a todos los niveles no creo que se haya cuantificado aún, pero que de seguro es absolutamente extraordinario. Mi reconocimiento a todos ellos y mi agradecimiento más profundo. Espero que con este trabajo ayude, cuanto menos, a divulgar todo este maravilloso ecosistema tecnológico de posibilidades sin fin.

Por último, quiero agradecer a mis compañeros Nick, Allan, Michiel, Manolo y Rocío, de mi nueva etapa laboral en Sunntics, por su generosidad al permitirme compaginar los desarrollos tecnológicos realizados en la empresa con el desarrollo de esta tesis. Ambos proyectos se han retroalimentado mutuamente en gran medida, y sin duda la posibilidad de depurar ciertos subsistemas en un entorno tan exigente como el de Sunntics ha ayudado a elevar la calidad global del resultado de esta tesis. Muchas gracias por confiar en mí para tanta responsabilidad y por lo que aprendo con todos vosotros cada día.

Índice general

Prefacio III

Sumario V

Abstract VII

Agradecimientos IX

Índice general XI

Índice de figuras XIII

Índice de tablas XV

Índice de códigos XVII

I PLANTEAMIENTOS GENERALES 1

I.1 Justificación y contextualización 3

I.1.1. Una primera aclaración terminológica: "rejilla", "tesela" y "rejilla multiescalar" . . . 4
I.1.2. El alcance de las TIG en el curriculum de la Geografía . . . 5
I.1.3. "¿Deberían los geógrafos aprender a programar?" . . . 6
I.1.4. La importancia la estructura en los datos . . . 9
I.1.5. La información geográfica . . . 12
I.1.6. Realidad, sistemas y modelos . . . 13
I.1.7. Modelos de datos en Geografía . . . 14
I.1.8. Un nuevo modelo de datos para la información geográfica . . . 22
I.1.9. Ventajas e inconvenientes de los modelos clásicos . . . 23
I.1.10. Conversión entre modelos . . . 24
I.1.11. Contexto actual de estructuras de datos y tecnologías geográficas . . . 25
I.1.12. *Machine Learning* . . . 29
I.1.13. *Big Data* . . . 31
I.1.14. La *web* moderna: un nuevo paradigma de *software* . . . 33
I.1.15. Experiencia del grupo de investigación . . . 37

I.2 Hipótesis, objetivos, área de estudio y estructura de la tesis 41

I.2.1. Hipótesis y objetivo general 41
I.2.2. Hipótesis y objetivos específicos 41
I.2.3. Área de estudio 43
I.2.4. Estructura de la tesis 44

II FUENTES DE DATOS Y METODOLOGÍA 45

II.1 Bases metodológicas 47
II.1.1. Conceptos previos 48

II.2 Datos originales: selección e importación 51
II.2.1. Datos puntuales 53
II.2.2. Datos poligonales 70
II.2.3. Importación y transformación de datos originales 74
II.2.4. Bases de las estructuras de datos para la plataforma 75
II.2.5. Subida de la información a la plataforma 77

II.3 Definición de la rejilla 79
II.3.1. Características generales de la rejilla 79
II.3.2. La rejilla asimétrica 80
II.3.3. Definición matemática de la rejilla 83
II.3.4. Topología e indexación de la rejilla 83

II.4 Teselado 87
II.4.1. Los análisis de teselado 87
II.4.2. Adaptación de las fuentes originales para los análisis de teselado 96
II.4.3. El motor de teselado 97
II.4.4. Configuración del análisis de teselado inicial 102
II.4.5. Teselado en la plataforma 103

II.5 *Machine Learning* 107
II.5.1. Formato de datos necesario para alimentar algoritmos de *Machine Learning* 108
II.5.2. Algoritmos de *Machine Learning* seleccionados 110
II.5.3. Uso de los algoritmos de *ML* sobre la rejilla 116

II.6 Mecanismos de difusión 117
II.6.1. Visualizaciones de datos en entorno *web* 117
II.6.2. *Application Programming Interface* 122

II.7 Visualización 131
II.7.1. Visores para la rejilla definida matemáticamente 131
II.7.2. Funcionalidades del visor 133

III RESULTADOS 135

III.1 Introducción a los resultados 137
III.1.1. Unas notas introductorias sobre el código del programa y su ubicación y consulta . . . 137

III.2 Arquitectura de microservicios 141
III.2.1. Diseño de la arquitectura de microservicios de *Cell* . . . 141

III.3 Implementación de la base de datos de gestión de *Cell* 145
III.3.1. *PostGIS* . . . 146
III.3.2. Capacidades *NoSQL* de la *PostgreSQL*: los tipos de dato *JSON* y *JSONB* 150
III.3.3. Modelo de datos para la plataforma *Cell* . . . 152
III.3.4. La librería *SQL* . . . 160

III.4 Definicion de la rejilla 167

III.5 Datos originales 175
III.5.1. Población . . . 175
III.5.2. Modelo Digital del Terreno (MDT) . . . 177
III.5.3. Hábitats de Interés Comunitario (HICS) . . . 177
III.5.4. Catastro . . . 178
III.5.5. Otra información de contexto . . . 181
III.5.6. *ETL* a *PostGIS* . . . 182
III.5.7. Creación de la base de datos principal para los datos de origen . . . 184
III.5.8. *ETL* de datos de contexto . . . 184
III.5.9. *ETL* de MDT . . . 185
III.5.10. *ETL* de datos de catastro . . . 185
III.5.11. *ETL* de los datos de Hábitats . . . 185
III.5.12. *ETL* de datos de población . . . 186

III.6 La librería libcellbackend 187
III.6.1. Elección del lenguaje de programación . . . 188
III.6.2. *Python* . . . 191
III.6.3. *JavaScript / TypeScript* . . . 192
III.6.4. *Java* . . . 196
III.6.5. Evaluación del lenguaje . . . 197
III.6.6. Dependencias de la librería *libcellbackend* . . . 197
III.6.7. Clases base de la librería *libcellbackend* . . . 199
III.6.8. Objetivos funcionales de los *Gridder Tasks* . . . 205
III.6.9. Clases para realizar análisis de adscripción en *libcellbackend* . . . 207
III.6.10. Implementación de *GridderTasks* en *libcellbackend* . . . 209
III.6.11. Diseño de una clase de la familia *GridderTask* . . . 209
III.6.12. Variables y catálogos . . . 210
III.6.13. Variables de indexación . . . 213
III.6.14. La clase base *GridderTask* . . . 215
III.6.15. Clases de la familia *GridderTask* . . . 217

III.7 Teselado **235**
III.7.1. Teselación con utilidades *CLI* . . . 235
III.7.2. Configuración final de *GridderTask* para la adscripción de las fuentes originales . . . 240
III.7.3. Teselación con la arquitectura de microservicios . . . 250
III.7.4. Evaluación del proceso de teselado . . . 266
III.7.5. Formato final de la información adscrita . . . 280

III.8 *Machine Learning* **295**
III.8.1. La clase *MlTask* . . . 295
III.8.2. Clases de la familia *MlTask* . . . 296
III.8.3. Otras clases de la librería . . . 298
III.8.4. Evaluación del proceso de *Machine Learning* . . . 298

III.9 Visualización **313**
III.9.1. La librería *libcellfrontend* . . . 313
III.9.2. *Mapbox GL* . . . 314
III.9.3. Histograma de frecuencias de las variables de adscripción con *C3 / D3* 316
III.9.4. Resultados con *Mapbox GL* . . . 317
III.9.5. Resultados con *CARTO* . . . 326

IV DISCUSIÓN, CONCLUSIONES Y LÍNEAS DE FUTURO 333

IV.1 Discusión **335**

IV.2 Conclusiones **339**

IV.3 Líneas de futuro **341**
IV.3.1. Mejoras en los microservicios de datos . . . 341
IV.3.2. Nuevos análisis de teselado . . . 342
IV.3.3. Machine Learning . . . 343
IV.3.4. Visualización . . . 343

V BIBLIOGRAFÍA Y ANEXOS 345

V.1 Bibliografía **347**

V.2 Repositorios GitHub **349**
V.2.1. Repositorios de datos . . . 349
V.2.2. Repositorios de la plataforma *Cell* . . . 350

Índice de figuras

I.1.1. *Jupyter Notebook: código documentado* 10
I.1.2. *Jupyter Notebook: visualización* 11

I.2.1. Zona de estudio: Comunidad Autónoma de Andalucía 43

II.1.1. Esquema metodológico 49

II.2.1. *Triangular Irregular Network* 54
II.2.2. MDT regular 55
II.2.3. Datos originales en formato ráster del MDT del IECA 56
II.2.4. Datos del MDT adscritos como puntos al centroide de las celdas del ráster 57
II.2.5. Secciones censales de la ciudad de Sevilla 59
II.2.6. Datos originales de los centroides de las celdas de población 60
II.2.7. Vista de detalle de los centroides de las celdas de población 61
II.2.8. Vista de las parcelas de catastro a nivel autonómico 63
II.2.9. Vista de detalle de las parcelas de catastro 64
II.2.10. Vista de las unidades constructivas de catastro del centro de Sevilla . . . 65
II.2.11. Vista de detalle de unidades constructivas de catastro en el centro de Sevilla 66
II.2.12. Cálculo del centro de gravedad constructivo volumétrico urbano 67
II.2.13. Cálculo del centro de gravedad constructivo volumétrico rural 68
II.2.14. Comparativa de asignación de centroides en catastro urbano / rural . . . 68
II.2.15. Vista de los datos originales de Hábitats de Interés Comunitario 72
II.2.16. Vista de detalle de los datos originales de Hábitats de Interés Comunitario 73

II.3.1. Origen de coordenadas de una definición de rejilla 81

II.4.1. El análisis de teselado y su relación con otros elementos de la plataforma 88
II.4.2. Pasos funcionales de un análisis de teselación 90
II.4.3. Ejemplo de geometrías originales puntuales colisionantes 92
II.4.4. Ejemplo de geometrías originales poligonales colisionantes 93
II.4.5. Ejemplos de colisión *IDW* 94
II.4.6. Esquema conceptual de microservicios de teselado de la plataforma 98
II.4.7. Pasos funcionales del proceso de teselación en la plataforma 104

II.5.1. Muestra del popular set de datos de entrenamiento para aplicaciones de visión computacional *Chiuauas vs Muffins* 108
II.5.2. *Iris sibirica*, una de las muchas especies del género *Iris* 109
II.5.3. Pasos de una clasificación de *clusters Jenks* 111

II.5.4. Ejemplos de ajuste a distintos tipos de distribución de diferentes algoritmos de *Machine Learning* para *clustering* . . . 113
II.5.5. Árbol de decisión generado a partir de los datos de supervivientes del *Titanic* . . . 115

II.6.1. Esquema simple de una aplicación cliente - servidor . . . 119
II.6.2. Pasos funcionales para la realización de procedimiento de *Machine Learning* 129

III.1.1. Esquema de resultados . . . 138

III.2.1. Arquitectura de microservicios de la plataforma *Cell* . . . 142

III.3.1. Modelo de datos . . . 153
III.3.2. Distribución de densidad de información en un índice *GIST* . . . 161

III.4.1. Rejilla utilizada en pruebas de teselado, resolución de 100 kilómetros . . . 169
III.4.2. Rejilla utilizada en pruebas de teselado, resolución de 50 kilómetros . . . 170
III.4.3. Rejilla utilizada en pruebas de teselado, resolución de 10 kilómetros . . . 171
III.4.4. Rejilla utilizada en pruebas de teselado, resolución de 5 kilómetros . . . 172
III.4.5. Rejilla utilizada en pruebas de teselado, resolución de 1 kilómetro . . . 173

III.5.1. Datos originales de población . . . 175
III.5.2. Cobertura autonómica de los datos de población . . . 176
III.5.3. Esfericidad del municipio de Cádiz . . . 178
III.5.4. Esfericidad del municipio de Villablanca . . . 179
III.5.5. Espacios Naturales Protegidos de Andalucía . . . 181

III.6.1. Ránking *Tiobe* de lenguajes de programación . . . 189
III.6.2. Ránking de lenguajes de programación de la encuesta de *Stack Overflow* . 190
III.6.3. Ránking de lenguajes de programación de *GitHub* . . . 190
III.6.4. *Offset* de una tesela . . . 203

III.7.1. Interpretación gráfica de la interpolación *IDW* . . . 251
III.7.2. Visualización de la capacidad de herencia inter-resolución de un conjunto de datos . . . 270
III.7.3. Secuencia de teselado por herencia 1 . . . 271
III.7.4. Secuencia de teselado por herencia 2 . . . 271
III.7.5. Secuencia de teselado por herencia 3 . . . 272
III.7.6. Secuencia de teselado por herencia 4 . . . 272
III.7.7. Secuencia de teselado por herencia 5 . . . 273
III.7.8. Secuencia de teselado por herencia 6 . . . 273
III.7.9. Secuencia de teselado por herencia 7 . . . 274
III.7.10. Reparto de un análisis de teselado entre microservicios trabajadores, resolución 50 km . . . 274
III.7.11. Reparto de un análisis de teselado entre microservicios trabajadores, resolución 10 km . . . 275
III.7.12. Reparto de un análisis de teselado entre microservicios trabajadores, resolución 5 km . . . 275

III.7.13. Reparto de un análisis de teselado entre microservicios trabajadores, resolución 1 km . . . 276
III.7.14. Reparto de un análisis de teselado entre microservicios trabajadores, resolución 500 m . . . 276
III.7.15. Reparto de un análisis de teselado entre microservicios trabajadores, resolución 250 m . . . 277
III.7.16. Reparto de un análisis de teselado entre microservicios trabajadores, resolución 125 m . . . 277
III.7.17. Tratamiento del MDT por medias de puntos dentro de tesela a 125 m . . . 278
III.7.18. Tratamiento del MDT por medias de puntos dentro de tesela a 25 m . . . 279
III.7.19. Tratamiento del MDT por medias de puntos dentro de tesela a 5 m . . . 279
III.7.20. Teselación en Granada . . . 281
III.7.21. Análisis del teselado: Granada . . . 286
III.7.22. Análisis del teselado: Granada, selección de teselas de control . . . 287
III.7.23. Análisis de teselado: Granada, municipios . . . 288
III.7.24. Análisis de teselado: Granada, secciones censales . . . 289
III.7.25. Análisis de teselado: Granada, núcleos de población . . . 290
III.7.26. Análisis de teselado: Granada, población . . . 291
III.7.27. Análisis de teselado: Granada, EENNPP . . . 292
III.7.28. Análisis de teselado: Granada, Hábitats de Interés Comunitario . . . 293
III.7.29. Análisis de teselado: Granada, MDT . . . 294

III.8.1. K-Means sobre la estructura de la población a 10 km . . . 301
III.8.2. K-Means sobre la estructura de la población a 5 km . . . 301
III.8.3. K-Means sobre la estructura de la población a 1 km . . . 302
III.8.4. K-Means sobre la estructura de la población a 500 m . . . 302
III.8.5. K-Means sobre la estructura de la población a 250 m . . . 303
III.8.6. Zonas de entrenamiento para *Random forest* . . . 305
III.8.7. Zona de entrenamiento: Casco Antiguo y Triana y Los Remedios . . . 306
III.8.8. Zona de entrenamiento: Los Bermejales . . . 307
III.8.9. Zona de entrenamiento: barrios nuevos y antiguos en Tomares, Sevilla . . . 308
III.8.10. Clases del Random forest aplicadas a Málaga . . . 309
III.8.11. Clases del Random forest aplicadas a Málaga, detalle . . . 310
III.8.12. Clases del Random forest aplicadas a Lora del Río, Sevilla . . . 311

III.9.1. Visor *Mapbox GL* . . . 319
III.9.2. Visor *Mapbox GL* . . . 320
III.9.3. Visor *Mapbox GL* . . . 321
III.9.4. Visor *Mapbox GL* . . . 322
III.9.5. Visor *Mapbox GL* . . . 322
III.9.6. Visor *Mapbox GL* . . . 323
III.9.7. Visor *Mapbox GL* . . . 323
III.9.8. Visor *Mapbox GL* . . . 324
III.9.9. Visor *Mapbox GL* . . . 324
III.9.10. Visor *Mapbox GL* . . . 325
III.9.11. Visor *CARTO* . . . 327

III.9.12. Visor *CARTO* 328
III.9.13. Visor *CARTO* 329
III.9.14. Visor *CARTO* 330
III.9.15. Visor *CARTO* 331
III.9.16. Visor *CARTO* 332

Índice de tablas

III.5.1. Estadísticas de capas de contexto (poligonales). Fuente: elaboración propia. 182

III.7.1. Variables de adscripción generadas durante la adscripción de los datos de contexto de polígonos de provincias. Fuente: elaboración propia. 241

III.7.2. Extracto del catálogo generado por un análisis de teselado de tipo "DISCRETEPOLYTOPAREA" sobre los municipios de Andalucía, formando el valor del catálogo con el nombre del municipio y su provincia entre paréntesis. Fuente: elaboración propia. 241

III.7.3. Extracto del conjunto de variables generadas por un análisis de teselado de tipo "DISCRETEPOLYAREASUMMARY" sobre los municipios de Andalucía, formando nombre y descripción de las mismas con el nombre y la provincia del municipio. Fuente: elaboración propia. 242

III.7.4. Extracto del conjunto de variables generadas por un análisis de tipo "DISCRETEPOLYAREASUMMARY" sobre las áreas censales de Andalucía, formando nombre y descripción con el código de la sección. Fuente: elaboración propia. 243

III.7.5. Extracto del conjunto de variables generadas por un análisis de tipo "DISCRETEPOLYAREASUMMARY" sobre los núcleos de población de Andalucía, utilizando el nombre del núcleo y su nivel para componer nombre y descripción. Fuente: elaboración propia. 244

III.7.6. Extracto del conjunto de variables generadas por un análisis de tipo "DISCRETEPOLYAREASUMMARY" sobre los Espacios Naturales Protegidos de Andalucía, utilizando el nombre y la categoría del espacio para su nombre y descripción. Fuente: elaboración propia. 245

III.7.7. Extracto del conjunto de variables generadas por un análisis de tipo "DISCRETEPOLYAREASUMMARY" sobre los Hábitats de Interés Comunitario de Andalucía, utilizando el nombre del hábitat para su nombre y descripción. Fuente: elaboración propia. 247

III.7.8. Extracto del conjunto de variables generadas por un análisis de tipo "POINTAGGREGATIONS" sobre la rejilla de población del IECA, simplificándola de polígono a punto mediante el uso de los centroides de las teselas originales, definiendo un conjunto de 14 variables demográficas para los 7 años de datos presentes en los datos originales. Fuente: elaboración propia. 248

III.7.9. Extracto del conjunto de variables generadas por un análisis de tipo "POINTAGGREGATIONS" sobre los datos de catastro. Fuente: elaboración propia. 249

III.7.10. Conjunto de análisis de teselado llevados a cabo, indicando el nivel de resolución alcanzado y el total de variables creadas por cada uno de ellos. Especialmente importante es el resultado de la columna "Tiempo proceso", que recoge el tiempo máximo de proceso de una tesela de 50 kilómetros de lado para el análisis en cuestión. La teselación del MDT a 5 metros (*) ha sido una prueba piloto en una tesela de 10 kilómetros de resolución, siendo el tiempo de calculo previsto una extrapolación a la tesela de 50 kilómetros utilizado en los otros análisis. Fuente: elaboración propia. 266
III.7.11. Número potencial de teselas para una cobertura completa del área de estudio en cada nivel de resolución. La columna "Teselas" indica el total de teselas para el nivel de resolución indicado, mientras que la columna "Total acumulado" muestra el número total de teselas inscritas en la tabla *data* en el caso de cubrir todos los niveles de resolución de forma jerárquica. Fuente: elaboración propia. 267
III.7.12. Resultados de teselación para cada análisis contemplado. Se recogen el número de teselas resultante para cada uno de ellos en cada nivel de resolución. Fuente: elaboración propia. 267
III.7.13. Total de teselas y estadísticas de variables por cada una de ellas en los distintos niveles de resolución. "Total teselas" representa el número total de teselas presentes en el juego final de datos adscritos, "Máximo variables", "Mínimo variables" y "Media variables" son el número máximo, mínimo y de media de variables encontradas para el conjunto de teselas del nivel de resolución, mientras que "Media Sevilla", "Media sierra" y "Media campiña" corresponde a la media de variables por tesela en tres ámbitos distintos: un centro urbano denso, un medio rural serrano y otro de la Campiña del Valle del Guadalquivir. Fuente: elaboración propia. . . 268
III.7.14. Estadísticas de complejidad de los datos de origen con tratamiento poligonal. Se muestra el tamaño original del dato, el número de polígonos presentes tras el procesamiento *ETL* de adaptación para los requerimientos del análisis de teselado, "SP" corresponde a la media del segmento perimetral, "Área" es la media de área de los polígonos y, por último, "Esf." es la media de esfericidad de los mismos. "Cobertura" muestra, en porcentaje, el recubrimiento del área de estudio. Fuente: elaboración propia. 269
III.7.15. Estadísticas de complejidad de los datos de origen con tratamiento puntual. Se muestra el tamaño original del dato, el número de puntos y su densidad en el área de estudio. Fuente: elaboración propia. 269
III.7.16. Comparativa de las variables de adscripción encontradas en los datos de las dos teselas de control de un kilómetro, de coordenadas 4/411/176 (entorno urbano) y 4/424/182 (entorno rural). Fuente: elaboración propia. 282

Índice de códigos

II.6.1. **Código *pyproj* para la transformación de una coordenada.** 124
III.3.1. **Ejemplo de *JSON*.** 150
III.3.2. **Tipo *SQL cell___cell*.** 153
III.3.3. **Tabla *grid*.** 154
III.3.4. **Estructura *JSON* para la descripción de un nivel de resolución de una definición de rejilla.** 155
III.3.5. **Tabla *pg_connection*.** 155
III.3.6. **Tabla *gridder_task*.** 156
III.3.7. **Tabla *variable*.** 157
III.3.8. **Tabla *catalog*.** 157
III.3.9. **Tabla *data*.** 158
III.3.10. **Función *SQL*.** 163
III.3.11. **Función *SQL*.** 163
III.4.1. **Inserción de una rejilla en la base de datos.** 167
III.6.1. **Ejemplo de un programa escrito en *JavaScript*.** 193
III.6.2. **El mismo programa en *TypeScript*.** 194
III.6.3. **Configuración *JSON* de un *GridderTask* de tipo "DISCRETEPOLYTOPAREA".** 218
III.6.4. **Configuración de un *GridderTask* de tipo "DISCRETEPOLYAREASUMMARY".** 222
III.6.5. **Configuración de un *GridderTask* de tipo "POINTIDW".** 225
III.6.6. **Configuración de un *GridderTask* de tipo "POINTAGGREGATIONS".** 228
III.6.7. **Configuración de un *GridderTask* de tipo "MDTPROCESSING".** 232
III.7.1. **Ejemplo de *JSON* de configuración de la herramienta *CLI gridderseтup*.** 236
III.7.2. **Ejemplo de *JSON* de configuración de la herramienta *CLI gridder*.** 238
III.7.3. ***JSON* con las características de una rejilla para las entradas *API GET*, *POST* y *PATCH* de la familia de entradas */grid*.** 255
III.7.4. ***JSON* con las características de un origen de datos *PostgreSQL* para las entradas *API GET*, *POST* y *PATCH* de la familia de entradas */sourcepg*.** 256
III.7.5. ***JSON* con las características de un análisis de adscripción para las entradas *API GET*, *POST* y *PATCH* de la familia de entradas */griddertask*.** 256

III.7.6. ***JSON*** **con una petición de procesamiento de teselas para un análisis de teselado para la entrada** ***API GET /gridderjob/:gridderJobId.*** 258
III.7.7. ***JSON*** **con las características de una variable de adscripción entradas** ***API*** **de la familia** ***/variable.*** 260
III.7.8. ***JSON*** **con un catálogo tal y como lo devuelve la entrada** ***API /catalog/:gridderTaskId/:variableKey.*** 260
III.7.9. ***JSON*** **con una petición de procesamiento de** ***Machine Learning*** **para un vector de adscripción la familia de entradas** ***API /ml-job/:mlJobId.*** 261
III.7.10. ***JSON*** **con una petición de teselas para la entrada** ***API GET /query.*** 262
III.7.11. ***JSON*** **de respuesta de la entrada** ***API GET /query.*** 263
III.7.12. Datos de adscripción de la tesela de control urbana 4/411/176. . 280
III.7.13. Datos de adscripción de la tesela de control rural 4/424/182. . . 282
III.8.1. ***JSON*** **que representa las zonas de entrenamiento para la clase** ***RandomForestMlTask.*** 297

Parte I

PLANTEAMIENTOS GENERALES

I.1 Justificación y contextualización

> Lo perfecto es enemigo de lo bueno.
>
> *Voltaire*

Esta tesis tiene como objetivo general el diseño y creación de una plataforma *cloud* para la gestión de estructuras de rejillas (teselas) multiescalares asimétricas. Las rejillas multiescalares abstraen la información geográfica en una estructura de información que facilita el análisis temático multicriterio, así como proporcionan un medio para el almacenamiento eficiente de información y su tránsito entre servidores y clientes. Adicionalmente constituye una base firme sobre la que implementar muchos métodos de computación paralela sobre información geográfica, difíciles de implementar sobre la información vectorial dado su inherente propiedad de interrelación topológica. La plataforma de *software*, que hemos denominado *Cell*, constituye un sistema integral basado en *Software Libre* (Lakhani et al., 2003) para construir, gestionar, analizar y visualizar información geográfica almacenada en este nuevo modelo de datos geográficos propuesto. Esta tesis, por tanto, aborda una posible solución técnica a un clásico problema del análisis espacial: el de la integración espacial de datos multitemáticos para el análisis multicriterio. Además, como se verá, la solución propuesta permite adaptar la información geográfica a la estructura de datos nativa para los procedimientos de *Machine Learning*, que copan el interés de las técnicas de análisis avanzado de datos en los últimos años (Hastie, 2001).

La presente tesis Doctoral, presentada en el contexto de un Programa de Doctorado en Geografía, puede parecer, a priori, un tanto peculiar para la disciplina. Esto se debe a estar centrada en el desarrollo de un *software*, un conjunto de programas informáticos que prototipan e implementan los conceptos y objetivos del presente trabajo. Es, por tanto, una tesis Doctoral que se enmarca en el campo de las Tecnologías de la Información Geográfica (TIG), es decir, en el cruce de las disciplinas geográfica y de la ingeniería de *software*. Como tal, el presente documento debe ser apreciado sólo como una parte del trabajo, siendo el desarrollo de un prototipo de plataforma de *software*, llamado *Cell*, el otro producto de investigación que acompaña a éste documento y sin el cual no tiene sentido. Desarrollar en un documento como este todo el código generado para la programación de dicha plataforma carece de sentido, puesto que sería un documento del doble de páginas que el presente. Existen excelentes herramientas en *Internet* no sólo para la publicación de código abierto, como es el caso del *software* descrito en este documento, sino también para exponerlo a

la comunidad de desarrolladores y usuarios potenciales e incentivar la colaboración en el marco del *Software Libre*.

Por lo tanto, animamos al lector, aunque no tenga experiencia en el desarrollo de *software*, a visitar los distintos repositorios de *software* que se encuentran indexados en:

```
https://cell.37north.io
```

donde se detalla la función y características de cada uno de ellos.

Las TIG constituyen unas herramientas muy potentes para el desarrollo de la Geografía. En este campo, los geógrafos tenemos la tremenda suerte de que el *Software Libre* ha calado con fuerza (Coetzee et al., 2020), existiendo un completo, variado y útil ecosistema de programas, *frameworks* y librerías informáticas para llevar a cabo casi cualquier proyecto que se pueda necesitar. La disponibilidad de librerías, el constituyente básico de los programas informáticos, de alta calidad en nuestro campo, hace que el desarrollo de nuevos conceptos, ideas y prototipos sean planteables en términos de recursos y tiempo asequibles a nivel de pequeños equipos e incluso personal, como es este caso. Sistemas como *PostGIS*, *QGIS*, *GRASS* o *GDAL / OGR* ponen a disposición del desarrollador de TIG una panoplia de funcionalidad y capacidades que hace 20 años sólo eran posibles con caras licencias comerciales. Tributarios de estas capacidades son las plataformas *Software as a Service* comerciales que han crecido alrededor de este ecosistema, acercando al no desarrollador sus capacidades en entornos de fácil uso y funcionalidad. Póngase como ejemplo de ellos *CARTO* (CARTO Contributors, 2021a) o *Mapbox* (Mapbox, 2021a).

I.1.1. Una primera aclaración terminológica: "rejilla", "tesela" y "rejilla multiescalar"

En este trabajo de tesis se alude constantemente a los términos "rejilla" y "tesela", por lo que es necesario, antes de entrar en materia, dejarlos bien acotados.

Entendemos por ***rejilla*** a una estructura regular de unidades territoriales o espaciales, organizadas según un patrón geométrico repetitivo, que tiene unas características de forma, medida y extensión bien definidas por unas reglas geométricas determinadas de antemano, y que por lo tanto pueden definirse matemáticamente. La rejilla, como conjunto de unidades regulares territoriales o espaciales, segmenta una región del espacio geográfico en unidades medibles, deterministas y de forma, posición y tamaño fácilmente predecibles gracias a las reglas geométricas que determinan la confección de la rejilla.

Dichas unidades espaciales regulares que crea la rejilla sobre el espacio es lo que en este trabajo de tesis denominamos por ***teselas***. A veces, en el texto, se utiliza ***celda*** como término sinónimo. Las teselas de las que están compuestas los modelos de rejilla más populares son cuadrados o rectángulos, hexágonos o triángulos, pero éstos elementos geométricos tienen que tener siempre el mismo tamaño en una rejilla determinada.

La ***rejilla multiescalar***, por su parte, es un término desarrollado en esta tesis y que hace referencia a una estructura de datos en la que se usan varias rejillas, entendidas según se ha definido en este apartado, de tamaños distintos (100 kilómetros, 10 kilómetros, 250 metros, por ejemplo) pero que trabajan conjuntamente para estructurar los datos geográficos según las características y fines que se desarrollarán en esta tesis.

I.1.2. El alcance de las TIG en el curriculum de la Geografía

El debate acerca del papel de las TIG en la Geografía es una constante en la disciplina desde hace ya algunos años (Chuvieco et al., 2005). Para plantearnos la cuestión, el primer consenso que habría que afrontar es claro y obvio: ¿consideramos que las computadoras pueden ser de ayuda al quehacer geográfico? Con esto no incluimos tareas computacionales evidentes como la redacción de textos, sino al análisis de datos geográficos en su concepción más actual del mismo, lo que se ha dado en llamar *ciencia de datos*. Si la respuesta es no, el debate termina inmediatamente. Si la respuesta es sí, la respuesta es que el análisis geografico debe entrar en una fase instrumental más tecnificada en la que el dato, tratado con rigor, sea un eje fundamental, entonces tenemos que abordar qué alcance deben tener las TIG en la instrumentalización de la disciplina.

Hoy en día la tecnología es abundante y está por todos lados. Existen tecnologías para hacer de todo, muchas de ellas en el campo del *Software Libre* y, por tanto, accesibles de forma gratuíta en cuanto a inversión económica se refiere[1]. Existen tantos sistemas, tantas opciones y alternativas para abordar problemas en la llamada *ciencia de datos* que muchas veces se pierde la perspectiva y se cae en el error de considerar la tecnología como un fin en si misma, en lugar de ser una herramienta al servicio de un fin científico. No es difícil ver en el maremagnum de cursos de *Internet* cursos que se publicitan con el lema "convierteté en un científico de datos". ¿En un científico de datos de qué? En muchas ocasiones, los alumnos preguntan como hacer un currículum. Muchos currículums se están convirtiendo en una lista enorme de proclamas de conocer tal o cual sistema informático, pero sin reseñar su campo de aplicación. El conocimiento temático es fundamental, sin conocimiento temático la aplicación de tecnología no tiene sentido. Muchas veces pareciera, a tenor de lo transmitido en infinidad de blogs técnicos de *Internet* y cursos, que la aplicación de esta o aquella tecnología de última generación (sobre todo en el campo del *Machine Learning*) hará de la comprensión e interpretación de los datos un proceso cuasi-místico en el que la tecnología resuelve el problema sólo. Esto es un gran error, por supuesto. La irracionalidad de aplicar tecnologías muchas veces sobredimensionadas a problemas con una interpretación y profundidad temática considerable se justifica por la propia aplicación de la tecnología. Más vale un buen especialista temático con una buena base estadística y una hoja *Excel* que un no especialista con todo el arsenal de *Inteligencia Artificial* y *Machine Learning* a sus espaldas. Claro está que un buen especialista temático con una buena base estadística y con capacidad instrumental de ese nivel sería lo deseable.

[1] Que no en inversión de tiempo, claro. Aprender una tecnología nueva lleva tiempo, sea o no de *Software Libre*, por eso hay que tener muy claro que la apuesta que estamos haciendo en esa inversión de tiempo es productiva. No todo el *Software Libre* lo es, al igual que el comercial.

En este sentido, esta tesis se enmarca en un contexto muy instrumental, precisamente más en la línea de una solución tecnológica instrumental que creemos puede abrir nuevas capacidades analíticas para los especialistas temáticos.

I.1.3. "¿Deberían los geógrafos aprender a programar?"

En el mundo de las empresas tecnológicas, la fusión de conocimiento temático más capacitación técnica avanzada es el perfil profesional más buscado. En este contexto profesional, ¿hasta qué punto debería llegar la formación de un geógrafo en el uso y posible desarrollo de *software*, más allá de lo que incorporan actualmente los currículos académicos en materia de SIG de escritorio? (Chuvieco et al., 2005, Toro Bonilla, 2014, Etherington, 2016). El título de este apartado hace referencia a un hilo de discusión recurrente en foros en el que muchos geógrafos involucrados en empresas tecnológicas aportan sus experiencias y opinión.

Para empezar, existe un consenso en los currículos académicos en que la formación instrumental de un geógrafo debe cubrir una buena capacitación en Sistemas de Información Geográfica tradicionales de escritorio. Ya sea con plataformas comerciales (*ArcGIS*) o con plataformas de *Software Libre* (*QGIS*), el geógrafo debería ser capaz de aprovechar las capacidades analíticas de estos sistemas lo máximo posible. Su entorno amigable de interfaz gráfico "con botones" hace que la curva de aprendizaje sea aceptablemente suave.

El problema de los interfaces gráficos "con botones" es que esos "botones" están programados para cumplir tareas muy específicas y, por tanto, altamente inflexibles. Por mucho que una marca comercial o de *Software Libre* se empeñe en hacer estos sistemas lo más flexibles posible, siempre hay un límite a dicha flexibilidad. Siempre habrá una eventualidad en la que querríamos que el "botón" hiciera las cosas ligeramente distintas, o nos encontraremos con que el programa no es capaz de reproducir un determinado análisis que tengamos en mente. Pasa igual con sistemas como, por ejemplo, *Excel*: a pesar de la tremenda flexibilidad que demuestra a la hora de componer graficos estadísticos, siempre se llega a un punto en el que la modificación pretendida en el gráfico simplemente no es posible. Ningún programador pensó que nadie querría hacer esta u esta otra cosa con un gráfico estándar.

Por lo tanto, llegados a este punto, ya hay que pasar al siguiente nivel, y aquí es donde se suele poner esa frontera a la que hace referencia el título: la programación. Esta palabra asusta, y con razón, porque la programación de ordenadores no es una actividad con una curva de aprendizaje precisamente suave. Pero lo importante es entender qué queremos decir con *programación*.

Programación es un término genérico que se usa en el contexto de darle a un sistema una nueva funcionalidad que no posee o crear desde cero (o casi cero) un nuevo sistema que haga algo diferente. Para conseguir estos objetivos, sin embargo, las posibilidades son múltiples. Podemos, por tanto, identificar varias etapas o modalidades de *programación*, que podemos ordenar de más fácil a más difícil:

1. la "programación"[2] de análisis de datos en lenguajes de consulta altamente expresivos como *SQL*;

2. la programación (esto ya sí es programación de verdad) de flujos de análisis de datos en formato *Notebook* interactivos en sistemas como *Python* o *R*;

3. la programación de módulos para sistemas ya existentes, como por ejemplo la creación de extensiones para sistemas de escritorio como *Excel* o, en el campo de los SIG, *QGIS* o *ArcGIS*;

4. la programación de nuevos sistemas desde cero[3], usualmente altamente complejos.

Las dos últimas "modalidades" de programación caen en el ámbito de la ingeniería de *Software* y por tanto precisan de una capacitación profesional acorde. Sin embargo, los dos primeros están en el ámbito de eso que se ha dado en llamar *ciencia de datos* y que es accesible para todos los ámbitos científicos. De hecho, se podría reformular este apartado con el título *"¿Qué sector del saber está libre de la necesidad de programar?"* o similar. No ya sólo las ciencias físicas, de la salud, la biología, de la tierra o las sociales, sino también en campos como la filología (el procesamiento de lenguaje natural es una línea estrella en IA) o el derecho (Dyevre, 2020, Hidalgo, 2019). Todos los aspectos del saber pueden beneficiarse de estas nuevas técnicas y herramientas.

El acceso a esas dos primeras modalidades no tiene una curva de aprendizaje tan marcada y las ventajas y el potencial instrumental que ofrecen superan con creces lo que pueden aportar los conjuntos de "botones" de cualquier SIG de escritorio. Flujos de trabajo que pueden afectar a decenas de operaciones en un SIG de escritorio se pueden resolver unas decenas de líneas de código de consultas *SQL* o con un pequeño *script Python* o *R* utilizando las librerías adecuadas. Esta forma de abordar instrumentalmente el análisis tiene una gran ventaja, que es capital en la práctica de las estas técnicas analíticas: la reproducibilidad. Si bien es cierto que los SIG de escritorio proporcionan ciertas herramientas para fomentar hasta cierto punto esta reproducibilidad (por ejemplo, el módulo *Graphical Modeler* para *QGIS* o el *Model Builder* de *ArcGIS*), son herramientas, lógicamente, muy ligadas a estos sistemas y no demasiado portátiles. Un *script Python* o *R* es ejecutable en cualquier sistema con exactamente el mismo entorno de ejecución gracias a tecnologías como *Docker*.

Especialmente importante es la formación en la primera "modalidad de programación", el lenguaje *SQL*. Esta herramienta no sólo proporcionaría a los geógrafos, gracias a sistemas de *Software Libre* como *PostGIS*, una extraordinaria capacidad analítica, sino que también

[2] La manipulación de datos con el lenguaje *SQL* no es estrictamente programación, ya que es un lenguaje de tipo *declarativo*, es decir, mediante el cual a la máquina se le solicita un resultado, sin indicarle cómo debe llegar a él (lenguajes *imperativos*), pero se suele percibir como tal por el público no especialista por su uso de código.

[3] Hoy en día, casi ningún nuevo sistema informático se crea *desde cero*. Un programa informático moderno se crea por composición de las funcionalidades aportadas por *librerías*, que son conjuntos de funcionalidad programados por terceros dentro de un ámbito temático determinado. Por ejemplo, si se quiere hacer un programa que sea capaz de reproyectar coordenadas, incluiré en el mismo la librería *PROJ*, que resuelve este complejo problema, y si se quiere dibujar escenas en 3D pues recurriré a un motor gráfico especializado como *VTK*, por citar uno.

traería, como efecto necesario, un mayor sentido de *Cultura del Dato* a la disciplina, es decir, una mayor consciencia de que los datos, si se quiere que sean útiles en un contexto computacional, tienen que ser recogidos, codificados y tratados con reglas muy esctrictas, no sólo desde el punto de vista estadístico, por supuesto, sino también desde el punto de vista formal, ya que los ordenadores no son capaces de procesar información en *cualquier* estado.

La importancia de llegar a esa primera modalidad de "programación" se debe a que la formación en SIG de escritorio tradicionales no consigue dar el salto conceptual en el entendimiento de los datos al nivel requerido. Dado que la mayoría de los formatos que se usan frecuentemente en estos sistemas son de tipo monotabla (la "mentalidad SHAPEFILE"), el geógrafo en formación acaba adquiriendo muchísimos vicios relacionados con una mala estructuración de los datos, sobre todo la alta redundancia en los mismos y la falta de atomicidad. Las limitaciones de esta visión sobre los datos son demasiado estrictas para llegar a soluciones de modelado que propicien un análisis más avanzado. Como resultado de ello, ese salto conceptual a las estructuras de datos multitabla, donde el modelo está bien *normalizado*, es imposible, dejando al geógrafo en una posición de desventaja a la hora de enfrentarse a proyectos reales del mundo de las TIG donde la solidez del modelo de datos es fundamental. Este paso no sólo le aporta al geógrafo una gran capacidad analítica, sino que es una parte insoslayable de la educación en técnicas de análisis de datos para dar el salto al resto del ecosistema de herramientas de este campo. Este nivel aporta una capacidad de análisis de datos, tanto en capacidad como en volumen, muy superior al ofertado por la tradicional aproximación de los SIG de escritorio.

Con una buena base en el primer nivel descrito el geógrafo tendría un arsenal de herramientas a su disposición muy avanzado para abordar las técnicas tradicionales de geoprocesamiento, para poder así implementar las técnicas clásicas de análisis geográfico de una forma más eficiente, flexible, reproducible y capaz de abordar cantidades masivas de datos. Sin embargo, las nuevas técnicas de *Machine Learning* y *Big Data* (Gutiérrez Puebla, García Palomares y Salas Olmedo, 2016, Gutiérrez Puebla, García Palomares, Romanillos et al., 2017) se apoyan en estos sistemas para realizar una parte del trabajo (repositorio, preparación de datos y difusión de los mismos), pero éstas técnicas más novedosas ya no se realizan en el seno de una base de datos relacional geográfica, sino que se llevan a cabo utilizando diversas librerías de *software* en lenguajes muy sencillos de aprender como *Python* o *R*. Estos lenguajes, en este contexto, se utilizan de una forma *declarativa*, es decir, que no requieren unos conocimientos muy avanzados de ingeniería de *software* para ser utilizados, ya que el usuario se limita a crear *scripts*, que son programas enfocados no a la construcción de un sistema reutilizable de *software*, sino a la ejecución puntual de un flujo de trabajo. En este modo de programación, el usuario no está construyendo un programa propiamente dicho, actividad que quedaría para los dos últimos niveles descritos, sino que está usando, de una forma muy sencilla y hasta mecánica, las funcionalidades de dichas librerías, en un entorno muy interactivo, para la exploración y análisis de datos. Tecnologías como los *Jupyter Notebooks* (figuras I.1.1 y I.1.2) (Project Jupyter Contributors, 2021), que pueden ser utilizados con estos dos lenguajes, proporcionan un entorno de computación científica que permite combinar, de forma muy didáctica, código de manipulación y visualización de datos con explicaciones, con objeto de que el lector

pueda no sólo seguir el discurso científico de la publicación, sino también experimentar con ella en tiempo real. Su aplicación didáctica es enorme (Rey et al., 2015).

I.1.4. La importancia la estructura en los datos

Si ya hemos aceptado la utilidad de las computadoras en el quehacer del análisis geográfico, hemos de aprender a comunicarnos con ellas de forma eficiente. Los ordenadores[4] tienen ciertas características que los hacen muy diferentes a las personas a la hora de abordar datos y problemas:

1. son máquinas **finitas**, porque sus recursos lo son, como por ejemplo la memoria. Las máquinas de hoy en día están tan avanzadas que parece que sus recursos son a veces inagotables, pero se agotan;

2. son máquinas **discretas**, es decir, están sujetas a una precisión numérica finita más allá de la cual no pueden ir, por lo que finalmente un número debe caer a un lado u otro de dicha precisión. Igual que en el punto anterior, las capacidad de las modernas máquinas son tan inmensas que este efecto no se nota en la enorme mayoría de las aplicaciones;

3. son máquinas **deterministas**, basadas en los famosos "ceros y unos", por lo que la probabilidad no se encuentra en su núcleo analítico más básico.

En resumen, todo esto, a efectos prácticos, significa que a la máquina hay que darle los datos de una forma unívoca, precisa y desagregada. Esto se consigue creando estructuras de datos.

Para los ordenadores, la estructura en el dato es la base de su procesamiento. A las personas nos cuesta verlo porque precisamente nuestra unidad de procesamiento, el cerebro, ha evolucionado con exactamente el funcionamiento opuesto al de las máquinas: somos capaces de entender y manejar simbólicamente conceptos como infinito o el dominio de los números reales, somos capaces de entender de forma natural el concepto de *probabilidad* y nuestro cerebro es altamente resistente a la información parcialmente incompleta y/o errónea, rellenando fácilmente los huecos en la misma y detectando y subsanando de forma natural e inconscientemente inconsistencias en su estructura. Por eso, para comunicarnos información, un *Post-It* lleno de garabatos nos sirve. Hacemos diagramas constantemente, nos apoyamos en pizarras en las que dibujamos representaciones simbólicas de conceptos complejos y todo esto lo acompañamos, encima, con el lenguaje oral y gestual humano. Teniendo en cuenta que una de las ramas con más actividad en Inteligencia Artificial es que las máquinas entiendan nuestra escritura manual, la distancia que nos separa es grande.

[4] "Ordenador" es el término que se le suele dar en la variante castellana del español a las computadoras. Es el término preponderante en esta variante, mientras que en las variantes latinoamericanas el término más común es "computadora". En puridad, "computadora" da una mejor visión general de lo que hace la máquina, puesto que "computa" operaciones numéricas. Sin embargo, el término castellano hace referencia a otra capacidad analítica estrella de estas máquinas, la capacidad de "ordenar" datos. Veremos a lo largo de esta tesis que esa capacidad, la de "ordenar" datos, es de suma importancia para el rendimiento en análisis de datos.

Figura I.1.1: En un *Jupyter Notebook* los contenidos se estructuran en celdas de contenido, que pueden servir para desarrollar texto en el que se explica qué hace el cuaderno o para introducir código reproducible *Python* o *R*. En la imagen, un cuaderno destinado a explicar cómo funciona la clusterización *K-Means* con una aplicación a teledetección. El lector puede ir ejecutando las celdas de código, pero lo más importante es que puede modificarlas para experimentar con el ejercicio o el flujo de proceso planteado. Fuente: elaboración propia.

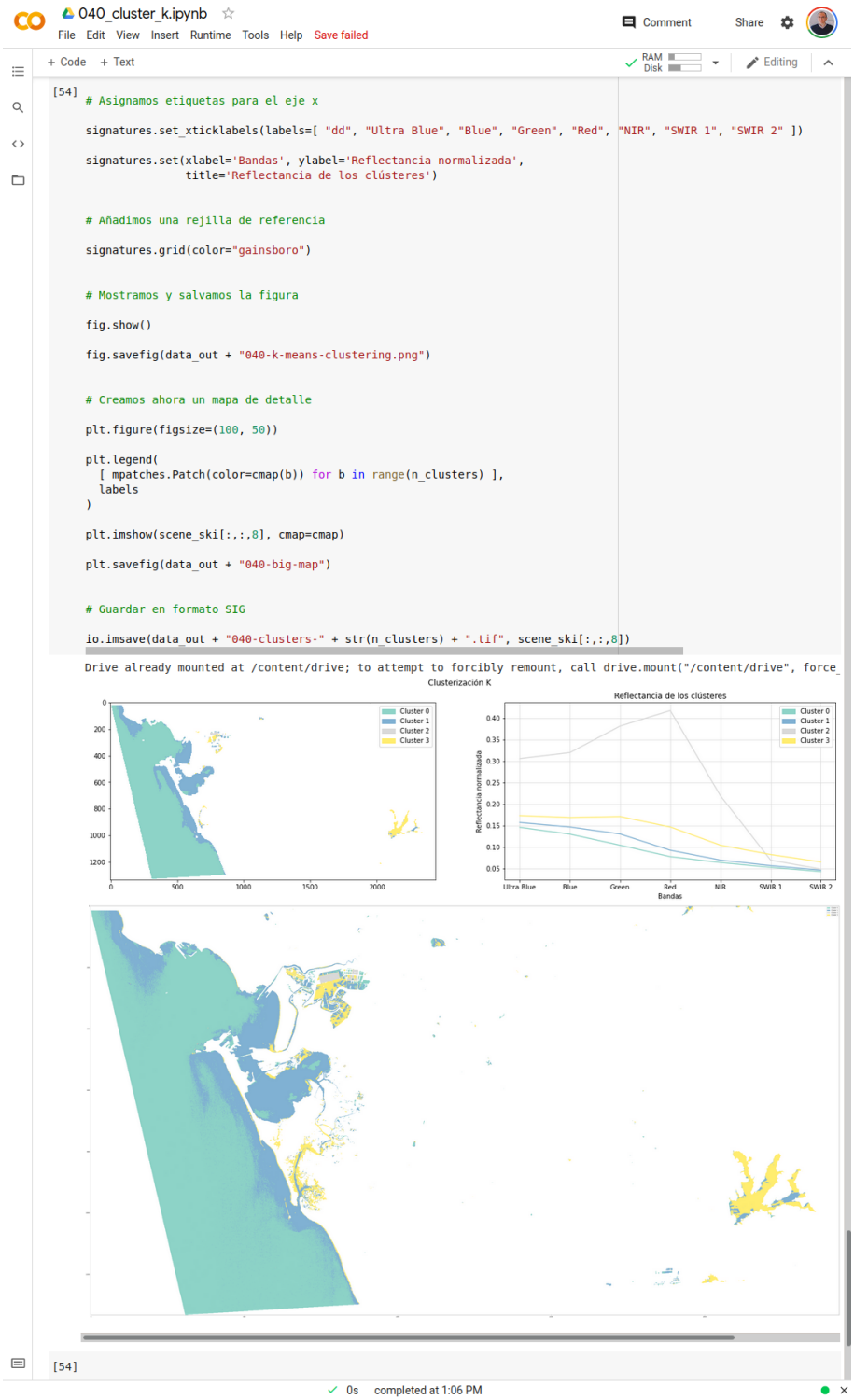

Figura I.1.2: Los cuadernos *Jupyter* combinan eficientemente visualizaciones de datos complejas, incluidos mapas dinámicos, con el código de tratamiento de datos para la exploración de resultados finales o parciales. En la imagen, estudio de una clusterización *K-Means* sobre los tipos de agua encontrados en una escena Landsat, con una comparación de las signaturas espectrales encontradas en cada *cluster*. Fuente: elaboración propia.

Las personas también estamos constantemente estructurando la información. Por ejemplo, la primera estructura de información que aprende un niño es la lista. Después, el siguiente paso, es estructurar listas en filas, con lo que tenemos una tabla con coordenadas X e Y en las que estructuramos la información mediante cruces de filas y columnas, y ya, después, entramos en estructuraciones más complejas como gráficos estadísticos o cartografía. En todas las ciencias, estructurar la información es parte de la metodología, ya sea en ciencias naturales o sociales. La creación de buenas encuestas, por ejemplo, es un ejemplo de estructuración de la información en ciencias sociales. Le damos estructura en forma de ítems, en lugar de transcribir una conversación a modo de entrevista, porque el objetivo de la encuesta es la homologación de las respuestas para posibilitar su contraste. Esta es la mentalidad de la máquina, y esto se consigue creando *estructuras de datos*.

Obviamente, todo lo que sucede en una máquina está regido por datos. La confección del texto de esta tesis se realiza por la introducción manual de secuencias de datos textuales que juntos componen el texto. Esos datos van a la memoria del ordenador y éste es capaz de usarlos, por ejemplo, para transformar posteriormente en PDF lo escrito en su memoria. El ordenador está siempre gestionando datos mientras trabaja.

Eso no quiere decir, no obstante, que el ordenador *pueda* hacer un uso sofisticado de estos datos textuales. El ordenador no entiende de qué va el texto, ni puede extraer ideas o palabras clave del mismo. En bases de datos documentales, los textos van acompañados de unos *metadatos* introducidos por expertos temáticos mediante los cuales se le comunica a la máquina, de forma explícita, los contenidos fundamentales del texto cuya salvaguarda se le encomienda. En general, una forma de expresión como el lenguaje natural humano tiene una complejidad tal que, aunque son datos estructurados, puesto que nos comunicamos con ellos, sus reglas son tan tremendamente intrincadas que sólo con mucho esfuerzo las aplicaciones de Inteligencia Artificial más sofisticadas están comenzando a ser capaces de extraer el significado de textos en lenguaje natural sencillo. Désele a dicho texto giros del lenguaje y/o referencias culturales (como un texto literario, no digamos poético) y la máquina volverá a estar completamente perdida.

Por tanto, el primer objetivo es crear dichas estructuras de datos. Las estructuras de datos estarán especialmente diseñadas para un fin y ámbito temático concreto.

I.1.5. La información geográfica

Ya colocados en el ámbito de la geográfia, para crear estructuras de datos para nuestro campo lo primero que hay que hacer es examinar qué características tiene la información geográfica. La información en los distintos campos del conocimiento tienen sus propias características. Por ejemplo, en la información meteorológica, la variable "tiempo" juega un papel fundamental, ya que es un hecho altamente dinámico y cambiante, y es precisamente ese cambio y su predicción lo que ocupa gran parte de la actividad analítica en la disciplina.

En información geográfica, por lo tanto, lo primero que salta a la atención de un analista es que la información geográfica tiene una ***naturaleza dual***:

1. por un lado, la información geográfica tiene una vertiente **geométrica:** las cosas

suceden o se emplazan en un lugar y, además, depende de la escala considerada o de la utilidad que se le quiera dar al dato, esa ubicación puede ser meramente adimensional, sin forma ni extensión (un punto) o tener estas características;

2. por otro, la información geográfica tiene una vertiente **temática** o **alfanumérica**, es decir, los objetos y fenómenos que emplazamos sobre el territorio tienen características no territoriales asociadas. La toponimia es uno de los ejemplos más claros.

I.1.6. Realidad, sistemas y modelos

Como en toda actividad científica, el objetivo de explicar, opcionalmente controlar y predecir la deriva futura de un fenómeno real pasa por la *sistematización* del mismo. Un *sistema* es un conjunto de elementos identificados a partir del estudio del fenomeno real que están vinculados por relaciones de intercambio de energía, materia y/o información, formando un todo en el que el estado total del sistema no puede ser definido por el estado aislado de algunos de sus componentes. Un clima, por ejemplo, no puede ser caracterizado teniendo en cuenta sólo el régimen de temperaturas.

A partir de ese sistema se crea un modelo, que es una representación en base a información descriptiva de un conjunto de los elementos del sistema y sus relaciones destinados a su cuantificación. En el modelo, se toman en consideración sólo los elementos de información de los componentes que son de interés para la resolución de una tarea analítica concreta. Por ejemplo, de la gravitación hay dos modelos, el cuántico y el clásico newtoniano. Mientras que con el segundo se consigue describir la mecánica celeste del Sistema Solar con una sóla ecuación, éste modelo se desmorona al tener en cuenta los efectos cuánticos de la materia a la hora de estudiar las estructuras subatómicas, por lo que hay que crear un nuevo modelo, el cuántico. El fenómeno es el mismo, pero utilizamos dos modelos bien distintos para su comprensión. No existe, por lo tanto, el modelo perfecto, sólo modelos convenientes a unas aplicaciones u otras.

El término "modelo" toma un significado muy concreto en ingeniería de *software*, ya que por modelo entendemos en esta disciplina la implementación, ya sea como elementos de código de un programa o como estructuras de datos destinadas a ser gestionadas por estos últimos, de las características de datos descriptivas de los objetos reales que se pretenden simular en el programa o de su comportamiento. Existen metodologías de diseño de programas que trabajan alrededor de este concepto, como por ejemplo la "programación orientada a objetos", donde la programación se centra en emular con código unidades funcionales con un alto nivel de independencia que definen características y propiedades funcionales de los elementos del sistema y cómo éstos establecen relaciones entre sí.

Por tanto, en ingeniería de *software*, y dadas las características anteriormente descritas de las máquinas, la creación de estos modelos, tanto a la hora de diseñar los programas como las estructuras de información que los hacen funcionar, es de suma importancia, ya que, de no existir o hacerlo sin el detalle suficiente, simplemente no se conseguirá el objetivo de hacer funcionar a la máquina para el fin que pretendemos. Los modelos de programas y de datos ayudan y permiten a la máquina entender el sistema del mundo real en base a un modelo concebido por el desarrollador.

I.1.7. Modelos de datos en Geografía

La Geografía comienza su andadura en este campo con los conocidos desarrollos llevados a cabo por Roger Tomlinson[5](Geographic Information Science, 2020) en el curso de sus trabajos para el "Inventario de Usos del Suelo de Canadá" (*Canada Land Inventory*) en la década de los 60. Como parte de estos trabajos Tomlinson y su equipo tuvieron que idear y desarrollar estructuras de datos que fueran las adecuadas para la consignación, manejo, análisis y almacenaje de la información geográfica.

Lógicamente, para muchas disciplinas, las estructuras de datos computacionales emergen de estructuras de datos analógicas utilizadas previamente en la disciplina. Dichas estructuras de datos analógicas son las formas de organizar la información en soportes no digitales para su comunicación y entendimiento por otros especialistas. De esta manera, tenemos en todas las disciplinas científicas muchas metodologías distintas que implican la creación de gráficos, esquemas, formulaciones y demás maneras de organización de información compleja que ayuda a su sistematización y efectiva comunicación. En Geografía, por supuesto, la cartografía es el medio de comunicación y sistematización de análisis y datos más utilizado.

Al igual que muchas otras técnicas gráfico-esquemáticas de otras disciplinas, la cartografía es un procedimiento con un algo grado de abstracción de la información. La cartografía impresa está hecha, en origen, para ser interpretada con una sólida base de conocimientos semiológicos y técnicos[6]. Si observamos el espacio geográfico, que se compone de infinitos puntos, veremos que existe un concepto complejo que se encuentra en el núcleo de incontables debates en ciencia territorial: el límite. ¿Dónde está el límite de una ciudad? ¿y el límite de las riberas de un río? ¿cómo definimos la línea de costa? En la naturaleza los límites claros, nítidos y bien definidos son escasos, ya que lo que predominan son las transiciones. En las actividades humanas sobre el territorio, muchas veces los límites están bien reglados, como por ejemplo en el caso de los sistemas catastrales, pero en muchos otros fenómenos, como por ejemplo qué se considera urbano y qué no, no tanto. El espacio geográfico está lleno de transiciones complejas de interpretar. Y al igual que en dichas otras disciplinas, uno de los esfuerzos de la actividad científica es intentar, dentro de lo posible, definir o caracterizar esas transiciones para alcanzar cotas de abstracción superiores que permitan el avance y la sistematización de la realidad (Robinson et al., 1987).

Una de las tareas que hace la cartografía es tomar decisiones acerca de dónde se ubican esos límites. Dado que sería poco práctica una cartografía generalista que no delimitara de forma determinista la línea de costa, en ese caso el cartógrafo llega a un compromiso en el que determina, con una línea bien definida, y por tanto imponiendo un criterio binario tierra emergida / mar, dos zonas que en realidad están separadas por una transición mucho más compleja. Por tanto, la práctica de la cartografía es de por sí, en muchas ocasiones, un ejercicio de sistematización y simplificación de un fenómeno complejo, el espacio geográfico, que comienza a ser compatible con las necesidades de la máquina, así que parece ser un

[5]Tomlinson (1933 - 2014) es conocido como "El Padre de los Sistemas de Información Geográfica"

[6]Muchas veces, muy especializada en un sistema territorial concreto, como por ejemplo la cartografía geomorfológica o de navegación aérea, por poner dos ejemplos.

buen punto de partida. Cartografíar es sistematizar, cartografíar es modelar, cartografiar es imponer reglas para alcanzar un alto grado de determinismo territorial, tomando decisiones complejas de tipo si / no en base a un conocimiento científico. Incluso en la caracterización de transiciones o *buffers* se hace este ejercicio, graduando la transición en tramos bien delimitados, ajustando un sistema con potencialmente una precisión infinita a un modelo discreto y finito de posibilidades[7]. Es lo que un modelo informático de datos precisa como ejercicio previo de conceptualización del mismo.

Asimismo, un analista de información geográfica jamás debe de perder de vista el otro gran factor que la afecta: la escala. La escala determina, entre muchísimas otras cosas, cómo establecemos estos límites, su precisión o significación, y también incluso hasta que punto tiene sentido la delimitación del límite en sí mismo. Pongamos por ejemplo el caso de las ciudades. A una escala regional tiene sentido plantearse la determinación del límite entre lo que consideramos espacio urbano y lo que no, trazando una línea que delimita ambos espacios. Sin embargo, si consideramos una escala nacional o continental, la significación de dicho límite está muy por debajo de la escala del mapa: por muy grande que sea una ciudad, la determinación de sus límites se escapa de la significación de los objetivos plantedos por la cartografía. En estos casos, la ciudad suele ser representada por un simple punto, una ubicación que se ha considerado como significativa para abstraer la extensión real de la ciudad y marcar su posición. El punto, en cartografía, es adimensional y sin extensión. Simplemente designa la posición de un fenómeno, pero abstrayendo su forma y extensión. Es un definitivo ejercicio de discretización de información continua y compleja.

La escala también es importante porque puede tener un efecto determinante en la vertiente alfanumérica de la información geográfica. A medida que se aumenta la escala, por ejemplo desde una perspectiva autonómica a una municipal o una de sección censal, los datos alfanuméricos también van aumentando en definición, por lo que la distribución estadística de la información va variando, cambiando sus patrones, teniendo serias repercusiones en su interpretación. Este efecto se conoce como el *Problema de la Unidad Espacial Modificable*, o *MAUP* por sus siglas en inglés (*Modifyable Areal Unit Problem*). La estructura propuesta en esta tesis permite ver este efecto en acción y esperamos que contribuya a su comprensión e interpretación (Chasco Yrigoyen, 2003, Openshaw et al., 1979, Openshaw et al., 1981).

Por tanto, ya en la cartografía clásica quedan patentes dos formas de abstraer y representar la información:

- **con límites bien definidos, en forma binaria:** trazando líneas que determinan un cambio en las características del territorio, o definiendo puntos que abstraen la extensión de un fenómeno y sólo consideran significativa su posición, digamos promedio, sobre el territorio;

[7]Por ejemplo, el tratamiento de un fenómeno de precisión infinita y continua en el espacio, como la elevación. Cuando dibujamos isolíneas de elevación en un mapa, estamos determinando que para la isolínea la altura es conocida, pero estamos discretizando la cuantificación de la elevación entre ellas. Entre la isolínea de los 10 metros y la de 20 la altura es un rango de 10 a 20 metros, pero cada punto dentro de la banda marcada por las isolineas tiene una altura desconocida.

- **con continuidad espacial:** sin trazar líneas ni puntos, sino representando sobre el mapa la variabilidad bi o tridimensional de la variable mediante la aplicación de tramas o esquemas de colores que reflejan la intensidad de la misma en cada punto, pero con dicha variabilidad ya sujeta a un ejercicio de discretización de la información.

En ambos casos, cualidades muy apropiadas para crear estructuras de información en una máquina.

A la información temática asociada a la información cartografíada le pasa lo mismo. La información temática complementa la información geométrica indicando qué características no geométricas tiene el objeto referenciado gráficamente en la cartografía. De esta manera, el cartógrafo también tiene que sistematizar, con un sistema de símbolos, colores y toponimia, las características temáticas de la información geométrica, tomando decisiones sobre qué información es relevante o qué gradación o categorización le va a dar a los fenómenos territoriales. Por ejemplo, representar la jerarquía de las ciudades en el sistema urbano de un territorio (capital, grandes ciudades capitales subregionales, etc.) también conlleva un ejercicio de sistematización y modelado muy acorde con las reglas de la computadora.

Modelos clásicos de estructuras de datos para información geográfica

Existen muchos modelos para el modelado de información geográfica, pero sin duda son dos los modelos principales de datos utilizados para consignar información geográfica con objeto de ser analizada por computadoras: el modelo ráster y el modelo vectorial. Ambos poseen una larga tradición en Sistemas de Información Geográfica, siendo implementados en la mayoría del *software* de este tipo (Olaya, 2016).

El modelo vectorial

El **modelo vectorial** se utiliza cuando se quiere modelar la información territorial en términos de límites binarios. Se apoya en la definición de puntos, líneas y polígonos basados en secuencias ordenadas de coordenadas. Las geometrías básicas del modelo vectorial son:

- **puntos:** unas coordenadas que definen una ubicación en el espacio geográfico, sin extensión;
- **líneas:** una secuencia ordenada de puntos, entre un punto inicial y otro final, que definen una serie de segmentos enlazados;
- **polígonos:** un área delimitada por una línea, como la anterior, pero que tiene la propiedad de que su punto inicial y final son el mismo, por lo que se cierra sobre sí misma.

Estas geometrías básicas pueden, dependiendo de la complejidad del modelo topológico propio de cada *software*, tener variantes en las que se contemplan las versiones multigeométricas de las mismas. Asimismo, los polígonos también pueden complejizarse al tener en cuenta si pueden alojar en su interior islas o no.

También cabe destacar que el modelo vectorial puede ser implementado teniendo en cuenta una estructura de almacenamiento topológica o con geometrías replicadas, lo que se conoce popularmente por el *modelo spaghetti*[8]. Ambas modalidades del sistema vectorial tienen distintas ventajas y desventajas, cuyo desarrollo ya está más que documentado en la bibliografía (Robinson et al., 1987, Olaya, 2016).

Este modelo ha sido ampliamente implementado por muchos sistemas de información geográfica a lo largo de las décadas desde los años 60, dónde se pone tradicionalmente el nacimiento de los SIG. Cada sistema ha implementado a su manera las ideas básicas tras el modelo vectorial, incidiendo o aportando, en muchos casos, interesantes variantes y/o características adicionales sobre el modelo básico. Aún así, hoy en día el *Open Geospatial Consortium* (*OGC*)[9] ha definido una implementación estándar del modelo vectorial, llamada ***Simple Features*** con el que son, en diferentes estados de conformidad, compatibles la mayorías de los *software* SIG más importantes usados actualmente. Generalmente, es este modelo el que, de una forma u otra, es implementado en las aplicaciones SIG de *Software Libre*, en las que se centra esta tesis.

En cuanto a la vertiente alfanumérica de la información geográfica, el modelo vectorial la almacena, en la mayoría de las implementaciones existentes, en forma de tabla de base de datos. En estas tablas[10], cada fila está relacionada con una geometría puntual, lineal o poligonal, y las columnas hacen referencia a las propiedades temáticas que se quieren almacenar para el juego de datos. Así, por ejemplo, en una capa de ciudades representadas por puntos, cada fila de la tabla de datos representaría una ciudad individual[11], identificada por una clave primaria usualmente numérica, con columnas que almacenarían, para cada una de ellas, su nombre, su población, su extensión, etc.[12].

El modelo ráster

El modelo ráster, en contraposición al vectorial, modeliza la información territorial con una aproximación continua del espacio. Es la elección correcta para aquella información territorial en la que no se quieren abstraer límites más o menos arbitrarios, o en la que, en la definición del problema o la aplicación, es fundamental contar con dicha aproximación continua en el territorio. Un ejemplo clásico de juegos de datos ráster son la altura del relieve o las temperaturas.

La vertiente geométrica del modelo ráster se basa en la segmentación de la totalidad del

[8] Llamado así porque la posible duplicación de geometrías comunes a varios objetos geográficos hace que la información geométrica del conjunto de datos parezca precisamente eso, algo tan desordenado como un plato de spaghettis.

[9] El OGC es la organización de referencia de estándares libres en aplicaciones TIG.

[10] A un juego de datos espaciales, por analogía de uso en la mayoría de los SIG, se le suele llamar *capa*.

[11] Un concepto muy importante en este modelo es el de *clave primaria*, es decir, un dato que representa unívocamente, dentro de la colección de datos, a cada dato individidual.

[12] Este modelo de tabla única, muy extendido debido a la preponderancia que durante décadas ha tenido el formato vectorial *SHAPEFILE* diseñado por el fabricante de SIG *ESRI*, es altamente ineficiente en cuanto al almacenamiento de información alfanumérica, en términos de calidad y seguridad en los datos almacenados. Estas deficiencias han sido corregidas en sistemas multitabla como la base de datos relacional geográfica *PostGIS*, y es muy aconsejable diseñar sistemas complejos de información geográfica con tecnologías como esta en lugar de basarlos en *SHAPEFILES*.

territorio objeto de estudio en unidades espaciales homogéneas (tradicionalmente de forma cuadrada o rectángular), de un tamaño uniforme y fijo. Dicha cuadrícula se superimpone al territorio, creando segmentos o áreas territoriales en los que se va a producir el proceso de abstracción y uniformización de la variable considerada. Evidentemente, cuanto menor sea el tamaño de la rejilla, más precisión se obtendrá en la modelización y abstracción resultante[13].

Una vez decidida la precisión de la unidad espacial homogénea, el siguiente paso es tomar los valores de la variable y asignarle a cada unidad un dato final. Esta asignación puede ser definida de muchas formas distintas. Por ejemplo, en el caso de datos puntuales, si varios puntos caen dentro de una tesela de la rejilla, se puede obtener una media de los datos y asignarle dicha media al valor final de la variable procesada. O en el caso de que la información de base a rasterizar sea de formato área, podría asignarse a la tesela aquella categoría de área que sea predominante en la unidad. Es, por tanto, una simplificación de la información original, perdiéndose siempre en el proceso parte de la variabilidad de la variable en aras de la modelización.

Físicamente, los formatos de datos ráster guardan matrices numéricas. Las matrices numéricas tienen una larga tradición en algoritmia dado que muchos problemas de continuidad se modelan de esta manera. Los modelos geofísicos, como los meteorológicos y oceanográficos, por ejemplo, consumen y producen estas matrices numéricas. La matriz numérica se guarda como una sucesión de posiciones de memoria de un tamaño, siempre homogéneo, apropiado a la naturaleza numérica del valor de la variable (un *byte*, dos, etc.). Estas matrices se acompañan de una cabecera que especifica el geoposicionamiento, la rotación y el tamaño de las unidades uniformes que componen el ráster, por lo que los sistemas TIG pueden, a partir del almacenamiento en memoria de la matriz, posicionar con precisión cada unidad sobre el territorio.

Muchos de estos formatos, como el *GeoTIFF*, el *HDF* o el *NetCDF*, están diseñados para albergar, bajo el contexto común de una de estas cabeceras de geoposicionamiento comentadas previamente, varios rásters en un mismo paquete de datos. Cuando esto ocurre, se dice que el ráster es *multicapa* o que forma un *cubo*. Las imágenes de satélite, por ejemplo, aprovechan esta técnica de almacenamiento, teniendo una escena satélite tantas capas ráster como bandas de sensibilidad radiométrica tenga el sensor. De esta manera, para cada unidad espacial homogénea, se tienen varios valores radiométricos, uno por banda del sensor, almacenadas en distintos rásters del mismo tamaño.

Esta forma de organizar los rásters en cubos multicapa es similar y fuente de inspiración para la estructura de datos propuesta en este trabajo. Cuando se extraen como conjunto, cada unidad homogénea tiene un valor numérico en cada capa del ráster, con lo que se conforma un *vector* de datos. Estos vectores de datos son el formato natural de entrada de datos para los algoritmos de *Machine Learning*.

[13]Como es obvio, también se han de tomar en cuenta a la hora de elegir el tamaño de dicha rejilla consideraciones como la precisión de origen de los datos a abstraer. Y además, hay que tener en cuenta que a mayor precisión (menor tamaño) de la rejilla más ocupa el juego de datos en la memoria del ordenador y por tanto más difícil es procesar la información.

Otros modelos derivados de los anteriores

Estos dos modelos clásicos, que se encuentran ampliamente desarrollados en la literatura sobre SIG, están implementados con soluciones diversas en distintos SIG en uso actualmente (*GRASS*, *QGIS*, *ArcGIS*) y dan soporte a otros modelos de información espacial, más especializados, que cubren campos temáticos que precisan de detalles adicionales.

Un ejemplo de estos modelos adicionales son los modelos de redes geométricas. Con claras semejanzas al modelo vectorial, una red geométrica[14] es un conjunto de líneas vectoriales llamadas **ejes** que se conectan en sus extremos en puntos llamados **nodos**. Las redes, para ser eficaces, deben estar digitalizadas con un alto grado de calidad **topológica**[15], cuidando especialmente que la conectividad en los nodos sea precisa. Las redes se almacenan internamente en la memoría de los SIG con un modelo topológico similar al que se usa en algunos modelos de datos vectoriales con objeto de potenciar el cálculo de las conectividades entre ejes y así poder, de forma eficiente, hacer cálculos sobre la red. Las redes son cruciales en aplicaciones como la hidrología, el análisis de vías de comunicación, la modelización de redes logísticas o de servicios, los estudios de accesibilidad, etc.

Otro ejemplo muy destacable es el implementado por las llamadas **bases de datos relacionales geográficas**[16]. Una base de datos relacional geográfica es una base de datos relacional a la que se le ha dotado de un módulo de procesamiento topológico. Las bases de datos relacionales son, posiblemente, los sistemas de procesamiento de datos interrelacionados más flexibles a la hora de modelizar problemas para producir información por análisis de interrelación entre ellos. La clave de su éxito es la enorme capacidad de modelado de problemas de toda índole con un conjunto de reglas que, si bien no son triviales, son lo suficientemente sencillas para poder ser utilizadas por especialistas temáticos sin formación como ingenieros de *software*. Esta versatilidad las convierte en eficientes productoras de información.

La base de las técnicas de modelado en bases de datos relacionales se centra en varios principios, pero que pueden ser resumidos básicamente en estos:

- los datos introducidos en el modelo se introducirán en una única ubicación del mismo y nunca en más de una, con lo que conseguimos reducir el peligro de la redundancia de datos, que desemboca indefectiblemente en la inconsistencia de la información que genera la base de datos en sus análisis;

- el modelador tiene que explotar al máximo la interrelación que se produce entre los datos introducidos en el modelo. Cuanto más interrelación de datos se modelen, más información podrá generar la base de datos al analizarlos.

[14] Técnicamente, una red geométrica es un grafo, y existe una especialidad de la matemática, la Teoría de Grafos, centrada en su estudio y análisis. Los grafos tienen muchas y útiles aplicaciones técnicas.

[15] La topología es el conjunto de relaciones espaciales intrínsicas que los elementos geométricos establecen entre sí gracias a una función distancia determinada en un espacio multidimensional.

[16] Este autor lleva ejerciendo su actividad docente en este campo desde hace 15 años y considera que esta técnica ha de ganar, a pesar de su a veces dura curva de aprendizaje inicial, más visibilidad entre los estudiantes de Sistemas de Información Geográfica como la herramienta correcta para crear grandes bancos de información geográfica de calidad, entendida aquí la calidad como calidad en la consignación y manejo de los datos, desde una perspectiva estrictamente técnica y operacional, no temática.

Este análisis de información es llevado a cabo por un sofisticado componente de las bases de datos, el **motor relacional**. Basado en una rama de la matemática llamada **Álgebra Relacional**, el análisis no obstante es muy accesible dado que se produce gracias a la escritura de un lenguaje de consultas, el ***Structured Query Language*** (SQL), que se asemeja al lenguaje natural humano[17], por lo que es fácil de aprender y utilizar. Si el modelador ha encontrado lazos de relación entre juegos de datos distintos, una BDR es la herramienta perfecta para ponerlos en relación y extraer información novedosa. Este motor relacional se haría cargo, en el contexto de la información geográfica, de analizar la información geográfica en su vertiente temática.

Una base de datos relacional geográfica sigue todos estos conceptos, pero se le añade un nuevo componente analítico, también bastante complejo, denominado **motor topológico**. El motor topológico es a la vertiente geométrica de la información geográfica lo que el motor relacional es la vertiente alfanumérica, es decir, el motor topológico es capaz de inferir las relaciones de índole espacial que existen entre los elementos geométricos de la información geográfica.

Ciñéndonos al caso del modelo vectorial en este caso[18], la topología, desde un punto de vista matemático, es la definida por la distancia euclídea en un espacio tridimensional, por tanto, euclídeo. En este espacio euclídeo existe la propiedad de que si dos puntos tienen una distancia cero entre ellos, son el mismo punto[19]. Esta sencilla consecuencia que emana de la realidad del espacio geográfico es la base de cálculo de todas las relaciones topológicas existentes entre elementos geométricos del modelo vectorial:

- si dos puntos están a distancia cero son el **mismo punto**;
- por tanto, si un punto está a distancia cero de cualquiera de los infinitos puntos que componen una línea, ese punto está **sobre la línea**;
- si dos puntos de los infinitos puntos que componen dos líneas están a distancia cero esas dos líneas **intersectan**;
- si uno de los infinitos puntos de una línea está a distancia cero de uno de los infinitos puntos que componen el límite de un polígono esa línea es **tangente** al polígono;
- si muchos puntos de una línea están a distancia cero de un número análogo de los infinitos puntos interiores a un polígono, esa línea a **penetrado** en el polígono;
- si todos los puntos de una línea están a distancia cero de los puntos de un polígono, esa línea está **contenida** en el polígono;
- si varios puntos interiores a dos polígonos están a distancia cero los unos de los otros, esos polígonos **solapan**;

[17] Al lenguaje natural humano inglés, eso sí.

[18] Los cálculos topológicos en el modelo ráster son mucho más sencillos, ya que son intrínsicos a la propia naturaleza de su modelo geométrico: una tesela tiene 8 vecinas, en direcciones norte, sur, este y oeste y en sus diagonales, siempre a una distancia bien conocida. La distancia entre celdas también es fácilmente calculable en función, por ejemplo, de sus centroides.

[19] Esto, que parece evidente, no es siempre así en la topología matemática.

- la **dirección** (también llamada **acimut**) en la que se encuentran los elementos geométricos los unos con respecto a los otros también es una relación topológica a tener muy en cuenta en muchos modelos territoriales, como por ejemplo el análisis hidrográfico;
- la **conectividad en red**, modelizada por relaciones topológicas de conexión y contacto entre los nodos (puntos) y los arcos (líneas) de un grafo, es también una forma de utilizar la topología para modelizar múltiples problemas y aplicaciones.

Existen decenas, si no cientos, de posibles relaciones topológicas (resaltadas en negrita) que se establecen de forma ***intrínseca*** entre las geometrías de los modelos vectorial y ráster.

Aquí el concepto clave es **intrínseco**: estas relaciones topológicas existen tanto lo queramos como si no. Dos elementos geométricos, por el simple hecho de compartir el espacio geográfico y estar definidos por coordenadas sobre el mismo, están relacionados topológicamente por la función distancia euclidea (sobre el plano, sobre el elipsoide la función distancia cambia), lo deseemos o no[20]. El motor topológico es muy eficiente calculando estas distancias entre puntos pertenecientes a geometrías vectoriales y por tanto es capaz de contestar preguntas como:

- ¿en qué polígono que representan municipios está el punto X que representa una ciudad?
- ¿Por qué polígonos que representan espacios naturales protegidos discurre un curso fluvial definido por una línea?
- ¿En cuánta superficie se solapan las áreas de influencia de 10 kilómetros de una instalación logística con las áreas de reparto de 20 kilómetros cuadrados de sus centros de reparto?

La potencia de las bases de datos relacionales espaciales está no sólo en la increible facilidad con la que pueden modelizar y responder preguntas de índole topológico y alfanumérico sobre información geográfica, sino precisamente en la capacidad intrínseca de relación topológica.

En los modelados de información geográfica, el modelador debe encontrar o buscar un nexo de relación entre los datos que quiere relacionar. Pongamos un ejemplo: queremos hacer, a nivel nacional, un estudio municipal socio-económico. Para ello, contamos con una base de datos social y con otra, separada de esta, de datos económicos. Las bases de datos han sido desarrolladas y digitalizadas por modeladores distintos que no se han coordinado previamente. La primera es capaz de contestar, por medio del motor relacional, una ingente cantidad de preguntas de índole social. La segunda hace lo propio con preguntas económicas.

Si queremos contestar preguntas de índole socio-económico, debe existir un nexo de unión entre ambas bases de datos. Un código nacional municipal, como los proporcionados

[20] Existen formatos de datos, sobre todo vectoriales, que explicitan la topología, pero ese es otro tema bien distinto. Aquí estamos hablando de topología matemática.

para España por el Instituto Nacional de Estadística (INE), sería lo apropiado. Pero si los modeladores no han tenido algo así en cuenta, la relación entre datos puede llegar a ser muy costosa, si no directamente imposible.

La información geométrica de la información geográfica está intrínsecamente relacionada gracias a la topología. Aún en el hipotético caso de que no contáramos con un vínculo alfanumérico entre las anteriores bases de datos, si contamos con las geometrías de los municipios contemplados, pasaríamos de un escenario de relación alfanumérica basada en códigos a un escenario de relación topológica basada, por ejemplo, en la coincidencia de centroides de los polígonos de los municipios.

Es esta capacidad de establecer relaciones entre juegos de datos dispares, gracias a las técnicas de análisis territorial clásicas, y su reproducibilidad en modelos personalizados para cada juego de datos y aplicación o estudio a realizar, lo que convierte, hoy por hoy, a las bases de datos geográficas en los más avanzados sistemas de análisis de información geográfica, sobrepasando en capacidad (si bien no en facilidad de uso) a los Sistemas de Información Geográfica tradicionales de los llamados *de escritorio*.

Entre las bases de datos geográficas relacionales plenamente desarrolladas con toda su funcionalidad caben destacar dos:

- **Oracle Spatial:** *software* privativo con una fuerte inversión inicial;
- ***PostgreSQL* con la extensión *PostGIS*:** *Software Libre*.

I.1.8. Un nuevo modelo de datos para la información geográfica

El objetivo fundamental de esta tesis es la creación de un prototipo que demuestra, a modo de prueba de concepto, la utilidad de la estructura de datos propuesta, la rejilla multiescalar asimétrica, para la gestión de información geográfica armonizada geométrica y temáticamente, que facilite el análisis con técnicas de *Big Data* y *Machine Learning*.

El primer objetivo fundamental de este trabajo de tesis es la descripción e implementación de un nuevo modelo de información geográfica, en cierta medida similar al ráster, diseñada con los siguientes objetivos:

1. asimilar y adscribir información diversa, de origen vectorial o ráster, bajo un marco geométrico común y homogéneo, y por tanto comparable entre sí, que permita crear unidades de información homologables a nivel geométrico y temático;
2. minimizar el espacio de almacenamiento de información con respecto al ráster tradicional, consignando la información en menos espacio;
3. agilizar el trasvase de información entre servidores y clientes y, de esta forma, hacer clientes *web* ligeros y ágiles a la hora de mostrar información;
4. crear una estructura de datos que sea compatible con la ejecución de algoritmos propios del campo del *Machine Learning* sobre información geográfica.

Posteriormente, como segundo objetivo, está la implementación de un *software* prototipo que construya, gestione, analice, publique y visualice información en dicho formato.

I.1.9. Ventajas e inconvenientes de los modelos clásicos

Los modelos de información geográfica clásicos ráster y vectorial poseen una serie de ventajas e inconvenientes que nos acercan o nos alejan de los objetivos mencionados.

El modelo vectorial tiene como una de sus cualidades que produce información muy compacta en tamaño. Al definir los fenómenos territoriales con objetos vectoriales definidos por secuencias de coordenadas, la información a almacenar es relativamente compacta. Si el modelo vectorial utilizado es de naturaleza topológica, esta compactación de la información es aún más notable. El modelo vectorial, al ser una aproximación de naturaleza discreta binaria a los fenómenos territoriales, sólo guarda información para aquellas zonas en las que el fenómeno existe. Si creamos una capa de suelo urbano, por ejemplo, sólo consignaremos polígonos de urbano en aquellas áreas que lo sean. En el continuo espacio territorial, aquellas áreas que no sean urbanas simplemente no se consignarán en la capa. Esto favorece los objetivos 1 y 2.

El ráster, por su parte, es más expansivo en cuanto a espacio de almacenamiento de la información. Al ser de naturaleza continua, el ráster precisa de un dato para cada unidad de segmentación territorial. Si modelamos en ráster la capa anteriormente descrita, aquellas celdas con presencia urbana obtendrán el dato "presencia de urbana", pero aquellas celdas que no contengan el fenómeno deberán, igualmente, estar presentes en el juego de datos aunque sea con un dato de información nula. Este dato de información nula, aún así, ocupa espacio de almacenamiento. El ráster ocupa siempre el mismo tamaño para una resolución, zona de estudio y variable determinadas, independientemente de la extensión real del fenómeno digitalizado en el área considerada. Esto, por tanto, va contra los objetivos 1 y 2.

El modelo vectorial, al ser de naturaleza discreta, abstrae los fenómenos territoriales imponiendo límites en los que el fenómeno está presente o no lo está en absoluto. Es un mal modelo para modelar transiciones o fenómenos continuos. Los límites que modeliza el modelo vectorial, además, no tienen por qué coincidir entre fenómenos que comparten zona de transición. El límite finalmente decidido dependerá de los datos de origen o del criterio técnico / temático del especialista que lo define. Por lo tanto, lo normal es que los límites de las zonificaciones vectoriales sean difícilmente coincidentes. Esto genera un problema de analogía de la cantidad de información por cada unidad de área: es complejo establecer una zonificación unívoca cuando consideramos varios fenómenos territoriales de distintas fuentes. Esto va contra el objetivo 1.

El modelo ráster, por contra, parte de un proceso inicial de segmentación uniforme de la zona de estudio, dividiendo ésta en unidades mínimas de información que constituyen la base de resolución final del conjunto de datos a modelar. Esta segmentación impone al digitalizador de información un rígido marco geométrico que le obliga a tomar decisiones temáticas de abstracción y consignación de la información bajo un contexto homogéneo que permite una comparativa entre unidades de información fiable. Al tener cada tesela

del ráster las mismas características geométricas de extensión, la medición otorgada a la variable para cada una de ella es comparable a la de sus compañeras de ráster. Esto es inviable en el vectorial puesto que cada unidad de datos lineal o poligonal[21] tiene unas características geométricas potencialmente únicas, anulando la posibilidad de establecer comparaciones homogéneas, al menos sin hacer un análisis matemático de ratios o similares previamente. Esta característica del ráster es muy importante para cumplir con el objetivo 1.

La no uniformidad espacial del modelo vectorial también va en contra del objetivo 4. Al no disponer de unidades espaciales homologables, los posibles valores temáticos asociados a ella tampoco lo son, por regla general. En el modelo ráster, no obstante, al ser homologables las unidades espaciales que genera (todas las celdas tienen el mismo tamaño), los posibles valores almacenados en sus datos temáticos sí son directamente homologables entre sí, por lo que se pueden utilizar sin modificación en procedimientos de análisis de los descritos en el objetivo 4.

I.1.10. Conversión entre modelos

Es posible transformar, asumiendo por regla general una pérdida de precisión, la información geográfica entre modelos ráster y vectorial.

Para transformar el modelo vectorial en ráster, se superimpone la estructura de celdas sobre la capa vectorial. Una vez hecho esto, hay que decidir qué método de asimilación de la información vectorial sobre cada tesela se utilizará. Las posibilidades son muchas, por poner algunos ejemplos:

- para capas poligonales, se puede especificar por tesela la presencia o no del fenómeno, binariamente, sin cuantificar la magnitud de la presencia;
- para capas poligonales con un atributo temático discreto, se puede ver qué categoría temática ocupa más área de la tesela y asignarle a la tesela completa el valor temático de la categoría más extensa. La información del resto de las categorías presentes en la tesela se perdería;
- en el caso anterior, se podrían crear varios rásters con el porcentaje de ocupación por tesela de cada categoría;
- si la variable a tener en cuenta es continua, se pueden aplicar medidas de tendencia central estadística a los valores numéricos presentes en la tesela, ponderadas frecuentemente por el área que ocupa cada una. Por ejemplo, si un polígono ocupa con un valor numérico 4 el 75 % del área de la tesela y el 25 % restante la ocupa un valor 2, la media ponderada por superficie sería de 3,5;
- para capas puntuales, se pueden también aplicar distintos métodos de conversión, como por ejemplo presencia / no presencia, recuento de puntos por tesela, media de valores continuos asociados a los puntos que caen en la tesela, suma de valores de puntos en la tesela, etc.

[21] En los puntos, al ser adimensionales, esto no se cumple.

Como siempre sucede en el modelo ráster, cuanto mayor es la resolución espacial del juego de datos[22], menor es la pérdida de información original vectorial y viceversa.

Estas conversiones de vectorial a ráster tienden a aumentar mucho el tamaño del set de datos. Un sólo polígono que delimita un área con un valor temático discreto, como por ejemplo el nombre de un municipio, acaba transformado en una potencialmente ingente cantidad de celdas con dicho valor temático repetido. En el ejemplo 3, por ejemplo, el ráster se ve repetido varias veces. Recordemos que los juegos de datos ráster no pueden obviar celdas: todo el espacio del área de estudio debe estar recubierto por ellas, aunque sea sólo para almacenar un valor nulo o "sin datos"[23].

En cuanto a las conversiones inversas, ráster a vectorial, la mecánica es mucho más sencilla y determinista: simplemente consiste en generar en geometría vectorial el área recubierta por las celdas y asignarles a cada polígono el valor temático asignado a la tesela original, o incluso, en ocasiones, se transforma la tesela en un punto, tomando como referencia el centroide de la tesela. En el caso de convertir la tesela a polígono, se realiza un paso final consistente en fusionar todos los polígonos de celdas adyacentes que compartan el mismo valor temático.

I.1.11. Contexto actual de estructuras de datos y tecnologías geográficas

La publicación y disponibilidad de datos para el público vive en los últimos años una verdadera revolución en la cultura del dato gracias a las iniciativas *Open Data*, enmarcadas en las recomendaciones de la UNESCO para *Open Science* (Toro Bonilla, 2014, UNESCO, 2020, Das, 2020). Cada vez son más los organismos de muy diversa índole que proporcionan datos abiertos en todos los ámbitos. El dato cada vez está mejor metadatado y es más accesible para el público en general, y sus licencias de uso son cada vez menos restrictivas, lo que permite a los sectores académicos realizar mejor ciencia y al sector privado generar valor con su puesta en explotación.

Por lo tanto, hoy en día han proliferado la publicación y accesibilidad de grandes conjuntos de datos públicos y privados. Muchos de estos conjuntos de datos están levantados por medios automáticos, como por ejemplo el LIDAR, las redes de dispositivos móviles, el tráfico en grandes redes sociales en *Internet*, etc. Estos conjuntos de datos son de tal tamaño que son inabarcables sin la tecnología adecuada. Las técnicas de *Big Data* (Gutiérrez Puebla, García Palomares y Salas Olmedo, 2016, Toro Bonilla, 2014) se especializan en el cribado, transformación y análisis masivo de estos datos, muchas veces incluso en tiempo real. Si a ello se suma la progresiva incorporación de la componente espacial, aunque sólo sea con una posición geolocalizada, su tratamiento y análisis se hace aún mas complejo.

[22] Es decir, cuanto más pequeña es la tesela.

[23] El valor "sin dato" suele ser conocido como *null* en la terminología de la ciencia de datos, y significa "dato desconocido". Se debe tener en cuenta que no es lo mismo una cadena de texto vacía, que es un dato en sí mismo, ni un cero, que es un valor numérico, ni el concepto de "no aplicable", que en un buen modelo de datos no debe de existir.

Estos datos, procedentes de fuentes tan heterogéneas, son publicados bajo muchos formatos distintos y con esquemas de datos[24] muy distintos. Por supuesto, también está siempre la consideración de la escala, que también ha de ser trabajada para alcanzar un estado de equivalencia. A pesar de la puesta a disposición de datos en abierto, la integración de información espacial es un problema clásico (Knapp, 1980, Vargas et al., 2019) de nuestra disciplina, por otra parte necesaria para abordar la mayor parte de los problemas territoriales y ambientales (Paneque et al., 2018, Ruíz Sinoga et al., 2015 y Martínez Murillo et al., 2017). Una de las posibilidades propuestas en muchos foros y reuniones científicas es la integración espacial en teselas multiescalares y multidimensionales. La mayor parte del *software* TIG ofrecen esta posibilidad, generalmente con teselas multiescalares sobre superficies proyectadas y, en menor medida, sobre teselas sobre superficies elipsoidales basadas en datum globales (sobre todo el WGS84).

Para abordar estos retos es necesario un entorno de trabajo flexible y escalable. En los últimos años se han popularizado el uso de entornos *cloud*, basados en microservicios que facilitan el necesario procesado en paralelo (*Parallel Computing*) para garantizar una operatividad asumible en términos de tiempo de computación. Si hablamos de información geográfica, la mayor parte del *software* más utilizado (*Google Earth*, *ArcGIS Cloud*, *CARTO*, *Mapbox*, por citar los más populares) están migrando su oferta a este nuevo entorno tecnológico.

Otra tendencia generalizada en el tratamiento de datos de todos los ámbitos de aplicación es la irrupción y desarrollo de nuevas tecnologías ligadas a la Inteligencia Artificial (IA), especialmente, las ligadas al *Machine Learning*, que constituye una sección del concepto más general de IA. La información geográfica no puede quedarse al margen de esta revolución[25], necesaria para abordar estos conjuntos de datos masivos, por lo que es necesario explorar nuevas vías instrumentales para adaptarse a este nuevo paradigma. Precisamente, la integración espacial de los datos geográficos originales en teselas multiestacalares y multidimensionales es una de las estructuras de datos geográficos que mejor se adaptan a los requerimientos para potenciales análisis de IA o *Machine Learning*, ya que estos algoritmos precisan de ser suministrados con estructuras de datos muy específicas en las que la homogenización de sus componentes es fundamental. Esta homogeneización es difícil de procesar en información gegráfica, sobre todo sobre el modelo vectorial, puesto que uno de sus puntos fuertes es precisamente la posibilidad de saltarse la necesidad de dicha homogeneidad. Esta propiedad, tan celebrada y necesaria en cartografía digital y el análisis clásico espacial, es un gran obstáculo para estas nuevas técnicas de análisis.

Por otra parte, se han producido igualmente grandes avances en la geovisualización de datos en clientes ligeros *web*. Navegadores *web* más rápidos, más interactivos y más autónomos de los servidores de los que indefectiblemente dependen en última instancia,

[24] *Esquema de dato*, en este contexto, es el término técnico para llamar a las estructuras de datos a las que se ha estado haciendo referencia a lo largo de esta introducción.

[25] Como nota histórica, la información geográfica ya se quedó al margen de la última gran revolución en el tratamiento de datos, en los años 70: el modelado entidad-relación y las bases de datos relacionales. La información geográfica tardó 30 años en dar ese salto. Culpa de ello fue, por un lado, la dificultad técnica inherente al aspecto geométrico de la información geográfica, pero también una aparente apatía por incorporar nuevas herramientas de análisis que fueran más allá del cómodo entorno del SIG de escritorio.

gracias a la revolución en el mundo *web* que ha posibilitado la implantación del estándar *HTML5* (Anthes, 2012), hacen que la exploración y el trabajo *online* con información geográfica tenga más posibilidades que nunca. Ya no se expresa a través de mapas estáticos, sino a través de estos clientes *web* (geovisores), a los que se unen un conjunto de nuevas técnicas de exploración de datos[26] que contribuyen no solo a la geovisualización de los mismos, sino a su exploración interactiva a través de la creación de *dashboards* altamente interactivos.

Antecedentes de la rejilla multiescalar

El concepto de rejilla multiescalar se inspira obviamente en el modelo ráster de información geográfica. En este modelo ya podemos encontrar muchas de las ventajas buscadas en la nueva estructura de datos propuesta, sobre todo la homogeneización y discretización del espacio geográfico y la discretización, en consecuencia, de la información alfanumérica adscrita al mismo.

Desde que la UE propusiera en el marco de la Directiva Inspire (INSPIRE Directive, 2007) este formato para la distribución de información asociada a datos poblacionales (EuroStat, 2012), son varios los autores y organismos que han utilizado esta estructura para la difusión de la información temática. Casos específicos son las estructuras de rejilla utilizadas para la representación de indicadores climáticos por el *Global Climate Monitor* (Álvarez Francoso, Camarillo Naranjo et al., 2014) o para el análisis de densidad de población de España (F. J. Goerlich et al., 2012). Aparte de la propia INSPIRE, que publica el marco de referencia de rejilla común europea, organismos como el Instituto de Estadística y Cartografía de Andalucía (IECA) (Enrique et al., 2013) o el Instituto Nacional de Estadística (INE) (INE, 2021b) han encontrado en este formato de publicación una valiosa herramienta para la publicación de información de detalle, como por ejemplo demográfica, sin incurrir en quebrantamientos de las leyes de secreto estadístico (BOE, 2006a).

Dado lo anterior, no es de extrañar por tanto, que en marzo de 2017 se presentase una Proposición de Ley por la que se modifica la Ley 14/2010, de 5 de julio (BOE, 2006b), sobre las infraestructuras y los servicios de información geográfica en España con el fin de subsanar deficiencias de la trasposición de la Directiva INSPIRE en España. Entre las medidas a subsanar destaca las recogida en el artículo 5 que insta a la inclusión como Equipamiento Geográfico de Referencia Nacional el "sistema de cuadrículas". Esto supone la normalización y estandarización de un conjunto de rejillas multiescalares para representar diferentes tipos de información territorial (población, usos de suelo, etc.), garantizándose que todas las celdillas, y a cualquier escala, tengan un origen común y con ello proporcionando coherencia geométrica a futuros análisis espaciales. Este hecho revela la importancia de este tipo de estructuras para el análisis espacial de datos temáticos en el futuro y, en cierta medida, justifica los objetivos de esta tesis.

[26] En ciencia de datos existe un subsector, el *DataViz*, cuyo objeto de estudio son las técnicas interactivas de exploración y visualización de gráficos de todo tipo, con el objetivo de comunicar interpretaciones y patrones presentes en los datos con más eficacia.

Tipos de recubrimientos teselares

Las aplicaciones del recubrimiento teselar del espacio geográfico tienen ya una trayectoria en la literatura científica (Goodchild y Shiren, 1992), y es un problema que sigue actualmente vigente (Bondaruk et al., 2019), no sólo en el ámbito de la Geografía, sino en todos los sectores de las Ciencias de la Tierra. La ingente acumulación de información espacial procedente no sólo ya de la inmensa cantidad de sensores disponibles (redes de seguimiento en tierra y océanos, satélites de todos los tipos y aplicaciones), sino de las salidas de los cientos de modelos matemáticos que se corren sobre ellos diariamente[27], cada uno con una escala y resolución distinta, ha llevado al *Open Geospatial Consortium* (*OGC*) a la creación de un nuevo estándar, el *Discrete Global Grid Systems* (Sistema de Teselado Global Discretizado) (Purss et al., 2016), basado en teselación hexagonal, para que toda esa información heterogénea se armonice sobre un estándar único que facilite la realización de nuevas investigaciones sin tener que estar lidiando constantemente con no triviales problemas de adscripción de datos heterogéneos.

La adscripción de los datos a un sistema de rejilla, sea esta del tipo que sea, no es trivial. No basta con identificar una relación topológica de inclusión y hacer una media estadística o cualquier otro cálculo de centralidad, sino que cada problema de adscripción de información a un modelo de rejilla debe ser estudiado por especialistas en la materia que tratan los datos para no cometer ninguna aberración estadística que desvirtúe completamente el sentido e interpretación posterior de los datos. Este es un gran obstáculo para la integración de datos provenientes de distintas fuentes, ya que precisa de un elevado conocimiento temático en todas ellas. Por tanto, queda patente el interés y las ventajas de que estos productores especializados de información aborden la tarea de la adscripción a un marco de referencia de rejilla común a nivel mundial, diseñado por una organización de prestigio y consenso como es el *OGC*, para que los científicos de otros campos puedan integrarlos en sus algoritmos y hacer ciencia de una forma más sencilla.

Este recubrimiento teselar propuesto por el *OGC* utiliza una rejilla hexagonal porque ésta ofrece ventajas frente al más común tramado cuadrangular heredado del ráster:

1. los hexágonos crean un recubrimiento más uniforme del espacio geográfico, ya que sus distintas diagonales a los vértices y sus perpendiculares a sus lados no difieren tanto en distancia como las que se producen en celdas cuadradas;

2. un hexágono tiene relaciones topológicas con sus vecinos más limpias, puesto que, al contrario que en un tramado cuadrangular, los centros de los vecinos de un hexágono, que son seis, están siempre a la misma distancia de su propio centro. En una rejilla cuadrangular, que tiene ocho posibles vecinos, los cuatro que se encuentran lindando con los límites de la celda se encuentran a menor distancia que los cuatro que lindan con las esquinas. Esto desvirtúa mucho la topología inherente a la rejilla para ciertas aplicaciones;

[27] Como ejemplo, tengamos en cuenta que el *Met Office* británico tiene un archivo de más de 5 *petabytes* en datos originales y en ejecuciones de modelos. Si esto no dice nada, quizás el hecho de que sólo su mantenimiento cuesta 2 millones de libras al año si lo haga.

3. al tener más lados, el desarrollo elipsoidal del hexágono se ajusta mejor y con menos error medio a la superficie del mismo que su contrapartida cuadrangular.

El hexágono, sin embargo, no está exento de problemas, ya que no pueden descomponerse en un número finito y con recubrimiento completo por réplicas de si mismo de menor tamaño, es decir, un hexágono no es descomponible en otros hexágonos. Por contra, una celda cuadrangular si es divisible en un número finito y con recubrimiento completo de celdas con sus misma forma y proporción.

Esta debilidad le resta un tanto de interés al hexágono porque no es posible obtener relaciones topológicas entre resoluciones de rejilla distintas. Esto quiere decir que no hay una traslación directa entre una rejilla hexágonal donde los hexágonos tienen 1 kilómetro de lado frente a otra en la que tienen 500 metros de lado. Sin embargo, en la rejilla cuadrangular esta relación inter-resolución si es viable, ya que una celda de 1 kilómetro de lado es divisible en 4 celdas de 500 metros de lado de una forma geometrica y matemáticamente trivial. Esta propiedad permite agilizar mucho los cálculos de adscripción, como se verá, y es ampliamente utilizada en la solución propuesta en esta tesis, que opta por explorar el desarrollo de las rejillas cuadrangulares y sus propiedades.

Otras soluciones de recubrimiento del espacio geográfico optan por triángulos. Los triángulos pueden darse en redes regulares o irregulares (las *Triangular Irregular Networks*, TIN). El recubrimiento por triángulos es muy utilizado para modelizar espacios en sistemas computerizados puesto que tienen muchísimas utilidades en Inteligencia Artificial. Por ejemplo, es la base del movimiento de robots en espacios conocidos de antemano, y se utiliza mucho para identificar y modelar rutas óptimas a partir de la delimitación del espacio transitable. Las triangulaciones regulares se basan en triángulos equiláteros que sí tienen la propiedad de ser descomponibles en un conjunto de 4 triángulos equiláteros, lo que lo convierten en una interesante opción ya que sí tienen la propiedad de autocontención de niveles de resolución inferiores. La topología, no obstante, es menos interesante que la del hexágono por tener el triángulo tan solo tres vecinos. Sin embargo, las redes triángulares son objeto de una gran cantidad de algoritmia de geometría computacional, ya que se utilizan para muchísimas aplicaciones computacionales, siendo una de las más evidentes y comunes la composición de objetos tridimensionales. Es, por tanto, una opción muy interesante a considerar en el campo que nos ocupa.

I.1.12. *Machine Learning*

El término *Machine Learning* hace referencia a una subrama de la Inteligencia Artificial, que cubre muchísimos campos, centrada en el aprendizaje y entrenamiento de modelos de inferencia basados en datos. El objeto de las técnicas de *Machine Learning* es el análisis de un conjunto de datos multidimensionales, es decir, multitemáticos, para detectar los posibles patrones de relaciones que pueden producirse entre las dimensiones temáticas de los datos.

Las técnicas de *Machine Learning* cubren muchos ámbitos, desde la estadística más básica hasta las técnicas del *Deep Learning*, redes neuronales que intentan asemejarse en su funcionamiento a un cerebro biológico. Entre las más sencillas podemos encontrar

regresiones polinomiales de diferentes grados, por ejemplo. Otro ámbito de aplicación de estas técnicas es el estudio de componentes principales, algoritmos destinados a reducir la dimensionalidad de un problema, descartando elementos temáticos del fenómeno que no tienen una relevancia destacada a la hora de diferenciar unos elementos de otros. Otras técnicas más avanzadas tratan de crear clasificaciones complejas categóricas dentro de los datos multidimensionales, infiriendo la pertenencia de nuevo dato a una de las clases aprendidas en base al estudio de sus características y de las características de los grupos aprendidos. Básicamente, estas técnicas caen dentro de dos grupos principales (Raschka et al., 2019):

1. **técnicas de aprendizaje no supervisado:** estas técnicas tratan de inferir la clasificación a partir del estudio de un subconjunto de los datos, sin intervención de un experto humano que asista el proceso de aprendizaje. Un ejemplo de esta técnica es el clasificador *K-Means*;

2. **técnicas de aprendizaje supervisado:** estas técnicas infieren la clase de pertenencia de los nuevos datos a partir de un aprendizaje en el que se les ha suministrado grupos de control seleccionados que se sabe pertenecen a las clases objetivo. A partir de estos grupos de control el algoritmo aprende a distinguir los distintos elementos del grupo de datos a partir de sus características. Un ejemplo de esta técnicas son los *Random forest*.

Para trabajar, estos algoritmos necesitan *vectores*. Los vectores son estructuras de datos que presentan las dimensiones temáticas del objeto de estudio en base a una serie de componentes, que son numéricas. Estos vectores entrenan a los algoritmos y, una vez el proceso de entrenamiento ha finalizado, son capaces de asignar a las clases aprendidas nuevos vectores que se les proporcionan como entradas.

Aunque se podría aplicar estas técnicas, por ejemplo, a un grupo de datos vectoriales geográficos que tuvieran un conjunto homogéneo de datos temáticos, estos algoritmos, para funcionar bien, precisan de una preparación previa de los datos que los conviertan en homologables. Los datos deben estar normalizados, por ejemplo de una escala de 0 al 1, si los datos de partida no son homologables entre sí por su naturaleza. Esto pasa mucho con los datos vectoriales. Imaginemos el caso de datos de población vinculados a municipios. Los polígonos de municipio no son homologables entre sí, ya que tienen áreas distintas, por lo que la población que contienen tampoco parte de una situación de igualdad: se puede razonar que en municipios más extensos cabe a priori más población, pero se puede dar incluso lo contrario. Esto se soluciona normalizando el dato de población según el área, es decir, utilizando densidades. Sin embargo, este enfoque también tiene sus propios problemas, y no siempre tiene sentido aplicarlo.

La adscripción a una rejilla homogénea soluciona este problema, ya que todas las teselas que la componen tienen las mismas características geométricas. Si la información de partida ha sido adscrita, asignada, a la tesela con las mismas reglas para todas, entonces los datos de una tesela son perfectamente homologables, sin más transformación, a la de sus vecinas. Y como esto es válido para todas las dimensiones temáticas que queramos asignar a la tesela, tenemos que es un vehículo muy apropiado para ser entradas de datos

geográficos a estos algoritmos.

I.1.13. *Big Data*

El *Big Data* es otra de las tendencias actuales en análisis de datos. El *Big Data* tiene como objetivo desarrollar metodologías y herramientas de análisis de datos especialmente preparadas para el tratamiento de bancos de datos masivos, muchas veces en tiempo real. El grado de sensorización que están alcanzando las sociedades de los países más desarrollados, con el uso generalizado de *Internet*, redes sociales y dispositivos móviles de todo tipo[28], hace que la generación de datos en bruto sea tan grande que las máquinas no pueden operar con ellos de forma convencional.

La capacidad de un determinado sistema informático, algoritmo o flujo de trabajo de trabajar con un volumen creciente de datos se denomina *escalabilidad*. Muchos procesos escapan a la capacidad de cálculo y recursos electrónicos de un ordenador personal, por lo que, ante esta disyuntiva, se puede optar por dos soluciones:

- **escalabilidad vertical:** la escalabilidad vertical consiste en aumentar las prestaciones, tanto en *software* como sobre todo en *hardware*, de la máquina para intentar acaparar todo el trabajo. Ciertas mejoras de *hardware*, como el número de núcleos de procesamiento o, sobre todo, el aumento de la memoria, permiten a una sóla máquina aumentar su capacidad de procesamiento y por tanto abordar la aplicación de algoritmos complejos a un mayor volumen de datos. Estas máquinas suelen ser servidores, usualmente máquinas bastante caras, de altas prestaciones;

- **escalabilidad horizontal:** en contraposición a la anterior, esta estrategia de escalabilidad se centra en usar *clusters de computación*, es decir, un grupo de máquinas que se reparten el trabajo entre ellas para así abarcar tareas costosas en tiempo y procesamiento.

La escalabilidad vertical tiene un techo duro difícil de romper. Los microprocesadores han llegado ya al límite de la física del microtransistor, por lo que, por efectos cuánticos, ya no se pueden miniaturizar más ni añadir más a un microprocesador, es decir, el chip de silicio ya ha llegado a su tope de crecimiento. Ahora las máquinas se hacen más potentes añadiendo más microprocesadores que trabajan en paralelo, pero esto tampoco se puede estirar indefinidamente, así como la memoria. Además, desde el punto de vista operativo, cuanto más grande sea la máquina más riesgo hay de caida del sistema si esta falla. Es más ventajoso, por motivos de redundancia, escalar los sistemas horizontalmente, ya que en caso de fallo de una máquina otras pueden asumir su lugar. Más vale, por tanto, 10 máquinas modestas trabajando en *cluster* que una sola máquina con la potencia de todas ellas.

[28] Generalmente se hace referencia a los teléfonos, pero no sólo éstos están sensorizados. Las flotas de vehículos, los dispositivos de las *Smart Cities* y últimamente las tecnologías y desarrollos en el campo de la *Internet of Things* (IoT, *Internet* de las cosas) está haciendo que la sensorización sea ubícua en nuestro entorno.

Las técnicas de *Big Data*, por lo tanto, ponen el foco en metodologías de proceso de datos distribuidas por escalabilidad horizontal. La mayoría de las metodologías y herramientas de analítica *Big Data* siguen un esquema conocido como *Map - Reduce*. Esta metodología consta de dos etapas, referidas con los nombres anteriormente citados. En la etapa *Map*, el sistema controlador del *cluster* de computación estudia la composición y tamaño del total de datos y adopta una estrategia de segmentación del mismo ventajosa para crear varios paquetes de trabajo. Estos paquetes de trabajo son asumidos por otras máquinas[29] del *cluster*, que los procesan, obteniendo un resultado final. Este resultado final es devuelto al controlador, quien, cuando los tiene todos, comienza la etapa del *Reduce*, que consiste en aplicar una estrategia para la creación de un resultado final a partir de los resultados parciales, usualmente por agregación o agrupamiento de los mismos.

En lo que a información geográfica se refiere, la cosa se complica un poco. Para empezar, hay que matizar lo que significa *Big Data* en información geográfica. Se dice que tenemos que recurrir a una solución en el ámbito del *Big Data* cuando nuestros datos son tan grandes que una máquina sola no puede con ellos. Esa es la definición estándar y lo más publicitado en prensa, por ejemplo. Sin embargo, no sólo tenemos un problema de *Big Data* cuando los datos son muy voluminosos (Gutiérrez Puebla, 2018), sino también cuando, no siendo tan voluminosos como otros, el procesamiento que se les realiza es altamente costoso en términos de computación, alargándose mucho en el tiempo. Esto es lo que suele suceder en *Big Data* geográfico: si bien los volúmenes de datos pueden parecer modestos en comparación con otras aplicaciones, el procesamiento geométrico de los datos vectoriales son procesos muy costosos computacionalmente y que se alargan mucho en el tiempo.

Por ejemplo, la base de datos del catastro en continuo de Andalucía tiene un tamaño, bastante respetable, de 110 *gigabytes*. Un juego de datos de este tamaño podría ser procesado, si se tratara por ejemplo de operaciones sencillas con texto, por una sóla máquina en un tiempo razonable. Sin embargo, cuando se trata de hacer operaciones topológicas con los polígonos del catastro, como por ejemplo detectar posibles inconsistencias como solapes, la carga de trabajo excede con mucho la de una sola máquina. En este momento, adoptar una estrategia *Map - Reduce* sería ventajoso.

Sin embargo, la adopción de estas estrategias no son fáciles en información geográfica vectorial. La razón es la siguiente: la información geográfica está intrínsecamente relacionada entre sí gracias a la topología. La etapa *Map* de la metodología *Map - Reduce* funciona bien cuando los subconjuntos de datos a fragmentar para su suministro a distintas máquinas están completamente desconectados los unos de los otros, es decir, no hay relación entre sus miembros. Esto quiere decir que el proceso que haga la máquina "A" de su lote de trabajo está complemente desligado y es autónomo[30] de lo que pueda estar pasando en el lote de trabajo procesado por la máquina "B".

Esto no sucede en información geográfica. La topología relaciona a todos los elementos

[29] Esto es una cuestión de escala. Muchas veces no son máquinas las que asumen el paquete de trabajo, sino otros núcleos del microprocesador de una sólo máquina. Como las máquinas actuales cuentan con varios núcleos de ejecución en sus potentes microprocesadores, son capaces de realizar un trabajo grande fragmentado en paquetes de trabajo más pequeños. El principio metodológico es el mismo, en cualquier caso.

[30] En terminología de datos se dice "análisis atómico".

del conjunto de datos de forma implícita, queramos o no. Estas relaciones por distancia pueden ser vitales o no para el problema en cuestión. Si no son vitales, es decir, si en el procesamiento que hay que hacerle a las geometrías no interfiere ninguna relación topológica con ninguna otra, el proceso es seguro y podemos hacer el *Map* como nos plazca, ya que el procesamiento es atómico para cada geometría. Pero si las relaciones topológicas juegan un papel en el análisis (por ejemplo, porque sea un análisis de áreas de influencia), la metodología *Map* ya no es tan directa, sino que hay que plantear una estrategia para dicha fase específica del problema en cuestión. Lo mismo pasa para el *Reduce*: en esta fase, la reagrupación de resultados puede depender de una relación topológica entre los resultados en sí tan pesada de computar como el cálculo original sobre las geometrías. Esto convierte a cada problema de *Big Data* geográfico en casos únicos que hay que estudiar y solucionar por separado, sin una metodología común a todos ellos, lo que dificulta la aplicación de estas técnicas.

Por todo ello, la discretización del espacio geográfico en teselas pertenecientes a una rejilla ayuda en gran medida a consolidar soluciones más estándares para las aplicaciones de *Big Data* geográfico. La rejilla, al imponer una homogeinización a nivel geométrico y temático en la información de origen, facilita mucho el proceso de *Map*, dado que las teselas están muy acotadas espacialmente y, por lo tanto, su capacidad de relación topológica se reduce mucho. Dada esa acotación espacial, las teselas incurren en relaciones topológicas de contacto exclusivamente con sus ocho vecinas (en el caso de teselas cuadrangulares). No incurren en relaciones topológicas de solape y contención (siempre dentro de un mismo nivel de resolución, se entiende), ni tampoco en relaciones que no sean la distancia con el resto del universo teselar. Por lo tanto, aún en el caso de problemas controlados por una distancia límite, es muy fácil y factible seccionar el espacio geográfico en grupos de teselas en las que tenemos garantizado que el análisis va a ser atómico y autocontenido. Esto es una grandísima ventaja de esta estructura de datos geográficos, ya que abre la puerta a avanzadas formas de procesamiento distribuido, que es la tendencia en las aplicaciones masivas de nube.

I.1.14. La *web* moderna: un nuevo paradigma de *software*

La *web* ha cambiado mucho desde su concepción en los años 70 - 90 (Leiner et al., 2009). Lo primero que tenemos que entender es qué es la *web* y la *Internet* y un poco de su historia.

Para el público en general, "*web*" e "*Internet*" son sinónimos, pero desde un punto de vista técnico no lo son en absoluto. La "*Internet*" es la encarnación moderna de una red que comienza a diseñarse en los años 70, orientada a conectar los centros militares y académicos de los Estados Unidos. Esta red, a diferencia de otras anteriores, se diseñó para mejorar a sus predecesoras sobre todo en términos de robustez y redundancia. El objetivo de su diseño era tener un control exhaustivo sobre el destino de los pequeños paquetes de datos en los que se descomponía un fichero y ser consciente, en todo momento, de si éstos habían llegado a su destino o no, y si no, volver a mandarlos e intentar, si una parte de la red se caía, mandarlos por una ruta distinta. La *Internet*, a gran escala, es un intenso parloteo entre máquinas que están constantemente mandandose mensajes acerca

del destino de estos paquetes de información[31]. Con el tiempo, esta tecnología acabó en el dominio civil por su probada fiabilidad. Por lo tanto, *Internet* es la red básica, un mecanismo de transmisión de datos, pero sin un fin concreto.

La "*web*", sin embargo, es el nombre por el que se conoce la *World Wide Web* (*WWW*) (Toro Bonilla, 2014, Berners-Lee et al., 1994). Esta tecnología funciona sobre la red *Internet* para cumplir, esta vez sí, un propósito concreto. Inicialmente, este propósito no era otro que el que los científicos del CERN en Ginebra tuvieran una forma fácil de compartir, en formato digital, publicaciones científicas internas a la propia institución consultables en una forma precursora de las páginas *web* modernas. Los padres de la *WWW* son el británico Sir Timothy Berners-Lee y el belga Robert Cailliau, y para hacer funcionar el concepto debieron crear muchos subsistemas que hoy son familiares para el público no especialista: los servidores *web*, el protocolo *HTTP*, el formato de hipertexto *HTML*, las direcciones de *Internet URL*, los nombres de dominio (servicios *DNS*, *Domain Name Servers*) y un largo etcétera quizás ya no tan conocidos, pero que en conjunto hacen que la *web* funcione. Así, ahora se entenderá que la *web* es un componente importante del tráfico que se mueve por *Internet*, pero no el único. Los correos electrónicos, los vídeos de YouTube y sistemas mucho más exóticos utilizan los recursos base que proporciona la *Internet* para, con sus propios protocolos y subsistemas, funcionar sobre ella.

La *web*, en origen, era muy sencilla. Su objetivo era crear documentos de hipertexto, es decir, texto que puede soportar enlaces que los vinculan con otros documentos de hipertexto, permitiendo una navegación entre ellos. Entre los inventos de estos dos investigadores estaba, por supuesto, el primer navegador *web*, que le permitía al usuario conectarse a los servidores del nuevo protocolo *HTTP* y controlar la navegación entre los documentos que contenían para ser servidos bajo demanda, en modo consola. Recordemos: texto enlazado. A posteriori llegó *Mosaic*, desde el *National Center for Supercomputing Applications* de Illinois, que fue el primer navegador gráfico, del que descienden todos los actuales. Y a alguien se le ocurrió, entonces, la *frivolidad* de incluir imágenes en los documentos de hipertexto. Pero el foco del sistema seguía siendo texto enlazado más alguna que otra imagen.

Y a partir de ahí la historia va en escalada geométrica. Las grandes casas de *software* se interesan por el concepto y comienzan a desarrollar y comercializar sus propios sistemas para replicar este esquema: servidores *web* para los especialistas y cada vez más potentes navegadores *web* para el público. Llegó la *Guerra de los Navegadores* de finales de los 90 entre *Microsoft / Explorer* y *Netscape / Navigator*. La *web* seguía creciendo, llevando al límite constantemente una tecnología originalmente concebida para intercambiar texto e imágenes, intentando con ello hacer juegos, vídeo y aplicaciones que cada vez se parecieran más a las aplicaciones de escritorio que instalábamos en nuestras máquinas. Pero obviamente la tecnología estaba muy limitada y era constantemente parcheada, sin demasiado control. Cada fabricante de servidores o de navegadores incluía sus propias innovaciones,

[31] Nótese la inmensa utilidad militar de esta característica. En un escenario dominado por la Guerra Fría y el miedo al concepto de la "Destrucción Mútua Asegurada", la habilidad de una red informática de transportar, de forma inteligente, información evitando nodos de la red que ya no existen (es decir, ciudades e instalaciones militares arrasadas en el intercambio nuclear) es más que evidente para mantener vivas las comunicaciones en este escenario.

muchas veces incompatibles con los sistemas de otros fabricantes. Fueron años duros para las tecnologías *web*. La llegada del producto *Macromedia Flash*, que consistía en incrustar un *plugin* dentro de los navegadores con funcionalidad multimedia, permitió hacer cosas impensables de otra forma, a costa de depender del desarrollo de una única compañía, *Macromedia*, desarrollo que por supuesto era propiedad industrial suyo y por tanto, no abierto.

Afortunadamente, esta situación comienza a encauzarse con la llegada de un nuevo estándar, *HTML5*, orientado a ofrecer, con estándares abiertos y transparentes a los que todos los fabricantes puedan acogerse para crear sus productos, una completa renovación de la *web*, modernizándola de forma radical en todos los aspectos, incorporando tecnologías como vídeo, sonido, gráficos tridimensionales, superficies de dibujado libre con control pixel a pixel y un larguísimo compendio de mejoras. Esto convierte a los navegadores *web* en mini-sistemas operativos dentro de los sistemas operativos, unas *sandbox*[32] de cada vez más potencia. Si a esto añadimos los extraordinarios avances acaecidos a lo largo de las décadas en lo que a velocidad de transmisión de datos se refiere (por ejemplo, la fibra óptica), el panorama de la *web* se transforma radicalmente.

Pensemos en un programa de escritorio de los que llamamos "pesados", es decir, un programa de altas prestaciones con consumos de recursos de computación locales (memoria, tarjeta gráfica, microprocesador) intensos. Por ejemplo, *Photoshop* o, en nuestro ámbito, *ArcGIS*, o un videojuego en tres dimensiones, que es un *software* que demanda recursos de computación como pocos. Todo se ejecuta en nuestra máquina. Pueden conectarse a *Internet* para examinar el portfolio de plantillas artísticas de la empresa *Adobe*, o para explorar un servicio de imágenes de satélite, o para participar en una partida entre varios jugadores, pero la computación principal se sigue ejecutando en nuestras máquinas. La única forma de que esto funcione es instalando el programa principal en ella. Este programa puede utilizar servicios accesorios en *Internet*, pero el núcleo está instalado en la máquina local. Esto, obviamente, permite hacer cosas como piratear el *software*.

Esto está cambiando y va a cambiar más todavía en el futuro cercano. La combinación de las nuevas tecnologías en el cliente (navegador), el ecosistema *HTML5*, permite crear aplicaciones *web* casi indistinguibles ya de la experiencia de usuario de una aplicación nativa de escritorio instalada en nuestra máquina. Los grandes avances en conectividad y velocidad en la red hace que el tránsito de grandes volumes de datos sea factible entre los servidores de la compañía y la máquina cliente. De esta forma, no es muy descabellado tener un *Photoshop* que funcione dentro del navegador, o una versión de *ArcGIS* 100 % *online* (*ArcGIS Online*) o incluso jugar a un videojuego por *streaming* en *Google Stadia*.

Todos estos sistemas tienen algo en común: el programa principal ya no se ejecuta en la máquina local. La máquina local se limita a mostrar al usuario lo que está pasando en las máquinas de la compañía, donde el programa realmente se ejecuta, y proporcionarle

[32] Técnicamente, el término *sandbox* hace referencia en ingeniería de *software* a un entorno controlado y aislado dentro de un sistema donde los procesos que se ejecutan en él están completamente aislados del resto del sistema y, por tanto, son incapaces de hacer daño al sistema anfitrión. Los navegadores *web* se diseñan según este principio. Hay que tener en cuenta que el navegador *web* va a recibir código desde el exterior de la máquina que es ejecutable en la máquina receptora. Hay que limitar el daño que un posible código malicioso pueda hacer, aislando el navegador del resto del sistema de forma radical.

un método de interaccionar con él. Al aplicar un filtro fotográfico, ya no hay un *Photoshop* con la foto almacenada en nuestra memoria y un programa aplicando sobre ella el filtro, ni al realizar un *buffer* sobre unos polígonos, ni tan siquiera en el complejísimo proceso de dibujado de una escena tridimensional de un juego de alta gama a 60 *frames* por segundo, acelerado con tarjetas gráficas de última generación[33]. Todo esto se ejecutará en los servidores de las compañías, y los clientes simplemente verán el resultado y seguirán trabajando en remoto con el programa.

Paradójicamente, esto supone un movimiento pendular espectacular en la forma de concebir la computación. La computación ya era así en los años 50, 60 y 70, antes del advenimiento del *ordenador personal*. Las grandes instituciones científicas, académicas y gubernamentales compraban enormes máquinas, llamadas *mainframes*, que servían tareas de computación, a través de terminales "tontas" (una pantalla de texto de fósforo verde con un teclado, eso sí, de una calidad de los que ya no se fabrican). Varias decenas o cientos de usuarios se conectaban remotamente al ordenador central e interaccionaban con él de forma similar a la descrita.

Las ventajas de esta forma de comercializar el *software* son muchas. Primero, el pirateo de las propiedades intelectuales de las compañías es prácticamente imposible a no ser que se produzca una brecha de seguridad seria en los servidores de la compañía. Dado que el *software* que realmente contiene el núcleo de la propiedad intelectual (el algoritmo del filtro fotográfico, el motor de geometría computacional de un SIG o el código del videojuego) se ejecuta siempre en los servidores de la compañía y jamás es instalado en las máquinas clientes, el acceso al mismo es a priori imposible. Esto hace que los usuarios deban adquirir por fuerza licencias comerciales de uso. Y, por otro lado, se abre muchísimo el abanico de posibilidades de articular estas licencias. Por ejemplo, la compañía tendrá ahora detalladísimos datos sobre el uso y la experiencia de uso de los usuarios interaccionando con el programa[34], por lo que puede comercializar el programa con opciones de "pago por uso". Así, al final del mes, el usuario recibe una factura acorde al uso que ha hecho del programa ("300 filtros fotográficos a razón de 0,04€ el filtro: 12€", por ejemplo, o "60 buffers sobre 700MB de datos vectoriales, a razón de 0,001€ el *megabyte* por buffer: 42€", etc.).

Esta nueva forma de comercializar el *software* se llama *Software as a Service* (*SaaS*), es decir, "*software* como servicio", en contraposición al modelo anterior del "*software* como producto". Todas las grandes compañías de *software* están creando versiones de sus productos bajo esta modalidad, que será la predominante en el futuro muy cercano.

Aparte de las ventajas para la compañía, el usuario también recibe beneficios. Primero, la facilidad de uso aumenta, puesto que estas plataformas vienen acompañadas de una panoplia de servicios accesorios: datos siempre disponibles en la nube y siempre seguros, *software* siempre actualizado, etc. Sin embargo, los que más ganan son los usuarios más técnicos, porque este tipo de servicios no sólo se ofrece en forma de usable y ergonómico producto *web* para facilitar su uso, sino también como *API* de servicios (Toro Bonilla, 2014).

[33] En teoría. Eso es lo que les gustaría a los promotores de *Stadia*. Los resultados no son muy espectaculares por ahora pero sin duda lo serán en el futuro muy cercano. Es cuestión de tiempo.

[34] Algo que, de por sí, ya tiene un valor muy importante.

Una *API* de servicios proporciona al ingeniero de *software* un canal directo con las funcionalidades de la plataforma que puede aprovechar en sus propios programas y flujos de trabajo. En lugar de tener que abrir el *ArcGIS Online* para hacerle buffers a 300 capas cada día, el programador puede crear un programa en, por ejemplo, el lenguaje *Python* que le permite acceder a la operativa de *buffers* de la plataforma de forma automática y ejecutar los procesos en cadena. Pero este concepto va más allá: este programa en *Python* podría hacer esta operación de los *buffers*, obtener cartografía, y después, con esa cartografía, conectarse a la *API* de servicios de *Photoshop* versión "nube" y aplicarle un filtro fotográfico distinto a cada mapa. El programador estaría enlazando en un flujo de trabajo los servicios de dos compañías *SaaS* con productos y orientaciones muy distintas para hacer un tratamiento de datos complejo.

Y éste es el futuro de la computación. Para el usuario, todas las aplicaciones serán *web*. Deberá estar constantemente conectado a la red (ya lo estamos) y tendrá que pagar por todos los servicios que use. La necesidad de tener ordenadores potentes se reducirá muchísimo puesto que la computación intensa ya se hace "en la nube". Para el programador, tendrá a su disposición un inmenso mercado de *API* de servicios extraordinariamente especializados que combinar y hacer trabajar juntas, posiblemente para crear sus propias *API* de servicios de sus propios productos y enriquecer estos con capacidades que costaría muchísimo desarrollar desde cero, si es que esto es tan siquiera posible.

En este trabajo de tesis se sigue este esquema. El producto de *software* descrito e implementado durante su desarrollo es una muestra sencilla, pero realista en sus planteamientos, de cómo se construye y diseña este nuevo *software* orientado a servicios escalables en la "nube".

I.1.15. Experiencia del grupo de investigación

El grupo de investigación al que se adscribe el autor de esta tesis ha centrado parte de su investigación en el uso integrado de fuentes de información espacial, cuyo volumen (imágenes de satélite, LIDAR, etc.) y/o periodicidad (catastro, población, datos estadísticos, etc.) ha ido aumentando sin pausa por lo que se podría hablar de *Big Data espacial* y para su gestión y análisis se han tenido que ir incorporando nuevas tecnologías, especialmente las bases de datos relacionales espaciales, y nuevas aproximaciones metodológicas, como el *Machine Learning*, a la vez que, en todos los casos, se han tenido que ir incorporando nuevas herramientas de geovisualización.

En este sentido, el grupo de Investigación "Ordenación litoral y Tecnologías de Información Territorial" (registrado en el PAIDI como RNM-177), ha estado vinculado a las Tecnologías de Información Geográfica (TIG) desde su creación. En la segunda mitad de la década de los ochenta se situaba entre los grupos de investigación pioneros en el uso de la teledetección espacial, inicialmente en sus aplicaciones a la gestión ambiental y, especialmente, al análisis del medio costero (Moreira Madueño et al., 1992). En este tipo de aplicaciones se incorporaron imágenes de satélite de media resolución (Landsat-MSS y Landsat-TM), especialmente adaptadas al dinámico medio marino (Ojeda Zújar, Sanchez et al., 1995), para evolucionar en la actualidad a nuevas aplicaciones con imágenes de alta

resolución y tratamientos de imágenes más sofisticados (Sánchez Carnero et al., 2014). En la década de los 90 se incorporaron a los trabajos del grupo de investigación nuevos datos y aplicaciones (uso de MDT, análisis espacial sobre datos vectoriales, análisis de redes y accesibilidad, evaluación de riesgos naturales...) al incorporar el *software Arc-Info* (Ojeda Zújar y Vallejo Villalta, 1995) como herramienta para el análisis de datos geográficos en proyectos de gestión territorial que necesitaban de la integración de datos del medio natural y variables antrópicas, desarrollándose en la actualidad gran parte de estas tareas de integración de datos y análisis (Ojeda Zújar, Álvarez Francoso, Martín Cajaraville et al., 2009, Ojeda Zújar, Vallejo Villalta et al., 2007) con paquetes de programas más potentes tanto privados como de código abierto (*ArcGIS* o *QGIS*). En la primera década de este siglo se incorporaron a los proyectos y trabajos del grupo (Pérez Alcántara, Díaz Cuevas et al., 2016, Pérez Alcántara, Ojeda Zújar et al., 2017, Fraile Jurado et al., 2017, Ojeda Zújar, Fraile Jurado et al., 2021) el uso de grandes conjuntos de datos masivos (parcelas y edificios catastrales, geocodificación de población, etc...) para cuya gestión se incorporaron las bases de datos espaciales en todos los procesos metodológicos (*PostgreSQL / PostGIS*) y otros *software* de código abierto (*GRASS*) para el procesamiento masivo de datos ráster o nubes de puntos *LIDAR* con *LAS*[35] (Fernandez Núñez et al., 2017, Ojeda Zújar, Vallejo Villalta et al., 2007). La incorporación de nuevos miembros al grupo, ligados a la aplicación de la teledetección al comportamiento fenológico de la vegetación ha posibilitado desarrollar proyectos y publicaciones (Rodríguez Galiano et al., 2016) centrados en datos masivos de series temporales de variables proporcionadas por nuevos sensores y plataformas espaciales, incorporándose igualmente nuevas técnicas y metodologías de análisis centrados en *Machine Learning* (Caparros Santiago et al., 2020). Por otra parte, en todos los proyectos de investigación se ha considerado que para, la difusión de los datos, además de las publicaciones científicas, era esencial la publicación de cartografía temática (Ojeda Zújar, 2010), una tradición que inevitablemente ha ido evolucionando hacia la difusión de información geográfica (Iglesias Campos et al., 2011) a través de *Internet* (servicios de mapas, atlas digitales), conllevando la toma de contacto con nuevas tecnologías y nuevo desarrollos metodológicos (servidores de mapas, diseño de clientes *web*, etc.) orientados a la geovisualización y herramientas de exploración de datos a través de la *web* (Ojeda Zújar, Álvarez Francoso, Díaz Cuevas et al., 2013, Álvarez Francoso, Ojeda Zújar et al., 2020, Prieto Campos et al., 2018).

En esta tesis se trata de dar un paso más en la necesaria evolución de las metodologías y tecnologías que se utilizan en el Grupo de Investigación, especialmente en 4 aspectos claves:

- dar cabida a las nuevas fuentes de información que por su volumen, velocidad de actualización y variedad (vector, ráster) podrían catalogarse de *Big Data* espacial y para ello, en primer lugar, se intenta profundizar en el concepto de dato geográfico (geographical data science) y en el correcto uso de las BBDD espaciales (tanto *SQL* como *NoSQL*) para su almacenamiento y gestión;

- la necesaria evolución en los métodos y tecnologías que posibilitan la necesaria in-

[35] *LAS* (*LASer*) es un formato de intercambio de nubes de puntos *LIDAR* (LAS Working Group (LWG) Contributors, 2021).

tegración espacial de los datos. Se intenta dar un salto evolutivo apostando por el uso de teselas multiescalares asimétricas (ver más adelante sus ventajas), tanto en su proceso de generación, como en los diferentes algoritmos que permiten la adscripción a las mismas de las diferentes variables y estructuras de datos geográficos;

- una vez se obtengan los datos integrados espacialmente en teselas multiescalares y multidimensionales explorar la adecuación de estas estructuras para la utilización de técnicas de IA y *Machine Learning*;
- por último, quizás el más innovador, realizar todas la tareas anteriores en nuevo entorno optimizando las posibilidades de *Internet*. Para ello, esta tesis se centra en dos aspectos críticos:
  - desarrollar una plataforma *cloud*, basada en microservicios, que serán los encargados de todos los procesos computacionales y analíticos ligados en los puntos anteriores, haciendo un uso masivo de *Parallel Computing*;
  - desarrollar un nuevo tipo de clientes que a través de geovisores y dashboard permitan una mayor interacción con los datos originales.

Todo el proceso de desarrollo de esta tesis se ha ido gestando en un largo periodo de tiempo en el que el autor ha estado integrado en proyectos del Grupo, entre los cuales han tenido un papel central el proyecto de la Consejería de Obras Publicas, Transporte y Vivienda de la Junta de Andalucía (*Georreferenciación, caracterización estadística y estrategias de difusión del espacio residencial en Andalucía*) iniciado en 2014 y, sobre todo, el proyecto del Plan Nacional (*CSO2014-51994-P*) en el que el autor fue investigador contratado (FPI) durante 4 años hasta 2019.

I.2 Hipótesis, objetivos, área de estudio y estructura de la tesis

I.2.1. Hipótesis y objetivo general

De lo comentado en el epígrafe I.1.11 "Contexto actual de estructuras de datos y tecnologías geográficas" (pag. 25) se deriva la casi obligación de orientar los trabajos con conjuntos de geodatos masivos, así como el obligado y demandado proceso de su integración espacial, en un nuevo entorno de trabajo en *Internet* (entornos en la nube o *Cloud*). Por ello, la hipótesis general de este trabajo es testar las posibilidades de desarrollar una plataforma *Cloud* que, a través de microservicios, realice las tareas del modelado y almacenamiento en bases de datos de los procesos de integración espacial de la geoinformación ambiental (Ruíz Sinoga et al., 2015) y socio-económica, de su análisis espacial, así como su difusión *web* y el acceso a datos y resultados a través de clientes *web* ligeros altamente interactivos orientados al mapa y al dato. Por lo tanto, el objetivo general de este trabajo es el desarrollo de una plataforma *Cloud* que, de forma integral, posibilite la realización de todos los procesos comentados con anterioridad.

I.2.2. Hipótesis y objetivos específicos

Como objetivos específicos de este trabajo de tesis tenemos los descritos a continuación:

1. **el dato geográfico: nuevas metodologías y tecnologías para su almacenamiento:** la proliferación de nuevas fuentes de información (*Big Data*, repositorios públicos de geoinformación bajo la filosofía *Open Data*, datos provenientes del cada vez más sensorizado *Internet of Things* (*IoT*), redes sociales, mobiles, etc.) es una realidad hoy en día y, por su volumen, velocidad de generación y actualización y variedad (datos estructurados y no estructurados), podrían entrar, en su mayor parte, dentro de la categoría de *Big Data* espacial. La hipótesis de trabajo en relación a los datos es evaluar si las ventajas que presentan las nuevas funcionalidades de bases de datos (especialmente las BBDD espaciales) podrían facilitar el almacenamiento, gestión e integración espacial de los mismos. Por lo tanto, un objetivo específico para ello será profundizar en el concepto de dato geográfico (*Geographical Data Science*) y en el correcto uso de las BBDD espaciales (tanto clásicas *SQL* como las más recientes *NoSQL*), para seleccionar aquellas más adecuadas para conseguir el objetivo general (Stonebraker, 2010);

2. **capacidad de la teselación multiescalar para la integración de información geográfica armonizada:** ya se ha comentado con anterioridad que, aunque se ha realizado un enorme esfuerzo en la normalización y estandarización de los datos geográficos a nivel individual (INSPIRE Directive, 2007), queda aún pendiente y es un tema persistente en los foros sobre generación y análisis de los datos espaciales, la reflexión y normalización de las adecuadas estructuras espaciales para facilitar su integración espacial. La hipótesis de trabajo se basa en considerar si las estructuras de teselas multiescalares y multidimensionales podrían ser estructuras adecuadas para los procesos de integración espacial de datos que generalmente utilizan diferentes formatos adaptados a los dos modelos existentes para modelar la realidad territorial (vector y ráster). Por lo tanto, un objetivo específico y, quizás el de mayor componente de reflexión metodológica, será evaluar la viabilidad de estas estructuras de teselas multiescalares, especialmente las que cubren exhaustivamente un espacio planar (teselación regular regulares, básicamente cuadrados, hexágonos o triángulos) y optar por la más adecuada para facilitar los procesos de integración de los conjuntos de variables temáticas derivadas de los grandes conjuntos de datos temáticos (catastro, población, espacios protegidos, etc) utilizados en esta tesis, así como desarrollar los diferentes algoritmos que permiten la adscripción a las mismas de las diferentes variables y estructuras de datos geográficos;

3. **capacidad de la teselación multiescalar de facilitar los análisis de *Machine Learning*:** en el contexto del análisis espacial, la irrupción y desarrollo de nuevas tecnologías ligadas a la Inteligencia artificial (IA) y, especialmente, las ligadas a las técnicas de *Machine Learning*, ha abierto nuevas posibilidades de análisis geográfico mejor adaptado a la situación actual de proliferación de geodatos masivos de uso abierto. La hipótesis de trabajo ligada al análisis espacial se basa en evaluar si la integración espacial de los datos geográficos originales en estas estructuras de teselas multiescalares y multidimensionales podrían facilitar los trabajos analíticos con técnicas de IA y *Machine Learning*. Por lo tanto, otro objetivo específico de este trabajo será testar varios procesos analíticos con estas técnicas sobre las estructuras de teselas multiescalares y multidimensionales generadas y evaluar su adaptación a los requerimientos para potenciales análisis de IA o *Machine Learning*;

4. **posibilidades de geovisualización de la teselación multiescalar:** finalmente, también se han comentado los grandes avances producidos en los últimos años en la geovisualización de datos en clientes *web* ligeros (más rápidos, más interactivos y más autónomos), de tal forma que la conclusión de cualquier trabajo geográfico, donde los datos sean masivos, ya no se expresa exclusivamente a través de mapas estáticos (o su equivalente como servicio de mapas), sino a través de clientes *web* que integran tareas de geovisualización y semiología (geovisores), con un conjunto de nuevos componentes gráficos (*widgets*) que contribuyen no solo a la geovisualización, sino a la exploración interactiva de los datos a través de paneles de mando (*dashboards*). La hipótesis de trabajo en relación a la difusión *web* de los datos integrados en teselas multiescalares y multidimensionales se basa en evaluar si estas estructuras se adaptan bien a los requerimientos de las nuevas tecnologías para el desarrollo de clientes *web* modernos (*HTML5*, *WebGL*, *Canvas*, *D3*, etc.) con una

Figura I.2.1: La Comunidad Autónoma de Andalucía, delimitada en naranja. En el interior, sus ocho provincias. Fuente: elaboración propia.

orientación predominantemente al dato. Por lo tanto, un objetivo específico final para este trabajo, será el diseño y desarrollo de una *API* de acceso a los datos nativos de las teselas y el de clientes *web* que consuman estos datos para poder testar la adaptación de las estructuras de teselas multiescalares a los requerimientos de los diferentes componentes de los clientes *web* (geovisores, *widgets*, gráficos interactivos y *dashboards*).

I.2.3. Área de estudio

Para el desarrollo de esta tesis su utilizarán conjuntos de datos masivos originales procedentes de organismos públicos públicos que tengan la propiedad de cubrir toda la superficie de la Comunidad Autónoma de Andalucía. Andalucía cuenta con una superficie 87.268 kilómetros cuadrados y una altitud media de aproximadamente 380 metros. Cuenta con 910 kilómetros de costa y una gran variedad de ecosistemas distintos.

I.2.4. Estructura de la tesis

Esta tesis está estructurada en cinco partes.

- **Parte I: Planteamientos generales:** una introducción a las diversas temáticas y estado de la cuestión del objeto de estudio de la tesis, incluyendo específicamente las hipótesis, objetivos, área de estudio y estructura de la tesis.
- **Parte II: Fuentes de datos y metodología:** en esta parte se desarrolla la selección de los juegos de datos que se han elegido para las pruebas de la plataforma y metodología propuestas, así como la metodología a seguir en el diseño de la plataforma, sin entrar en detalles de implementación ni en tecnologías escogidas para ello. Se desarrolla en el orden en el que se plantea el diseño de la plataforma: características de los datos originales a teselar, definición de las características rejilla multiescalar asimétrica, metodología de teselado a seguir, aplicación de técnicas de *Machine Learning*, planteamientos de mecanismos de difusión y, finalmente, vías de visualización de la información teselada.
- **Parte III: Resultados:** en esta parte se entra en los detalles de la implementación de la arquitectura de microservicios que hace funcionar a la plataforma y en las tecnologías utilizadas. Se discute en profundidad también el tratamiento que se le ha hecho a los conjuntos de datos seleccionados para someterlos al proceso de teselado, así como de los métodos de visualización utilizados. Sigue más o menos la estructura de la Metodología: descripción de la arquitectura de microservicios, creación y estructura de la base de datos que da soporte de almacenamiento a la plataforma. Formalización de las características de la rejilla multiescalar asimétrica, transformación y tratamiento de los datos originales, descripción del núcleo de implementación de *software* de la plataforma, la librería *libcellbackend* e implementación de los flujos de teselado. Con los resultados finales de los procesos de adscripción se llevan a cabo análisis *Machine Learning* (*K-Means* y *Random forest*) sobre los datos, con sus resultados, y, finalmente, se explica la creación de aplicaciones cliente para la visualización de la información teselar (geovidores y *widgets* con las plataformas *Mapbox* y *CARTO*).
- **Parte IV: Discusión, Conclusiones y Líneas de futuro: discusión** de resultados obtenidos al teselar y visualizar los conjuntos de datos seleccionados con la plataforma descrita, estructurándola en función de los objetivos específicos definidos, y contextualizándolos con las funcionalidades existentes en las plataformas actualmente disponibles. Incluye las **conclusiones** finales acerca de la metodología propuesta y su implementación en el prototipo *Cell*. También incluye **Líneas de futuro**, donde se examinan vías de mejora y ampliación de la metodología y el prototipo de plataforma propuesto, de cara a su puesta en producción en nuevos proyectos.
- **Parte V: Bibliografía y anexos:** consta de la **bibliografía** citada y un **anexo** dedicado a la estructura de los repositorios de código *Git* que corresponden a la implementación del prototipo de la plataforma propuesta y de los datos de origen seleccionados.

Parte II

FUENTES DE DATOS Y METODOLOGÍA

II.1 Bases metodológicas

El Diablo está en los detalles.

Proverbio inglés

Comenzamos esta sección de la tesis destinada a la descripción de la metodología y los datos utilizados para demostrar el funcionamiento de la plataforma con un necesario, pero breve, desarrollo de las líneas maestras de la metodología, que será necesario para el entendimiento de los posteriores capítulos.

La metodología propuesta es compleja, integrando dos aspectos principales:

- desarrollo de una metodología de adscripción de datos geográficos a rejillas multiescalares para el desempeño de análisis multivariante normalizado geométricamente;
- el desarrollo, para implementar la anterior metodología, de una plataforma de *software*, con una parte pública interactiva (el *frontend*) ejecutable en un navegador *web* y otra parte privada (el *backend*) para su ejecución en servidores y cuyo punto de acceso exterior es una *API REST*. Esta implementación, en su parte servidora, toma la forma de una arquitectura de microservicios que trabajan coordinadamente para llevar a cabo tareas de tratamiento y publicación de datos.

Como resumen general, podemos diferenciar una serie de fases metodológicas, que se muestran en el esquema II.1.1. En dicho esquema se pueden diferenciar las siguientes fases principales:

1. **adquisición y transformación de datos:** selección, estudio y procesamiento de fuentes de datos originales que sirvan para ejemplificar el funcionamiento de la plataforma;
2. **análisis de las necesidades y propiedades del proceso de teselado:** definición de las características del proceso de teselado de los datos originales, que dependen de la utilidad que se les quiera dar;
3. **proceso de teselado:** implementación de las definición del teselado y ejecución del procedimiento;
4. ***Machine Learning*:** realización de procesos de *Machine Learning* sobre los resultados;

5. **difusión de resultados:** difusión de los resultados obtenidos.

II.1.1. Conceptos previos

Es muy importante delimitar terminológicamente algunos conceptos que se van a utilizar a partir de este capítulo de forma sistemática.

Datos o información original Los datos que se pretenden transformar al formato de rejilla asimétrica.

Rejilla Por "rejilla" se hace referencia a la definición del marco de referencia geométrico que va a ser utilizado para la transformación de los datos originales. La rejilla tiene una definición matemática que determina sus propiedades geométricas.

Tesela Una "tesela" es una unidad de adscripción determinada por la definición matemática de la rejilla. En una rejilla pueden existir conjuntos teselares de muchas resoluciones (tamaños) distintas.

Adscripción Por "adscripción" se entiende el proceso de convertir o transformar información temática vinculada a las geometrías de los datos originales a las teselas definidas por la rejilla. Existen muchos posibles mecanismos de adscripción, que dependen fundamentalmente de la naturaleza de las dimensiones temática y geométrica de la información geográfica de origen y su interacción con las teselas determinadas por la rejilla.

Análisis de adscripción Las distintas metodologías posibles de conversión de la información original a las teselas de la rejilla, como se ha descrito en la entrada "adscripción".

Variable de adscripción El resultado final de un análisis de adscripción. Un análisis puede devolver una o muchas variables de adscripción.

Vector de adscripción El vector de adscripción es el conjunto de variables de adscripción calculadas y almacenadas en una tesela en concreto. "Vector" aquí no se refiere al concepto geométrico de vector tal y como solemos entenderlo en información geográfica, sino al concepto matemático, es decir, un grupo de elementos ordenados como lista [variable0, ..., variableN]. Es un término heredado del campo del *Machine Learning* e Inteligencia Artificial, donde constituye un elemento de datos orientado a la modelización de las características de un fenómeno, y necesario para su clasificación y/o entrenamiento de los modelos de *Machine Learning*.

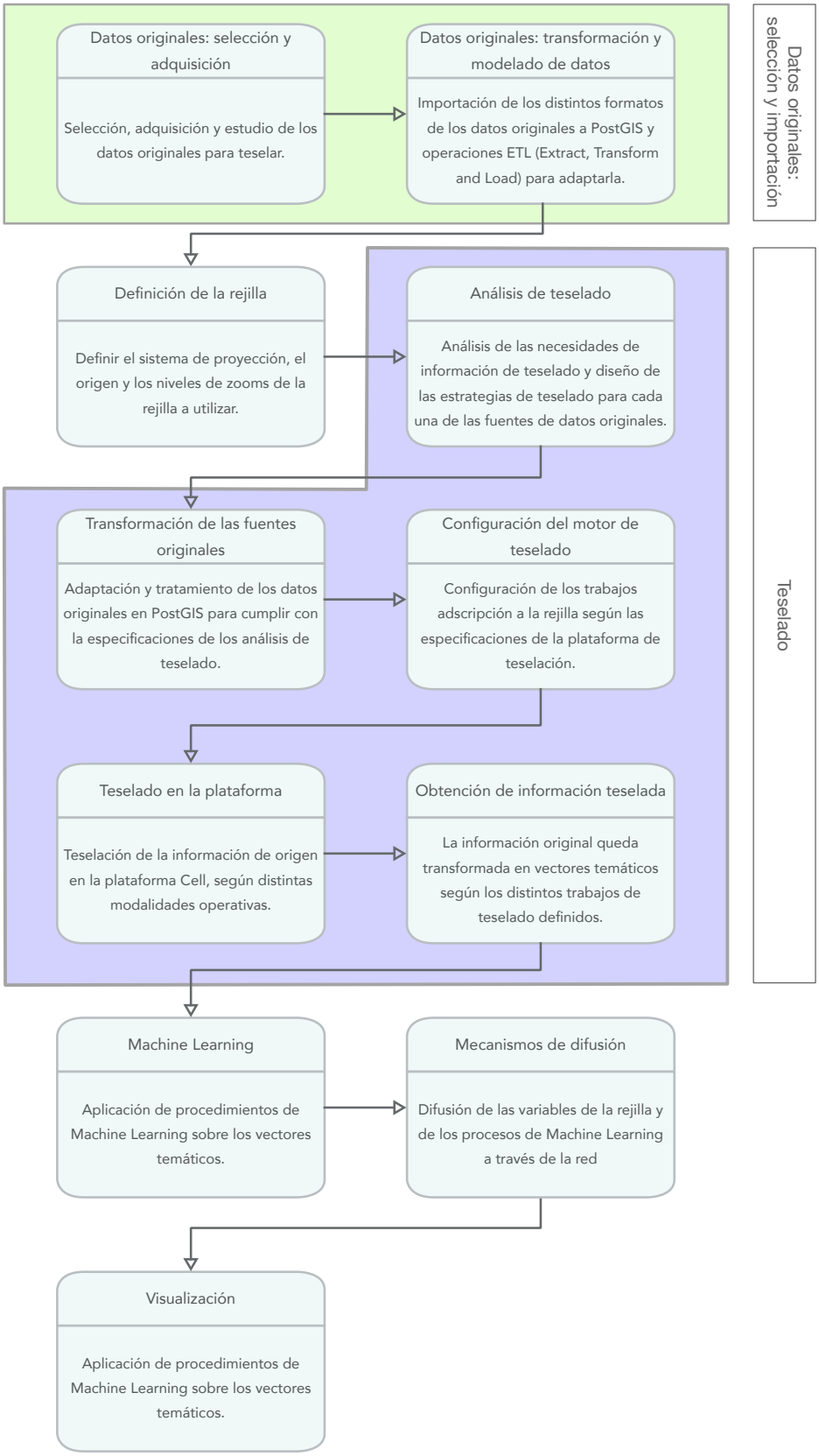

Figura II.1.1: Esquema metodológico. Cada recuadro corresponde a un capítulo de esta parte de la tesis, mientras que los dos grandes bloques señalados son pasos comentados en los capítulos cuyo nombre se muestra a la derecha, en vertical.

II.2 Datos originales: selección e importación

En este capítulo se describen los conjuntos de datos seleccionados para ser procesados por la plataforma propuesta, su estructura y su adaptación para las necesidades de la plataforma.

En la elección de datos para mostrar las capacidades de la plataforma se han observado los siguientes criterios:

1. **su extensión territorial:** los conjuntos de datos debían ser completos para el área de estudio seleccionada, es decir, para todo el conjunto de la Comunidad Autónoma de Andalucía;

2. **su interés temático:** se han seleccionado conjuntos de datos que tengan sinergias temáticas entre sí para realizar tests y aplicaciones de las funcionalidades de *Machine Learning* con una vocación inter-temática. También se incluyen conjuntos de datos con potencial de creación de índices territoriales, así como información de contexto con objeto de filtrar la información teselada a ámbitos de interés territorial específico, como provincias, municipios, etc.;

3. **su tamaño:** a ser posible, se han seleccionado conjuntos de datos de elevado peso en cuanto a tamaño físico de los mismos. Aunque la información vectorial no es muy pesada en sí, su procesamiento sí lo es;

4. **presencia tanto de información vectorial como ráster:** es interesante probar la plataforma con las dos estructuras de datos más usuales de información geográfica. Dado que uno de los objetivos de este trabajo es describir y definir un prototipo de implementación de una nueva estructura de información geográfica, tiene mucho sentido que esta nueva estructura sea construible tanto a partir de información vectorial como ráster, aunque sea para su transformación;

5. **diversidad geométrica en información vectorial:** debido a sus características geométricas y topológicas, la adscripción de información vectorial puntual, lineal y poligonal suponen retos y metodologías distintas que han de ser evaluadas. Por ello, se intentará incluir fuentes de datos de los tres tipos geométricos vectoriales base;

6. **su complejidad geométrica:** este aspecto hace referencia especialmente a las capas de datos poligonales, donde su complejidad se expresa bien por el número de

polígonos presentes, bien por su tamaño medio, o bien por la densidad de vértices que presentan. Se espera que un mismo mecanismo de adscripción de datos sea más lento en capas con más complejidad geométrica que en las sencillas;

7. **distintas escalas de significación:** se han seleccionado fuentes de datos con distintas escalas significativas, desde las que se mueven en el ámbito métrico como las que se mueven a escalas autonómicas;

8. **presencia de datos tanto cuantitativos como cualitativos:** las dos tipologías de datos, como es usual en todas las facetas de su tratamiento (Olaya, 2016), precisan en esta metodología de estructuración de datos de técnicas de adscripción teselar diferenciadas;

9. **recubrimientos del ámbito de estudio tanto total como parcial:** aunque la mayoría de las técnicas de adscripción no varían según esta propiedad de los juegos de información seleccionados, sí es interesante contar con ambos para probar el comportamiento asimétrico, desde el punto de vista espacial, de la estructura de datos propuesta. Además, un recubrimiento total de juegos de información pesados, como puede ser el catastro, también cumple el objetivo de probar el funcionamiento de la plataforma propuesta y de la metodología con grandes conjuntos de datos;

10. **su densidad de datos temáticos:** se han seleccionado también paquetes de datos que presentan, dentro de un único paquete, un elevado número de datos temáticos.

Asimismo, otros conjuntos de datos se han seleccionado no por su componente temática, sino por ser divisores territoriales, como por ejemplo divisores administrativos, con el objeto de actuar como filtros para crear estudios de detalle en áreas específicas.

A continuación se describen los conjuntos de datos seleccionados. Puede extrañar no encontrar datos lineales en el conjunto. Esto se debe a que para las pruebas de la plataforma se ha optado por testear el procesamiento de la geometría más simple (los puntos) y la más compleja (los polígonos). Sin embargo, cabe destacar que la plataforma tiene un diseño muy flexible y abierto y que la creación de análisis de teselado o adscripción de fuentes de datos lineales no difiere de las implementadas para el caso de puntos o polígonos, por lo que su inclusión es perfectamente factible con un esfuerzo razonable.

Tampoco hay datos que sean directamente ráster, sino elaboraciones a partir de los mismos. Esto es perfectamente intencionado. Como se verá en los próximos capítulos, el hacer una plataforma que toma demasiadas decisiones acerca de los formatos y estructuras de los datos es contraproducente de cara a su versatilidad. La plataforma oferta un conjunto de análisis de teselado que funcionan con condiciones específicas en lo referente a la geometría de los datos de origen, ya sean, en las pruebas de prototipo realizadas, puntos o polígonos. De dónde se obtengan y por qué procedimiento estos puntos o polígonos para realizar los procesos de adscripción es responsabilida del operador de la plataforma, que tiene total libertad para decidir qué procesos de transformación va a aplicar a sus datos originales para adecuarlos a esos requerimientos específicos de los análisis de teselado implementados. De esta manera, un mismo juego de datos original, transformado de formas diversas, puede dar lugar a análisis de adscripción con objetivos distintos.

II.2.1. Datos puntuales

Los datos puntuales presentan un alto interés para su teselado, ya que esta geometría está cobrando mucha importancia dentro del *Big Data* geográfico, puesto que es la geometría más usual en la que se recogen grandes bancos de datos, como por ejemplos los pertenecientes a la geolocalización de dispositivos portátiles (teléfonos móviles y otros dispositivos con GPS[1]). Además, otras geometrías más complejas, como por ejemplo los polígonos, pueden ser abstraidos en ciertos escenarios como puntos, consideraciones de pérdida de precición mediantes, obteniéndose de los mismos un centroide de un tipo u otro. Esta técnica se utiliza por ejemplo cuando se quiere adscribir la información de un juego de datos que ya está en formato de rejilla a otra rejilla distinta, o cuando se teselan juegos de datos ráster.

Hay que destacar que, aunque en origen puede que no sean datos puntuales, algunas de las fuentes de datos seleccionadas en este apartado lo han sido porque el tratamiento que se les va a dar es puntual.

Los juegos de datos puntuales seleccionados se describen a continuación.

Modelo Digital del Terreno

Los modelos digitales del terreno modelizan el campo de elevaciones de un área geográfica en base a una trama de puntos con cotas de altura asociadas. Esta trama puede ser regular, en formato de rejilla, con los puntos espaciados a intervalos regulares, formando una malla (figura II.2.2), o irregular, con interpolación por TIN (*Triangular Irregular Network*) (figura II.2.1) (de Berg et al., 2000), con puntos de cotas discretos tomados en puntos que no tienen una correlación espacial regular entre ellos.

Los Modelos Digitales del Terreno (MDT) constituyen por tanto una aproximación discretizada por puntos de un fenómeno espacial continuo, y la espaciación de los mismos, en el caso de las mallas regulares, se basan en tomar, para un área de territorio determinado, siempre del mismo tamaño, un único punto central con un dato representativo para todo el área, como se hace en los datos ráster. En ese sentido, se puede hablar de la resolución del MDT, magnitud expresada en la unidad del sistema de referencia elegido, y que hace referencia a la separación entre los puntos regulares de cotas. Así, un MDT de resolución de 100 metros indica que, sobre un sistema de proyección proyectado que usa metros como unidad, cada punto de cota está separado de sus vecinos a distancias regulares de 100 metros.

Los MDT, independientemente de si son regulares o no, forman una red de datos anclados en el territorio que sirven para la estimación por interpolación de la magnitud temática (altura, en este caso) de cualquier otro punto del espacio en el que no hay un dato de cota. Existen muchos modelos de interpolación de variables continuas a partir de elementos puntuales discretos, y en la plataforma descrita se utilizará el método *Inverse*

[1] El punto es, sin duda, la geometría más fácil de entender y más utilizada en campos no especializados en TIG, ya que gran cantidad de sensores se abstraen bajo esta geometría. Véase por ejemplo casos de uso con gran cantidad de datos sensorizados como el seguimiento de flotas o las Smart Cities (Ferreira et al., 2013).

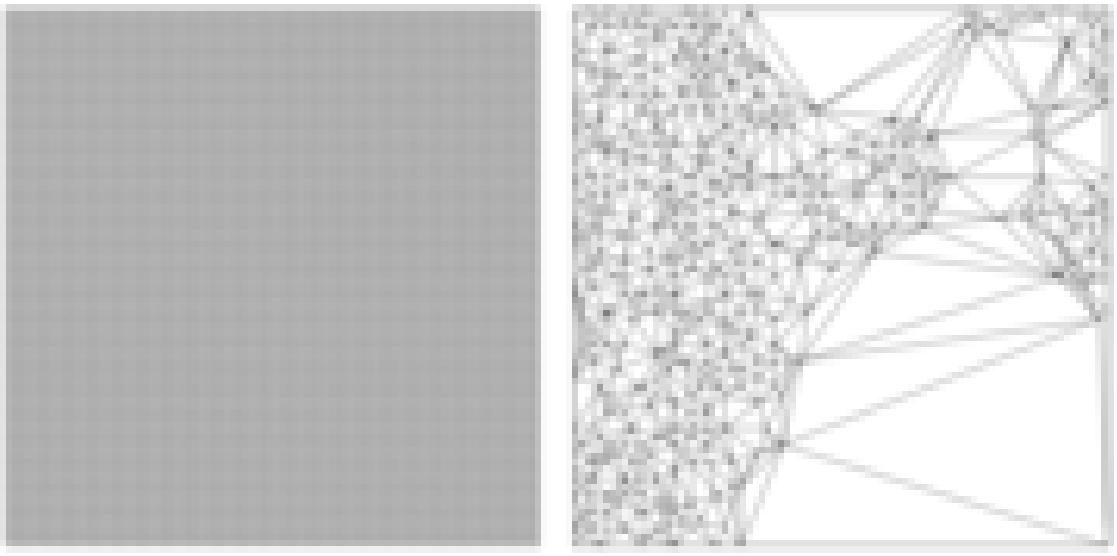

Figura II.2.1: A la izquierda, una malla regular de triángulos. A la derecha, una *Triangular Irregular Network*. Los triángulos son los más pequeños posibles, y el valor de un punto interior del triángulo es interpolable a partir de los tres vértices de dato conocido. Si el *TIN* representa un terreno, los triángulos determinan, por su vector normal, la orientación del terreno interno a él, lo que permite calcular, por ejemplo, valores de insolación y luminosidad. Fuente: `https://github.com/heremaps/tin-terrain/blob/master/docs/Mesh%20Generation%20Algorithms.md`.

Distance Weighting para obtener un campo interpolado de elevaciones bajo la resolución nativa de los datos de elevación originales.

Para este juego de datos se ha seleccionado el MDT proporcionado por el **Instituto de Estadística y Cartografía de Andalucía (IECA)**[2], que publica una malla regular con una resolución de 100 metros. El juego de datos se publica originalmente como un ráster en el producto **Datos Espaciales de Referencia de Andalucía (DERA)**[3], un conjunto de información geográfica básica para la CCAA, muy apropiado como marco genérico base para muchas aplicaciones.

El conjunto de datos original se presenta en formato *GeoTIFF*[4] monobanda, teniendo los datos de altura en números con decimales como valor de la banda ráster. El sistema de proyección utilizado es el ETRS89 UTM 30 Norte, y el tamaño del ráster es de 5317 por 3157 píxeles, para un total de 16,78 millones de píxeles. La resolución del pixel es de 100 por 100 metros, lo que da una densidad de datos de 100 píxeles por kilómetro cuadrado. La banda de datos tiene definido el valor ***NO DATA*** o dato nulo, ya que el formato *GeoTIFF* soporta dicha definición, por lo que la identificación de píxeles sin datos es trivial. El valor máximo encontrado en el ráster es de -149 metros, mientras que el mayor es de 3450.

Al tratarse de un conjunto de datos ráster, éstos pueden tratarse a nivel vectorial bien como polígonos (transformando las celdas del ráster en un campo poligonal) o bien como

[2]Instituto de Cartografía y Estadística de Andalucía (IECA), 2021c.
[3]Instituto de Cartografía y Estadística de Andalucía (IECA), 2021a.
[4]Instituto de Cartografía y Estadística de Andalucía (IECA), 2021b.

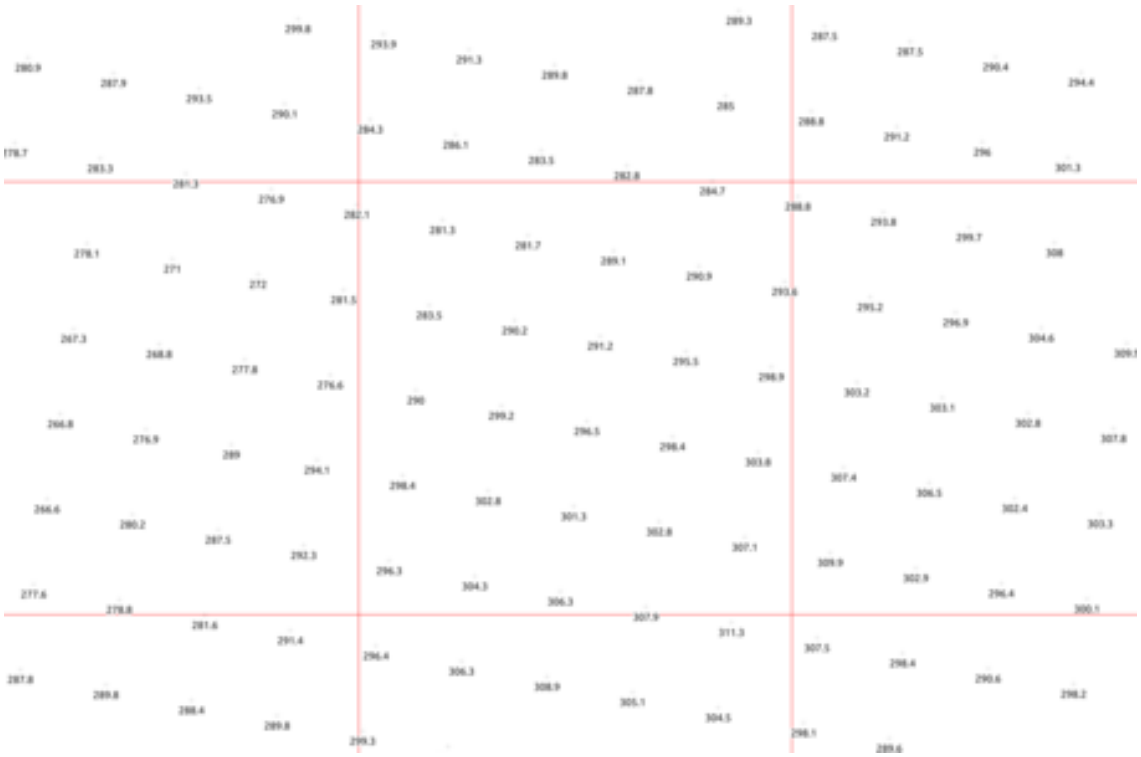

Figura II.2.2: El MDT del IECA, con una resolución de 100 metros, extraído como puntos del ráster original. Se puede observar, al proceder de una fuente ráster, que es un MDT en formato de rejilla. Las rejillas mostradas corresponden a una resolución de 500 metros. Fuente: elaboración propia.

puntos (tomando el centroide de la tesela ráster como anclaje geométrico). Dado que este juego de datos pretende ser utilizado para probar la implementación de un método de interpolación puntual como *IDW*, se ha incluido entre los datos puntuales al ser el ráster convertido a puntos según el segundo método citado (Olaya, 2016).

Este conjunto de datos satisface la condición de tener un recubrimiento completo del área de estudio. Su tamaño, sin ser desmesurado, también constituye una buena elección para poner a prueba la teselación por interpolación. Su complejidad geométrica no es significativa, al tratarse de puntos y al estar uniformemente espaciados, y su densidad de datos temáticos es muy baja, ya que sólo presenta un elemento de datos, la cota. La principal razón de su inclusión es la puesta a prueba de que la plataforma es capaz de generar datos continuos a menor resolución de la original por interpolación.

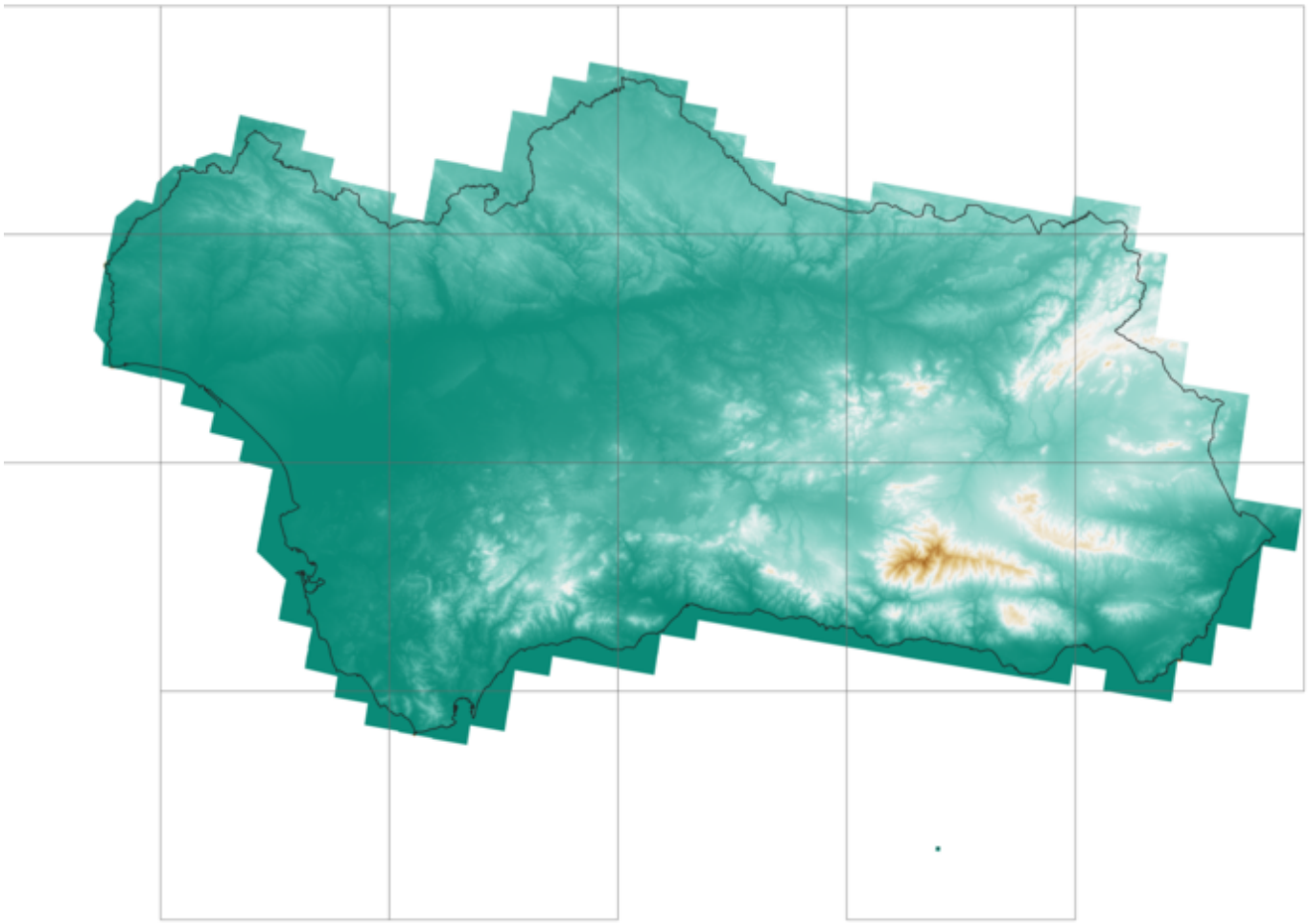

Figura II.2.3: Datos originales en formato ráster del MDT del IECA. Obsérvese las zonas sin datos donde el dato *NO DATA* está consignado a tal efecto. Las teselas superpuestas son del nivel de 100 km. Fuente: elaboración propia.

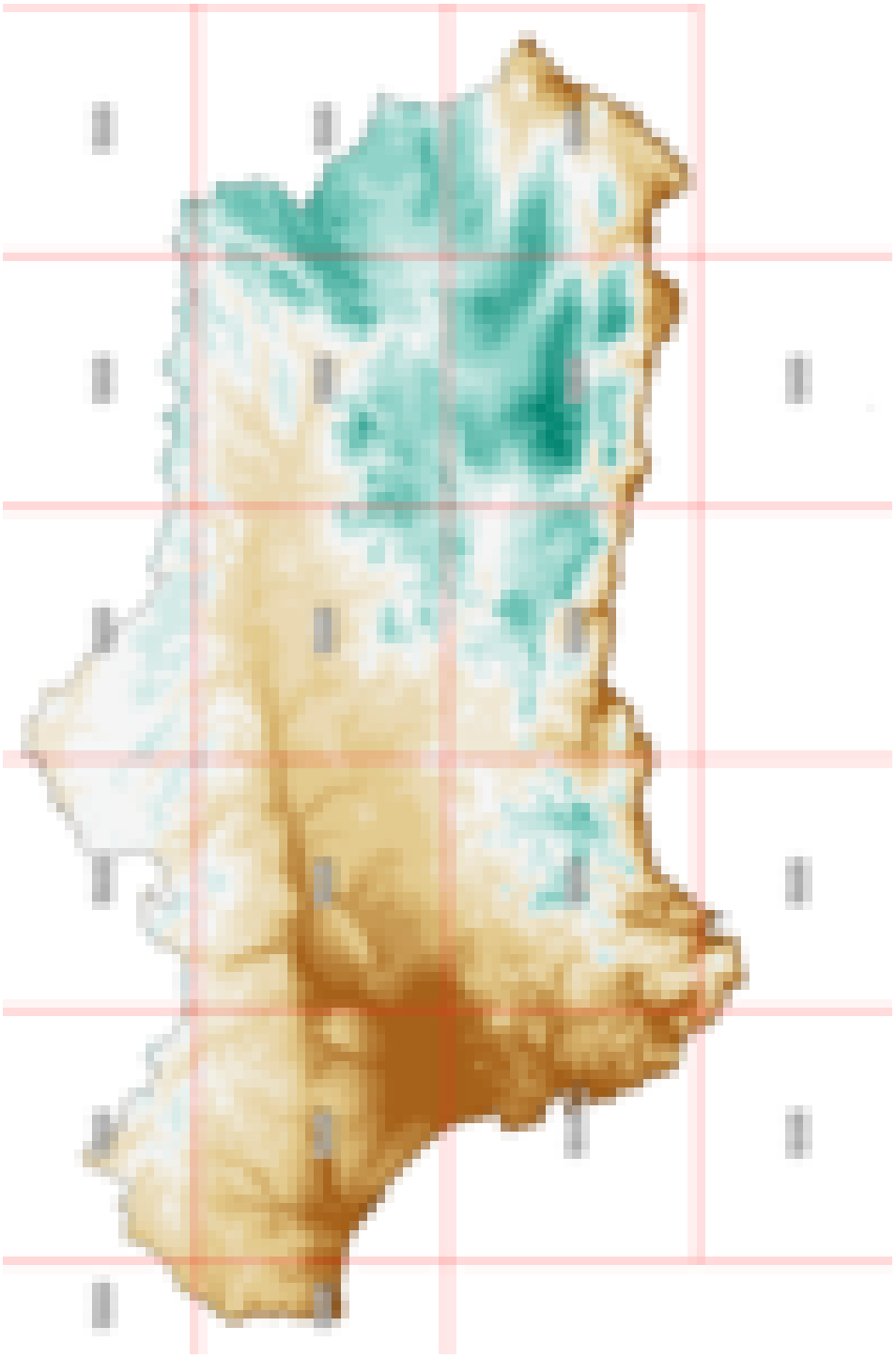

Figura II.2.4: Datos del MDT adscritos como puntos al centroide de las celdas del ráster. Fuente: elaboración propia.

Datos de población

Los datos de población incluidos en el conjunto de datos de prueba tienen una clara vocación temática, y es esta su principal razón de inclusión. Desde el punto de vista geométrico, los potenciales retos que esta capa de información puede ofrecer son mínimos: es una capa con una densidad de información baja, su tamaño es muy limitado y su complejidad geométrica, al tratarse de puntos, escasa. Sin embargo, tiene una extraordinaria importancia en su paquete de datos alfanuméricos, ya que incluye datos completos de muchas variables demográficas temáticas en distintos momentos temporales.

A diferencia de la información medioambiental, es de sentido común el hecho de que la información demográfica puede ser extraordinariamente sensible y que está sujeta a importantes condicionantes legales. La Ley de Protección de Datos (BOE, 2020) y de Secreto Estadístico de las Administraciones (BOE, 2006a), que constituyen los organismos encargados del levantamiento de detalle de esta información, imponen lógicas restricciones a su acceso y difusión. Por lo tanto, las administraciones, en este caso el IECA, han diseñado un ingenioso sistema de publicación de la información demográfica que cumple con todos los requisitos legales a la par que pone a disposición de público una valiosa información. Este sistema es en gran parte inspiración para los trabajos de esta tesis.

En origen, la información demográfica se geolocaliza en función de los datos de vivienda de cada uno de los ciudadanos. En los distintos registros administrativos, la dirección postal de la ciudadanía se recoge de forma textual. Gracias a los nomenclator y a los callejeros digitales[5], se pueden llevar a cabo operaciones de geocodificación postal, esto es, la asignación de una coordenada geográfica (el portal) a una dirección postal[6]. Sin embargo, la publicación a ese nivel de detalle de la información supondría un quebranto, dado su detalle y desagregación, de las normativas anteriormente citadas.

Tradicionalmente, la información demográfica se ha publicado agregada en municipios o en secciones censales (F. J. Goerlich et al., 2013) (ver figura II.2.5). Este nivel de desagregación es inconveniente para aplicaciones de detalle por varias razones:

1. **se referencia a geometrías no homologables:** los polígonos de cada municipio son distintos, por lo que hay que recurrir a indicadores derivados basados en la densidad por área;
2. **la definición espacial es demasiado gruesa para muchas aplicaciones:** puede ser apropiada a niveles provinciales, regionales o nacionales, pero constituyen un bloque monolítico de datos para estudios de detalle.

El INE, por ejemplo, publica la pirámide completa de población a nivel nacional en secciones censales (INE, 2021a). Dado que la sección censal es una unidad administrativa menor que el municipio, en ciertos escenarios proporciona una mejor resolución espacial de información[7].

[5] *Portal CartoCiudad* 2021, González Jiménez et al., 2012, Merchán et al., 2014, Instituto de Cartografía y Estadística de Andalucía (IECA), 2021d

[6] Los procesos de geocodificación no son triviales y suponen un considerable reto tanto técnico como de administración y levantamiento de datos.

[7] Una sección censal es una delimitación que debe ser siempre interna a un municipio y que debe agregar

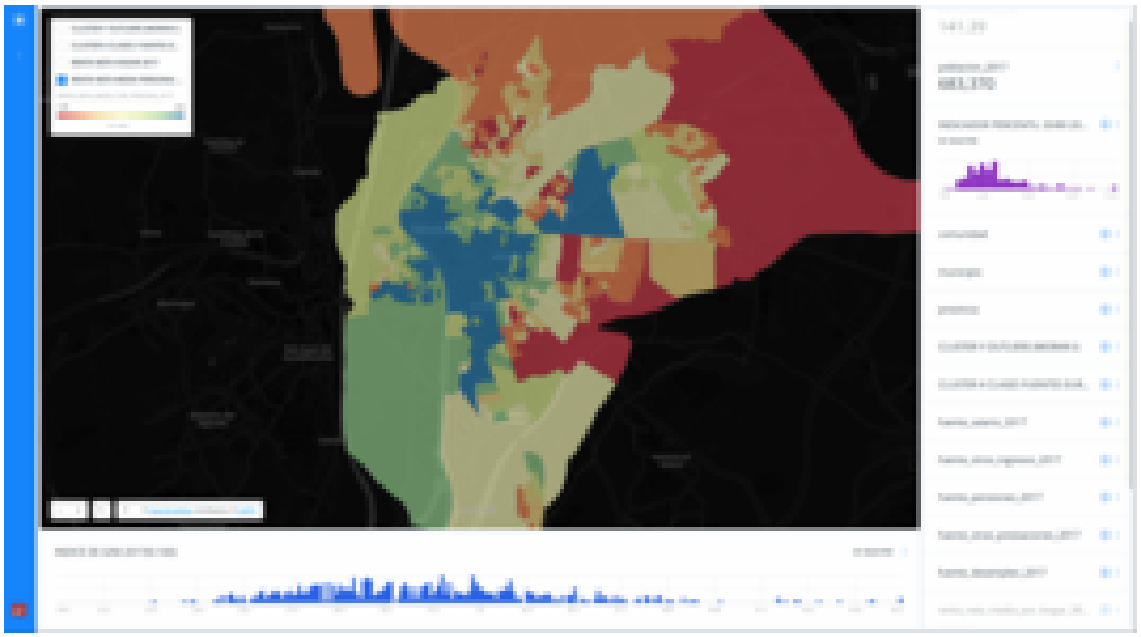

Figura II.2.5: Secciones censales de la ciudad de Sevilla y parte de la aglomeración urbana. Debido a la alta densidad de población, la sección censal se convierte en una interesante opción en estudios de estas zonas por dos razones: a) la densidad de 1000-2500 habitantes se alcanza en un espacio reducido y b) a este nivel el Instituto Nacional de Estadística (INE) publica datos demográficos completos. Fuente: visor desarrollado en el marco del proyecto de investigación *Plataforma Cloud para la integración espacial de geoinformación ambiental y socioeconómica (Spatial Big Data) y clientes web interactivos (geovisores, dashboard) para su difusión, acceso, explotación y análisis (Maching Learning). Proyecto singular de actuaciones de transferencia CEI RIS3. Junta de Andalucía. Andalucía Tech CEI-9.*

Para superar esta barrera de resolución, y siguiendo el procedimiento de EUROSTAT, la información demográfica se presenta agregada en celdas de 250 metros de resolución, con un recubrimiento completo del territorio andaluz (Enrique et al., 2013). La información original anclada en los portales se agrega sobre el área de la tesela que los contiene, presentando como dato el total de cada magnitud para la tesela. No obstante, aún así han de cumplirse muchos requisitos de secreto estadístico, como por ejemplo no mostrar la información si el número total de efectivos poblacionales está por debajo de un cierto límite.

En la dimensión geométrica, por tanto, la información se presenta como un conjunto de polígonos regulares de 250 por 250 metros generados a partir de la subdivisión de la rejilla oficial europea (Gallego, 2010) de un kilómetro cuadrado. Sólo se incluyen los polígonos de las teselas donde se encuentran datos, ya que existen celdas en los datos originales que tienen todos sus atributos a nulo a causa del secreto estadístico.

Esta capa de datos, al igual que el ráster, no es en origen una capa de puntos, sino poligonal. Sin embargo, por razones que ya se verán cuando se discutan los análisis de teselado (“Los análisis de teselado”, II.4.1, pág. 87), esta capa se ha incluido en los datos

entre 1000 y 2500 habitantes, por lo que en áreas densamente pobladas, como grandes ciudades, da una resolución espacial muy interesante para muchas aplicaciones de detalle.

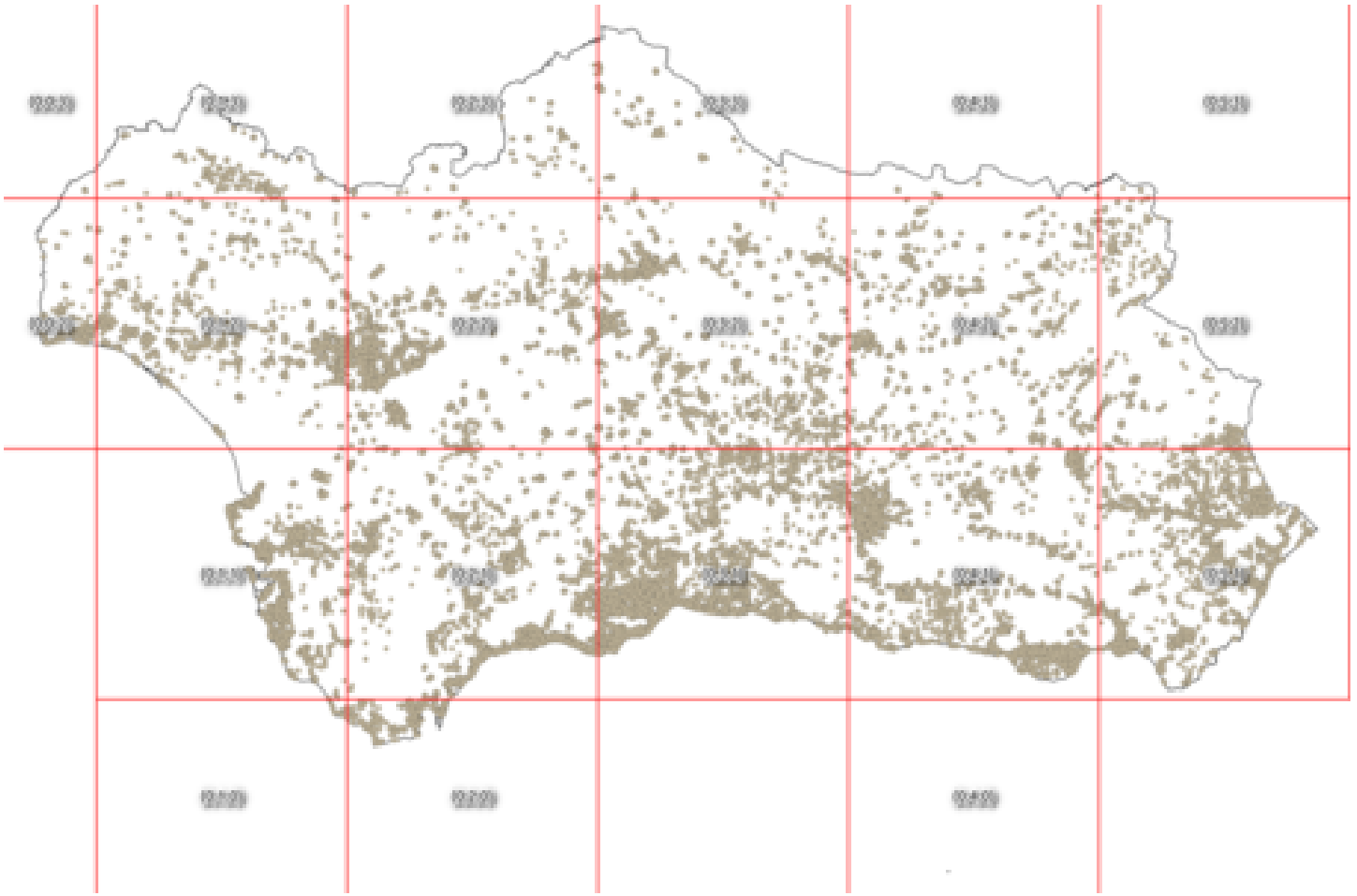

Figura II.2.6: Datos originales de los centroides de las celdas de 250 metros de población publicadas por el IECA. Las teselas mostradas son de 100 kilómetros de lado. Fuente: elaboración propia.

Figura II.2.7: Vista de detalle de los centroides de las celdas de 250 metros de población publicadas por el IECA. Las teselas mostradas son de 10 kilómetros de lado. Fuente: elaboración propia.

puntuales puesto que será el centroide de las celdas originales el que se usará como anclaje geométrico para la adscripción a la rejilla a implementar. Por tanto, sobre este apoyo geométrico puntual se encuentran las variables temáticas, que son las siguientes:

- **población total:** número total de efectivos poblacionales en la tesela;
- **población de mujeres:** número total de mujeres en la tesela;
- **población de hombres:** número total de hombres en la tesela;
- **población con edades entre 0 y 15 años:** número total de habitantes en este rango de edad;
- **población con edades entre 16 y 64 años:** número total de habitantes en este rango de edad;
- **población con edades superiores a los 64 años:** número total de habitantes en este rango de edad;
- **población de origen español:** número total de habitantes de origen español;
- **población de origen en el Espacio Europeo Schengen:** número total de habitantes con origen en el Espacio Europeo Schengen;
- **población de origen magrebí:** número total de habitantes con origen en el norte de África;
- **población de origen americano:** número total de habitantes con origen americano;
- **otros orígenes:** número total de habitantes de otros orígenes.

Estos datos se proporcionan para los años 2002, 2013, 2014, 2015, 2016, 2017 y 2018.

Los datos son numéricos, en este caso números enteros por tratarse de efectivos poblacionales. No todas las celdas tienen valores en todos los datos. Como ya se ha comentado, las implicaciones de secreto estadístico son importantes, por lo que muchas posiciones de datos están marcadas con un dato nulo, que en este caso se representa, de una forma unívoca, con un -1. Los datos originales proporcionados por el IECA se publican en formato *SHAPEFILE*.

Catastro

El Catastro de España constituye, sin género de dudas, uno de los bancos de datos geográficos más complejos, completos y útiles de cuantos administra una agencia gubernamental. La IDE del catastro es un sofisticado conjunto de servicios de difusión y análisis de información geográfica de gran valor para los sectores público y privado, y su catálogo de servicios es muy amplio. Su ubicación es la Oficina Virtual del Catastro de España (Dirección General del Catastro, 2021).

El catastro es posiblemente la fuente de datos seleccionada que más criterios cumple para los objetivos planteados en esta tesis:

Figura II.2.8: Vista de las parcelas de catastro a nivel autonómico. El juego de datos pesa varios *gigabytes*, pero lo que realmente pesa son las relaciones topológicas y los geoprocesamientos susceptibles de ser calculadas sobre él para su análisis. Fuente: elaboración propia.

Figura II.2.9: Vista de detalle de las parcelas de catastro. Obsérvese la diferencia en densidad de información entre el catastro rural y el urbano. Fuente: elaboración propia.

Figura II.2.10: Vista de las unidades constructivas de catastro del centro de Sevilla. A más intensidad de tinta, mayor volumen constructivo. Fuente: elaboración propia.

Figura II.2.11: Vista de detalle de unidades constructivas de catastro en el centro de Sevilla. A más intensidad de tinta, mayor volumen constructivo.
Fuente: elaboración propia.

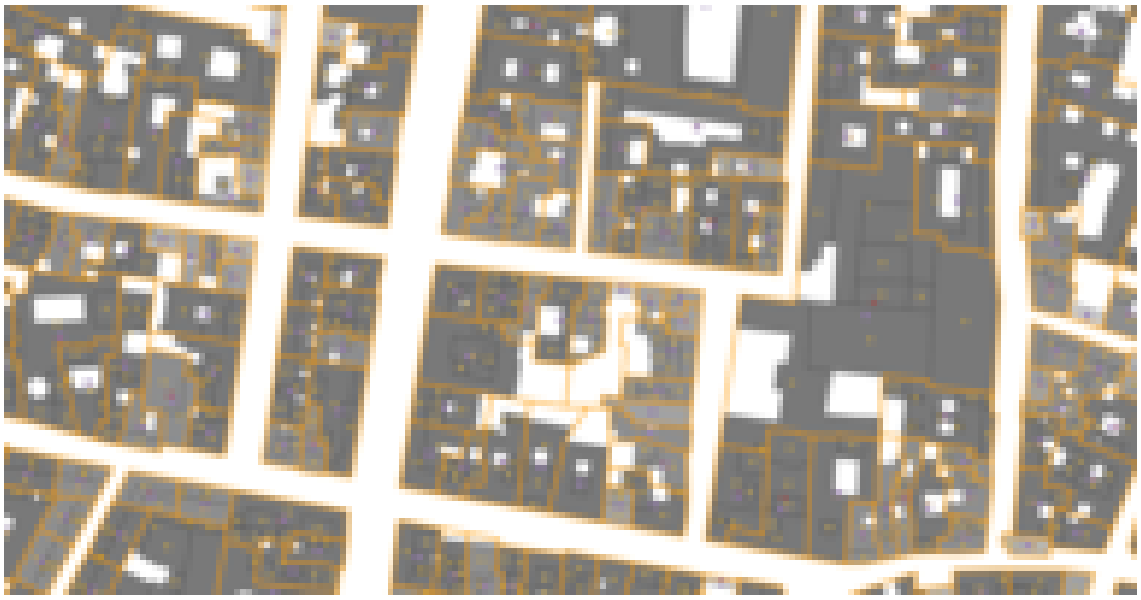

Figura II.2.12: Detalle del análisis de los centros de gravedad volumétricos de las unidades constructivas en catastro urbano. En naranja, la delimitación de la parcela, la unidad de análisis. Los pequeños puntos amarillos corresponden a los centroides de cada unidad constructiva, a los que se les ha calculado el área y el volumen. El punto grande morado corresponde al centro de gravedad volumétrico, mientras que los pocos círculos ocres que se observan son el centro de gravedad por área. En tonos grises, las unidades constructivas, con una tinta tanto más intensa como volumen tiene. Fuente: elaboración propia.

1. su extensión territorial es completa para todo el territorio autonómico, aunque administrativamente el catastro está dividido entre Catastro Rústico y Urbano;

2. el interés temático es innegable, puesto que de la explotación temática del mismo se extraen una gran cantidad de indicadores temáticos sobre vivienda, usos, etc.;

3. el tamaño del juego de datos es ingente (ver figuras II.2.8, II.2.9, II.2.10 y II.2.11);

4. la complejidad geométrica es alta;

5. tiene una alta densidad temática.

El catastro es descargable en base a los municipios. Desde la Oficina Virtual del Catastro se pueden descargar las capas y ficheros con toda la información catastral del municipio seleccionado. Esto hace que para conseguir un juego de datos en continuo para todo el territorio autonómico sea necesario descargar la información municipio a municipio y después componerlos en un juego de datos unificado.

La capa de catastro que se utiliza en este trabajo es el resultado de un proyecto realizado por el Grupo de Investigación del autor y la Consejería de Vivienda y Fomento de la Junta de Andalucía (Ojeda Zújar, 2015), destinado a extraer de los datos catastrales información de temática residencial, con objeto de identificar y categorizar el espacio residencial andaluz. El resultado final de dicho proyecto fue la generación de múltiples índices, de los que destacamos como de especial interés para este trabajo los siguientes:

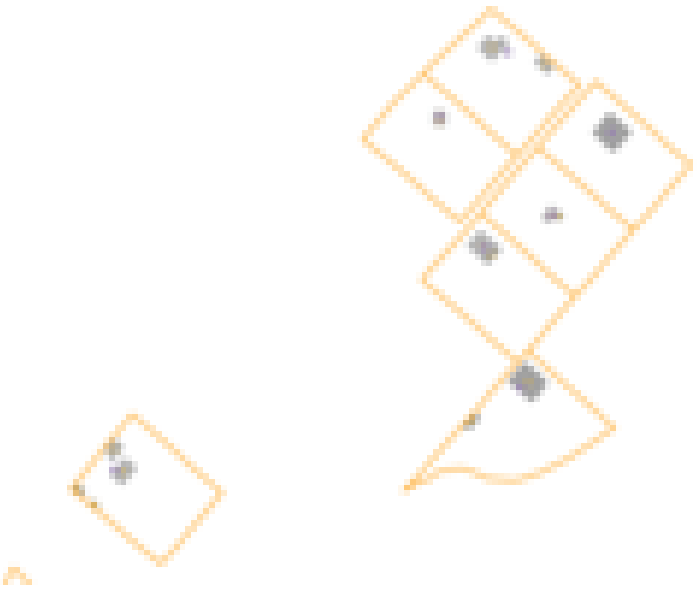

Figura II.2.13: Idéntico planteamiento al de la figura II.2.12 pero en catastro rural. Fuente: elaboración propia.

Figura II.2.14: Una comparativa del cálculo de los distintos centroides en parcela y unidades constructiva del catastro. Observesé como el hecho de adjudicar los indicadores calculados para parcelas y construcciones al centroide volumétrico tiene más sentido que hacerlo al más tradicional de la parcela, sobre todo en entornos rurales. Fuente: elaboración propia.

- total de superficie de fincas con construcciones;
- superficie total construida, siendo esta igual a la suma de las superficies de los elementos constructivos de la finca (incluidos porches y terrazas);
- total de superficie construida sobre rasante;
- total de superficie construida bajo rasante;
- total de bienes inmuebles;
- bien inmueble más antiguo;
- bien inmueble más moderno;
- total de bienes inmuebles con uso principal “Almacén - Estacionamiento”;
- total de bienes inmuebles con uso principal “Residencial”;
- total de bienes inmuebles con uso principal “Industrial”;
- total de bienes inmuebles con uso principal “Oficina”;
- total de bienes inmuebles con uso principal “Comercial”;
- total de bienes inmuebles con uso principal “Deportivo”;
- total de bienes inmuebles con uso principal “Espectáculos”;
- total de bienes inmuebles con uso principal “Ocio y Hostelería”;
- total de bienes inmuebles con uso principal “Sanidad y Beneficencia”;
- total de bienes inmuebles con uso principal “Cultura”;
- total de bienes inmuebles con uso principal “Religioso”;
- total de bienes inmuebles con uso principal “Obras de Urbanización y Jardinería, suelos sin edificar (solares)”;
- total de bienes inmuebles con uso principal “Edificio Singular”;
- total de bienes inmuebles con uso principal “Almacén Agrario”;
- total de bienes inmuebles con uso principal “Industrial Agrario”;
- total de bienes inmuebles con uso principal “Agrario”;
- superficie media de inmuebles;
- total de construcciones reformadas;
- media de usos diferentes por parcela;
- total de solares;

- total de direcciones postales;
- total de unidades constructivas de uso residencial;
- total de unidades constructivas;
- total de inmuebles con al menos una construcción con destino residencial.

El anclaje geométrico de estos índices es la unidad constructiva, no la parcela. Los datos proporcionados a nivel de parcela, pero que pertenecen a las construcciones que albergan, han sido asignados a las mismas por un procedimiento de asignación basado en el cálculo del centro de gravedad volumétrico de las unidades constructivas, bajo la presunción de que, en el ámbito rural, el volumen constructivo en parcelas con usos residenciales suele pertenecer a la vivienda (ver figuras II.2.12, II.2.13 y II.2.14).

II.2.2. Datos poligonales

La lógica indica que los datos poligonales serán los que más estresarán computacionalmente los procesos de adscripción. En el catálogo de las relaciones topológicas y los geoprocesamientos que se pueden dar entre los distintos tipos de geometrías, es evidente que los que afectan a los puntos, por su adimensionalidad, son los más rápidos de procesar (de Berg et al., 2000), ya que suelen ceñirse a la comprobación de distancias y/o inclusiones en polígonos. En cuanto al geoprocesamiento, un punto, de por sí, no es procesable, también debido a su carácter adimensional. Puede existir o no existir, pero nada más.

Sin embargo, los polígonos tienen un catálogo de geoprocesamientos posibles más extenso, y las modificaciones a las que pueden verse sometidos son computacionalmente complejas. Las intersecciones y las uniones de polígonos complejos, por ejemplo, se realizan con algoritmos que, a pesar de que están descritos ampliamente en la literatura y están muy probados[8], no son computacionalmente triviales, necesitando el ordenador un tiempo considerable en solventarlos.

Hábitats de Interés Comunitario

Los Hábitats de Interés Comunitario de Andalucía (HIC) es un juego de datos fruto de la aplicación en la Comunidad Autónoma de la Directiva del Consejo Europeo 92/43/CEE de 1992, "relativa a la conservación de los hábitats naturales y de la fauna y flora silvestres" (BOE, 2021), (REDIAM, 2021).

La inclusión de esta capa de información se debe a su temática ambiental, pero también a su extraordinaria complejidad geométrica. Es una capa poligonal con un alto detalle en la digitalización, a gran escala, sobre ortofoto. Esto hace que su escala significativa sea muy grande, así como su resolución espacial, produciendo polígonos muy detallados, con una alta densidad de vértices por unidad de área, y extraordinariamente fragmentados. Esta

[8]El estudio de los algoritmos de geoprocesamiento cae dentro de una especialidad de la matemática que es la geometría computacional. El manual de de Berg et al. que se puede encontrar en la bibliografía es un magnífico curso introductorio a la disciplina, ya que en él se repasan todos los algoritmos clásicos de la misma.

complejidad supondrá, como se verá más adelante, un reto en el procesamiento poligonal, ya que a polígonos más pequeños y complejos, más trabajo de geometría computacional. Su extensión es autonómica, pero su recubrimiento no es total.

La fuente de datos original es una *File Geodatabase* del sistema *ArcGIS*.

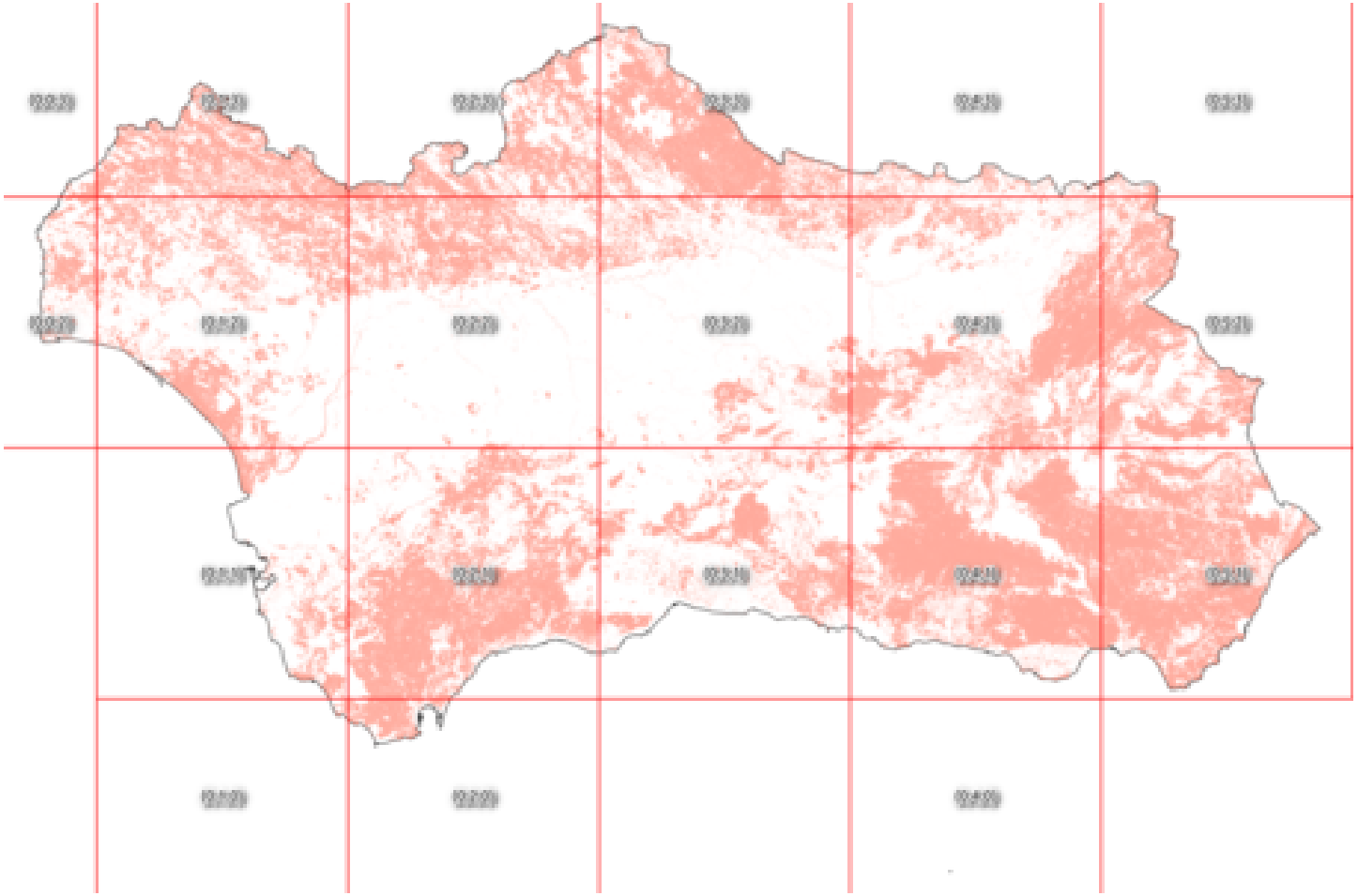

Figura II.2.15: Vista los datos originales de los Hábitats de Interés Comunitario. Las teselas mostradas son de 100 kilómetros de lado. Fuente: elaboración propia.

Figura II.2.16: Vista de detalle de los datos originales de los Hábitats de Interés Comunitario. Fuente: elaboración propia.

II.2.3. Importación y transformación de datos originales

Como cualquier sistema informático, la plataforma propuesta en esta tesis para la teselación de información geográfica habilita una *Application Programming Interface* (*API*). Una *API* (Ofoeda et al., 2019) es un conjunto de métodos, operaciones y estándares que permiten a un agente externo al programa (ya sea una persona que lo opera u otro programa informático que interopera con él) entender de qué manera y bajo qué circunstancias el programa funciona y admite y transmite información.

En ingeniería en general, abstraer de los detalles del funcionamiento de un subsistema complejo el cómo éste interacciona con el resto del sistema es una de las bases de su práctica. Los subsistemas trabajan en base a especificaciones operativas (por ejemplo, límites tolerados de presión, temperatura, etc.) y en base a cómo dicho subsistema se conecta con el resto del sistema (por ejemplo, especificaciones de tipos de válvulas, tuberías, etc. que conectan unos componentes con otros). Todos estos detalles están medidos y documentados cuidadosamente por el fabricante. Un mal uso, en contra de las especificaciones, puede dañar o hacer que el subsistema rinda por debajo de lo esperado.

En ingeniería de *software*, el proceso es análogo: las librerías (subsistemas de código informático) se diseñan con unas especificaciones. En la práctica, un programador que use una librería no tiene por qué saber ni estar interesado en cómo hace las cosas internamente. Lo único que le interesa es cómo conectarla al resto del sistema y qué especificicaciones de entrada de datos, qué operaciones puede realizar y qué estructura tienen los datos computados de salida. Eso es la *API*, y tienen que estar bien documentadas (Meng et al., 2018). Un programador se referirá a la documentación de la *API* de una librería (subsistema) para interaccionar con ella e integrarla en su programa. Para ejemplos de documentaciones de *API* de productos TIG en la nube, véase *Mapbox* (Mapbox, 2021b) y *CARTO (CARTO Contributors, 2021b).*

Un sistema informático, por tanto, por muy sofisticado que sea, no puede aceptar la información bajo cualquier formato o estructura. Un programa podrá aceptar una o varias formas o estructuras de información, pero éstas estarán siempre acotadas por la propia arquitectura interna del sistema y otras circunstancias de diseño. En el caso de la plataforma planteada en este trabajo podemos destacar:

1. **el conjunto de estándares existentes dentro del campo de aplicación del sistema:** a veces, sobre todo en sistemas muy novedosos, son los propios creadores del sistema los que acaban imponiendo al resto de los prácticantes de la misma especialidad sus estándares y estructuras de datos[9]. En ámbitos de aplicación maduros, sin embargo, lo normal es que ya exista una plétora de formatos, estructuras de datos y modelos de operación bien conocidos y aceptados por la comunidad de usuarios y/o investigadora. En el diseño de un nuevo sistema se debe hacer un análisis previo de dichos estándares para acogerse a los mismos siempre que se pueda;
2. **la accesibilidad y el formato de publicación de los datos con los que el**

[9]Por ejemplo, el *KML* es un estándar de facto popularizado por *Google*, mientras que el *SHAPEFILE*, que no destaca precisamente por su calidad ni robustez, ha sido durante décadas el formato más extendido de intercambio de información geográfica.

sistema va a operar: la interoperatividad del sistema, y por tanto su facilidad de uso y sus probabilidades de aceptación por parte de terceros dependerá en gran medida de que el sistema sea capaz de ingerir, con la menor transformación posible, los formatos de información que, aunque no estén regidos por estándares, sean de común uso dentro del campo de aplicación[10].

En este aspecto, la plataforma descrita en este trabajo se adapta a las herramientas existentes en el amplio ecosistema de plataformas, fabricantes y estándares presentes. La plataforma propuesta, por tanto, utiliza estándares de operación para llevar a cabo el crucial trabajo de la preparación e importación de datos. Los condicionantes de diseño a la hora de abordar estas tareas son:

1. **el uso nuclear en la arquitectura de la aplicación de la base de datos relacional de *Software Libre PostgreSQL*:** *PostgreSQL* (Obe et al., 2017) es una de las base de datos relacionales de *Software Libre* más potentes[11] y versátiles. Su elección es clara puesto que en el ámbito de aplicación de las TIG aporta la extensión ***PostGIS*** (Hsu et al., 2015), que la transforma en una base de datos relacional geográfica insustituible en este campo;

2. **el uso de formatos de intercambio de información geográfica:** formatos muy extendidos y utilizados en casi todos los sistemas información geográfica de escritorio de escritorio, como pueden ser el formato *SHAPEFILE* o el *KML*;

3. **la operativa de los trabajos de teselación:** como se describirá más adelante, la plataforma implementa una serie de tareas de teselación que se adaptan a diversas estructuras de información geográfica y que la teselan con un fin u otro. La forma en la que estas tareas de teselación entienden las estructuras de información en vital para el diseño de la *API* de datos la plataforma.

Aún así, el hecho de que la plataforma esté preparada para trabajar con *PostgreSQL* o con *SHAPEFILE* y *KML* aporta el marco de referencia en cuanto a formatos de información, pero no en cuento a estructura de los datos a aportar a la plataforma.

Ya en el capítulo introductorio se habló de la importancia de las estructuras de datos en todo sistema informático, de como éstos eran una parte del diseño del sistema muy importante y condicionante. Por lo tanto, los usuarios de la plataforma deben comprender cómo deben formatear su información y aportarla en los formatos descritos para que el sistema pueda procesarla.

II.2.4. Bases de las estructuras de datos para la plataforma

Fundamentalmente, e independientemente del formato original de los datos (*KML*, *GeoJSON*, *SHAPEFILE* o *PostgreSQL*), lo más importante para entender la estructura

[10] Por ejemplo, si un colectivo de usuarios está muy acostumbrado al uso de una herramienta externa como por ejemplo el *Excel*, muy común, el sistema debería aceptar datos en formato *CSV*.

[11] Si bien no la más utilizada o desplegada, ya que *MariaDB* (anteriormente llamada *MySQL*) es sin duda la más utilizada ya que está orientada sobre todo al desarrollo de aplicaciones *web*.

de datos a proporcionar al sistema viene condicionado por las características técnicas de las tareas de teselado que se vaya a seleccionar para su procesamiento. El usuario de la plataforma debe conocer y entender las peculiaridades de cada tarea de teselado[12] para poder configurarlas en base a la estructura de datos de la información original.

Por tanto, los condicionantes estructurales para cada tarea de teselado son:

1. **agregaciones de puntos:** para realizar agregaciones sobre nubes de puntos, la plataforma necesita que se le proporcione una capa de información de geometría puntual. En cuanto a la información alfanumérica, esta dependerá del tipo de operaciones estadísticas que se quieran hacer sobre ellas. Como mínimo, se ha de proporcionar un campo de información temática numérica continua[13] para definirle una operación estadística de agregación, como pueden ser conteo, suma, medidas de tendencia central, etc.;

2. **interpolación de nubes de puntos:** estas tareas de teselado son muy sencillas, ya que sólo precisan de datos geométricos puntuales y una única variable numérica continua para hacer su labor de interpolación. Su objetivo es interpolar una nube de puntos de una resolución determinada a un nivel de rejilla inferior a dicha resolución[14];

3. **cálculo de áreas poligonales cubiertas por categorías temáticas:** esta familia de tareas de teselación tienen como objetivo contar la proporción del área de cada tesela que cubre un conjunto de categorías temáticas adscritas a una cobertura poligonal. Son fáciles de configurar, ya que todo lo que precisan es una geometría poligonal y un campo con una variable discreta, que puede ser textual.

Teniendo esto en cuenta, queda al arbitrio del usuario proporcionar a la plataforma una tabla *PostgreSQL* en un servidor accesible a la misma[15], en el caso del prototipo desarrollado, aunque por supuesto es posible implementar la subida de datos a la plataforma en otros formatos gracias sobre todo al concurso de la librería *GDAL / OGR*.

Un usuario con un perfil clásico en Sistemas de Información Geográfica tiene muchas opciones para llevar a cabo esta tarea de transformación de datos con herramientas estándar. Los SIG de escritorio más usuales (*QGIS*, *ArcGIS*), siempre que no se expongan

[12] Conocer, en definitiva, su *API*.

[13] Hay que destacar la capacidad de esta tarea de teselado de realizar a la vez muchas operaciones de agregación estadística de puntos, por lo que cuantas más se configuren sobre una única capa puntual más eficiente será el teselado. Merece mucho la pena, por tanto, trabajar externamente a la plataforma primero la información (con *Excel*, una base de datos o un SIG de escritorio como *QGIS* (Graser, 2013) o *ArcGIS* (Law et al., 2019)) para unir a una única nube de puntos todos los datos estadísticos deseados y después configurar una única tarea de teselado de este tipo que varias de ellas.

[14] Por ejemplo, interpolar un campo de temperaturas a 250 metros de resolución entre puntos a partir de medidas originales espaciadas un kilómetro.

[15] Generalmente, es un grave problema de seguridad dejar abierta de forma pública una base de datos con información sensible, y la enorme mayoría de las aplicaciones distribuidas no lo hacen como primera medida básica contra el robo de información. En ámbitos muy controlados, como por ejemplo la red interna de un organismo, los distintos servidores sí pueden accederse entre ellos, pero no desde el exterior, por lo que a priori no sería un riesgo. Esta opción de suministro de información a la plataforma, por tanto, es la más restrictiva, así como la más técnica, pero la más accesible al desarrollo de un prototipo.

a conjuntos de datos demasiado masivos (del orden de varios gigas de tamaño), aportan herramientas fáciles e intuitivas para la transformación de información geográfica. Es muy interesante, por ejemplo, la capacidad de hacer uniones[16] de fuentes de datos geográficas con datos alfanuméricos extraidos de formatos *CSV* o *Excel*, en función de un campo común de información compartido por ambos juegos de datos, que permite de esta manera enriquecer los datos originales de la tabla con nuevos atributos (Olaya, 2016).

II.2.5. Subida de la información a la plataforma

Una vez el usuario ha modelado y estructurado la información original que pretende teselar en función a las especificaciones descritas por la tarea de teselado que pretende aplicarle, la información sería subida a la plataforma mediante una *API* de servicios[17]. Una vez la información ha llegado al servidor, esta es procesada de distintas formas:

1. primero, ha de ser importada de forma temporal en la *PostGIS* de trabajo interna de la plataforma. Los trabajadores de teselado sólo pueden leer la información desde una *PostGIS* u otra base de datos multiusuario, ya que es un proceso multiconcurrente[18] realizado por muchos de ellos a la vez, algo que tiene sus complicaciones en el caso de la lectura de ficheros. La importación del formato de origen dentro de *PostGIS* lo realiza la librería *GDAL*[19] (GDAL/OGR Contributors, 2021);

2. como paso posterior a la importación a la *PostGIS* de trabajo, ésta debe reproyectar el juego de datos al sistema de proyección nativo de la rejilla que se va a utilizar para teselar. Aunque *PostGIS* es capaz de hacer reproyecciones *al vuelo*, es decir, en tiempo real, siempre que sea necesario, en el caso de información compleja (polígonos, por ejemplo) es un paso computacional que ralentiza el proceso de teselado, por lo que es mejor tenerlo todo bajo el mismo sistema de proyección de antemano, algo que acelera mucho el proceso[20];

[16] Conocidos comunmente como *JOINS*.

[17] Este paso no es necesario en el caso de tener la información en *PostgreSQL*, ya que es un formato que la plataforma lee de forma nativa.

[18] Un proceso *multiconcurrente* es aquel que puede ser realizado en orden y con garantías de fiabilidad por varios clientes a la vez. Nótese que el uso de la palabra *cliente* en lugar de la palabra *usuario* es necesaria porque por *cliente* se entiende tanto a personas que hacen uso de un programa en un contexto persona-máquina como a programas que hacen uso del mismo en un contexto máquina-máquina.

[19] La librería *GDAL* es la principal librería de importación / exportación de *Software Libre* TIG y es utilizada en multitud de sistemas, tanto comerciales como de *Software Libre*. Posee más de 70 *drivers* para el trabajo con información vectorial y más de 150 para trabajo con fuentes ráster. Todos los formatos de uso común y estándar abierto son soportados por esta librería: *GeoJSON*, *KML*, *GeoPackage*, *SHAPEFILE*, etc. La lectura y escritura sobre *PostGIS* está también excelentemente bien soportada, por lo que, con la inclusión de esta librería, la plataforma tiene el potencial de utilizar como datos de origen información en una gran variedad de formatos.

[20] En el caso de suministrar la información directamente en una base de datos *PostGIS*, dado que no sería apropiado que la plataforma tuviera la capacidad de alterar la información original, es el usuario el responsable de proporcionar la información ya transformada. *PostgreSQL* posee el flexible método de creación de *MATERIALIZED VIEWS* para hacerlo de una forma poco intrusiva para los datos originales. La propia *PostGIS* realiza chequeos de sistema de proyección a la hora de geoprocesar, fallando los geoprocesos si se intentan ejecutar sobre sistemas distintos.

3. con estos dos pasos, se registra la fuente de datos original y queda disponible para definir sobre ella tareas de teselación.

II.3 Definición de la rejilla

En este capítulo se describe las características técnicas y la metodología de creación de la rejilla que va a imponer el marco discreto de adscripción de la información de las fuentes de datos originales.

II.3.1. Características generales de la rejilla

La rejilla propuesta en esta metodología impone el marco de referencia espacial geométrico sobre el que se va a producir la adscripción de la información de las fuentes originales de datos.

A grandes rasgos, la rejilla se parece a un ráster, es decir, induce una discretización regular sobre el ámbito de estudio, definiendo esta discretización el aspecto geométrico de la estructura de datos geográficos, mientras que el aspecto alfanumérico viene determinado por una adjudicación a cada tesela de un número de variables temáticas. En este sentido, la estructura de datos se asemeja a un ráster multibanda, es decir, un ráster con varias capas de información que comparten adscripción geométrica[1].

La rejilla propuesta, sin embargo, se diferencia de un ráster convencional en varios aspectos:

1. la estructura de datos es **asimétrica** tanto en el aspecto geométrico como alfanumérico de la estructura de datos;

2. es **multiescalar**, ya que la rejilla puede definir varios niveles de resolución distintos.

La rejilla a utilizar no es fija, sino es que definible por el usuario de la plataforma. Básicamente, en la definición de la rejilla se necesita aportar las características descritas a continuación. Es muy importante pensar bien de antemano estas cualidades geométricas de la misma ya que condicionarán mucho el uso posterior del juego de datos adscrito. Por ejemplo, es importante seleccionar niveles de escala que sean compatibles con rejillas oficiales ya existentes[2]:

Las características que definen la rejilla son, por orden de dependencia:

[1] Estos cubos multidimensionales, donde las dos primeras dimensiones son X e Y, se utilizan mucho por ejemplo en teledetección, siendo el resto de dimensiones las bandas radiométricas del sensor.

[2] Recordemos que, por ejemplo, EUROSTAT publica la información europea en una rejilla de 1 kilómetro, mientras que el Instituto de Cartografía y Estadística de Andalucía la reduce a 250 metros para muchas aplicaciones. Estos dos niveles de zoom, por lo tanto, son especialmente útiles.

1. su **sistema de proyección cartográfica**, determinado por su código *EPSG* (Nicolai et al., 2008): a la hora de seleccionar la proyección cartográfica hay que tomar en consideración todo lo que normalmente ha de tenerse en cuenta para seleccionar este importante parámetro cartográfico, como con cualquier otra aplicación. Hay que elegir sistemas oficiales bien soportados en la cartografía oficial y cuyas características sean apropiadas a la aplicación prevista[3];

2. su **origen de coordenadas**: el origen de coordenadas de la rejilla ha de venir expresado en las coordenadas del sistema de proyección seleccionado anteriormente. Este origen de coordenadas marca la esquina inferior izquierda de la primera tesela en todos los niveles de zoom. Dicha primera tesela tendrá las coordenadas de rejilla (0,0), como se explicará posteriormente. Ver figura II.3.1;

3. los **niveles de resolución**: el usuario puede definir todos los niveles de resolución, en la unidad del sistema de proyección elegido, que crea convenientes en la rejilla, pero con ciertas limitaciones operativas:

    1. han de ser definidos de mayor a menor: por ejemplo, 100 kilómetros / 50 kilómetros / 10 kilómetros, siendo cada una mayor que todas las siguientes;
    2. cada nivel de resolución debe contener completamente en unidades enteras al siguiente: por ejemplo, dividiendo entre 2 la resolución. De esta manera, una resolución de 100 kilómetros puede ser dividida en un nivel inmediatamente inferior en uno de 50, 25, 10, 5, etc.;
    3. cada nivel de resolución debe arrojar un número entero al ser dividido por el inmediatamente inferior: no es valido crear niveles de resolución en el que las divisiones arrojen números no enteros; por ejemplo, dividir una resolución de 100 kilómetros en 3 daría una resolución para el siguiente nivel de 33,3 periódico, algo muy inconveniente para los cálculos.

II.3.2. La rejilla asimétrica

Una de las características principales de la metodología propuesta, y que la diferencia de otras estructuras de datos a priori similares, como el ráster, es su asimetría.

En un ráster, que es una estructura de datos también teselar, la malla discretizadora del espacio geográfico es fija en geometría y escala, y además todas las celdas deben existir en el continuo del espacio, no obviando ninguna. De esta manera, si se hace un ráster a partir de un fenómeno espacial que no recubra por completo el área de estudio (por ejemplo, las extensiones de un uso del suelo en concreto), la estructura de datos define y guarda memoria computacional para todas las unidades discretas impuestas por la malla geométrica, tengan estas unidades presencia del fenómeno a digitalizar o no. Es por ello que la presencia y consignación de la categoría NO DATO[4] en un ráster sea tan

[3]Recordemos que las características principales de una proyección son *equidistancia* (se conservan las distancias), *equivalencia* (se conservan las superficies) y *conformidad* (se conservan los ángulos).

[4]Afortunadamente, los más modernos formatos de ráster de uso común, como el *GeoTIFF*, ya incorporan la noción de *NODATA* (sin dato) en su estructura. Previamente a la generalización de esta

Figura II.3.1: Origen de coordenadas de una definición de rejilla, concretamente la usada en las pruebas de la plataforma. Su origen de coordenadas está designado por el punto en la esquina inferior izquierda, y sus ejes de coordenadas son las dos líneas a 90 grados que parten de ella. Como referencia, para demostrar su compatibilidad, se muestra la rejilla oficial europea de 1 kilómetro de resolución en sistema Lamberth Azimutal Equiárea (*EPSG* 3035). Fuente: elaboración propia.

importante, ya que nos indica celdas vacías. Pero la tesela, aun así, existe en la estructura de datos.

En la metodología propuesta esto no es necesario. Si una tesela no existe, no se consigna en la estructura de datos. Por lo tanto, se produce una asimetría geométrica, ya que nos encontraremos sólo con aquellas unidades geométricas que tienen un dato real.

En cuanto a la vertiente alfanumérica de la información geográfica, también se cumple este principio de asimetría. En un conjunto de datos ráster o vectorial, una vez se definen las propiedades temáticas de un conjunto de datos geográficos, éstas son fijas para todos los elementos del juego de datos. La tabla de atributos temáticos, dentro de un mismo conjunto, no es asimétrica para cada elemento: si un elemento en concreto no tiene dato en una de las características del conjunto, éste debe marcarse como *NODATA*, pero aún así ocupa espacio en la estructura de datos (y, por tanto, en la memoria de los ordenadores, porque al *NODATA* también hay que dejarle su sitio).

La metodología propuesta soporta conjuntos de datos asimétricos en el aspecto temático. Dentro de las variables temáticas generadas en la plataforma, una tesela:

1. **puede no consignar datos para ninguna de ellas:** la tesela, por consiguiente, ni siquiera existe a nivel geométrico (asimetría geométrica);

2. **puede consignar datos para un conjunto arbitrario de ellas:** la tesela puede guardar los datos que realmente le sean propios, sin necesidad de rellenar con *NODATA* aquellas categorías de datos para las que no tenga una "colisión"[5] espacial o geométrica. De esta forma, en zonas con alta densidad de datos (por ejemplo, el centro de una gran ciudad), las teselas tendrán muchísimos datos asociados de determinadas categorías (demografía, urbanísticas), mientras que en zonas de baja densidad (zonas silvestres, por ejemplo), el conjunto de datos puede ser completamente distinto (usos del suelo, protección de espacios naturales, especies presentes, etc.).

De esta forma, la estructura de datos propuesta pretende garantizar el objectivo de crear una estructura compacta que, deseablemente, pueda, en escenarios de cobertura parcial, ser más eficiente en el almacenaje de información que el mismo conjunto de rásters.

---

funcionalidad en los formatos de datos más recientes, los productores de información ráster tenían que recurrir a soluciones que, en casos de inexistencia de documentación, podían dar lugar a equívocos, como por ejemplo consignar la ausencia de datos con un valor fuera del rango natural de la variable digitalizada, como por ejemplo un número negativo o similar. Es muy importante entender que, en ciencia de datos, el "dato" *NODATA* (llamado *NULL*, *UNDEFINED*, etc., depende del sistema) tiene que tener entidad de dato como cualquier otro dato conocido, y los sistemas y sus usuarios tienen que estar preparados para tratar y tener en cuenta en sus procedimientos analíticos la presencia de esta importantísima categoría de dato.

[5] "Colisión" en el sentido espacial del término, es decir, que una geometría del conjunto de datos original a teselar intersecte la geometría de la tesela y, por tanto, su influencia en la misma sea consignada como dato dentro de la categoría que le es propia.

II.3.3. Definición matemática de la rejilla

La asimetría a nivel geométrico es posible, al contrario que en el ráster, porque la rejilla no viene condicionada ni definida por el soporte físico de la estructura de datos en sí, sino que está definida matemáticamente y las celdas se recrean en tiempo real. En un ráster convencional, la topología de la vertiente geométrica de la información geográfica viene determinada intrínsecamente por la estructura de datos en memoria del propio conjunto de datos: cada dato ocupa una tesela de memoria y, por hacer una analogía gráfica, siempre existen sus "vecinos" en las 8 posiciones que rodean a la tesela[6]. Deben de existir, aunque sea para ponerles un *NODATA*. Esas posiciones topológicas no sólo son expresadas de forma geométrica en una imagen en la pantalla o con un pixel de color, sino que tienen que existir físicamente en la memoria del ordenador para que la estructura de datos sea coherente y consistente y funcione correctamente. Un ráster no puede tener filas o columnas de dimensiones irregulares, a las que les falten posiciones de datos.

Este no es el caso de la rejilla asimétrica propuesta. En lugar de generar de forma apriorística las geometrías de todas las posibles teselas que recubren el área de estudio, indistintamente de si estas van a contener o no un dato, sólo se consigna en la estructura de datos su coordenada en relación a un nivel de resolución y la posición con respecto al origen de coordenadas decididos en la definición de la rejilla. La plataforma puede recrear en tiempo real la geometría de una tesela cuando esta sea requerida, con un estrés computacional mínimo.

De esta manera, cuando se adscribe, como se comentará en capítulos posteriores, un dato geométrico a la rejilla, la plataforma recrea en memoria la geometría de la tesela objetivo, comprueba las colisiones topológicas que puedan existir con la información a teselar, según la lógica interna de la tarea de teselado seleccionada, y calcula el valor de la variable objetivo. Si no hay colisiones topológicas, esa tesela con coordenadas (X,Y) se olvida, desaparece de la memoria y no se le consigna ninguna nueva información alfanumérica. En caso de que si que exista colisión, se toman las coordenadas de la tesela y se graba su nueva información alfanumérica. La geometría se desecha[7].

Las teselas generadas se identifican, como ya se ha adelantado parcialmente, gracias a tres coordenadas:

```
[ nivel de resolución, X respecto al origen, Y respecto al origen ]
```

Llamaremos a estas coordenadas a partir de aquí [R,X,Y].

II.3.4. Topología e indexación de la rejilla

La limitaciones acerca de la elección de los niveles de resolución de la rejilla obedecen también a razones de autoindexación de la información en la rejilla. Por otro lado, al igual

[6] Salvo, claro está, en los límites de la matriz bidimensional que constituye el ráster.

[7] En la versión de desarrollo se conserva, por motivos de depuración de resultados y de visualización fácil y rápida en *QGIS*, como tipo *geometry* en *PostGIS*.

que sucede en el ráster, las relaciones topológicas entre teselas vienen determinadas por las coordenadas [R,X,Y] en un mismo nivel de resolución, siendo trivial encontrar relaciones espaciales de distancia y contigüidad entre teselas. Sin embargo, la rejilla propuesta no sólo establece relaciones topológicas en las dimensiones [X,Y] dentro de un nivel de resolución (relaciones topológica intra-resolución), sino también en la dimensión R, es decir, relaciones topológicas inter-resolución, poniendo en relación de contención / contenido a las teselas de distintos niveles de resolución.

La indexación de los datos es de suma importancia en toda estructura de datos para mejorar su eficiencia. La ventaja que ofrece la estructura piramidal que establecen los niveles de resolución entre sí, con los distintos niveles contenidos de forma jerárquica, proporciona una base sólida para aprovecharla como indexadora de los datos.

Por ejemplo, la plataforma puede, con cálculos computacionalmente ligeros, calcular para una tesela:

1. las coordenadas [R,X,Y] de la tesela contenedora en cualquier nivel de mayor tamaño de la estructura de resoluciones;
2. las coordenadas [R,X,Y] de todas las teselas contenidas en ella a cualquier nivel de menor tamaño de resolución;
3. las coordenadas [R,X,Y] de las teselas situadas a menos de distancia N;
4. las coordenadas de (X,Y), no de tesela, sino de coordenadas geográficas (recordemos, en unidades del sistema de proyección) de las esquinas y el centro de una tesela;
5. los vecinos de una tesela [R,X,Y].

Estas relaciones topológicas, especialmente las inter-resolución, son de gran ayuda para acelerar y optimizar ciertas operaciones de teselado que las pueden aprovechar. Por ejemplo, pongamos el caso de la operación de teselado que calcula, para una capa poligonal, el porcentaje de recubrimiento de una categoría temática:

1. obviamente, si para un determinado nivel de resolución no hay colisión topológica, tampoco lo habrá para los niveles inferiores. Se ahorra comprobaciones inútiles de colisiones topológicas, que son computacionalmente muy estresantes para la máquina;
2. si hay un solape parcial del polígono contra la tesela, se consigna la variable y se baja a la siguiente resolución para comprobar posibles colisiones con las teselas de dicho nivel;
3. si la cobertura es total, todas las teselas hijas de los niveles de resolución inferiores también tienen una cobertura total. Se les asigna, en cascada, el 100 % de recubrimiento a todas sin necesidad de procesar las colisiones topológicas.

Estas características autoindexativas de la estructura de datos, fruto de la topología inter-resolución, se aprovechan al máximo en todas las tareas de teselado en las que tiene sentido considerarlas.

Existen, sin embargo, casos en los que no es posible sacarles partido. Por ejemplo, en el caso de las interpolaciones de puntos mediante *IDW* a resoluciones por debajo de la resolución natural del dato de origen. En este caso, lo que se persigue es dotar a la tesela, representada por su centro para el cálculo de las distancias, de un valor interpolado en función de los puntos vecinos que están dentro de una cierta distancia objetivo X, ya que no es representativo interpolar cuando los vecinos más próximos están demasiado lejos[8]. En estos casos:

1. que una tesela a una resolución R0 tenga puntos cercanos a su centro no garantiza que todas sus teselas hijas en la resolución R1 los tengan;

2. y el caso contrario: si una tesela R1 tiene puntos cercanos no significa que una tesela madre R0 los tenga.

En un caso como este, hay que solucionar el algoritmo de adscripción para cada tesela independientemente.

[8]Todo en función de la resolución natural de la nube de puntos base de la interpolación.

II.4 Teselado

En este capítulo se discute el proceso de teselado, que consta de varios pasos. Primero, se discutirá el concepto de *Análisis de teselado*, pues es necesario tener claro este concepto antes de abordar los siguientes contenidos. Posteriormente, se discutirá la adecuación de los datos originales a estos análisis de teselado comentados. Finalmente, se examinará cómo configurar el motor de teselado para que realice el trabajo propiamente dicho.

II.4.1. Los análisis de teselado

Es importante comenzar estableciendo una serie de nociones o conceptos que van a dictar la implementación de estos análisis de teselado en la plataforma. Un *análisis de teselado*, también llamado *análisis de adscripción*, es un proceso formado por una serie de pasos bien definidos, que se detallarán a continuación, y cuyo objetivo es la generación de nueva información de adscripción sobre las teselas determinadas por la definición de una rejilla, tal como se vió en el capítulo anterior. Las unidades en las que se descompone esta nueva información adscrita a la tesela se denominan *variables de adscripción*, y corresponden a un aspecto temático calculado para la tesela en su conjunto.

Es importante el matiz de que un análisis de teselado actúa sobre *una única tesela*[1]: se ejecutarán en la plataforma tantos análisis de teselado como teselas se quieran calcular a partir de la información original.

El objetivo final de un análisis de teselado será la generación de nueva información alfanumérica en una tesela. Dicha tesela puede contener ya información generada por otros análisis de teselado o no tener ninguna. Recordemos que la información geométrica de las teselas está generada en tiempo real, lo que implica que si una tesela no tiene información alfanumérica ni siquiera está consignada en el sistema. En cualquier caso, el hecho de que la tesela contenga información previa o no es indiferente[2].

Previo a la descripción de los pasos que realiza un análisis de teselado hay que considerar qué forma y estructura van a tomar tanto los datos a ser suministrados al análisis

[1] Se podría, aunque no está contemplado actualmente en los planteamientos de este trabajo, crear análisis de teselado que tengan en cuenta relaciones topológicas con teselas que ya tengan ciertos atributos temáticos. Por ejemplo, en el caso de querer hacer cálculos de accesibilidad sobre la rejilla, esto sería necesario. También se podrían plantear análisis de teselado que hicieran cálculos con topología en función de atributos temáticos ya existentes, generados por otros análisis de teselado.

[2] Véase nota anterior. De la misma forma, un análisis de teselado podría verse condicionado por la información ya contenida en la tesela, generada por análisis de teselado previos.

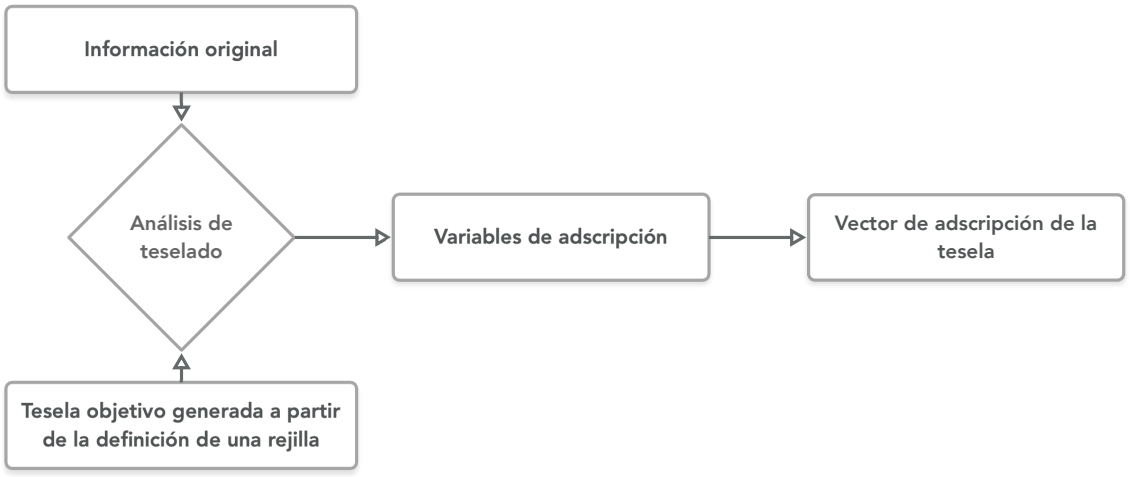

Figura II.4.1: El análisis de teselado y su relación con otros elementos de la plataforma. Al análisis de teselado se le proporciona una fuente de datos original adecuada a sus especificaciones, tanto en la vertiente geométrica como alfanumérica. Se le suministra también una tesela objetivo que está generada a partir de una coordenadas sobre la definición de la rejilla. El resultado del análisis son un conjunto de variables de adscripción, que acaban siendo introducidas en el vector de adscripción de la tesela.

como la que tendrá finalmente la información adscrita que va a ser consignada en la tesela. En este punto, introducimos el concepto *variable de adscripción*. Una *variable de adscripción* es uno de los resultados que generará un análisis de teselado y que será añadido a la información alfanumérica de la tesela procesada. De esta manera, un análisis de teselado puede generar, al final de su ejecución, una o varias variables de adscripción sobre la tesela objeto del análisis, añadiéndolas a su vector de adscripción.

El número de variables generadas por un análisis es variable y depende de sus objetivos de diseño y detalles de implementación, aparte, por supuesto, de depender mucho del tipo de dato original a adscribir. Por ejemplo, no es igual el tipo de variable de adscripción que generará el proceso de análisis de una variable continua que el de una variable discreta y categórica.

Así pues, es muy importante definir, en el diseño de objetivos de cada análisis de teselado, qué variables y con qué características va a generar.

Selección del análisis de teselado

A la hora de seleccionar un análisis de teselado para aplicarlo a unos de datos de origen se deben tener en cuenta las siguientes consideraciones:

1. el tipo de geometría original de la capa;
2. el tipo de datos temáticos, especialmente si son discretos o continuos;
3. los detalles de cálculo que ofrece cada tipo de análisis implementado en el sistema, que condicionan el tipo de variable de adscripción a obtener tras el procesamiento de los datos originales.

Variables de adscripción

Como ya se ha descrito, el objetivo final de los análisis de teselación es la generación, en la estructura de datos de las teselas definidas por una rejilla, de una o varias variables de adscripción.

Las características finales de una variable de adscripción dependen del tipo de dato que se va a procesar para dar origen a la misma. De esta manera, no son iguales las variables de adscripción fruto de un índice, por ejemplo, entre dos números, que arrojará un número con más o menos decimales, que la adjudicación de una categoría temática que se considere, dentro de un conjunto de las mismas, como definitoria de la tesela en su conjunto.

Básicamente encontraremos, por lo tanto, estos dos tipos de variables:

1. **continuas:** valores numéricos dentro de un determinado dominio;
2. **discretas:** variables categóricas dentro de un dominio de categorías.

Las variables del primer tipo no presentan más problemas: el cálculo final numérico se almacenará dentro de la estructura de datos de la tesela sin más ceremonia ni complicaciones. Sin embargo, en el caso de las discretas, se plantea un problema de espacio clarísimo que hay que solucionar.

Consideremos un sencillo ejemplo: asignarle a cada tesela una variable de adscripción con el nombre del municipio con más superficie en la misma. Esto podría ser solucionado asignando un código a cada categoría, como el código INE, pero esto requeriría a posteriori un procesamiento por parte del usuario. Lo interesante es almacenar el nombre (categoría) en sí. En Andalucía tenemos aproximadamente 770 municipios con nombres, muchas veces, nada cortos. Si se almacena la cadena de texto del mismo, el tamaño de la información final adscrita puede ser notable.

Para ello, las variables categóricas precisan, antes de ser utilizadas en un análisis de teselado, de un paso previo de *catalogación*. Este proceso examina el campo de la información original a teselar que contiene los datos categóricos, encuentra las ocurrencias únicas de cada uno de ellos y con ellos crea un catálogo, en el que asigna a cada valor del dato un pequeño código unívoco que será el que se consigne en los datos de la tesela. La plataforma aporta estos catálogos para que sean leídos en ambos sentidos: del código al dato y del dato al código. De esta manera, el espacio del dato teselado final se reduce mucho.

Pasos de un análisis de teselado

Una vez se tenga claro qué tipo de dato original va a procesar un análisis de teselado y qué conjunto y qué características tendrán sus variables de adscripción resultado, podemos detallar el conjunto de pasos que da un análisis de teselado para la consecución del cálculo de sus variables. Estos pasos son siempre los mismos aunque, dependiendo de la naturaleza del análisis, algunos pueden omitirse puesto que no juegan un papel analítico en el proceso de un análisis determinado. Sin embargo, estos pasos forman un marco estructural de la

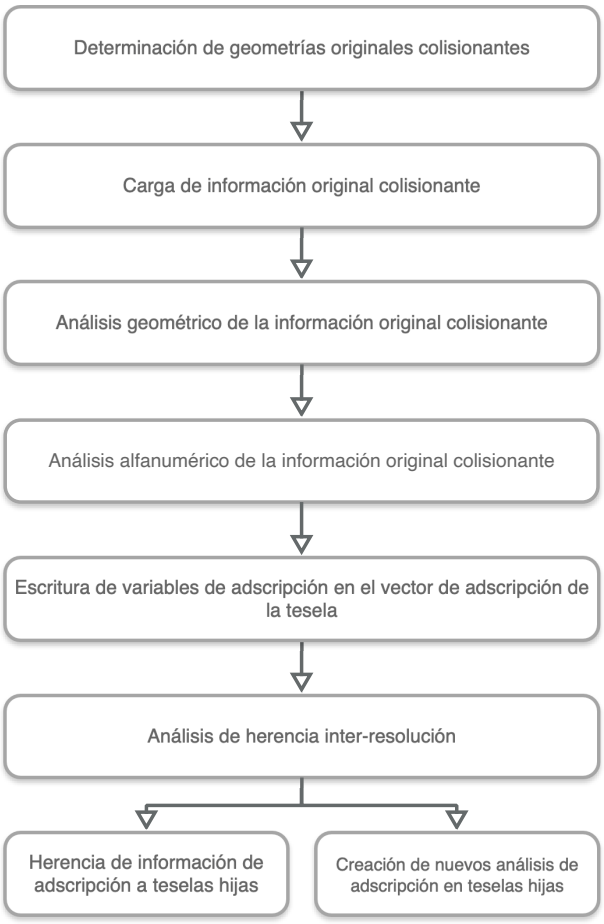

Figura II.4.2: Pasos funcionales de un análisis de teselación.

arquitectura que un programador[3] debe tener en consideración para el desarrollo e implementación de un nuevo tipo de análisis. Los pasos son lo suficientemente genéricos como para dar cabida a prácticamente cualquier tipo de análisis en el que la única consideración sea la tesela a procesar, es decir, no se establezcan relaciones topológicas con otras teselas (análisis atómico)[4].

Para cumplir este objetivo, el análisis de teselado debe dar los pasos descritos en la figura II.4.2, que pasamos a describir. En cada paso, sin entrar en detalles de implementación, que se darán en capítulos posteriores, sí se detallará una valoración del estrés computacional del mismo.

[3]Como veremos más adelante, la implementación de un nuevo análisis de teselado es un trabajo demasiado abstracto que debe caer en manos de un programador. Sería extremadamente complejo crear un sistema por el cual un no programador pudiera crear desde cero un análisis de teselado nuevo.

[4]En cualquier caso, los añadidos o modificaciones al esquema presentado para el caso de análisis no atómicos en el que existan influencias topológicas de teselas vecinas no sería demasiado complejo en el futuro.

Determinación de geometrías originales colisionantes

El primer paso analítico en cualquier análisis de teselado es determinar qué geometrías del juego de datos original colisionan con la tesela. Por "colisión" entendemos el cumplimiento de una condición topológica entre la geometría de la tesela y las geometrías originales, propia del análisis de teselado.

En la mayoría de los casos, esta condición topológica es una simple condición de solape, contención o intersección. Por ejemplo, en el caso del análisis de nubes de puntos, todos los puntos que caigan dentro de la tesela, o, en el procesamiento de geometrías originales poligonales, un solape. Sin embargo, esta condición topológica puede ser cualquiera. Por ejemplo, en el caso de las interpolaciones *IDW*, no basta con tener sólo los puntos originales que caen dentro de la tesela, sino que además hace falta extender esta, es decir, ampliarla en una distancia dependiente de la distancia máxima de influencia gravitacional de la nube de puntos original para tener en consideración todos los posibles puntos que puedan afectar a la tesela objetivo (figuras II.4.3, II.4.4 y II.4.5).

El estrés computacional de este paso es alto. En nubes de puntos, donde los algoritmos de indexación son muy eficientes (Nguyen, 2009), detectar colisiones es rápido incluso en conjuntos de muchos millones de puntos originales. Sin embargo, en el caso de los polígonos, por supuesto una buena indexación de los datos originales es determinante, pero aún así, tras encontrar la primera aproximación gracias al índice, el análisis final de intersección entre el polígono de la tesela y los polígonos originales es un algoritmo muy costoso en tiempo de computación.

Carga de información original colisionante

Una vez se hayan encontrado las geometrías originales que satisfacen la condición topológica de colisión en la fuente de datos de origen, éstas son cargadas en la memoria del análisis de teselado para su procesamiento.

Este paso tiene un estrés computacional bajo.

Análisis geométrico de la información original colisionante

Con los datos colisionantes en memoria, el análisis de teselado debe ahora implementar un tratamiento a su aspecto geométrico. Recordemos, por supuesto, que todos estos pasos son opcionales y que dependen de la naturaleza algorítmica de cada análisis de teselado.

Por ejemplo, en el caso de las nubes de puntos, dado que los puntos son adimensionales, el procesado geométrico suele ser nulo y este paso se obvia. Los puntos ya seleccionados por la condición de colisión ya están en la memoria de trabajo del análisis y por tanto listos en su forma original para casi todas las aplicaciones puntuales[5]. Los polígonos, sin embargo, sí son susceptibles de mucho procesamiento geométrico. Por ejemplo, para muchos análisis lo que nos va a interesar es lo que sucede en cada polígono exclusivamente en el contexto de la tesela considerada, por lo que por lo general los polígonos suelen ser interseccionados

[5] Y, de las aplicaciones consideradas, de todas.

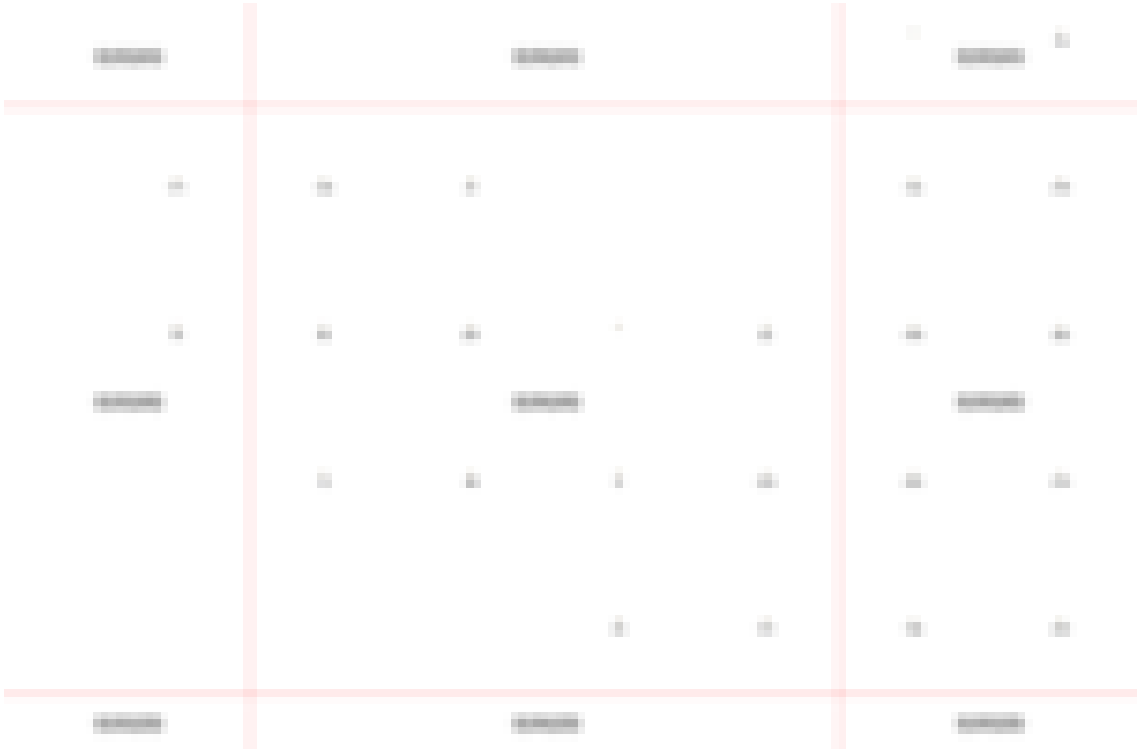

Figura II.4.3: En este ejemplo se puede observar un conjunto de puntos (éstos, pertenecientes a los datos originales de población, mostrando su atributo temático de población total para el 2002), colisionando por inclusión con una tesela de un kilómetro de lado. Fuente: elaboración propia.

por la geometría de la tesela, quedándose el análisis sólo con el área solapante del mismo y desechando el resto.

Este paso es estresante computacionalmente hablando en el caso de los polígonos, ya que la mayoría de los análisis de teselado van a requerir de su recorte con la geometría de tesela objetivo y éste es un proceso costoso.

Análisis alfanumérico de la información original colisionante

Una vez el aspecto geométrico original ha sido procesado, llega el momento de hacer lo propio con el aspecto alfanumérico de los datos originales. Este es quizás el paso más importante del algoritmo de procesamiento del análisis de teselado, ya que es donde se va a implementar la lógica del cálculo de la variables o las variables de adscripción, que son el objeto de todo el proceso.

En este punto, con la información original colisionante ya tratada y procesada en el contexto de la tesela objetivo, el algoritmo tiene acceso al juego completo de información alfanumérica asociada y a los resultados geométricos también del procesamiento geométrico del punto anterior[6], con lo que se pueden generar estadísticas[7] y/o índices entre dichos

[6]Por ejemplo, en el caso de los polígonos, el área total de afectación de los mismos en la tesela objetivo.
[7]Medidas de centralidad y dispersión estadística, recuentos, etc.

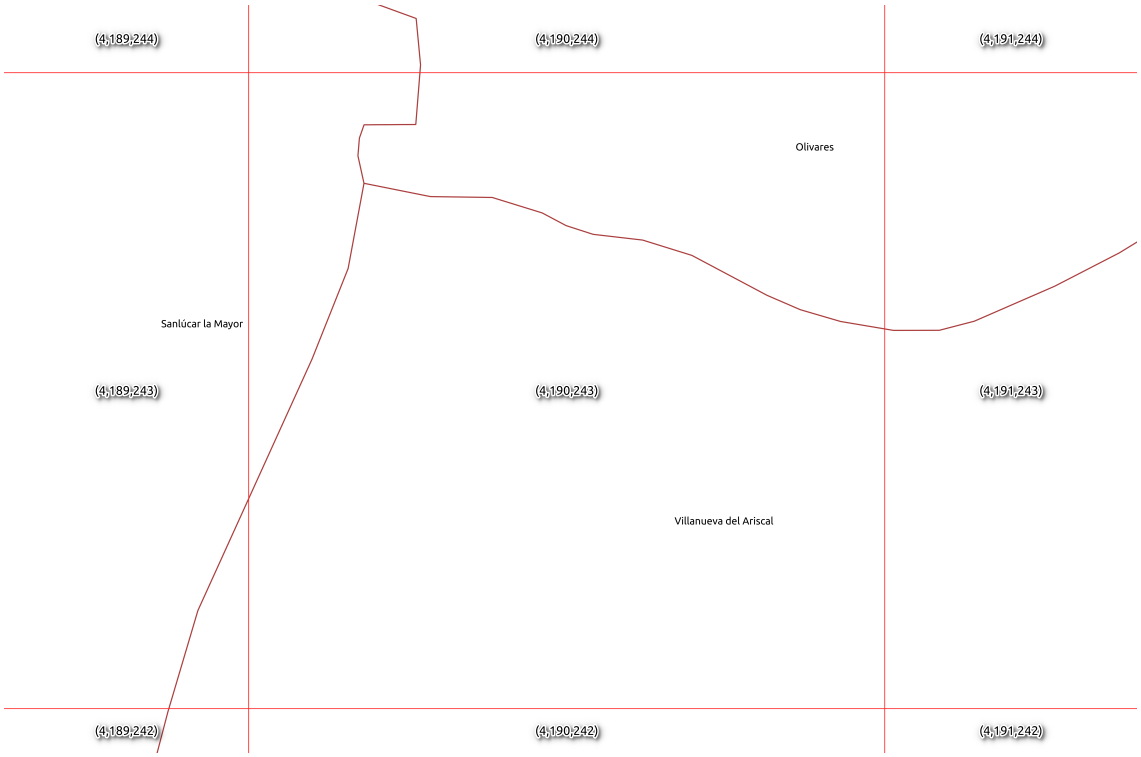

Figura II.4.4: En este ejemplo se puede observar un conjunto de polígonos (pertenecientes a los datos originales de municipios), colisionando por solape con una tesela de un kilómetro de lado. Fuente: elaboración propia.

valores.

Cada resultado obtenido por estos cálculos generará una variable de adscripción distinta que se añade a los resultados finales del análisis de teselado para la tesela objetivo.

Este paso no suele ser por lo general costoso, ya que no exige de operaciones de geoprocesamiento (ya se realizaron en el paso anterior) y con toda la información alfanumérica en la memoria del análisis de teselado suele ser rápido y eficiente.

Análisis de la herencia inter-resolución

Para cuando se llega a este paso del proceso el objetivo de calcular las variables de adscripción para la tesela objetivo está cumplido y se podría dar por finalizado el proceso. Sin embargo, como ya se ha comentado, existen pasos en el proceso del análisis que son muy estresantes computacionalmente hablando, y hay que aprovechar las características autoindexativas y de topología inter-resolución de la estructura de la rejilla multiescalar para minimizarlos todo lo posible.

Dado que existen relaciones topológicas implícitas entre los niveles de resolución contiguos en la definición de la rejilla, éstas pueden ser, dependiendo de la naturaleza del análisis de teselación, aprovechadas para agilizar la aplicación del análisis a las teselas hijas en el nivel inmediatamente inferior, aprovechando el propio procesamiento realizado

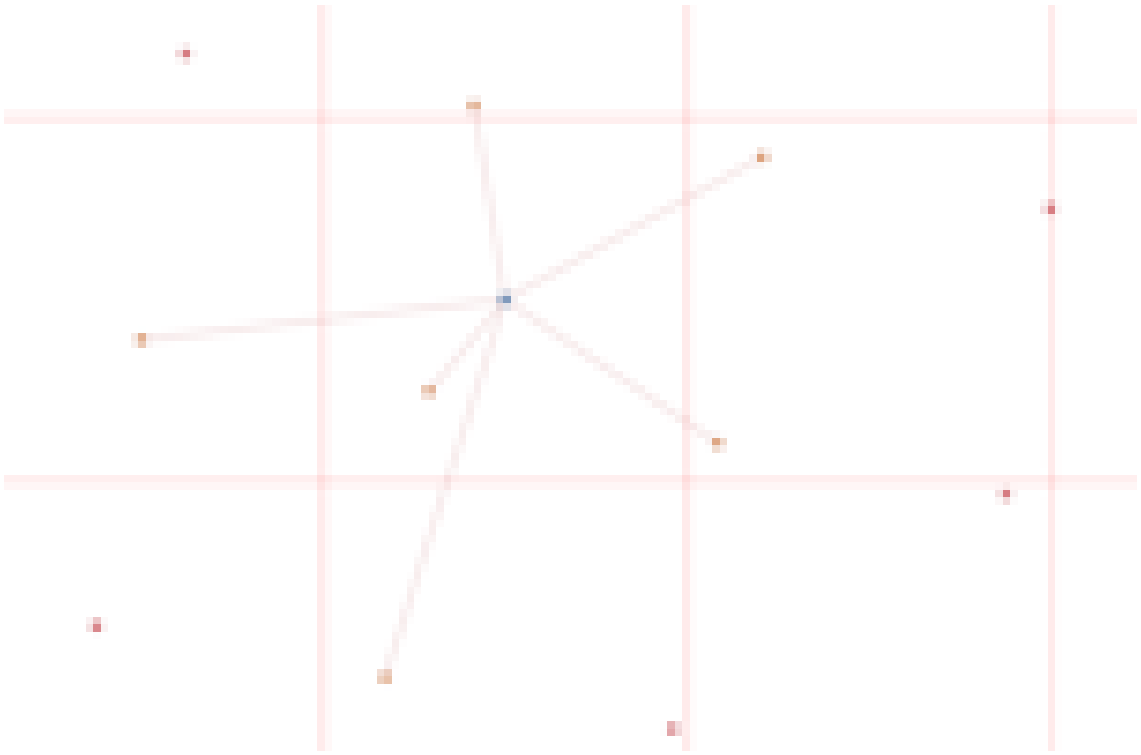

Figura II.4.5: A veces, el concepto de "colisión" no es tan evidente como en el caso de las figuras II.4.3 ó II.4.4. En este ejemplo se muestra el concepto de colisión de una convolución *IDW* para interpolar el punto central de la tesela. Las geometrías originales "colisionantes" (puntos unidos al central con una línea) son los 6 puntos más cercanos al centroide de la tesela, aunque la mayoría de ellos ni siquiera estén dentro de la tesela objetivo. Fuente: elaboración propia.

para la tesela objetivo.

En este paso, por tanto, se determina la estrategia de herencia de información hacia las teselas hijas. Un caso paradigmático es el análisis de la categoría temática discreta predominante en la tesela[8]. Si en el cálculo de la tesela objetivo se encuentra que una categoría temática obtiene el 100 % de cobertura de la misma, es trivial que en todos los niveles de resolución superiores (con teselas más pequeñas) éste valor va a ser también del 100 %. Por lo tanto, un análisis con este objetivo analítico simplemente iría ascendiendo en la escala de resolución de la rejilla adjudicando a todas las teselas hijas esa misma categoría al 100 %. Gracias a ello, el sistema se está ahorrando una buena dosis de estrés computacional no teniendo que volver a calcular colisiones y geoprocesamientos de recorte extremadamente costosos. En el otro extremo se encuentran, por ejemplo, las interpolaciones de nubes de puntos, que no suelen ser muy eficientes en este aspecto, necesitando recalcular todas las colisiones de nuevo por distancia objetivo con la nube de puntos original, dado que los cálculos se hacen con respecto al centro de la tesela objetivo.

Las estrategias para realizar este proceso son tan variadas como variados son los posibles análisis de teselado, y debe estudiarse con detenimiento para intentar aprovecharlas

[8]Por ejemplo, ver qué municipio es el que ocupa más área en la tesela.

al máximo.

Este es un proceso que puede llegar a ser costoso computacionalmente si el número de teselas hijas que heredan información de la tesela objetivo es alto, pero aún así, muy inferior al coste de procesarlas sin aprovechar las ventajas descritas en este apartado.

Escritura de las variables de adscripción generadas en la información alfanumérica de la tesela

En este se añade a la estructura de información que contiene las variables de adscripción para la tesela objetivo el resultado del análisis, que será una o varias nuevas variables de adscripción nuevas.

Generación de posibles nuevos análisis de teselado en teselas hijas

En función del análisis de herencia del paso anterior pueden darse tres situaciones:

1. el análisis de teselado en curso es capaz de asignar un valor para todas las variables de adscripción para sus teselas hijas y de esta manera calcular de una pasada toda la adscripción de sus descendientes;

2. el análisis puede aprovechar de forma parcial resultados de su propio procesamiento para reducir el estrés computacional de la aplicación del análisis de teselado en sus descendientes;

3. no existe lógica geométrica y/o alfanumérica mediante la cual los resultados del proceso en la tesela objetivo puedan ser de ayuda para el procesamiento de sus descendientes.

El primer caso, el más conveniente, permite resolver la adscripción en todas las teselas descendientes sin tener que lanzar sobre la plataforma ningún nuevo análisis de teselación.

El segundo permite guardar en la memoria de la plataforma información ya procesada que agilizará en gran medida la aplicación del análisis de teselación a los descendientes. Un caso fácil de visualizar vuelve a ser el de los municipios: si la aplicación del análisis a una tesela de gran tamaño ya ha determinado cuáles son los polígonos originales colisionantes y ya ha realizado un primer trabajo de recorte[9] de dichos polígonos con el polígono de la tesela, es más eficiente que sus teselas hijas hereden dichos recortes y no tengan que ir de nuevo a la fuente original de datos a comprobar colisiones y realizar recortes, sino que lo hagan ya con el contexto potencial de afectación determinado por su tesela madre de resolución inferior, lo que seguramente reduzca muchísimo el estrés computacional de dichas operaciones[10]. Si tenemos en cuenta que este proceso se va repitiendo en cada nivel de

[9] Conocido entre los algoritmos de geoprocesamiento popularmente como *clipping*.

[10] Aunque, por un lado, y ya a nivel de implementación, se pierde el efecto de los potentes índices de un sistema como *PostgreSQL*. Pero si tenemos en cuenta que el área de búsqueda se ha acotado enormemente gracias al recorte de la tesela madre, y que además los polígonos han sido muy simplificados por dicho recorte (grandes líneas rectas en los límites de la tesela madre), el balance es más que positivo aún sin índices.

resolución con cada vez conjuntos de datos más fragmentados, la ganancia puede ser importante. En este caso, el análisis de teselado en curso generará nuevos análisis de teselado para las teselas hijas pero almacenando en la memoria de la plataforma esta información ya tratada que le será suministrada a las mismas para agilizar el procesamiento.

El tercer caso es el menos favorable. No se puede aprovechar ni a nivel geométrico ni a nivel alfanumérico los cálculos de la tesela objetivo, dado que la naturaleza del algoritmo obliga a comenzar el proceso desde cero. En este caso no queda más remedio que crear tantos nuevos análisis de teselación como celdas hijas existan y realizar el análisis de principio a fin.

II.4.2. Adaptación de las fuentes originales para los análisis de teselado

Uno de los requisitos descritos en el capítulo anterior en el esquema de diseño de un análisis de adscripción es decidir cual es la mejor estructura de datos necesaria para un eficaz desempeño computacional de la tarea. Este interfaz de entrada de datos, donde se impone un rígido marco estructural para la información de origen, ha de ser respetado para que el análisis pueda ser llevado a cabo.

Por lo tanto, es tarea del usuario de la plataforma formatear la información original de una forma que sea compatible con el análisis de adscripción que se pretende aplicar. En este sentido, las estructuras de datos a ser consumidas por los análisis deben ser las más sencillas posibles, incluso si esto supusiera una redundancia inducida en los datos de origen.

Muchos de los formatos más populares de información geográfica, por su sencillez, son de tipo monotabla, es decir, una única tabla asociada a una seria de geometrías. Este modelo monotabla, implantado en la práctica de la geografía computacional por la popularidad del formato *SHAPEFILE* de *ESRI* durante décadas, es altamente inapropiado para la consignación de la información con garantías de calidad. El modelo monotabla, donde además la entidad fuerte es la geometría, limita al modelador de datos de una forma tal que muchas veces las únicas soluciones posibles van en contra de todos los preceptos del buen diseño de datos[11]:

1. redundancia sistemática de datos complejos entre registros: varias filas de la tabla contienen el mismo dato complejo (por ejemplo, el nombre de una provincia);

2. redundancia sistemática de datos complejos a nivel de atributos: las conocidas columnas “municipio 1”, “municipio 2”, . . . , “municipio X”;

3. falta de atomicidad en el dato: las conocidas columnas que contienen datos separados por comas ([especie vegetal presente 1, especie vegetal presente 2, . . . , especie vegetal presente X]).

[11] La experiencia docente nos indica que la lucha contra esta “mentalidad *SHAPEFILE*” es el primer y principal obstáculo a la hora de enseñar modelado de datos.

En definitiva, con este modelo, aunque tiene la ventaja de ser extremadamente sencillo, se produce la panoplia completa de errores de diseño en lo que a modelado de datos se refiere[12].

El usuario, por tanto, tiene que trabajar la información original para desagregarla, atomizarla, y proporcionarla al análisis en condiciones sobre todo de atomicidad, aunque por lo general el diseño de los análisis tolera la redundancia.

Si el usuario controla la fuente original de datos, esta adaptación tendrá la forma generalmente de vistas materializadas (Obe et al., 2017) de *PostGIS*, donde la adaptación se puede hacer de forma no destructiva, respetando la estructura original de los datos.

II.4.3. El motor de teselado

De todas las ventajas de los sistemas basados en arquitecturas de microservicios (Jaramillo et al., 2016), la plataforma de teselado se beneficia sobre todo de las capacidades de escalado de las mismas a la hora de afrontar volúmenes de análisis de teselado grandes, bien sea por el volumen inherente de la información (mucha extensión, por ejemplo, o niveles de resolución muy finos) o por el estrés computacional del análisis de teselado seleccionado[13].

En la figura II.4.6 podemos ver un esquema conceptual de los microservicios que componen la plataforma de teselado:

1. **controlador:** la misión del microservicio controlador, como su propio nombre indica, es el control de la actividad del resto de los microservicios;

2. **almacén de datos:** la base de datos donde se guardan tanto los datos de origen como los datos teselados. Pueden ser obviamente fuentes de datos distintas pero a nivel conceptual forman una unidad;

3. **memoria compartida:** para aumentar la eficiencia, se precisa de un microservicio que implemente un sistema de memoria compartida entre los trabajadores y el controlador;

4. **trabajadores de teselación:** estos microservicios son los que se encargan de realizar el teselado propiamente dicho, ejecutando los análisis de teselación sobre las fuentes de datos originales.

Controlador

En una arquitectura de microservicios, una de las premisas es diseñar los mismos de forma que sean lo más autónomos posible. Por ello, el término controlador puede ser un poco equívoco.

[12] Con las hojas *Excel* pasa lo mismo en muchas ocasiones.
[13] O peor, por ambas a la vez.

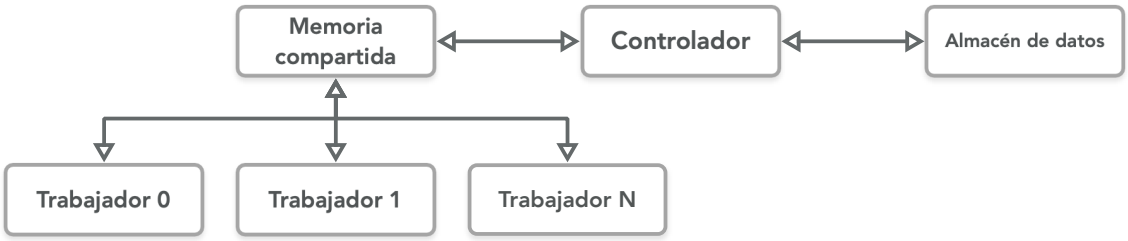

Figura II.4.6: Esquema conceptual de microservicios de teselado de la plataforma.

El controlador se encarga de llevar a cabo diversas tareas de gestión y mantenimiento automático en la plataforma. Es muy importante tener en cuenta que, en una arquitectura de microservicios, éstos pueden caer en cualquier momento. El *hardware* falla, los programas tienen errores, siempre puede surgir una eventualidad inesperada o con muy baja probabilidad de ocurrencia que haga que un microservicio falle. Por lo tanto, las arquitecturas de microservicios han de ser resistentes a las caídas de servicio de alguno de sus componentes y ser capaz de recuperarse de esa eventualidad.

Por otra parte, uno de los objetivos básicos de estas arquitecturas es potenciar al máximo la escalabilidad, es decir, redistribuir en cada momento los recursos de *hardware* disponibles hacia la tarea más prioritaria. Si diseñamos microservicios que hacen sólo una cosa[14], para darle más recursos computacionales de *hardware* a esa prioridad sólo habrá que levantar más microservicios de ese tipo, que serán destruidos al finalizar la tarea. Esto quiere decir que las arquitecturas de microservicios se diseñan teniendo en cuenta que son altamente concurrentes, es decir, que están sucediendo muchas cosas simultáneamente con la complicación adicional de que el orden en el que se ejecutan no es siempre predecible.

Por estas dos características, resiliencia frente a caídas parciales de componentes y coordinación de la concurrencia, es necesario que exista uno o varios componentes[15] que se encarguen de tareas de gestión.

El primer paso para alcanzar estos objetivos, sobre todo el de resiliencia a caídas de servicios, es, como se puede observar en el esquema, desacoplar los componentes. El controlador no está directamente conectado con los trabajadores: esta dependencia tan directa sería un inconveniente para el objetivo de autonomía tanto del controlador como de los trabajadores. En lugar de ello, utilizan un mecanismo centralizado, muy resistente a caídas y altamente recuperable, de memoria compartida. Gracias a este microservicio, la comunicación entre componentes no es directa, sino que se articula en base al intercambio de *mensajes*. La memoria compartida actúa como un servicio de mensajería, donde los microservicios se conectan para recibir mensajes o dejar mensajes destinados a otros servicios.

Las ventajas de la comunicación por colas de mensajes en microservicios de memoria

[14] De ahí lo de *micro*.

[15] Diseñar la arquitectura, siempre que se pueda, teniendo en cuenta que todos los microservicios pueden ser redundantes le dará a la misma un añadido de robustez muy necesario en sistemas de disponibilidad crítica, por ejemplo.

compartida son evidentes para la consecución de los objetivos de resiliencia y concurrencia:

1. los microservicios están inertes a la espera de algún trabajo que hacer. Estos trabajos se distribuyen por colas de mensaje en la que los microservicios que las consumen pueden leerlos de dos formas:

    a. **bloqueante:** los microservicios compiten entre sí por hacerse con el mensaje. Sólo el primero que llega se lo queda de forma exclusiva y ese mensaje no podrá ser recibido por ningún otro microservicio de su mismo tipo. Ideal para repartir pequeñas tareas computacionalmente intensas entre varios trabajadores, como es el caso de los análisis de teselado para cada una de las teselas objetivo. Tan sólo hay que colgar en la cola de mensajes órdenes de ejecución de dichos análisis y los microservicios trabajadores las irán cogiendo de una en una, a medida que van acabando, de forma ordenada y sin posibilidad de que por error dos hagan una misma tarea dos veces, así hasta que se acabe la cola;

    b. **sindicada:** este tipo de cola de mensaje es como un tablón de anuncios. El mensaje será enviado y recibido en todos los microservicios de un tipo determinado. De esta manera, por ejemplo, se les puede decir a los trabajadores, todos a la vez, que cuando terminen el análisis de teselado que están ejecutando en ese momento no acepten ninguno más, es decir, pausar o reanudar la teselación;

2. los microservicios informan periodicamente en una cola de mensajes especial (cola de *latido de corazón* o *heartbeat*) de su actividad y estado a intervalos de tiempo prefijados, una especie de anuncio de que sigue ejecutándose y que, por tanto, no ha fallado en lo que quiera que está haciendo. Estos mensajes *heartbeat* están destinados al controlador;

3. si un trabajador fallara en la ejecución de un análisis de teselado, éste saldría arrojando un error y dejaría de enviar mensajes a la cola *heartbeat*;

4. los microservicios de trabajadores que finalizan con éxito su análisis de teselación dejan los resultados del mismo en una cola de mensajes destinada al controlador para que este los procese y los almacene en el almacén de datos (con el que sí tiene una relación directa);

5. si se añaden[16], para acelerar trabajo, nuevas copias de trabajadores de teselado a la arquitectura, éstos sólo tienen que ser conscientes de la existencia de la memoria compartida. La existencia de microservicios de controladores, almacén de datos o sus otros compañeros trabajadores les es indiferente: consumen mensajes desde una cola y ponen resultados en otra. Esto permite que sean funcionales y productivos, sin necesidad de ninguna otra acción, con sólo activarlos;

6. si se cae un controlador, los trabajadores siguen dejando sus resultados en la memoria compartida. Cuando el controlador se recupere de su caida, tendrá trabajo por procesar pero lo importante es que éste no se habrá perdido.

[16]Hay que destacar que este proceso de añadir o quitar microservicios tiene la ventaja, si están bien diseñados y lo suficientemente desacoplados entre sí, de que se puede realizar *en caliente*, es decir, sin tener que apagar el sistema ni hacer ningún otro cambio en microservicios que ya están funcionando.

Como se puede observar, por tanto, el trabajo del controlador es ser la puerta de entrada a la arquitectura y coordinar el trabajo del resto de microservicios, especialmente de los trabajadores, gracias a los mecanismos de cola de mensajes. Para llevar a cabo estas tareas, el controlador, aparte de suministrar y atender requerimientos de forma inmediata a traves de las colas de mensaje, realiza las siguientes tareas a intervalos regulares:

1. examina el estado general del sistema (uso de la CPU, memoria, tamaño de carpetas críticas de datos en el disco duro, etc.);

2. examina el estado de los análisis en curso, junto a la información de *heartbeat* de los trabajadores, para detectar posibles errores y caídas de servicio y relanzar las tareas no realizadas con éxito.

Memoria compartida

Como se ha discutido ampliamente en la sección anterior dedicada al controlador, la memoria compartida juega un papel vital en la arquitectura como nexo de unión de todos los demás microservicios (Zhang et al., 2014). Por ello, su resistencia a fallos, caídas y su capacidad de recuperación frente a estas eventualidades son de máxima importancia para la estabilidad de la plataforma en su conjunto.

La memoria compartida cumple diversos papeles en la arquitectura:

1. es un almacén temporal de datos para intercambiar información entre microservicios, comportándose como una base de datos ultrarrápida y muy eficiente;

2. gestiona las colas de mensajes descritas en la sección anterior.

Para que este microservicio rinda con eficacia, lo ideal es que sea una solución 100 % en memoria, es decir, el almacén de datos no funcione desde el disco duro, como suele ser normal en bases de datos convencionales, sino que todos los datos se alojen en la memoria de la máquina[17].

La memoria compartida es lo que permite el desacoplamiento entre los microservicios. Dado que todos se conectan a ella para el intercambio de información y tareas, no hay necesidad de una conexión componente a componente directa (por ejemplo, entre el controlador y los trabajadores), que es de una naturaleza mucho más compleja e inestable. Todo lo que tiene que hacer un microservicio trabajador o controlador es conectarse a este microservicio y comenzar a escuchar y escribir en las colas de mensajes designadas.

Almacén de datos

El almacén de datos es una base de datos de un tipo u otro que se usa como el repositorio definitivo de la información teselada y como fuente de datos originales para la teselación. Nótese como su única conexión es con el controlador: dado que para estos

[17] Esto hace que, obviamente, el espacio de almacenamiento sea limitado. Sin embargo, su papel es el intercambio efímero de información, no la salvaguarda de la misma a largo plazo, así que bien gestionada y dimensionada no debe constituir un problema muy serio.

servicios la entrada / salida de información suele ser un cuello de botella en su rendimiento, así como el número de conexiones simultáneas, el hecho de que sea un número limitado de microservicios la que la usan y controlan el acceso y escritura de información, de una forma controlada[18], hace que este microservicio[19] tenga más posibilidades de rendir a buen nivel[20].

Trabajadores de teselación

Los trabajadores de teselación tienen la tarea de realizar los análisis de teselado. Éste es el microservicio que más se duplica, ya que, cuantos más haya, más análisis se podrán computar a la vez, aumentando, por concurrencia, el rendimiento de la tarea global. Este aumento, obviamente, está limitado por los recursos de *hardware* disponibles. Lo óptimo es dedicar a cada uno un núcleo de microprocesador[21]. Por supuesto, también usan la memoria, que es otro factor de *hardware* limitante.

Los trabajadores de teselación se comunican exclusivamente con la memoria compartida. Cuando un trabajador se activa, establece una conexión con dicho microservicio y se conecta a las colas de mensajes donde el controlador deposita los requerimientos de cálculo de análisis de teselación, con los datos necesarios para llevar a cabo el trabajo. El trabajador lee los datos de la cola[22] y procesa el análisis de teselación. Al terminar, éste deja los resultados en la cola de mensajes que lee el controlador y es éste el encargado de llevar la información procesada a la base de datos para su almacenamiento definitivo.

Este patrón de diseño, minimizando las interdependencias entre microservicios y centralizándolo en la memoria compartida, tiene la ventaja de tener una tramitación de puesta en marcha o desactivación de trabajadores mínimo. Cuando un trabajador se activa, lo único que tiene que hacer es recibir trabajos, de una forma completamente transparente para el resto del sistema. Las interdependencias son mínimas. De las misma manera, cuando termina un trabajo, puede optar por apagarse, concluyendo su ejecución. Este proceso puede realizarse incluso desde fuera de la máquina en la que se encuentra la memoria compartida, usando otra máquina donde se ejecutan trabajadores adicionales que, usando

[18] Por ejemplo, no rinde igual hacer 500 microtransacciones de datos a una base de datos, con 500 procedimientos individiduales de apertura de conexión, registro de credenciales de usuario, etc. que hacer esas 500 transacciones de datos de una sóla vez en la que la parte administrativa del proceso se ejecuta una vez y no quinientas.

[19] No debe llevarnos a equívocos aquí el término *micro*: una base de datos de varias decenas de gigas o un tera tiene de todo menos de *micro*. Pero a nivel conceptual es un microservicio, ciertamente.

[20] No debemos olvidar que una arquitectura de microservicios tiende a rendir en su conjunto como el peor de ellos. Son los llamados *cuellos de botella* de las arquitecturas. El rendimiento de una base de datos es un cuello de botella típico, por eso son microservicios a los que se dedican muchos recursos y sobre los que se habilitan muchas técnicas de mejora del rendimiento, como réplicas, balanceadores de carga, etc.

[21] Un *core* o núcleo de microprocesador es una unidad dentro del mismo capaz de gestionar un hilo de ejecución de un programa. Los microprocesadores tienen varios núcleos para poder realizar más tareas a la vez eficientemente. Cuando un programa se ejecuta, éste va abriendo hilos (*threads*) de ejecución, que tienden a acomodarse en núcleos inactivos si estos están disponibles. Si no, tienen que compartir tiempo de núcleo con otros hilos de ejecución, pero en este momento el rendimiento de la máquina comienza a degradarse.

[22] Recordemos que esta lectura es bloqueante: cuando un trabajador comienza a leer un mensaje de la cola, lo bloquea y ya no está disponible para otros trabajadores. Esto impide que dos trabajadores reciban el mismo trabajo.

la red que conecta a ambas máquinas, encuentra a la memoria compartida y comienzan a trabajar. De esta manera, la capacidad de procesamiento de la plataforma viene determinada por el *hardware* disponible y por el comportamiento y rendimiento de microservicios que pueden convertirse en cuellos de botella, como es el caso ya mencionado del almacén de datos.

Aunque ya se ha destacado que la interdependencia con la plataforma de un trabajador de teselado es mínima, aún así el controlador debe saber de su existencia para controlar el trabajo realizado por cada uno de ellos. Los análisis pueden fallar, haciendo a un trabajador terminar abruptamente su ejecución y saliendo del proceso con un error. En este caso, tendríamos un análisis de teselado que no está concluido satisfactoriamente, dejando por lo tanto una tesela o grupo de teselas sin datos. Esto es algo que hay que controlar de alguna manera, y el sitio correcto para hacerlo es el controlador.

Usando la memoria compartida, el trabajador usa una cola de mensajes para dejar a intervalos un pequeño paquete de datos acerca de su estado actual, que puede ser:

1. **inerte pero funcionando:** el trabajador está en funcionamiento pero actualmente no está realizando ningún trabajo, sino que está a la espera de que aparezcan nuevos análisis en la cola de mensajes de la memoria compartida;
2. **trabajando en un análisis:** procesando actualmente un análisis.

Estos mensajes de estado llevan una marca de tiempo, lo que permite al controlador tener una idea del estado de los trabajadores. Si se detecta que un trabajador deja de emitir señal durante un tiempo superior al del intervalo de esa misma señal (dentro de un márgen considerado seguro), el controlador dará al trabajador por terminado y dara parte del error para que sea examinado, volviendo a poner en la cola de trabajadores cualquier trabajo no concluido para que sea realizado por otro.

II.4.4. Configuración del análisis de teselado inicial

Para comenzar el proceso de teselado, la plataforma precisa una definición previa del trabajo a realizar para construir un grupo inicial de análisis de teselado que desencadenará todo el proceso. Para ello, dicha definición debe llevar las siguientes especificaciones:

1. **rejilla a utilizar en el teselado:** en una plataforma pueden convivir varias rejillas[23], así que hay que definir qué rejilla se va a utilizar en el análisis para adscribir;
2. **fuente de datos de origen:** datos de conexión a la fuente de datos de origen, donde residen los datos a adscribir convenientemente formateados según se describió en el capítulo II.4.2;
3. **características y configuración propia del análisis de teselado:** cada análisis de teselado implementado en el sistema tiene unas características algorítmicas distintas, generando al final del proceso una o varias variables de adscripción, por lo

[23] Sin embargo, no se han planteado aplicaciones inter-rejillas. Cada rejilla tiene sus datos completamente separados del resto.

que hay que definirle a la plataforma qué análisis de teselado queremos llevar a cabo y con qué configuración;

4. **el área a teselar:** hay que proporcionar en la configuración el área que se quiere teselar, que puede ser o bien una tesela de alto nivel[24] o bien un área definida por un polígono;

5. **el nivel de resolución mínimo por el que empezar:** es decir, la tesela más grande por la que comenzará la teselación;

6. **el nivel de resolución máximo en el que terminar:** el tamaño de tesela más pequeño donde terminará el proceso de teselado.

Estos dos últimos parámetros de teselación son de capital importancia puesto que determinan mucho la eficiencia global del teselado. Si se tienen en cuenta las relaciones topológicas inter-resolución descritas en el apartado II.4.1 del diseño de los análisis de teselado, comenzar por un nivel de resolución mínimo apropiado y terminar en el máximo que se considere necesario en un único teselado puede ser más eficiente, ya que se aprovechan al máximo las capacidades de herencia. Todo ello, claro, dependiendo de la propia naturaleza del análisis de teselado seleccionado y de las características geométricas de la información original.

Por ejemplo, sería muy poco eficiente teselar polígonos grandes entre resoluciones de 10 kilómetros y 250 metros y a posteriori decidir que se necesitan también las resoluciones de 125 y 25 metros. Llegar a estos niveles grandes de resolución por herencia es eficiente, pero teselar de nuevo un área grande comenzando por 125 metros puede ser costoso en el tiempo.

Con esta información, la plataforma ya puede abordar el teselado.

II.4.5. Teselado en la plataforma

El teselado en la plataforma se produce una vez se han tomado las acciones previas ya citadas:

1. se ha seleccionado una definición de rejilla para la teselación;

2. se ha elegido, entre los análisis de teselado disponibles en la plataforma, el que es más apropiado para cumplir los objetivos analíticos del proceso de adscripción a la rejilla y la naturaleza de los datos de origen;

3. se ha adaptado la fuente original de datos a las necesidades de información del análisis de teselado seleccionado.

[24] En las pruebas de la plataforma se consideró que las pruebas de teselación eran más fáciles y menos propensas a repetir teselaciones en áreas ya teseladas si el punto de partida del teselado era una tesela del primer nivel de resolución. Es una forma muy cómoda de organizar y llevar un seguimiento del trabajo de teselado en pruebas, así que se ha quedado como una opción en la plataforma final.

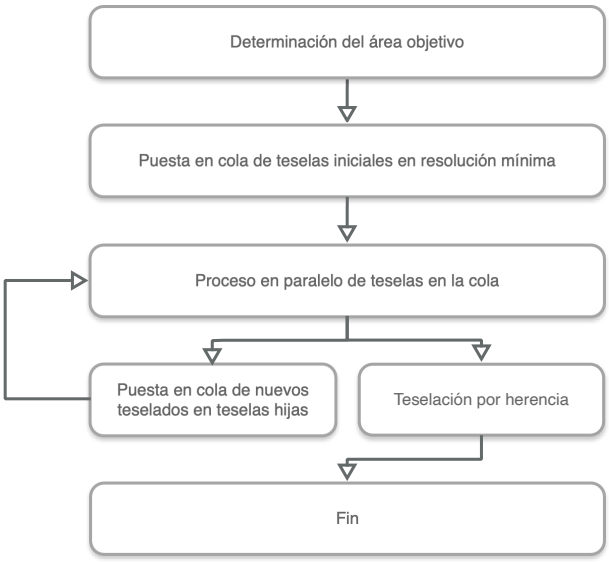

Figura II.4.7: Pasos funcionales del proceso de teselación en la plataforma.

Una vez hecho esto, se le puede dar la orden a la plataforma para que comience el teselado. El teselado conlleva una serie de pasos descritos en la figura II.4.7, que se describen a continuación.

Determinación del área objetivo

Para comenzar el teselado, la plataforma ha de hacerlo sobre un área objetivo, que puede ser proporcionada de distintas formas:

1. **no aportarla:** la plataforma simplemente comenzará a teselar todo el ámbito de los datos originales;

2. **una o varias teselas de resolución baja inicial:** las propias teselas de baja resolución son una buena forma de organizar los trabajos de teselado, y evidentemente muy natural para la plataforma. Indicarle una tesela de baja resolución (100x100 kilómetros, por ejemplo) es una forma eficaz de gestionar estos trabajos por lotes;

3. **indicando un área de interés:** se le proporciona a la plataforma el área que interesa teselar como polígono.

Selección de las resoluciones inferiores y superiores

Se han de seleccionar las resoluciones inferior (es decir, más gruesas, p.e. 100x100 kilómetros) y superior (más finas, p.e. 125x125 metros) entre las que se quiere ejecutar el analisis de adscripción.

Selección de teselas iniciales en resolución mínima

Una vez determinada el área objetivo y el nivel de resolución más bajo para la adscripción, la plataforma debe encontrar cuáles son las teselas de dicha resolución que colisinan con dicha área:

1. en el caso de no haber aportado área de interés, se estudia el ámbito completo de los datos de origen y se encuentran todas las teselas de baja resolución que colisionan con dicho ámbito;
2. en el caso de haber aportado una tesela o teselas de baja resolución, son exclusivamente estas teselas las que desencadenan el análisis;
3. en el caso de haber aportado un polígono de área de interés, se encuentran las colisiones con las teselas de baja resolución y son éstas teselas las que dan origen al análisis.

En cualquier caso, el resultado final de este paso del análisis es encontrar una o varias teselas iniciales en el rango mínimo de resolución que afectan al área considerada. Esta tarea la hace el controlador de la plataforma.

Teselado

Con el juego de teselas iniciales ya encontrado comienza el análisis de teselado propiamente dicho. Estas teselas se ponen a disposición de los trabajadores de la plataforma, que las cogen de una en una y las procesan, aplicando el análisis de teselado seleccionado.

El proceso de cada una de estas teselas iniciales puede dar lugar a dos situaciones:

1. **es posible el teselado de niveles superiores de resolución por herencia:** en este caso, un único trabajador resuelve la teselación tanto de la tesela inicial como de todas sus descendientes hasta el nivel de resolución máximo designado;
2. **no es posible resolver las teselas descendientes por herencia:** en este caso, el trabajador generará una serie de nuevas teselas hijas sobre las que aplicar de nuevo el análisis de teselado.

En el primer caso, un único trabajador acaba generando el resultado del análisis de adscripción para toda la columna tesela inicial - descendientes.

En el segundo caso, sin embargo, el trabajo se paraleliza con otros trabajadores que pudieran estar disponibles. Al finalizar el análisis de adscripción de la tesela inicial, el trabajador determina que la adscripción de las teselas descendientes no puede hacerse por herencia. En este caso, determina cuales son las teselas hijas en el siguiente nivel de resolución e instruye al controlador sobre la necesidad de poner el cola el proceso de las mismas. En este momento, el ciclo se repite, sufriendo estas nuevas teselas descendientes el mismo tratamiento que una tesela inicial. Este proceso es repetitivo hasta alcanzar el nivel máximo de resolución indicado, bien por herencia a partir de un determinado nivel de resolución, bien por repetición de puesta en cola de nuevas teselas, cada vez de más resolución, a analizar.

Finalización del proceso de teselado en la plataforma

Llega un momento, bien por proceso por herencia, bien por repetición del ciclo tesela madre - descendientes, en que todas las teselas en todos los niveles de resolución seleccionadas han sido resueltas. En este momento, cada tesela en el almacén de datos ya tiene calculadas todas las variables de adscripción fruto del análisis seleccionado, almacenados en su vector de adscripción.

Obtención de información teselada

Un análisis de teselado finalizado, como se ha comentado en apartados anteriores, deja una grupo de variables de adscripción listos en cada una de las teselas en el almacén de datos, en su vector de adscripción.

Una de las propiedades de la estructura de datos teselada propuesta es su asimetría en los dos aspectos de la información geográfica. En el aspecto alfanumérico o temático, esta asimetría se expresa en la propiedad que tiene la tesela de incorporar un número arbitrario de variables de adscripción, dependiendo de la presencia[25] de un fenómeno territorial concreto en ella. Aquellas teselas que no los contengan no tendrán esas variables de adscripción.

Por lo tanto, la implementación de la plataforma precisa de un medio de administración y almacenamiento de datos flexible en este aspecto. Consideraciones de implementación aparte, que se discutirán en próximos capítulos, lo importante es que dicho almacenamiento propicie una *vectorización* de las variables. En este caso no estamos hablando de *vectores* en el ámbito geométrico clásico de los sistemas de información geográfica, sino de vectores de datos.

Un vector de datos es un conjunto ordenado de datos individuales. El órden es importante, lo que lo distingue de un conjunto de datos, y el número de elementos también. Un vector, en cuanto a dimensiones se refiere, y para diferenciarlo de una matriz[26], tiene sólo una dimensión:

```
[ variable A, variable B, ..., variable N ]
```

La dimensión del vector, por tanto, es el número de elementos (llamados *componentes*) que poseen.

Es importante que la plataforma sea eficaz buscando variables de adscripción presentes en las distintas teselas y extrayendo un conjunto de las mismas para confeccionar un vector de variables. Una funcionalidad primordial de la plataforma es encontrar, por tanto, cuántas teselas en un área determinada tienen todas las variables de adscripción que se quieren incluir en el vector de datos y recuperar dicho vector a partir de ellas. Estos

[25] Presencia o influencia. Se pueden implementar análisis de teselado que calculan la influencia (por ejemplo, accesibilidad o distancia) a fenómenos territoriales que, no estando presentes físicamente en la tesela, no obstante la influyen e inducen una variable de adscripción en la misma.

[26] Las matrices pueden considerarse como un grupo de vectores de la misma dimensión.

vectores son la base para realizar los procedimientos de *Machine Learning* (Pereira et al., 2019) que se describen a continuación.

II.5 Machine Learning

Con la obtención de vectores de variables de adscripción a partir de la información contenida en cada tesela es posible abordar procedimientos de *Machine Learning*.

El *Machine Learning* es un conjunto de técnicas estadísticas y computacionales orientadas a la extracción de información a partir de un conjunto de datos complejo y multivariante. Son técnicas que ponen en relación estadística un conjunto de datos con grados de interdependencias variables para encontrar patrones y/o clasificaciones en sus distribuciones, sobre una población de elementos que tienen todas las características consideradas.

El *Machine Learning* y la Inteligencia Artificial están relacionadas (Raschka et al., 2019), considerándose las técnicas de *Machine Learning* como una parte importante del más amplio campo de la Inteligencia Artificial. Muchas técnicas de *Machine Learning* van encaminadas al análisis predictivo, mientras que otras se concentran en el análisis exploratorio de datos. Otras van encaminadas a la reducción de la dimensionalidad de un problema. En cualquier caso, la práctica del *Machine Learning* requiere una buena base previa en estadística descriptiva e inferencial.

Entre las técnicas de *Machine Learning* (*ML*) se hace una distinción entre aquellas que son supervisadas y no supervisadas (Pereira et al., 2019):

1. un algoritmo de *ML* es **supervisado** cuando se le proporciona experiencia previa contrastada por un experto, es decir, el algoritmo aprende de reglas ya existentes[1];

2. en cambio, un algoritmo de *ML* es **no supervisado** cuando trabaja directamente sobre la estructura de los datos sin ninguna regla previamente aprendida en busca de patrones en los mismos.

Una de las aplicaciones de los algoritmos de *Machine Learning* más importantes son los algoritmos de clasificación, es decir, algoritmos destinados a clasificar una población de elementos en un número determinado de grupos en función de sus características. Estos son los algoritmos que se van a implementar en la plataforma para analizar los datos de adscripción.

[1] Es decir, las reglas de clasificación o inferencia son aprendidas tras el análisis de conjuntos de datos seleccionados que corresponden con una categoría de clasificación conocida.

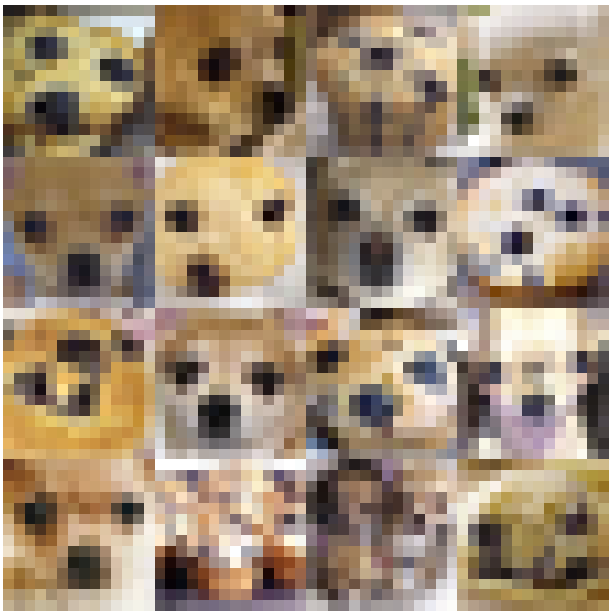

Figura II.5.1: Muestra del popular set de datos de entrenamiento para aplicaciones de visión computacional *Chiuauas vs Muffins*. Fuente: Togootogtokh et al., 2018.

II.5.1. Formato de datos necesario para alimentar algoritmos de *Machine Learning*

La gran mayoría de algoritmos de *Machine Learning* precisan de que la información suministrada esté en forma de vectores de datos (Kanevski et al., 2008). Estos vectores de datos describen las características de los elementos a analizar, y las componentes del vector deben ser consistentes en todos ellos, es decir, el orden en el que proporcionamos las variables de adscripción al algoritmo no suele ser relevante, pero lo que sí lo es es que todos los vectores tengan las componentes ordenadas de la misma manera.

Para poner un ejemplo, recurriremos a un ejemplo clásico en el aprendizaje de estas técnicas. Dependiendo de la técnica a implementar, existen una serie de juegos de datos de ejemplo que se consideran canónicos dentro de la disciplina. Estos conjuntos de datos, a veces muy pesados, se utilizan en el desarrollo de nuevos algoritmos, ya que, al enfrentar a algoritmos ya conocidos y estudiados frente a nuevos con el mismo conjunto de datos de prueba, se puede medir la eficacia de los nuevos métodos[2].

Para ejemplificar cómo funciona un algoritmo de clasificación de *Machine Learning* vamos a examinar uno de estos clásicos conjuntos de datos de contraste conocido como *The Iris Dataset* (el set de datos de las flores iris) (Yang, 2013). Las *iris* es un género de plantas con flores con muchísimas especies, hibridadas por la belleza de las mismas. Sus

[2]Un ejemplo realmente curioso de estos conjuntos de datos de contraste es el conocido como *Chiuauas contra Muffins* (Togootogtokh et al., 2018), una ingente colección de fotografías de chiuauas y muffins que se utiliza en la determinación del rendimiento de nuevos algoritmos de *Machine Learning* de visión computacional, es decir, de identificación de objetos en imágenes. Los algoritmos compiten por diferenciar chiuauas y muffins en condiciones, digamos, extremas (figura II.5.1).

Figura II.5.2: *Iris sibirica*, una de las muchas especies del género *Iris*. Fuente: Wikipedia, `https://en.wikipedia.org/wiki/Iris_(plant)`

flores, por tanto, se dan en una enorme combinación de formas y colores. El set de datos recoge los datos morfológicos de las flores de 3 especies distintas, con 50 muestras cada una. Las variables son:

1. longitud del sépalo;
2. anchura del sépalo;
3. longitud del pétalo;
4. anchura del pétalo;
5. especie a la que pertenece el individuo: *Iris setosa*, *Iris versicolor* o *Iris virginica*.

El último dato, obviamente, es de control. Los importantes para la clasificación son los 4 primeros. El objetivo del algoritmo de clasificación, por lo tanto, es determinar la especie de un nuevo individuo no identificado en función de sus características.

Hay dos formas de abordar este proceso, como ya se explicó anteriormente:

1. **supervisado:** previo a la clasificación, el algoritmo aprende las características claves de la morfología de cada especie gracias a que se le proporciona, del conjunto de 50 individuos de cada una, una pequeña muestra[3]. El algoritmo aprenderá a distinguir las tres especies a partir del estudio de las características de cada una y de esta manera será capaz de determinar a qué especie pertenece un nuevo individuo;

[3]El llamado *set de entrenamiento*.

2. **no supervisado:** sin aprendizaje previo, el algoritmo procesa todos los datos o parte de los mismos e intentará crear N grupos con características diferenciadoras a partir de ellos. Esta clasificación será la base del aprendizaje, ya que al intentar inferir a qué grupo pertenece un nuevo individuo se le asignará la clase con la que tenga más afinidad de las encontradas durante el proceso de clasificación.

Los algoritmos, para procesar la información, precisan de un vector de datos. En este caso, el vector sería:

```
[ longitud sépalo, anchura sépalo, longitud pétalo, anchura pétalo ]
```

El orden de los componentes no es generalmente relevante, pero, obviamente, este orden ha de ser respetado en todos los vectores suministrados al algoritmo.

II.5.2. Algoritmos de *Machine Learning* seleccionados

La aplicación de *Machine Learning* que resulta a priori más interesante, ya que ayuda a la interpretación y exploración de datos, es la de clasificación, como la descrita en el apartado anterior. Gracias a esta familia de algoritmos, se podrá someter a las teselas (nuestros individuos) a clasificaciones multivariantes gracias a la habilidad de la plataforma de encontrar aquellas que tienen todas las variables de adscripción sobre las que se quiere basar la clasificación. La aplicación generará, para cada tesela, un vector que será suministrado al algoritmo escogido.

Hay que tener en cuenta que los algoritmos, usualmente[4], requieren que estén presentes todas las variables en el vector, es decir, no es posible computar el algoritmo con vectores a los que les falten componentes, como por ejemplo un componente a nulo o *NODATA*. En el ejemplo anterior, si desconociéramos uno de los parámetros morfológicos de un individuo del género *Iris*, no podríamos someterlo a clasificación con muchos de estos algoritmos.

Los algoritmos escogidos son:

1. para clasificaciones no supervisadas, el ***K-Means*** (Likas et al., 2003);

2. para clasificaciones supervisadas, el ***Random forest*** (Breiman, 2001).

Algoritmo de clasificación no supervisada: *K-Means*

El *K-Means* es un algoritmo de clasificación no supervisada que realiza la clasificación de los vectores objetivo en un espacio multidimensional. El objetivo es estudiar la nube de puntos en dicho espacio multidimensional y encontrar grupos de puntos cercanos entre sí.

[4] Algunos algoritmos de *ML*, como las redes neuronales, son especialmente buenas trabajando con información parcialmente incompleta. El estudio y aplicación de este tipo de algoritmos constituye un subgrupo en el campo de la IA llamado *Deep Learning*.

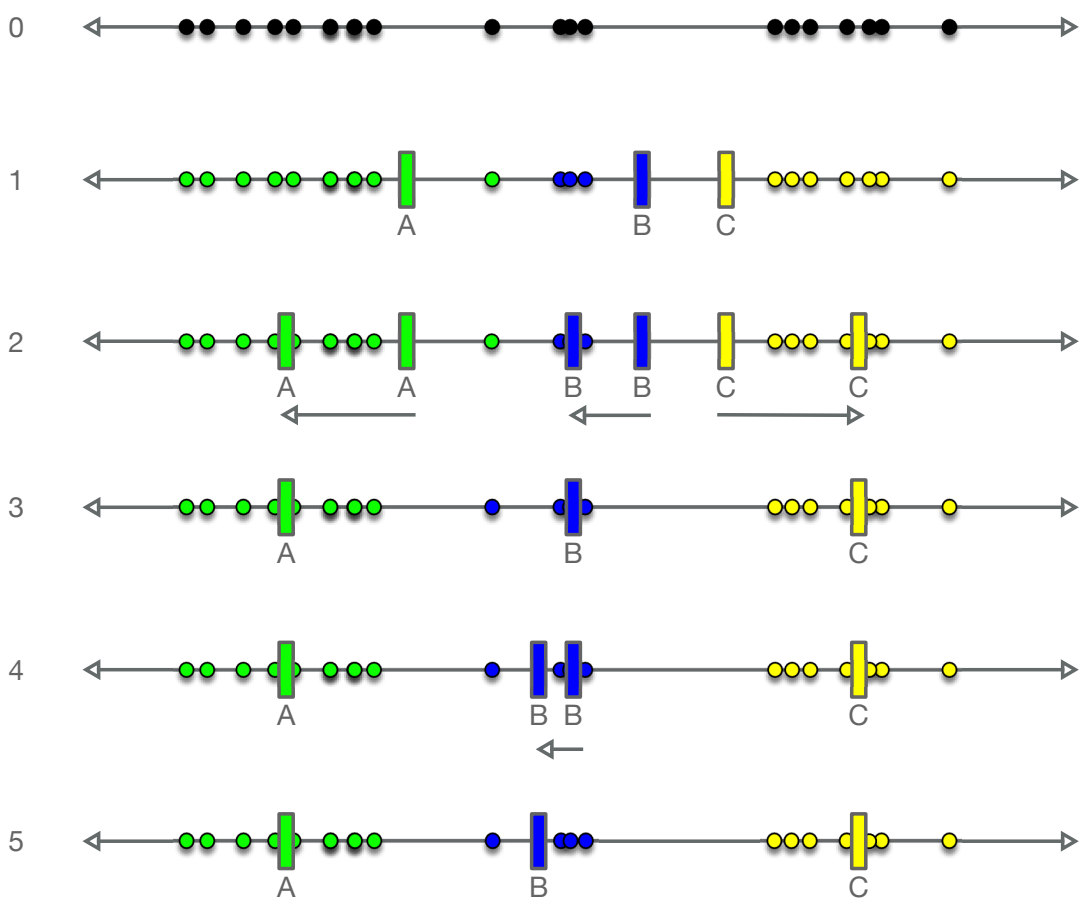

Figura II.5.3: Pasos de una clasificación *Jenks*, es decir, un *K-Means* unidimensional. En el paso 0, se distribuyen los valores del conjunto de datos en el espacio dimensional adecuado (en este caso, una línea, ordenados de menor a mayor). En el paso 1, aleatoriamente, se generan en el rango de la variable los N núcleos de agrupación deseados, en este caso 3. Se calcula, para cada valor, la distancia a cada uno de ellos, siendo asignado cada valor de la variable al núcleo más cercano. Esta es la primera propuesta de agrupación, no óptima. En el paso 2 se calculan los centros de gravedad (media geométrica, en este caso unidimensional, la media aritmética) partir de los puntos que pertenecen a cada grupo. El núcleo del grupo se desplaza a la media geométrica. En el paso 3, se vuelven a recalcular la pertenencia de los valores a cada uno de los núcleos, reorganizándose los grupos. Obsérvese como un punto del ejemplo cambia de grupo. En el paso 4 se vuelve a repetir el proceso del cálculo del centro de gravedad. En el paso 5, los núcleos ya son estables y ya no se mueven, por lo que se termina el proceso. Para inferir la pertenencia a un grupo según estos datos de entrenamiento tan sólo hay que ver qué núcleo tiene más cercano. Fuente: elaboración propia.

Para explicarlo, comencemos con un simple ejemplo unidimensional: el conocido en Geografía como *método Jenks*[5] (J. Chen et al., 2013). El *Jenks* no es más que la aplicación del *K-Means* en una sólo dimensión, es decir, con vectores de datos de una sola componente.

Los pasos para hacer una clasificación *Jenks* (y, por tanto, de un *K-Means*) son (figura

[5]El método *Jenks* se utiliza en semiología cartográfica para encontrar *clusters* de valores cercanos y obtener así una clasificación de los datos más cercana a sus divisiones naturales.

II.5.3):

1. distribuir los datos en un espacio de dimensiones equivalentes al vector: es decir, en el caso de *Jenks*, unidimensional, una línea;

2. distribuir aleatoriamente tantos núcleos de clase (*cluster cores*) como clases se quieran extraer de los datos a lo largo de la extensión dimensional de la variable, es decir, en este caso, entre el valor máximo y mínimo de la variable;

3. calcula qué puntos corresponden a cada núcleo en función de su cercanía;

4. dentro del conjunto de puntos asignados por cercanía a cada núcleo, calcular el centro de gravedad de los mismos, es decir, el punto que minimiza la media de distancia a cada punto del conjunto. Mover el núcleo de cada clase a dicho centro de gravedad;

5. repetir el punto 3: volver a calcular qué núcleo tiene más cerca cada punto;

6. volver a repetir el punto 4: recalcular el centro de gravedad y mover el núcleo al mismo;

7. repetir la secuencia de los pasos 3 y 4 hasta que ningún punto cambie ya de núcleo que tiene más cercano o los núcleos se muevan tan poco que se consideren como estables. Es lo que se llama la *convergencia* de los núcleos y, por tanto, del algoritmo. El estudio de estos procesos de convergencia en este tipo de algoritmo iterativos es de suma importancia, así como el tiempo en el que tardan en hacerlo y si este tiempo se alarga lineal, exponencial o logaritmicamente a medida que el número de datos aumenta. En ese momento se da por terminado el proceso y los núcleos son los que determinan las clases.

¿Cómo inferimos en qué clase va a caer un nuevo dato? Pues simplemente viendo qué núcleo tiene más cercano. Si el set de entrenamiento, es decir, los datos originales con los que se calcularon los núcleos, son lo suficientemente representativos y numerosos, la inferencia será de calidad. Si son pocos y poco representativos, entonces no será buena.

El *K-Means* hace el proceso anterior pero en un espacio multidimensional: con vectores de dos componentes, en un plano, con tres, en un espacio de tres dimensiones, con 20... en un espacio de 20 dimensiones[6]. En el ejemplo de los datos de las *Iris*, sería una espacio de 4 dimensiones. La disposición cuatridimensional de los puntos que caracterizan las propiedades morfológicas de las flores en este espacio forma cúmulos de puntos con aquellos que pertenecen a la misma especie. La identificación de los centros de gravedad de esos cúmulos constituye los núcleos de la clasificación.

El *K-Means* es un método sencillo pero no exento de problemas, por supuesto. Este tipo de algoritmos de clasificación tendrán mejores o peores resultados dependiendo de la distribución de los datos originales. Algunos son especialmente eficaces adaptándose a unas distribuciones u otras, como se puede observar en la figura II.5.4. En ella, el *K-Means*

[6] Aunque no podamos imaginar ni darle expresión a un espacio de 20 dimensiones, la matemática es exactamente la misma que para 1.

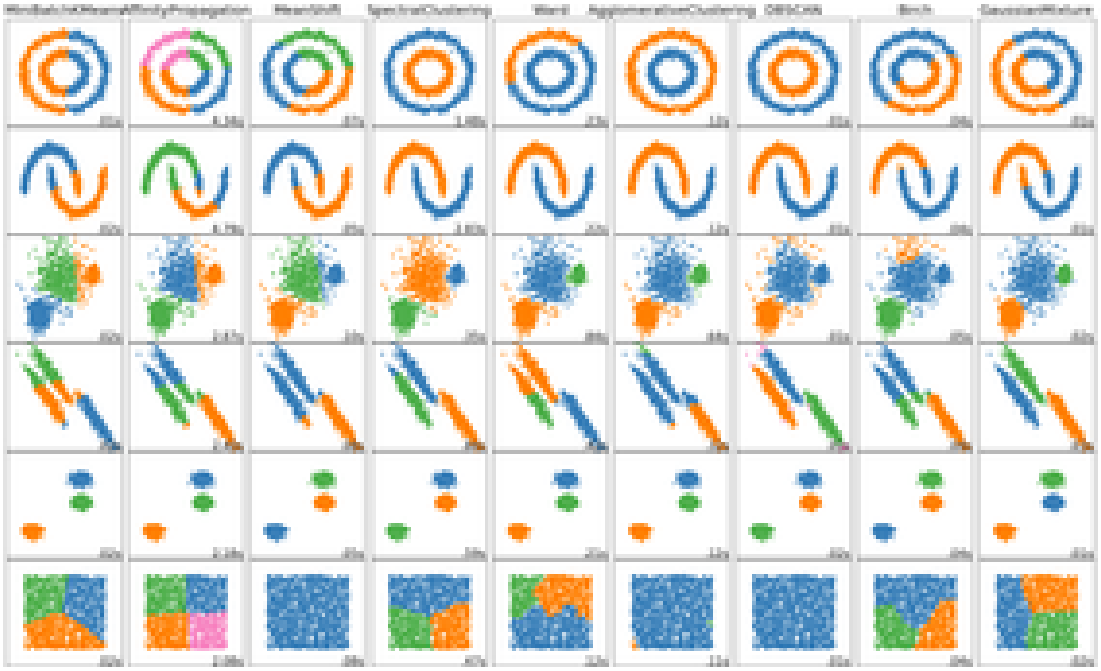

Figura II.5.4: Ejemplos de ajuste a distintos tipos de distribución de diferentes algoritmos de *Machine Learning* para *clustering*. El *K-Means* es la primera columna. Fuente: documentación de *Scikit-Learn* (Pedregosa et al., 2011) (`https://scikit-learn.org/stable/auto_examples/cluster/plot_cluster_comparison.html`).

es la primera columna: podemos observar que ofrece buenos resultados en distribuciones uniformes (tercera y sexta distribución), excelentes en distribuciones con grupos muy compactos (quinta distribución), pero no demasiado brillantes en distribuciones más concretas. Otros clasificadores arrojan mejores resultados según qué distribuciones.

También hay que tener en cuenta que, a priori, no siempre hay forma de saber cuál es el número óptimo de clasificadores. En el caso de los *Iris*, sabemos de partida que hay tres clases, por lo que la clasificación se hará teniendo en cuenta tres núcleos. En el caso de no saber o poder visualizar a priori (por ejemplo, por tener un gran número de dimensiones) cual es el número adecuado de núcleos iniciales, no queda más remedio que probar con un número creciente de los mismos. Entonces puede comenzar a suceder dos fenómenos que comparten todos los algoritmos de *Machine Learning*:

1. si el número de núcleos es demasiado bajo, se produce un fenómeno de excesiva generalización y las clasificaciones e inferencias de clases de nuevos individuos no serán muy significativas. Estaremos perdiendo detalles y matices presentes en la distribución de los datos;

2. por otro lado, si el número de núcleos es demasiado alto, se produce un fenómeno conocido en *Machine Learning* e inteligencia artificial como *overfitting*[7]: la clasificación es demasiado específica y el modelo de inferencia está, por lo tanto, desvirtuado

[7] *Sobreajuste* del modelo a los datos.

hacia la hiperespecialización, algo que tampoco es útil. El caso extremo es tener tantos núcleos como población en el set de datos: cada dato tendrá su propio nucleo y la clasificación carecerá de sentido.

El número óptimo de núcleos, por tanto, puede ser inferible mediante métricas de ajuste de los núcleos:

1. una desviación típica muy alta en la distancia de los puntos del set de aprendizaje al núcleo indicará un exceso de generalización: en ese núcleo están entrando puntos tanto muy cercanos como muy lejanos;

2. una desviación típica muy baja en la misma distancia puede ser un signo de sobreajuste del modelo, teniendo demasiadas clases para la distribución.

Por tanto, el ajuste y la bondad del modelo predictivo de un *K-Means* (y de cualquier otro algoritmo de *ML*) sólo se puede evaluar con una profunda revisión estadística del ajuste tanto de la población de entrenamiento[8] como de las pruebas de clasificación con individuos (vectores) no pertenecientes al mismo. Y por supuesto, con grandes dosis de sentido común y conocimiento temático[9].

Algoritmo de clasificación supervisada: Random forests

Como ejemplo de algoritmo de clasificación supervisada se ha seleccionado el llamado *Random forests* (Breiman, 2001). El *Random forests*, también llamado *árboles de decisión aleatorios*, es un método de *ML* que clasifica vectores en base a un entrenamiento previo en el que el modelo se entrena proporcionándole poblaciones de vectores de clases bien identificadas.

El *Random forests* utiliza internamente otro algoritmo de *ML* muy utilizado llamado *Decision tree* (árboles de decisión). En un árbol de decisión, que es también un método supervisado, se analizan las distintas componentes del vector (variables) para determinar su influencia en que dicho vector caiga en una clase u otra, creando un árbol de decisión que va testeando el valor de cada componente frente a unos valores umbral, ramificando las posibilidades hasta alcanzar una clase u otra[10]. Las variables que no son relevantes para la clasificación son obviadas por el árbol de decisión.

El problema que tienen los árboles de decisión es que son extraordinariamente propensos al *overfitting*: explican muy bien el juego de datos de entrenamiento, pero no

[8] Aunque el *K-Means* es un método no supervisado, se considera que tiene un set de entrenamiento cuando se toma una muestra aleatoria del conjunto de los datos para calcular los núcleos de agrupación. Éstos datos entrenan el modelo y los núcleos resultantes son utilizados para clasificar al total de los datos.

[9] Siempre hay que evitar la tentación de utilizar la fuerza bruta de las máquinas para explorar relaciones espúreas, por mucho que pueda parecer un ejercicio inocente. El peligro es llegar, a falta de conocimiento temático, a conclusiones falsas, y, desde luego, en entornos corporativos, a desperdiciar el dinero. Por ejemplo, una máquina para *Machine Learning* en la infraestructura *Cloud* de *Amazon* no baja de los 1700€ mensuales de pago por uso. Este pago por uso de infraestructura *hardware* es lo que se conoce como *IaaS* (*Infrastructure as a Service*).

[10] El set de entrenamiento clásico de los árboles de desición se basa en la lista de supervivientes y víctimas del hundimiento del *Titanic* (figura II.5.5).

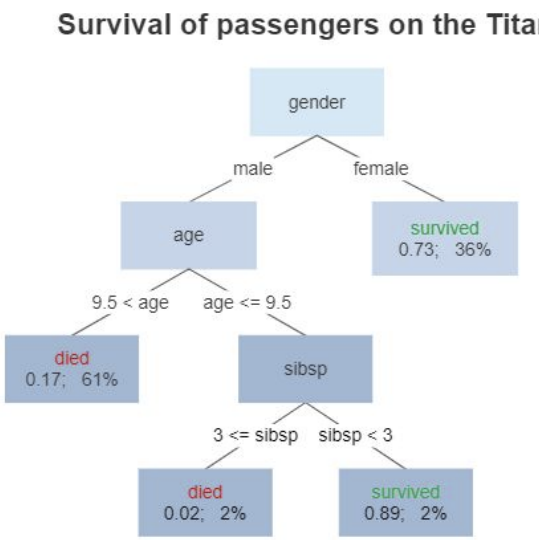

Figura II.5.5: El árbol de decisión generado tras el análisis de los datos de supervivientes del *Titanic*. El análisis determina que las probabilidades de supervivencia eran buenas si se era mujer o un hombre de menos de 9,5 años con menos de tres familiares a bordo. Fuente: Wikipedia, `https://en.wikipedia.org/wiki/Decision_tree_learning`.

son capaces de proporcionar buenas predicciones frente a nuevos datos. Para evitar esta propensión al *overfitting*, *Random forests*[11] crea, a partir de los datos de entrenamiento, varios árboles que modelizan los datos desde distintas aproximaciones, y que trabajan en conjunto para clasificar nuevos datos. Por ello, en muchas clasificaciones de algoritmos de *Machine Learning* se le coloca en la categoría de *ensemble algorithms*, por esa propiedad de generar varios modelos que permiten llegar a una clasificación aplicados conjuntamente. Cuando un nuevo vector es sometido a la clasificación, es clasificado por cada uno de dichos árboles, que han sido generados desde distintas perspectivas y con distintas parametrizaciones a partir de los mismos datos (y, que, por lo tanto, no están totalmente correlacionados, esta es la clave), obteniéndose un resultado de clasificación por cada uno de ellos. La categoría predominante en las predicciones individuales de cada árbol gana. Esta forma de aprendizaje por mayorías se puede ver en muchas otras técnicas de *ML*, como por ejemplo los algoritmos genéticos. El hecho de que la predicción salga de varios modelos no fuertemente correlacionados hace que los efectos de *overfitting* que puedan ocurrir en algunos árboles sean cancelado por otros.

El caso del juego de datos de los *Iris* se ajusta muy bien a este algoritmo supervisado. Las clases de entrenamiento son los conjuntos de datos que se sabe pertenecen a cada especie, y dichos conjuntos son proporcionados al algoritmo para su entrenamiento. El bosque de árboles de decisión generado se guarda y es utilizado para aplicarlos a cualquier nuevo vector de datos (los datos morfológicos de una nueva flor) para inferir la pertenencia a una especie u otra.

[11] Curiosamente, el término *Random forests* es una marca registrada.

II.5.3. Uso de los algoritmos de *ML* sobre la rejilla

La rejilla, al adscribir información con estructuras de datos muy diversas en origen, consigue el efecto de uniformizarlas en una única estructura de datos tanto a nivel alfanumérico como a nivel geométrico. A nivel geométrico, las teselas proporcionan una unidad geométrica base homologable a todas las unidades de información. En el nivel alfanumérico, toda la información adscrita ha quedado reducida a una estructura de variables escalares en forma de vector asimétrico, es decir, que cada tesela tiene su propio vector de datos con más o menos variables.

A partir de aquí, y según todo lo descrito, la aplicación de algoritmos de *ML* a las teselas viene de forma natural. La plataforma puede rastrear la presencia de un subvector de variables de adscripción determinado en el total del vector de cada tesela[12]. Estos vectores son el formato de datos nativo de los algoritmos de *ML*.

El usuario, por tanto, sólo tendría que seleccionar el conjunto de variables de adscripción que considera significativas para la clasificación a realizar y ejecutar el procedimiento. Por ejemplo, se podría clasificar en base a la distribución la pirámide de población, con lo que podríamos obtener un conjunto de clases representativo de la estructura de la misma en Andalucía. Dado que los datos están referidos a unidades geométricas (las teselas) homologables entre sí, no hay que hacer ninguna transformación a los datos, aunque sí es verdad que en muchos escenarios es necesario realizar primero un proceso de normalización a una escala (0, 1) o (-1, 1) para eliminar la influencia de los valores absolutos.

En el caso del *K-Means*, la aplicación es directa, ya que sólo hay que seleccionar una zona de estudio, designar el número de clases (*clusters*) a calcular y lanzar el algoritmo. Dicho algoritmo asignaría una clase a cada tesela a partir del análisis multivariante. Este tipo de análisis nos permitiría encontrar patrones geográficos multivariantes en un área de estudio, diferenciando zonas de características similares.

En el caso del *Random forests*, el usuario de la plataforma tendría primero que proporcionar regiones de entrenamiento de cada clase a discriminar. La plataforma extraería de las zona de entrenamiento los vectores de datos que las caracterizan y entrenaría el modelo de árboles con ellos. A partir de ahí, cuando se le proporcione al modelo el vector de cualquier otra tesela, esta será clasificada en una de las clases aprendidas. Este tipo de análisis nos permitiría encontrar las teselas que se parecen, para un área que no fuera de entrenamiento, a una clase u otra, encontrando ámbitos territoriales semejantes a las zonas de interés.

[12] Recordemos que estos algoritmos no pueden funcionar con información incompleta.

II.6 Mecanismos de difusión

En este capítulo vamos a examinar qué mecanismos de difusión de la información en rejilla nos proporciona la plataforma. Básicamente, son dos:

1. **uno orientado al usuario final:** visualizaciones de cartografía y datos en un entorno *web*;

2. **otro orientado al usuario con un perfil más técnico:** una *Application Programming Interface* (*API*).

II.6.1. Visualizaciones de datos en entorno *web*

Las visualizaciones de cartografía en entornos *web* son herramientas comunes muy presentes en *Internet* hoy en día (Álvarez Francoso, 2016). Los visores cartográficos son cada vez más potentes y cada vez hacen más uso de las enormes capacidades de las máquinas clientes[1] en términos de potencia de cálculo y gráfica (Álvarez Francoso, Camarillo Naranjo et al., 2014), llegando a usar los más avanzados incluso la tarjeta gráfica 3D de los ordenadores para crear aplicaciones cartográficas realmente sorprendentes[2].

Las primeras técnicas de visores cartográficos (Dragićević, 2004), en los tiempos anteriores a la revolución en las aplicaciones *web* clientes que supuso la llegada del estándar *HTML5* (Anthes, 2012), se basaban en el dibujado[3] de los mapas en la máquina servidora, es decir, era el servidor el que dibujaba el mapa que después era enviado a la máquina cliente para ser mostrado en una página *web*. Cada vez que el usuario interaccionaba con el mapa, éste tenía que volver a ser dibujado (un proceso ciertamente costoso en computación) por el servidor y ser servido de nuevo al cliente. El estándar *Web Map Service* del *OGC* (Sayar et al., 2005) es un buen ejemplo de esta forma de proceder.

[1] Recordemos que el término *cliente* en ingeniería de *software* hace referencia a otra máquina y/o programa que utiliza los servicios de otra máquina y/o programa, en este caso llamada *servidor*. En aplicaciones *web*, el *servidor* es el conjunto de máquina/programa que sirve los datos y el *cliente* la aplicación *web* que es cargada en nuestro navegador *web* (*programa cliente*) y que se ejecuta en nuestro ordenador de trabajo (máquina cliente).

[2] Sirva como pequeño ejemplo lo que el visor de la empresa *Mapbox*, *Mapbox GL*, es capaz de hacer: `https://docs.mapbox.com/mapbox-gl-js/example/`.

[3] En TIG, es muy común utilizar la palabra inglesa *render* (dibujado), adaptada al castellano en el verbo *renderizar*, para hacer referencia a una imagen generada por ordenador. Es un uso un poco cuestionable del lenguaje pero está muy generalizado.

Obviamente, con el aumento del volumen de uso de la cartografía en *Internet*, este método es inviable, por razones operativas, en aplicaciones de gran volumen y/o usuarios. Es inconcebible que *Google Maps* hubiese funcionado nunca como lo hace en el caso de que a cada cliente se le dibujara un mapa personalizado para cada pequeña interacción que efectuara sobre él. Las necesidades de computación para un servicio como ese, diseñado de esa manera, serían inasumibles. Por lo tanto, *Google* pasó a una estrategia intermedia: mientras esperamos que los clientes sean lo sufientemente sofisticados (Miller, 2006) para que dibujen los mapas por ellos mismos, lo que se va a hacer es predibujar el mapa a diferentes escalas, y teselados, y guardar esas teselas predibujadas que, servidas a un cliente, forman un mapa completo cuando se componen cual puzle se tratara. Más o menos la misma estructura de datos que se plantea en esta tesis, pero con pequeños fragmentos de mapa predibujados en lugar de vectores de datos alfanuméricos.

Sin embargo, tras la llegada del estándar *HTML5*, la *web* cambia. El impacto de este estándar de las tecnologías *web* es tan importante que permite hacer a nivel gráfico en una página *web* cosas que antes eran simplemente inviables sin recurrir a *plugins* o elementos externos al navegador[4]. Gracias a este estándar, ahora los desarrolladores *web* pueden recurrir a marcos de desarrollo perfectamente estándares como el *canvas*, que permite control de dibujado pixel a pixel en una región de la página *web*, o a *WebGL*, que permite el dibujado de gráficos en tres dimensiones con un gran rendimiento gracias al acceso a las capacidades de las modernas tarjetas gráficas.

Con este nuevo arsenal de posibilidades y herramientas proporcionados por el estándar *HTML5*, los desarrolladores pueden diseñar aplicaciones que hacen algo muy interesante: aprovechar al máximo las máquinas de los clientes. La lógica dice que, si por ejemplo una empresa como *Amazon* va a cobrar el tiempo de computación en sus máquinas, donde se alojan los servidores de las empresas, cuanto menos trabajen éstos por delegación de carga de trabajo a los clientes, más rentable será todo el proceso. Por lo tanto, los clientes *web* (los navegadores) han trascendido su objetivo de diseño original de ser clientes relativamente ligeros y muy dependientes de los servidores para convertirse en ordenadores dentro de ordenadores capaces de llevar a cabo muchas tareas, incluidas tareas de cálculo más o menos pesadas[5]. Una de las bases del diseño de sistemas *web* modernos es que por la red deben circular paquetes de datos lo más pequeños posible que alimenten a potentes aplicaciones cliente en navegadores que se encarguen integramente de la visualización de los mismos, sin depender para estas tareas de programas alojados en servidores. El objetivo

[4]El conocido *Flash* de la compañía *Macromedia* (hoy propiedad de *Adobe*) era la tecnología para crear contenido multimedia más popular en la década de los 2000 (Reinhardt, 2007). Lo que hacía *Flash* era crear, dentro de una ventana en una página *web*, un área de visualización ajena a las características nativas del navegador con su propio lenguaje de programación incrustado (*ActionScript*), que permitía a diseñadores y desarrolladores crear contenido multimedia altamente dinámico e interactivo imposible por otros medios. La decisión de Steve Jobs (Jobs, 2010) de no permitir el funcionamiento de *Flash* en los dispositivos portátiles de *Apple* fue el empujón definitivo a la adopción del estándar *HTML5*, que permitiría a las páginas *web* hacer todo lo que se hacía con *Flash* (y más) de forma nativa (sin *plugins*).

[5]Tanto es así que la empresa *Google* tiene una gama de ordenadores portátiles ligeros, la serie *ChromeBook*, que ni siquiera tienen sistema operativo como solemos entenderlos normalmente: lo único que ejecutan es un navegador *web Google Chrome* con el que conectar a los múltiples servicios de la compañía y hacer un trabajo 100 % *online* sobre un navegador de altas prestaciones que realiza buena parte del trabajo localmente.

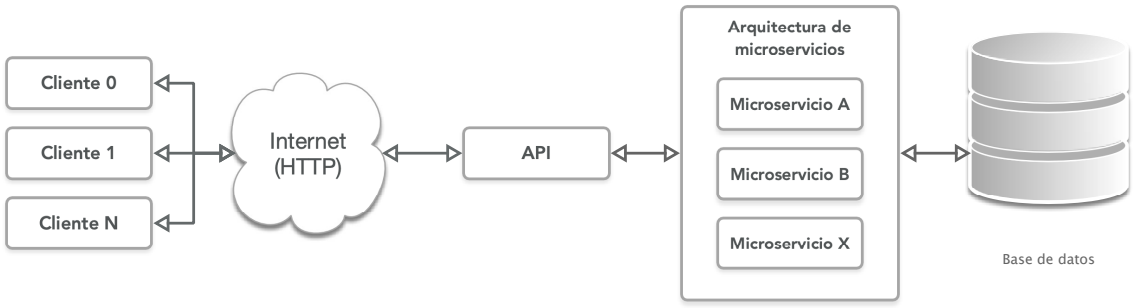

Figura II.6.1: Esquema simple de una aplicación cliente - servidor.

de éstos últimos, en una aplicación *web* moderna, debe ser proporcionar exclusivamente datos[6].

De esta manera, el proceso seguido por las aplicaciones cartográficas *web* se puede resumir como:

1. una primera etapa en la que el servidor tenía que estár al cargo del dibujado completo del mapa para mandárselo, una vez compuesto, al cliente, por carecer éste de capacidades de dibujado propias;

2. una etapa intermedia en la que dichos mapas, muy costosos de procesar de forma exclusiva para cada cliente que lo solicitara, eran pregenerados por servidores de dibujado y guardados en teselas cartográficas que era lo que se enviaba al cliente para ensamblar un mapa completo. Las capacidades de los clientes seguían sin ser muy potentes pero esto ahorraba importantes recursos de computación en la parte servidora;

3. la etapa *HTML5 / WebGL*, donde el navegador ha sido dotado de extraordinarias capacidades gráficas que le permiten de forma autónoma dibujar por si mismo los mapas a partir de datos vectoriales muy optimizados para su transporte por la red.

En esta última etapa *HTML5*, lo único que viaja entre el cliente y el servidor son los datos vectoriales que sirven para dibujar el mapa. Gracias a estas nuevas técnicas del estándar *HTML5*, se ha desarrollado un nuevo estándar en TIG, las *vector tiles* (teselas vectoriales), que son herederas de las teselas de fragmentos de mapa pregenerados de la segunda etapa, pero codificando información vectorial en lugar de imágenes (Netek et al., 2020).

Uno de los principios de diseño y desarrollo de plataformas *web* es reducir al límite el tránsito de datos en la red. El tráfico genera gastos a las empresas, por lo que es prioritario desarrollar estrategias que lo minimicen.

Hemos de entender el funcionamiento de una aplicación *web* como un intercambio de información entre una máquina servidora y una máquina cliente (figura II.6.1). El servidor

[6]Los *frameworks* de desarrollo *web* modernos, como los populares *React*, *Angular* o *Vue*, proporcionan unos marcos de diseño que llevan esta filosofía al extremo.

tiene una relación directa con el almacén de datos (usualmente, una base de datos de un tipo u otro), que guarda lo más valioso de la aplicación y que debe estar bien protegida y a salvo del contacto exterior. El servidor es el controlador de lo que sucede en la parte servidora de la aplicación[7], controlando el acceso a los datos. Los datos del mapa residen ahí, en el almacén de datos.

Imaginemos un mapa vectorial con el catastro de toda Andalucía. Esto son varios gigabytes de datos. Imaginemos ahora que queremos crear, con las nuevas tecnologías *HTML5*, un visor cartográfico que dibuje el mapa vectorial en el cliente. Para ello, el servidor debería mandar varios gigas de datos al cliente. Esto, obviamente, es absurdo. Primero porque tardaría mucho, segundo porque seguramente el usuario no está interesado en el 99 % de los datos que le acaban de llegar y no les va a dar ninguna utilidad, principalmente porque el nivel nativo de resolución vectorial de los mismos excedería con mucho la resolución de la propia pantalla del ordenador y su representación gráfica sería, por lo tanto un desperdicio de capacidad de cálculo.

Por lo tanto, hay que racionalizar este proceso de envío de información. Como ya se ha comentado, la solución es muy similar a la rejilla multiescalar descrita en este trabajo. Por un lado, el mapa se organiza en distintos niveles de escalas significativas. De esta manera, a una escala autonómica, no veríamos las parcelas catastrales, sino quizás las grandes divisiones administrativas del territorio, provincias, comarcas y principales núcleos de población. De esta forma, hemos reducido drásticamente el volumen de datos que va a viajar por la *web*: los datos vectoriales de estos elementos de información no son gran cosa, y le dan al usuario la información de contexto adecuada para ponerse en situación[8]. A medida que el usuario vaya interaccionando con el mapa se le va sirviendo información cada vez más territorialmente contextualizada en escala significativa, pero cada vez más rica en geometrías y elementos territoriales cartografiados: lo que ganamos de peso en datos por hacer la escala más grande lo perdemos porque la información cartográfica se densifica, por lo que el flujo de información tenderá a subir pero no mucho.

De esta manera, se consigue llegar a un equilibrio en el tránsito de información, que constituye uno de los puntales del diseño de aplicaciones *web*: darle al cliente la información que precisa en cada momento para ser autónomo durante unos minutos o segundos y no tener que recurrir constantemente al servidor para pedir más datos. Por ejemplo, en el caso del catálogo de una tienda, la aplicación *web* está obteniendo del servidor la información del producto que actualmente está consultando el usuario, pero seguramente, en el trasfondo, está también obteniendo la información de los 5 siguientes productos en la lista o de los 3 primeros recomendados para que cuando el usuario realice la próxima interacción más problable[9], que es pasar al próximo producto o pinchar en una de las tres primeras recomendaciones, la aplicación reaccione al instante.

[7] En ingeniería de *software*, esta parte de la aplicación suele recibir el nombre de *backend*.

[8] En realidad, si se piensa, hay una relación lógica y directa entre diseño cartográfico, la capacidad de percepción y asimilación de información humana y el volumen de datos generados por la máquina: un mapa con todo el catastro a nivel autonómico es ilegible, imperceptible e inútil para el usuario, cartográfica y semiológicamente disparatado y encima genera muchísimo tráfico de red mandar los gigas que pesa.

[9] El estudio de la interacción de un usuario con una página *web* u otro producto tecnológico es una categoría profesional propia dentro del mundo del diseño que se conoce como *diseño de UX*, siendo *UX* las siglas de *User eXperience*, *expeciencia de usuario*.

Con los mapas pasa lo mismo: las siguientes acciones más probables son hacer un pequeño zoom arriba o abajo o un desplazamiento. Por ello, el visor carga la información de las zonas adyacentes dentro de la misma escala, o de la misma zona a niveles de escala ligeramente por arriba y por abajo. Este balance de mandar la información precisa para hacer hasta cierto punto autónomo al cliente para satisfacer las necesidades de información e interacción inmediatas del usuario, pero sin pasarse de datos, es uno de los objetivos principales de los estudios de diseño de *UX*[10].

Por lo tanto, los mapas modernos vectoriales utilizan este estándar de teselas vectoriales en el que la información es transmitida al cliente muy poco a poco, según las interacciones que este realiza, e intentando, dentro de lo posible, adelantarse a las futuras interacciones del mismo. El peso de una tesela vectorial es muy pequeño, y la velocidad a la que un navegador moderno las dibuja más que adecuado.

La plataforma propuesta, por tanto, sigue también este modelo. Aquí es donde el hecho de que la rejilla, en la estructura de datos propuesta, tenga una definición estrictamente matemática y no necesite ser formalizada en un dato geométrico, vuelve a mostrar sus ventajes.

Recordemos que la rejilla viene definida por un origen de coordenadas, un sistema de proyección cartográfico y una serie de resoluciones jerarquizadas. La definición de esta información ocupa muy poco espacio. Por tanto, para trasladar desde un *backend* información de teselas, la parte geométrica se puede obviar, puesto que lo único que hay que hacer es enviar a través de la red un paquete de datos con los siguientes elementos:

1. **definición de la rejilla:** como se ha dicho, no ocupa prácticamente nada;

2. **coordenadas de las teselas a tratar en el cliente:** recordemos que las teselas vienen definidas por sus coordenadas [nivel de resolución, X, Y]. Aunque se pidan muchas teselas, este segmento de información no suele ser muy grande[11];

3. **variables de adscripción seleccionadas para las teselas:** para cada una de ellas se incluye el vector temático compuesto por una o varias variables de adscripción

[10] Se dan casos realmente curiosos relacionados con este campo. Una de las empresas que más recursos ha invertido históricamente en este campo, y con resultados más que notables, es *Apple*. Los ingenieros de UX llegaron a la conclusión de que el límite de la frustración humana frente a una pantalla en blanco de teléfono móvil era ridiculamente bajo, de menos de un segundo. Si el móvil se quedaba sin responder, por ejemplo refrescando la pantalla a la hora de cambiar de aplicación, la percepción de la experiencia de uso de los probadores caía en picado. Como los modelos de la época eran incapaces de cambiar entre aplicaciones, por lo general, tan rápidamente, decidieron que lo que iban a hacer era hacer una captura de pantalla de la aplicación en el momento que ésta era abandonada por el usuario para pasar a otra. Cuando el usuario volvía, se le presentaba dicha captura de pantalla, un conjunto de píxeles inertes incapaces de interaccionar con nada pero que satisfacía la frustración del usuario mientras el dispositivo hacía todo lo posible por tener la aplicación real funcionando en el trasfondo y mostrársela al usuario.

[11] Depende, por supuesto, de la extensión de la zona solicitada y la resolución deseada. Una extensión grande a mucha resolución ocupa mucho más espacio (más teselas) que una zona pequeña y/o un nivel de resolución más bajo. La plataforma puede, opcionalmente, estimar el tamaño y número de teselas involucradas en el paquete de datos para seleccionar una resolución más baja para hacer el paquete de datos más pequeño. En estos cálculos también hay que tener en cuenta el tamaño de los vectores de datos pedidos a las teselas.

seleccionadas por el usuario. Este segmento de datos puede ser grande dependiendo del número de teselas y/o del tamaño del vector temático, es decir, del número de variables de adscripción solicitadas.

Una vez este paquete de información se componga en el servidor, se mande por la red y sea recibido en el cliente *web*[12], éste tiene la capacidad, a partir de la definición de la rejilla, de transformar en geometrías las coordenadas de cada tesela para su representación gráfica en un visor. Además, su representación no tiene por qué circunscribirse a un polígono, que es la forma natural de la tesela, sino que puede representarse de distintas maneras. Por ejemplo, es trivial calcular el centro de cada una de ellas y representar dos magnitudes simultáneamente, una como color y otra como tamaño, o usar tres variables temáticas como componentes roja, verde y azul y componer un color *RGB*. Las posibilidades son múltiples.

II.6.2. *Application Programming Interface*

Para empezar, debemos entender el concepto de *interface*. La RAE recoge este término derivado del inglés como *interfaz*, y lo define como[13]:

1. f. Conexión o frontera común entre dos aparatos o sistemas independientes.

2. f. Inform. Conexión, física o lógica, entre una computadora y el usuario, un dispositivo periférico o un enlace de comunicaciones.

Estas dos definiciones cubren perfectamente las dos acepciones que se van a discutir en este apartado. Extendiendo un poco la definición propuesta por la RAE, podríamos decir también que el interfaz, en su segunda acepción, no sólo se refiere a la interacción entre una computadora y su usuario, sino a la interacción entre un usuario y cualquier máquina. El interfaz de un coche son el volante, la palanca de cambios, los pedales, etc.

La ingeniería de *software* aplica los procesos estándar de la práctica general de la ingeniería, como no podía ser de otra manera. Entre los principios de la ingeniería se encuentran la capacidad de poder mantener un sistema, de garantizar la integridad del mismo y la integración externa de dicho sistema con otros.

Consideremos por ejemplo la construcción de una central energética termosolar. Estas plantas energéticas utilizan espejos para concentrar la luz solar en lo alto de una torre, donde se ubica un componente esencial de la misma, el receptor, preparado para aguantar enormes temperaturas a la par que permite la conducción en su interior de sales que se fusionan y se calientan a temperaturas de aproximadamente 800°C. Estas sales en estado de fusión descienden de la torre y se guardan en enormes tanques de almacenamiento.

[12] En ingeniería de *software*, los clientes *web* suelen ser conocidos como *frontends*, en contraposición a los servidores, que son conocidos como *backends*.

[13] `https://dle.rae.es/interfaz`

Esta reserva de calor después puede utilizarse en multitud de aplicaciones: desalación de agua, generación de vapor para la generación de energía, etc. Para generar electricidad se usan turbinas de vapor. El receptor, el tanque de almacenamiento y la turbina de vapor pueden ser fabricados por distintas compañías, pero a la entrega, todas tienen unas especificaciones técnicas que describen como se conectan unos elementos con otros (calibres de tuberías, válvulas, etc) y qué condiciones y rangos de operación soportan: presión, temperatura, tiempo de funcionamiento, etc. Cada componente ha sido fabricado por compañías distintas que no tienen por qué saber del trabajo de las demás, ya que es el promotor de la planta el que tiene la responsabilidad de elegir componentes que sean capaces de trabajar entre sí sin fundirse o dañarse los unos a los otros por exceder esos límites operativos. Por otra parte, el promotor[14] no tiene por qué saber todos los detalles de fabricación o desarrollo de la turbina, puesto que se va a limitar a utilizarla en un contexto determinado. Ni la va a mejorar, ni va a experimentar con ella. La va a usar siguiendo las indicaciones del fabricante con un fin bien definido.

El *software* se comporta igual. Los programas informáticos grandes se construyen a piezas, siendo estas piezas el trabajo de otras personas o equipos y pudiendo ser de una complejidad interna notable. Otros programadores toman estas piezas, estos fragmentos de código, y los conectan con otros para hacerlos funcionar conjuntamente con unos objetivos bien definidos. Es muy posible que un equipo de desarrollo haya escrito una proporción muy pequeña del programa final, ya que se han limitado a conectar con código propio estas piezas y hacerlas funcionar como un todo.

Estas piezas de *software* que se empaquetan para ser usadas por otros programadores reciben el nombre de *librerías*. El diseño de librerías contempla entre sus objetivos los principios de ingeniería descritos anteriormente:

- **mantenimiento:** si se detecta que la librería tiene fallos, los autores de la misma pueden arreglarlos, muy posiblemente sin necesidad de afectar a los sistemas que la utilizan más allá de reemplazar la versión defectuosa por la nueva, ya libre del error. Si el motor de un coche falla, se puede reparar independientemente del estado de la transmisión;

- **integridad:** la integridad de la librería está garantizada por sus autores puesto que en su diseño hay partes que están expuestas al exterior pero otras que no lo están. Si están bien diseñadas, esas partes críticas donde se producen los cálculos clave de la librería estarán a salvo de manipulaciones externas indebidas, y además, el funcionamiento complejo de esas partes internas de la librería no tiene por qué afectar al interfaz de la misma, es decir, a cómo ésta se comunica con el exterior. Podemos cambiarle el motor a un coche, pero no necesitamos cambiar el volante y los pedales;

- **integración externa:** los creadores de una librería no sólo crean esas partes internas que hacen los cálculos clave de la librería, sino que tienen que crear un sistema que permita a la misma comunicarse con el exterior de una forma unívoca, es decir,

[14] Obviamente, esto es una simplificación: los promotores tienen expertos que saben estas cosas.

exponer al exterior una serie de comandos y funciones bien definidos[15] que permiten a otros programadores interactuar con las partes privadas de la librería y solicitarle tareas de computación. El motor de un coche se diseña de forma que la fuerza motriz que proporciona pueda ser transferida a la transmisión, idealmente, del modo más estándar posible.

Por seguir con la analogía con la turbina, un ingeniero que la adquiere para montarla en una central no tiene que estar al tanto de todos los detalles de su funcionamiento, sino sólo de aquellos que incumben a su integración con el resto de la planta y a su operación. Si la turbina hace su trabajo gracias a una tecnología u otra es hasta cierto punto irrelevante. Por ejemplo, una librería como *PROJ* (PROJ Contributors, 2021), que hace reproyecciones cartográficas, con cálculos complejos, permite a un no especialista en geodesia realizar estos cálculos sin conocer sus detalles matemáticos[16]. Lo que debe saber un programador que quiera utilizar la librería es cómo integrarla en su programa y cómo interaccionar con su capa pública, con su interfaz, para solicitarle tal o cual operación. También debe saber, por supuesto, en qué forma dicha librería le va a entregar el resultado de sus cálculos.

El diseño y forma final que toma dicha capa pública de interfaz es lo que se conoce en ingeniería de *software* como una *API* (Ofoeda et al., 2019). La *API* de una librería es como los manuales de instalación y operación de la turbina.

En la documentación de una librería, por lo tanto, se definen las operaciones y formatos de datos de entrada y de salida la que llevan a realizar por nosotros un determinado cálculo sobre nuestros datos. Siguiendo con el ejemplo de la *PROJ*, por ejemplo, en la documentación de su *API*, podemos encontrar que tiene una función llamada *Transformer.from_crs*[17] que permite transformar una coordenada de un sistema geodésico a otro:

```
1  # Importación de la libreria PyProj en el programa
2  from pyproj import Transformer
3
4  # Creacion de un transformador de coordenadas desde
5  # WGS84 geograficas (EPSG 4326) a ETRS89 UTM Huso
6  # 30 Norte (EPSG 25830)
7  transformer = Transformer.from_crs(
8    "EPSG:4326", "EPSG:25830")
9
10 # Uso del transformador para transformar coordenadas
11 # WGS84 geograficas a ETRS89 UTM Huso 30 Norte
```

[15] Y sobre todo, bien documentados. Una librería sin documentación es inservible.

[16] Obviamente, sabiendo qué es una proyección cartográfica y para que sirve, sus tipos, etc., pero no exactamente toda la matemática que implica, por ejemplo, una reproyección.

[17] En realidad, aquí estamos simplificando para no tener que explicar un buen número de conceptos de programación que entorpecerían la discusión. Para empezar, *PROJ* es una librería escrita en el lenguaje *C* y *C++*. El código que se muestra a continuación pertenece a su *binding Python*, *pyproj* (pyproj Contributors, 2021) (un *binding* es una capa de programación que permite utilizar una librería escrita en un lenguaje en otro lenguaje distinto al nativo de la propia librería), que posee una sintáxis mucho más clara, y, técnicamente hablando, *Transformer* es una clase de la librería, que expone un método llamado *from_crs()*, que crea un objeto transformador, llamado *transformer*, que es el que posee el método *transform()*, que en última instancia realiza la transformación de coordenadas.

```
12 | transformer.transform(37.3891, -5.9845)
```

Código II.6.1: Código *pyproj* para la transformación de una coordenada.

La línea 1 importa la librería *pyproj* en el programa para poder ser utilizada por el programa cliente[18]. La línea 3 crea un objeto transformador desde el sistema de referencia elipsoidal WGS84 geográficas al sistema proyectado UTM 30 norte sobre el elipsoide ETRS89. Por último, se hace uso en la línea 3 de la función *transform()*, que recibe como parámetros la latitud y longitud de un punto en WGS84 geográficas para arrojar finalmente un resultado de (235778,52039099712, 4142218,730057209), que son sus transformadas en ETRS89 UTM30N.

Toda esta información de cómo hacer funcionar a la librería se ha extraído de la documentación de la *API* de la misma. En ella se describe tanto los diferentes objetos y funciones que proporciona la capa pública de la librería y, también muy importante, cuales son las estructuras de datos que estas funciones admiten. Por ejemplo, la función *from_crs()* precisa que se le proporcionen dos *parámetros*, primero el sistema de referencia de origen y después el de destino. De la misma manera, se describe en la documentación que el método *transform()* requiere dos o tres parámetros, correspondiéndose a las coordenadas en el sistema de origen definido con una coordenada altitudinal opcional, que en este ejemplo no está siendo utilizada. La documentación describe que el resultado de esa operación será una estructura (X, Y, [Z]) con las coordenadas transformadas. Nótese que la coordenada altitudinal Z en la respuesta del método aparecerá en función de que fuera proporcionada o no en su llamada.

Con esta documentación, el usuario de una librería puede incluirla en sus programas y utilizarla, sabiendo lo que hace, pero no necesariamente conociendo todos los detalles de cómo lo hace[19]. Esta capa pública de la librería, conjunto de funciones, estructuras de datos de entrada y salida y documentación, es lo que se denomina la *API* de la librería. Las *API* habilitan el principio de ingeniería de *software* de *modularidad* y *encapsulamiento*, es decir, la ya referida habilidad de poder usar un *software* sin conocer los detalles de su implementación.

API a nivel de servicios

Este concepto de *API* no sólo se ciñe a las librerías de *software*. El concepto de librería de *software* presta todas sus características al relativamente nuevo modelo de modularización de *software* en la era de *Internet*: los servicios *web* (Barros et al., 2006). En el mundo

[18] Al igual que en lo explicado en la relación entre cliente y servidor en una aplicación *web*, cuando un programa utiliza una librería también se le denomina *programa cliente* (de la librería).

[19] Hay que indicar que, en muchos casos, este proceso de integración está lejos de ser trivial. Existen librerías que tratan problemas muy complejos, y aún la capa pública, sin entrar en sus detalles internos, ya son extraordinariamente complejas de entender, solo aptas, lógicamente, para especialistas en la materia. Hay librerías que exponen una gran cantidad de funcionalidad y ésta ha de ser ordenada de acuerdo a una arquitectura de objetos, métodos, estructuras de información, eventos, etc. que puede llegar a ser realmente compleja. Al fin y al cabo, las capas públicas de una librería (*API*) están diseñadas con una filosofía de diseño que puede parecer al principio artificiosa o antinatural, pero que seguro tienen una intencionalidad muy estudiada. O no. También hay librerías con *API* realmente mal diseñadas.

de *Internet*, el papel de las librerías ha sido substituido por los servicios. Un *servicio web* es un *software* funcionando en un servidor que, bajo el paradigma cliente-servidor, permite que otros *software* clientes se conecten a él para solicitar servicios.

Aunque el ejemplo más palpable de este modelo son las propias aplicaciones *web* que usamos en el día a día, en el que un *frontend web* que funciona en un navegador *web* (el cliente) se conecta a un servidor para solicitar datos y trabajar con los mismos, esos mismos servidores proporcionan servicios para desarrolladores, en contraposición al nivel de usuario final. Estos *software* instalados en servidores, como las librerías, exponen una *API de servicios*, es decir, un conjunto de funcionalidad y documentación que le permite a un cliente, que puede ser otro servidor, conectarse a ella y solicitar tareas de computación. En el ejemplo de la librería *pyproj* anteriormente citado, un programa cliente usaba la librería importándola mediante una instrucción *import*. Librería y programa cliente existen en la misma máquina y la primera es usada directamente por el segundo.

En los servicios *web* la relación no es tan directa, ya que los servicios no suelen estar en la misma máquina que los *software* clientes que pretenden utilizar sus servicios. En cambio, la comunicación se establece a través de los métodos propios de *Internet*, fundamentalmente el protocolo *HTTP*. Son las llamadas *web API*.

Existen varios estándares para el diseño de *Web API*, como por ejemplo *SOAP* o *REST* (*Representational State Transfer*). En la plataforma se opta por esta última, por lo que la implementación se dice que es *RESTfull*.

Web API REST

Una *API REST* (Biehl, 2016) es una forma de diseñar *web APIs* que se basa en el acceso a los recursos del servidor mediante el uso de procedimientos estándares propios del protocolo *HTTP* para intercambiar información entre el cliente y el servidor en formatos de datos como *XML* o *JSON*. Por recurso entendemos un conjunto de datos que describe un elemento de información guardado en el servidor, o una funcionalidad a realizar sobre un conjunto de datos que se provee junto a la llamada a la *API*. Por ejemplo, un recurso podría ser un modelo de datos de un usuario (nombre, dirección, email, etc.) alojado en el servidor, o una función que haga reproyecciones de coordenadas de un sistema de referencia a otro, a la que se le proporciona, a imagen y semejanza del ejemplo de *pyproj*, los dos sistemas de referencia y las coordenadas a transformar.

El protocolo *REST* usa para articular el acceso a estos recursos llamadas estándar del protocolo *HTTP*, siendo las más comunes las siguientes:

1. ***GET*:** obtiene una representación de un recurso alojado en el servidor. Por ejemplo, los datos de un usuario, suministrándole por ejemplo su correo electrónico para identificarlo;

2. ***POST*:** se solicita la realización de un proceso por parte del servidor, por ejemplo, el ya mencionado proceso de transformación de coordenadas, o, siguiendo el caso del usuario, la creación de uno nuevo proporcionando sus datos e identificándolo de forma unívoca con su email;

3. **PUT:** modifica el estado de un recurso, por ejemplo, cambiar la dirección postal de un usuario identificado por su email;

4. **DELETE:** borra un recurso, por ejemplo, los datos de un usuario identificado por su email.

El protocolo *REST*, además, es un protocolo *stateless*, es decir, sin estado. Esto quiere decir que el servidor no hace ningún esfuerzo por recordar al cliente cada vez que lo visita, es decir, no existe un *estado* definido de la relación entre cliente y servidor[20]. Aunque esto pueda parecer a priori contraproducente y contra-intuitivo, en realidad es la forma más fácil de gestionar un sistema en el que puede haber cientos de clientes haciendo solicitudes simultáneamente. El cliente, en cada comunicación, debe plantear su requerimiento a la *API* como si fuera la primera que realiza. Cada operación es atómica y autoconclusiva.

Estas *API REST* están diseñadas para ser utilizadas a nivel de desarrollo, es decir, para interconectar servicios de distintos desarrolladores para crear un nuevo servicio. Por ejemplo, la empresa *Mapbox* ofrece a los desarrolladores varias *API* para la realización de tareas con información geográfica, desde cartografía hasta navegación. De esta manera, una empresa no especializada en TIG puede fácilmente incorporar un mapa interactivo en su aplicación *web* utilizando las *API* cartográficas de esta empresa. Al igual que en el caso de las librerías, los detalles de implementación de todo el servicio cartográfico no tiene por qué ser relevantes, lo único que tiene que hacer un desarrollador es familizarse con la *API* de *Mapbox* para usar sus servicios. Asimismo, las *API* están diseñadas para la comunicación máquina-máquina, por lo que un programa implementado en un servidor se programa para conectarse y utilizar la funcionalidad de diversas *API* de servicios de otros proveedores.

Hoy en día, las *API* son ubicuas en *Internet*. Todas las aplicaciones *web* con las que interaccionamos a diario gracias a sofisticados interfaces *web* orientados a usuarios finales están utilizando por debajo llamadas a las *API* de servicio de dichas compañías para funcionar. Al igual que el interfaz *web* llama a las *API*, lo pueden hacer otros programas y servicios. Por ejemplo, cuando se utiliza el *frontend web* de *Gmail*, en realidad se están efectuando llamadas a la *API* de servicios de dicho producto. Si se estudia la *API*, se puede crear una aplicación que accede[21] a la funcionalidad del servicio para integrar la información contenida en los correos electrónicos del usuario.

Por lo tanto, en realidad, no supone un esfuerzo adicional proporcionar este acceso *API* en una plataforma como la que se plantea en este trabajo. La interacción de un usuario con la plataforma se realiza gracias a la misma, y, por lo tanto, está también disponible para ponerla a disposición de otros servicios.

[20] Es como si el servidor tuviera amnesia: cada vez que recibe una petición de un cliente es como si lo conociera por primera vez.

[21] Siempre que hablamos de acceso es obvio que dicho acceso está basado en la autorización del usuario, es decir, al igual que tenemos que acreditarnos en *Gmail* con nuestro usuario y contraseña, de manera similar hay que acreditarse para acceder directamente a los servicios de la *API*. Los mecanismos de autorización son un tanto distintos, pero las *API* también están cerradas al acceso anónimo.

Arquitectura *API* para la plataforma

La *API* planteada para la plataforma tiene que cumplir una serie de objetivos funcionales:

1. **creación y ejecución de trabajos de teselado:** creación de rejillas, definición y puesta en marcha de un trabajo de teselado;
2. **acceso a los datos teselados:** recuperación de datos teselados;
3. **realización de procedimientos de *Machine Learning*:** lanzamiento de procesos de *Machine Learning* sobre los datos teselados.

La *API* será de tipo *REST*. Para el primer objetivo, la *API* tiene que tener métodos para la definición de rejillas, la definición de análisis de adscripción y el lanzamiento de los mismos, tal y como ya se ha descrito en capítulos anteriores.

Para el segundo, el acceso se puede hacer de dos formas:

1. **con una estructura de datos propia de la plataforma:** este estándar transmite la información de las teselas en una forma que la tesela no transmite su información geométrica, sino sus coordenadas y su información alfanumérica en forma de vector temático. Esta forma de transmisión de la información no es estándar y necesita clientes implementados ex-profeso que la comprendan;
2. **mediante el servicio estándar *OGC WFS* (*Web Feature Service*):** este estándar (OGC, 2021b) sí transmite la información geométrica de la tesela y su información alfanumérica en forma de tabla convencional de datos. Esta forma de transmisión de la información es estándar y tiene la ventaja de que puede ser entendida por clientes *OGC*[22].

Para el tercer objetivo, la plataforma seguirá los pasos de la figura II.6.2:

1. **determinación del vector temático:** el usuario debe determinar qué variables de adscripción va a incorporar al análisis de *Machine Learning*;
2. **determinación del área de estudio:** el usuario selecciona un área de aplicación del modelo de *ML*. La determinación del área es importante, ya que tomar más área de la del área de estudio puede desvirtuar estadísticamente, contaminar, el modelo añadiendo datos no relevantes. No es lo mismo, como se puede entender, entrenar un modelo para el área metropolitana de una gran ciudad que para una provincia que incluye ámbitos urbanos y rurales;
3. **definición del modelo de *ML*:** en este paso, el usuario debe determinar qué proceso de *ML* va a utilizar (*K-Means* o *Random forest*) y configurarlo. En el caso del *K-Means*, se reduce a seleccionar el número de grupos o *clusters* a discriminar en los datos, mientras que en el caso del *Random forest* se han de determinar las áreas de entrenamiento y las clases de discriminación a las que cada una pertenece;

[22] Por ejemplo, SIG de escritorio como *QGIS* o *ArcGIS*.

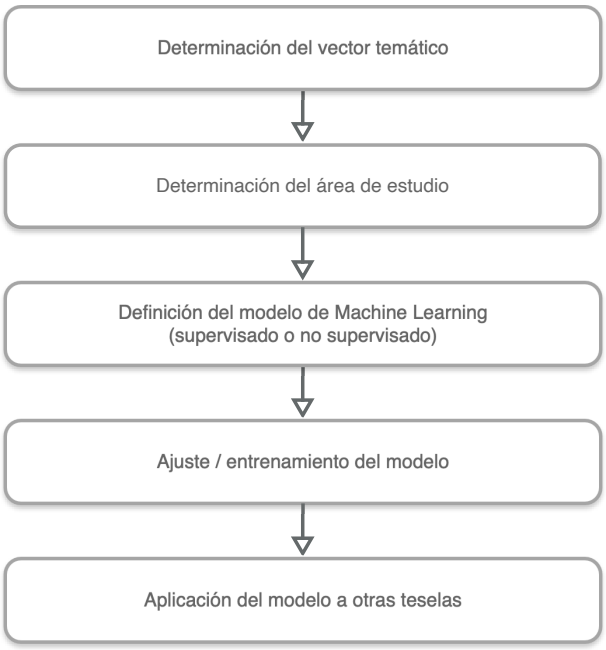

Figura II.6.2: Pasos funcionales para la realización de procedimiento de *Machine Learning*.

4. **ajuste del modelo:** en este paso, el algoritmo de *ML* seleccionado ajusta el modelo. En el caso del *K-Means*, esto implica tomar una muestra aleatoria de teselas en cada nivel de resolución de la rejilla y entrenar la discriminación de categorías en el número de grupos asignados dentro de la muestra. En el caso del *Random forests*, implica entrenar el bosque de árboles de decisión en base a las clases y las zonas de entrenamiento;

5. **aplicación del modelo a las teselas de la zona de estudio:** una vez el modelo está entrenado, este entrenamiento se guarda y se aplica a todas las teselas, de todos los niveles de resolución, del área de estudio. Recordemos que estos algoritmos no funcionan con vectores temáticos incompletos: las teselas tienen que tener todas las variables de adscripción seleccionadas.

II.7 Visualización

La visualización es el último paso de un proyecto TIG. La realización de cartografía es fundamental para el análisis visual de los datos y la detección de patrones espaciales (Robinson et al., 1987).

La visualización se basa en la representación, por un medio u otro de diseminación proporcionado por la *API*, de la información temática contenida en las teselas. De esta manera, el soporte geométrico de la visualización será aportado por la tesela, mientras que la información alfanumérica será aportada por el vector temático de variables de adscripción.

La *API* planteada puede ser capaz de difundir la información de la rejilla implementando bien el estándar *OGC WFS* o mediante un método propio que aprovecha la definición matemática de la rejilla para evitar el envío de las geometrías explícitamente. Es éste último mecanismo en el que vamos a centrar la discusión de este capítulo, puesto que la representación y el trabajo con información proveniente de un servidor que implementa el estándar *WFS* está ampliamente discutida en la literatura científica (Álvarez Francoso, 2016).

II.7.1. Visores para la rejilla definida matemáticamente

En el modo no estándar, la *API REST* devuelve, para todas las teselas que caen en la zona de visualización del visor[1], y para un nivel de resolución determinado, la coordenada **[R, X, Y]** (nivel de resolución, X, Y) de cada tesela y su vector temático, junto a la definición de la rejilla, que recordemos que incluye sistema de proyección, origen de coordenadas y niveles de resolución.

A partir de la definición de la rejilla, la librería que trabaja en el visor *web* recrea la geometría de la tesela. En muchas ocasiones, debido a las limitaciones de los visores *web*, esta geometría ha de ser reproyectada al sistema final de representación, que suele ser, casi invariablemente en los visores actuales, el designado por el *EPSG* con el código 3857, conocido con el nombre *Google Mercator*[2]. Una vez recreada la geometría de la tesela, esta

[1] Técnicamente se la conoce como *viewport*.

[2] Este sistema de proyección ha sido altamente cuestionado (Battersby et al., 2014) por la comunidad geodésica por su falta de rigor técnico. Básicamente, y simplificando la discusión, su principal (y única) propiedad es que hace que la proyección del mapamundi sea cuadrada, algo que los técnicos de *Google* buscaban para implementar su sistema de mapas multiescalares seccionados en imágenes de 256x256 píxeles. A pesar del poco rigor técnico de la propuesta, es posiblemente uno de los sistemas más utilizados del

ya puede ser utilizada sobre el área de visión del visor cartográfico.

Aunque la forma natural de la tesela sea cuadrada en la plataforma propuesta, ésto no implica que sea su única forma de representación. A partir de las coordenadas de la tesela es trivial encontrar su centro. Dicho centro puede ser utilizado, por ejemplo, para representar la tesela como un punto. También es factible encontrar dentro de la tesela dos o tres puntos, lo que puede resultar en mecanismos de visualización de información muy interesantes. Las opciones son múltiples:

1. **usar la geometría cuadrangular de la tesela:** estaríamos ante un mapa de coropletas. Las posibilidades de representación son usar una rampa de color o un conjunto de colores disjuntos para representar una variable continua o discreta, respectivamente;

2. **usar el centro de la tesela:** al usar el centro de la tesela para representar un punto, las opciones de representación suben a dos variables: una para el tamaño del punto (variable continua) y otra para el color del mismo (variables continuas y discretas);

3. **usar dos o tres puntos dentro de la tesela:** se pueden crear efectos interesantes como utilizar tres colores en modo transparente para generar un efecto similar al producido en una cuatricromía de imprenta, creando un mapa de colores que varía en tres variables cromáticas para representar el distinto peso de tres variables continuas en la tesela;

4. **usar el borde de la tesela:** el borde de la tesela también puede ser utilizado para representar ciertas cualidades discretas, como la presencia o no de variables de tipo booleano[3].

Es importante entender, desde el punto de vista del tránsito de los datos entre el servidor y el cliente, que lo que se recibe en este tipo de visores son datos, no mapas. El visor crea la representación cartográfica a partir de los datos, pero el servidor se comporta de una forma completamente neutra en cuanto a la representación cartográfica. El hecho de que el cliente reciba datos significa que las representaciones de datos no tienen por qué limitarse exclusivamente a la cartográfica, sino que también pueden utilizarse otros mecanismos para su representación, como por ejemplo histogramas. El uso de histogramas para ver la distribución estadística de una variable en intervalos sirve tanto para variables continuas como discretas, proporcionando un entendimiento de la variable muy complementaria con su visualización cartográfica.

Lo interesante de mezclar histogramas con su representación cartográfica es que ambos sistemas de visualización de datos pueden estar coordinados, es decir, que la distribución de frecuencias de una variable de adscripción visualizada en el histograma puede circunscribirse a lo visualizado en el mapa en ese momento, obteniéndose la distribución local a la

mundo, puesto que toda la información que se vuelca en sistemas como *Google Maps* debe ser reproyectado a dicho sistema. En poco tiempo adquirió, como suele ser norma con *Google*, el status de estándar de-facto, acabando el *EPSG* por asignarle un código en su estándar de identificación de sistemas de referencia.

[3]Del tipo si/no.

visualización actual. De la misma manera, una manipulación sobre el histograma también puede afectar al mapa, sirviéndo el primero como un filtro a la distribución de frecuencias de lo visualizado en el mapa. Este tipo de visualizaciones enlazadas son ya muy comunes en muchas plataformas de visualización y de exploración de datos. A nivel cartográfico, la plataforma *CARTO* (CARTO Contributors, 2021a) es la que lleva este concepto a su más evolucionada expresión, pero otras plataformas de análisis de datos como *Power BI* de *Microsoft* (Microsoft Corp., 2021b) o *Watson Analytics* (IBM Corp., 2021) de IBM lo hacen también, aunque sin darle a la componente cartográfica tanta importancia como CARTO.

II.7.2. Funcionalidades del visor

Por lo tanto, el visor debe cumplir los siguientes objetivos funcionales:

1. **exploración cartográfica interactiva de los datos de rejilla:** representación por reconstrucción matemática de la geometría de la rejilla y superimposición de la misma sobre una cartografía base. Esta visualización debe ser interactiva, permitiendo las operaciones clásicas de zoom y desplazamiento, así como de cambio de la cartografía base de referencia;

2. **control semiológico de las teselas, dictado por la representación de una o más variables de adscripción:** deben presentarse diversos mecanismos de representación semiológica de una o más variables de adscripción sobre los elementos geométricos aportados por las teselas;

3. **exploración de las variables de adscripción mixta gracias a la interacción entre histograma y mapa:** el histograma de frecuencia de la variable y el mapa deben estar coordinados.

Parte III

RESULTADOS

III.1 Introducción a los resultados

> La optimización apriorística es el germen de todos los males.
>
> *The Art of Computer Programming*
> *Donald Knuth*

En esta parte se entra en detalles sobre la implementación de la plataforma *Cell*, un conjunto de librerías y programas que conforman una arquitectura de microservicios destinados a realizar los trabajos computacionales descritos en la sección de Metodología.

Con el nombre *Cell* haremos referencia, por tanto, a toda la plataforma con sus distintas partes funcionales, que se pueden apreciar en la figura III.2.1 (pág. 142). La figura III.1.1, por su parte, secuencializa el orden en el que se expondrán los resultados. Estos dos diagramas son fundamentales para contextualizar los contenidos que se desarrollan a continuación.

III.1.1. Unas notas introductorias sobre el código del programa y su ubicación y consulta

Los resultados de esta tesis no pueden entenderse en su totalidad sin tener en cuenta la gran cantidad de código escrito para la implementación de la plataforma *Cell*. La plataforma *Cell* es el principal producto de esta tesis: un prototipo de plataforma para la creación, gestión y análisis aplicado a la rejilla multiescalar propuesta en este trabajo.

El código de una aplicación informática puede ser distribuido de muchas formas. Es, obviamente, demasiado voluminoso como para reproducirlo en este documento, y tampoco es el medio más adecuado para hacerlo.

Actualmente, la distribución del código de las aplicaciones informáticas, especialmente aquellas de *Software Libre* y, por tanto, sujeta a la participación de desarrolladores en un entorno colaborativo, se realiza gracias a aplicaciones llamadas *Sistemas de Control de Versiones* (*SCV*, *VCS* en inglés). Existen y han existido varias, pero sin duda la más utilizada actualmente es *Git*.

Alrededor de estas aplicaciones se han desarrollado plataformas comerciales, aunque de uso libre para proyectos de *Software Libre*, que ponen a la disposición de los desarrolladores

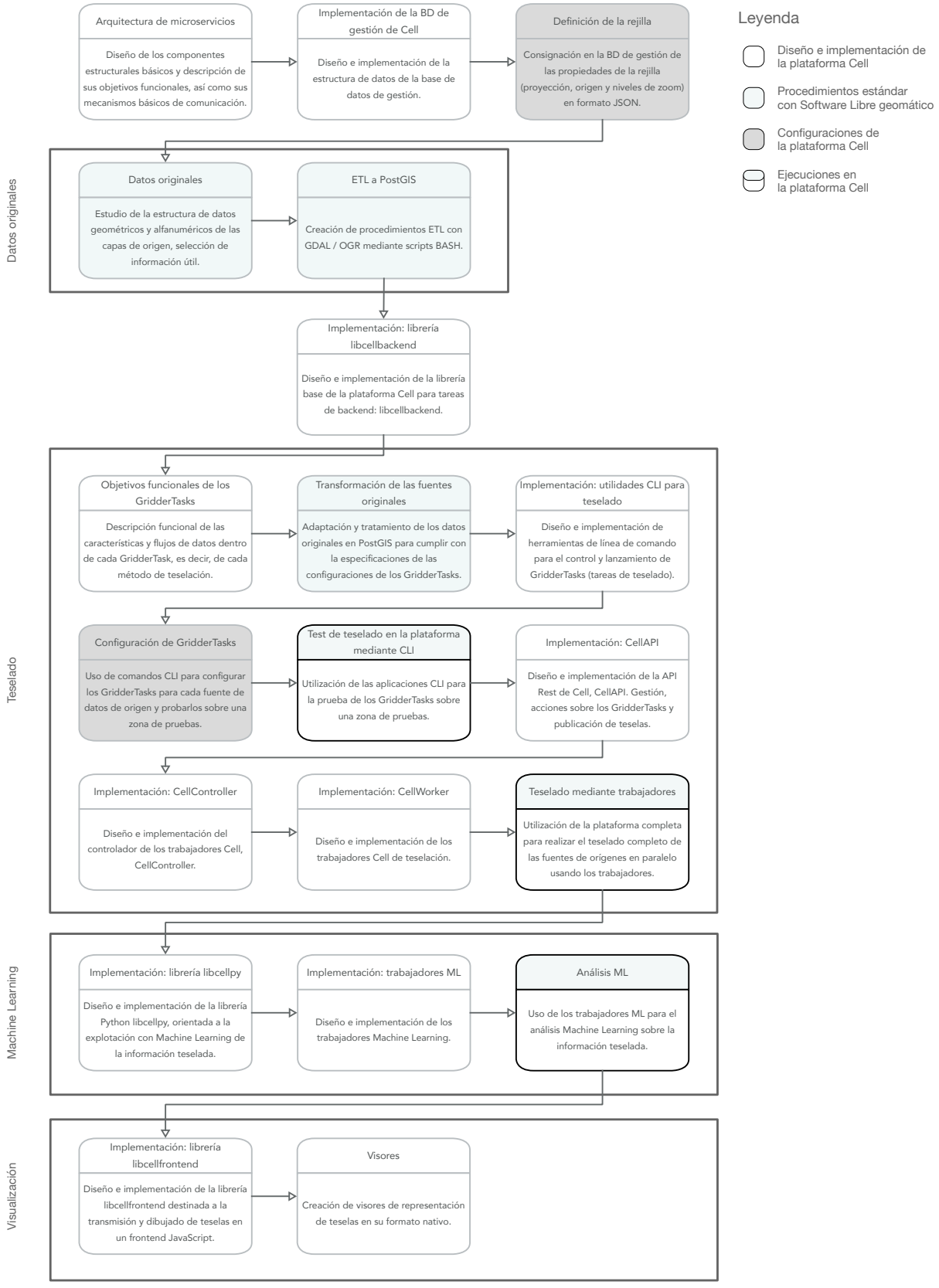

Figura III.1.1: Esquema con la secuenciación de resultados.

un ecosistema de herramientas completo que explotan y expanden las capacidades de *Git* como sistema de control de versiones, aportando muchos servicios adicionales. El código de *Cell* se encuentra alojado en la plataforma *GitHub*. Son varios los repositorios creados para la gestión del código y los datos de esta tesis:

1. **repositorios de datos de origen;**
2. **repositorio de la plataforma *Cell*.**

Todos estos repositorios pueden encontrarse centralizados en la página *web*:

`https://cell.37north.io`

Referimos al lector que quiera consultar el código con detalle a dichos repositorios. El anexo V.2 “Repositorios GitHub” (pág. 349) también proporciona más detalles acerca de la estructura de estos repositorios.

III.2 Arquitectura de microservicios

La plataforma propuesta, *Cell*, se basa en una arquitectura de microservicios, que aporta gran flexibilidad y *escalabilidad*[1] a la solución. Es este capítulo se dan las líneas maestras de esta arquitectura, que será descrita elemento a elemento a lo largo de los capítulos posteriores.

III.2.1. Diseño de la arquitectura de microservicios de *Cell*

Como ya se ha comentado, *Cell* es un sistema informático diseñado con una arquitectura de microservicios (Jaramillo et al., 2016), es decir, un sistema informático que, lejos de tener la forma de un gran programa monolítico destinado a ejecutarse en una sola máquina, está constituido por multitud de subsistemas (microservicios) entrelazados, pero independientes, que realizan aspectos concretos de la operativa global del sistema. La figura III.2.1 (pag. 142) muestra la arquitectura de *Cell*:

1. ***API*:** la *API* es el microservicio que hace las veces de cara pública de todo el bloque de componentes *Cell* que en la figura III.2.1 se muestra en *backend*, es decir, en los servidores. Es la capa de interacción entre posibles clientes externos y las funcionalidades del sistema, ejerciendo tareas de control de acceso y aislamiento de otros componentes, especialmente, como es regla en los servicios distribuidos, de la base de datos. La *API* se conecta con la base de datos *PostgreSQL* para leer datos de ella y poder enviárselos al usuario, como por ejemplo peticiones de datos de rejilla;

2. **controlador:** este microservicio, como ya se apuntó en "Metodología", capítulo III.7 (pag. 235), controla la actividad de los trabajadores y lleva a cabo otras tareas de gestión administrativa del sistema. El papel del controlador es fundamental en una arquitectura como la de *Cell*, en la que los trabajadores son un grupo de microservicios con un acoplamiento muy laxo al resto del sistema. Este acoplamiento laxo de los trabajadores hace que haya que monitorizar su salud y su funcionamiento, ya que si fallaran y dejaran de funcionar el resto del sistema no se enteraría de inmediato. Hay que chequearlos cada cierto tiempo para monitorizar su actividad

[1] Por *escalabilidad* se entiende en ingeniería de *software* la capacidad que tiene un sistema o una arquitectura de aumentar sus capacidades de proceso de datos. El aumento de esta capacidad puede ser o bien *vertical*, aumentando la potencia (memoria, CPU, etc.) de la máquina donde se ejecuta la arquitectura o sistema, o bien *horizontal*, aumentando el número de máquinas que trabajan conjuntamente, aportando cada una sus capacidades computacionales para que la capacidad global del sistema aumente. Obviamente, esta última opción requiere de arquitecturas descentralizadas y laxamente imbricadas, y aquí es donde la arquitectura de microservicios destaca.

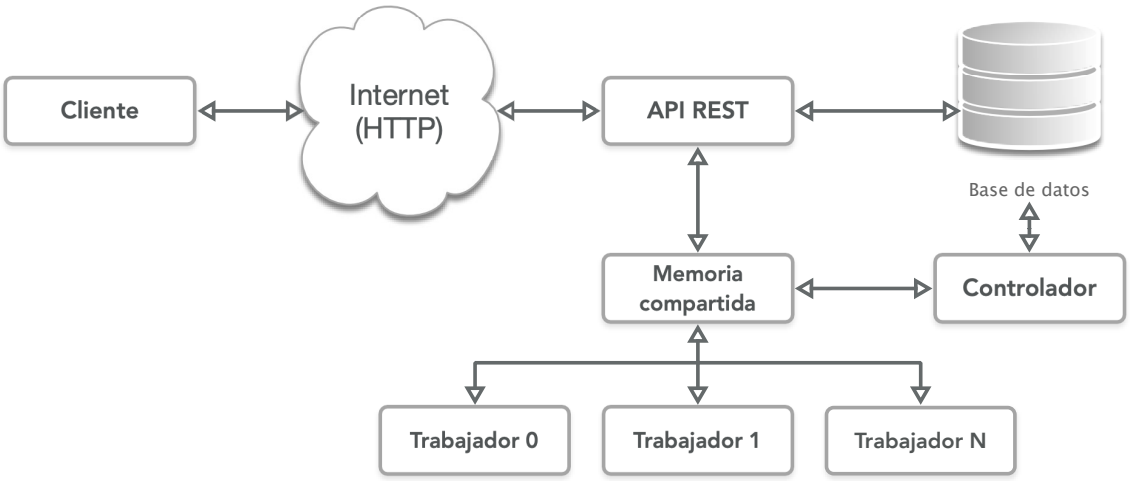

Figura III.2.1: Arquitectura de microservicios de la plataforma *Cell*.

y comenzar tareas de recuperación en caso de errores. Estas tareas administrativas son realizadas por el controlador a intervalos de tiempo prefijados. El controlador se comunica con la base de datos *PostgreSQL* para almacenar y leer datos y con la memoria compartida *Redis* (S. Chen et al., 2016) para compartir información con el grupo de trabajadores;

3. **base de datos *PostgreSQL / PostGIS*:** esta microservicio de base de datos es el repositorio centralizado de información de la plataforma. Aquí es donde se almacena la información adscrita a la rejilla y otros grupos de datos de interés, como por ejemplo el control de los trabajadores, el reparto de tareas a los mismos, los metadatos de las variables de adscripción, etc. A partir de aquí nos referiremos a ella simplemente como ***PostGIS***;

4. **memoria compartida *Redis*:** el microservicio *Redis* es un tipo de base de datos de los llamados ***NoSQL*** que implementa varias estructuras de almacenamiento de datos de alta eficiencia, con la particularidad de que su funcionamiento se centra en la memoria RAM de la máquina, no en el disco duro, por lo que la convierte en un sistema de rapidísimo acceso. Esto la hace ideal para cumplir dos papeles fundamentales: gestionar el intercambio de mensajes entre controlador y trabajadores para coordinar las cargas de trabajo y almacenar información que viaja hacia los trabajadores para su procesamiento o desde los mismos para su almacenaje en la base de datos *PostGIS*;

5. **trabajadores:** los microservicios de trabajadores son los microservicios replicados de la plataforma, ya que su número es variable y depende de los recursos de *hardware* disponibles. En la plataforma *Cell* vamos a encontrar dos tipos de trabajadores:

   - **trabajadores de adscripción:** son trabajadores programados en *TypeScript* que realizan los trabajos de adscripción de la información a la rejilla;
   - **trabajadores de *Machine Learning*:** son trabajadores programados en *Python* que realizan los trabajos de *Machine Learning*.

Estos microservicios se irán viendo con detalle en los siguientes capítulos.

III.3 Implementación de la base de datos de gestión de Cell

Todas las aplicaciones cliente - servidor (Oluwatosin, 2014), más allá de las más triviales, necesitan una base de datos de un tipo u otro en su capa más profunda. La base de datos es en muchas ocasiones lo más importante de una aplicación en la nube. Todos los demás componentes, servicios, etc. pueden ir renovándose[1], actualizándose, etc., pero los datos permanecen, se conservan de una versión a otra y en muchísimas ocasiones es el activo más importante de una compañía. Además, las bases de datos albergan información potencialmente confidencial. Por todo ello, las bases de datos están en la base operativa de todos los sistemas de *Internet*.

Las bases de datos de rango corporativo[2], como se las llama, son bases de datos que funcionan también según el paradigma cliente-servidor. Quizás una de las bases de datos más conocidas por el público sea *Microsoft Access* por ser parte de la *suite* ofimática *Microsoft Office*. Esta base de datos da una idea falsa de la forma en la que funciona una base de datos de las utilizadas en una aplicación en la nube, ya que se utiliza como un programa de escritorio: la tenemos instalada directamente en nuestros ordenadores, sus almacenes de datos son ficheros que podemos mover de un sitio para otro, tienen un ergonómico interfaz de usuario, etc. Su experiencia de uso no se diferencia demasiado de la de sus compañeros *Excel* o *Word*.

El caso de uso de una base de datos de rango corporativo no se parece en nada al del mencionado *Access*. Una base de datos de este tipo es un complejo programa (usualmente se le deja un servidor grande para él sólo) que se instala en un servidor y que no tiene, por lo tanto, un interfaz de usuario. Es un sistema cliente - servidor preparado para aceptar conexiones de programas clientes (tanto programas orientados al usuario final como otros

[1] Nótese que esta es otra gran ventaja de las arquitecturas de microservicios: al estar éstos desacoplados y comunicarse entre sí con estándares de comunicación (interfaces y *API*) bien establecidos, existe una gran libertad de acción para reescribirlos en otros lenguajes, mejorarlos, substituirlos por otros sistemas análogos (aunque esto no es trivial) o permitir que equipos de desarrollo distintos trabajen en ellos de forma más o menos independiente.

[2] Se les llama a sí a los grandes sistemas de bases de datos que pueden soportar grandes cargas de trabajo y almacenamiento de datos en un entorno multiusuario. Estas bases de datos se instalan en servidores para dar servicios centrales de almacenamiento y procesamiento a usuarios finales y otras aplicaciones informáticas. Como su ámbito de uso trasciende el monousuario (donde se ubica, por ejemplo, el *Access* de *Microsoft*), son bases de datos que se encuentran en empresas y departamentos gubernamentales, de ahí su nombre. Son sistemas complejos que requieren para su mantenimiento y gestión personal muy especializado. Además, las versiones comerciales suelen ser sistemas bastante caros.

programas que utilizan la base de datos) y dar servicios a cientos de usuarios simultáneamente. Además, debe de hacerlo poniendo todo el cuidado del mundo en la integridad de los datos. Debe estar preparada para desconexiones de red, caídas de sistemas, cortes de luz, fallos en los discos duros y todas las incidencias que puede sufrir un ordenador. Todo ello además asegurando que la potencial interacción simultánea de varios usuarios sobre el mismo datos no lo corrompe o lo deja en un estado de falta de integridad frente al resto de los datos del sistema.

Son, por lo tanto, sistemas extremadamente sofisticados y complejos, y cada fabricante toma estrategias y enfoques distintos para solventar todos estos grandes retos de ingeniería de *software*. Grandes fabricantes de estos sistemas son *Microsoft*, con *SQL Server*, y *Oracle*, uno de los líderes del sector, así como *IBM* y demás. Además, todos los operadores de infraestructuras de nube (*Amazon*, *Google*, etc.) tienen también sus propias tecnologías adaptadas a las necesidades de los nuevos paradigmas de creación de aplicaciones en proveedores de nube (Lewis, 2010).

Para *Cell* se ha seleccionado la base de datos relacional orientada a objetos de *Software Libre PostgreSQL*. *PostgreSQL* es un solvente proyecto de *Software Libre* con muchos años ya de desarrollo que proporciona una base de datos de grado corporativo con un alto rendimiento y panoplia de funcionalidad. Aunque no es la base de datos de *Software Libre* más desplegada del mundo[3], *PostgreSQL* es una base de datos *SQL* de tipo *ACID*[4], es decir, que garantiza la seguridad de las transacciones de datos entre clientes y servidor. Tiene como objetivos ser extensible y lo más compatible posile con el estándar *SQL*. La orientación a objetos de su nombre significa que un programador puede extender las capacidades de la base de datos por medios propios que no implican recompilar el código de la base de datos.

Es precisamente esta última cualidad la que la hace tan atractiva para las TIG. En el año 2001, la empresa canadiense *Refractions Research*, bajo la dirección técnica de Paul Ramsey (Hsu et al., 2015), comienza el desarrollo de una extensión para la *PostgreSQL* que permite que la base de datos entienda y sea capaz de operar con la vertiente geométrica de la información geográfica. Dicha extensión a la base de datos se llama *PostGIS*, está escrita en su mayor parte en *C* y *C++* y se vale de diversas librerías de *Software Libre* para transformar a la *PostgreSQL* en una base de datos relacional geográfica.

III.3.1. *PostGIS*

Una base de datos *SQL* no geográfica (la llamaremos a partir de aquí *convencional*) es un sistema diseñado para gestionar modelos de datos que se rigen por un sencillo

[3]Ese honor le corresponde a *MariaDB* (anteriormente llamada *MySQL*). Dado que *MariaDB* se centra en servir datos a aplicaciones *web* muy utilizadas de tipo *blog* y similar, es la opción más usual para desarrolladores de *Software Libre*. Sin embargo, *MariaDB* sacrifica funcionalidades y algunas medidas de seguridad de alto nivel en favor de una mayor velocidad (Pilicita Garrido et al., 2020).

[4]Acrónimo de *Atomicity*, *Consistency*, *Isolation* y *Durability* (atomicidad, consistencia, aislamiento y perdurabilidad), cualidades del sistema que garantizan que los datos no se van a corromper ni van a entrar en un estado de inconsistencia con las reglas del sistema cuando varios clientes intentan actuar sobre ellos a la vez.

sistema de reglas lógico-matemáticas. Este conjunto de reglas establece una metodología de modelado y diseño de estructuras de datos llamado *Modelado entidad-relación*[5,6] (Codd, 1983, P. P. S. Chen, 1976). Estos modelos entidad-relación se expresan, a nivel básico, en un conjunto de tablas que establecen relaciones entre sí gracias al intercambio de información común a las filas de ambas. Una vez la información de un problema o fenómeno está expresada en esta forma, el sistema puede aplicar otro sencillo conjunto de reglas lógico-matemáticas para relacionar los datos entre sí de la forma que el usuario quiera y, de esta forma, producir información, es decir, respuestas.

Esto lo hace la base de datos con información alfanumérica convencional, es decir, con la mitad de la información geográfica. A la información geográfica, por ejemplo en un SIG de escritorio tradicional, podemos hacerle preguntas de índole alfanumérico a la tabla de atributos de una capa (vertiente alfanumérica de la información geográfica). Por ejemplo, dame todos los municipios que tengan en su campo "provincia" el valor "Córdoba". Es una pregunta al aspecto alfanumérico de la información geográfica, y es algo que las bases de datos convencionales hacen extraordinariamente bien.

Sin embargo, en un SIG de escritorio también podemos interrogar a la vertiente geométrica de la información geográfica. Por ejemplo, "encuentra todos los polígonos de municipios que colindan con el polígono del municipio de Montilla". Es una pregunta al aspecto geométrico de la información geográfica, y lo resuelve un subsistema del SIG llamado *motor topológico*, es decir, un subsistema que sabe lo que es un vector, lo que es un polígono, y que sabe rastrear relaciones topológicas como "adyacencia" entre un conjunto de éstos.

Esa pieza, el motor topológico, les falta a las bases de datos convencionales. Las bases de datos convencionales tienen un sofisticado *motor relacional*, que contesta potencialmente enrevesadas preguntas hechas a un conjunto de tablas relacionadas, que es en lo que se expresa un modelo relacional. Sin embargo, no sabe lo que es un vector, ni mucho menos un polígono, y no digamos una relación topológica. No puede, por tanto, contestar preguntas hechas al aspecto geométrico de la información geográfica.

PostGIS proporciona dichos componentes a la base de datos *PostgreSQL*. Recordemos que es una extensión: es un programa que se instala sobre una instalación previa de *PostgreSQL*. Lo que hace es añadirle a la base de datos ese motor topológico[7], un segundo

[5] A pesar del auge de las bases de datos *NoSQL*, que sacrifican el modelado entidad-relación en pos de estructuras de datos predefinidas y, como tal, altamente eficientes, pero altamente inflexibles también, el modelado entidad-relación sigue siendo a día de hoy uno de los medios de expresión de modelos de estructuras de datos computacionales más seguro y flexible (Teorey et al., 1986). No es el más rápido, pero admite el modelado de la información de cualquier problema.

[6] Esta metodología de modelado es un trabajo seminal que constituye una de las mayores revoluciones en tratamiento de datos y que ha dado forma a cómo se ha abordado este área técnica en los últimos ya casi 50 años. La gestión de información con computadoras no se concibe en la actualidad sin la gran aportación que Ted Codd (1923 - 2003) realizó en 1970, desarrollando el álgebra y el cálculo relacional, que constituye la teoría matemática detrás de los motores relacionales de las bases de datos de este tipo. Ted Codd ha sido por ello reconocido con galardones tan prestigiosos como el Premio Turing en 1981 (Date, 2003, Chamberlin, 2012). Sin duda un desarrollo técnico - científico de una elegancia sin par y una utilidad incontestable. El único problema que tiene es que su escalabilidad es compleja, por lo que no es hasta la llegada de los grandes conjuntos de datos de la era de *Internet* más actual que se proponen metodologías *Big Data* complementarias a esta metodología.

[7] El motor topológico que usa la *PostGIS* es una librería escrita en *C* y *C++* llamada *GEOS* (GEOS

cerebro que le permite, trabajando en conjunción con el motor relacional ya existente, contestar a preguntas de índole topológico. *PostGIS* hace que la base de datos sepa lo que es una geometría, incorporando nuevos tipos de datos *GEOMETRY* con los que se pueden crear columnas en las tablas para alojar puntos, lineas, polígonos y demás, y operar con ellos. Además, añade una respetable cantidad de funciones que se pueden utilizar en las sentencias *SQL* que sirven para explorar, con las reglas lógico-matemáticas de ese lenguaje de consulta relacional, las relaciones topológicas entre datos de tipo *GEOMETRY*. Lo que tenemos, al final, es una potente base de datos relacional orientada a objetos geográfica, con un potencial analítico difícil de igualar[8].

Una pieza clave del rendimiento de una base de datos es la indexación. Por indexación entendemos la creación de estructuras de datos accesorias a los propios datos (es decir, metadatos) que ordenan los datos según distintas metodologías y criterios. Imaginemos una base de datos con el censo de toda España. Son más de 45 millones de registros. Si quisiéramos buscar todos los "Antonio" el sistema tendría que recorrer los datos de los 45 millones de registros para compararlos con el criterio de búsqueda, desde el primero hasta el último. Sin embargo, si le hemos calculado al campo "nombre" de la tabla de datos de personas un índice de tipo alfabético, el sistema lo utilizará primero para buscar en aquellos nombres que comienzan por "A". El resto serán obviados. El incremento en el rendimiento es notable. La información geográfica también se indexa espacialmente para encontrar con rapidez relaciones topológicas de distancia y, dada la alta exigencia computacional de las operaciones geométricas, este paso es fundamental[9]. Sin embargo, un exceso de índices en información no clave en los análisis tiene el paradójico efecto de poder ralentizar en exceso el sistema, puesto que el mantenimiento de los índices también exige tiempo de computación[10].

Por si todas estas ventajas de las bases de datos relacionales en general y *PostGIS* en particular no fueran suficientes, en los últimos años (Petković, 2017) *PostgreSQL* ha ido ganando en versatilidad gracias a la adopción de algunas funcionalidades tradicionalmente vinculadas a las bases de datos *NoSQL*. Entre ellas, la más importante es la inclusión, como tipo de dato nativo, del formato *JSON* (Peng et al., 2011), que permite almacenar documentos no estructurados en un campo de una tabla. Esto permite, por lo tanto, diseñar almacenes de datos con un enfoque mixto (Liu et al., 2016): el uso de la tradicional metodología de modelado y operación tan versátil y probado durante décadas de las bases

Contributors, 2021).

[8] Hay otra extensión, *PL/R*, que permite integrar el conocido entorno estadístico *R* dentro de la base de datos, con lo que tendríamos una *base de datos relacional orientada a objetos geoestadística*. También se le puede añadir otra extensión para incorporar código en *Python*, por lo que el potencial de incluir librerías de *Machine Learning* de dicho ecosistema directamente dentro de la base de datos es otra posibilidad a tener muy en cuenta.

[9] Las diferencias de hacer análisis con y sin índices espaciales son abismales en cuanto a rendimiento. Sin indexar la información, análisis que duran milisegundos sobre información indexada se van perfectamente a los minutos.

[10] Este equilibrio depende muchísimo de la dinámica y el ciclo de vida de los datos. En una base de datos que no recibe actualizaciones de los mismos con frecuencia o a la que no se le incorpora nuevos paquetes de información este extra de computación es bajo, ya que el sistema es perfectamente capaz de trabajar con información parcialmente indexada. Sin embargo, en bases de datos muy dinámicas donde la entrada, borrado y/o modificación de datos es una constante esto puede ser un problema y la estrategia de indexación ha de ser muy bien estudiada.

de datos *SQL* junto a las ventajas que en algunos escenarios tienen los documentos no estructurados o de estructura laxa, como es el formato *JSON*. Para que el trabajo con *JSON* sea operativo, obviamente, la base de datos no sólo incluye la capacidad de almacenar datos en dicho formato, sino que pone a disposición del usuario una serie de índices que permiten indexar la información no estructurada en *JSON* y recuperarla en tiempos razonables[11].

Cell hace un uso intensivo de esta capacidad. La estructura de datos que implementa la plataforma sobre *PostgreSQL* para el almacenamiento y gestión de la información tiene una base *SQL* altamente estructurada en sus fundamentos, aprovechando el alto rendimiento demostrado por el sistema en este aspecto (Stancu-Mara et al., 2008). Esto permite que ciertos aspectos críticos en la recuperación de teselas rindan adecuadamente: filtro por resolución, filtro por ámbito geográfico, etc. Sin embargo, para el almacenamiento de la información temática de adscripción de las teselas, el vector de adscripción, se utiliza un campo *JSON* de estructura laxa, ya que el principio de diseño asimétrico de la información temática vinculada a la tesela así lo exige. Una tesela en un entorno rural puede tener una decena de variables temáticas relacionadas con el medio ambiente, las divisiones administrativas y demás, mientras que una tesela de un entorno urbano densamente poblado posiblemente tenga muchísimas más variables temáticas al incorporar información, como es el caso de las fuentes de información de prueba seleccionadas, de población, catastro, etc.

Por lo tanto, para la comprensión de los próximos capítulos es importante tener en mente que:

1. *Cell* utiliza un microservicio de almacenamiento de datos basado en la base de datos *PostgreSQL*;

2. *Cell* utiliza la base de datos *PostgreSQL* para varios fines: gestión y administración de la información y almacenamiento del producto final del sistema, es decir, de la información teselada;

3. *Cell* utiliza la base de datos *PostgreSQL* como base de datos común de tipo *SQL* para aprovechar su alta eficiencia en indexación geográfica y temática, pero además utiliza sus relativamente recientes funcionalidades *NoSQL*, específicamente el uso que hace del formato de estructura de información laxa *JSON*, para almacenar la componente temática de las teselas.

III.3.2. Capacidades *NoSQL* de la *PostgreSQL*: los tipos de dato *JSON* y *JSONB*

Antes de entrar en detalles sobre la implementación del modelo de datos de la plataforma hay que hacer un obligado inciso en las mencionadas capacidades *NoSQL* de la

[11] *PostgreSQL*, comprensiblemente, no es el sistema que mejor haga este trabajo. Existen sistemas especializados en este tipo de estructuras laxas de información con un rendimiento en la indexación muy superior a *PostgreSQL*. No obstante, ha demostrado se lo suficientemente eficiente para los trabajos descritos en esta tesis.

PostgreSQL y en qué se expresan.

Para ello, primero hemos de revisar la noción de *tipo de dato*. En ingeniería de *software*, un *tipo de dato* es el modelo de memoria utilizado para albergar un dato determinado en un programa. Por ejemplo, no es lo mismo lo que ocupa un número de 0 a 255 (un *byte*) que un número de 0 a 65535 (dos *bytes*), o una cadena de texto de tamaño variable o una geometría vectorial. Para optimizar el espacio en memoria y, en consecuencia, los cálculos, el ordenador debe saber de antemano qué tipo de dato tiene cada dato para crear estructuras eficientes en memoria que los gestionen. En las bases de datos relacionales, cuando se diseña una tabla, siempre hay que especificar qué tipo de dato va a alojar cada columna de la misma: qué tipo de número, cuántos caracteres, etc.

En la base de datos no sólo se pueden almacenar números y palabras, que son los tipos de datos que nos pueden ser más familiares a priori. La base de datos tiene la capacidad de poder almacenar también información en binario, es decir, datos que tienen estructuras complejas internas destinadas a un fin computacional. Por ejemplo, la extensión *PostGIS* crea en la *PostgreSQL* un tipo de dato llamado *GEOMETRY*, que es de este tipo, y cuyo cometido es almacenar geometrías vectorias secuenciadas según el estándar *OGC Simple Features* (OGC, 2021a). Esta estructura de datos no está concebida para ser leída directamente por una persona, sino por la máquina, optimizando el espacio en memoria y ateniéndose a una estructura determinada por un estándar que permite a un programa leer y escribir la información en ese formato. Al incorporarse ese tipo de dato a la base de datos, *PostGIS* también incorpora una gran cantidad de funciones que lo entienden y saben trabajar con él: saben como leer un polígono almacenado en dicho formato y devolver el área que ocupa, por ejemplo.

Las capacidades *NoSQL* de la *PostgreSQL* se centran en la inclusión de dos tipos de datos de estructura laxa que gestionan información en *JSON*. El *JSON* es un estándar de *Internet* vinculado al lenguaje de programación *JavaScript* que se centra en la creación de estructuras jerarquizadas de información que sean fáciles de leer[12] y escribir, así como de entender por personas y no sólo por máquinas. El *JSON* se puede manejar tanto en una representación de texto plano (apta para las personas) como en un representación binaria (apta sólo para las máquinas), más compacta y que aumenta el rendimiento.

Una estructura en *JSON* presenta este aspecto:

```
1  {
2
3    "empresa": "ACME LTD",
4
5    "personal": [
6
7      {
8        "nombre": "Antonio",
9        "apellido": "Fernández",
10       "departamento": "márketing",
```

[12] En ingeniería, esos procesos de lecturas se suelen denominar con el préstamo del inglés *parser*, castellanizándolo en el verbo *parsear*.

```
11        "estado": "baja paternal"
12      },
13
14      {
15        "nombre": "Eva",
16        "apellido": "Hernández",
17        "departamento": "ingeniería"
18      }
19
20    ],
21
22    "departamentos": [
23
24      {
25        "nombre": "márketing",
26        "ubicacion": "Edificio A"
27      },
28
29      {
30        "nombre": "ingeniería",
31        "ubicación": "Edificio C"
32      }
33
34    ]
35
36  }
```

Código III.3.1: Ejemplo de *JSON*.

Como se puede ver, los principios del *JSON* son muy sencillos:

1. comienzo de la estructura jerarquizada y anidada de la información con una llave de apertura, en la línea 1;

2. la información se codifica en base a pares clave / valor. Por ejemplo, en la línea 3 se identifica el valor de dato “ACME LTD” con la clave “empresa”;

3. en la línea 5 se identifica con la clave “personal’ ’ una lista de paquetes de información, de ahí la presencia de los corchetes, que denotan el comienzo y final de una lista de elementos;

4. cada elemento de la lista anterior (líneas 7 y 14) son unidades de información compuestas por varios pares clave / valor. Nótese que no tienen por qué ser todos iguales: el primer elemento tiene un ítem de información extra (“estado”) que no posee el segundo. Esto es perfectamente normal, por eso se dice que el *JSON* es una forma laxa de estructurar la información;

5. en la línea 22 comienza otra lista;

6. se cierra la estructura de datos con la última llave en la línea 36.

Como se puede observar, *JSON* permite la creación de estructuras de datos:

1. **laxas:** la información no está sujeta a un esquema[13] rígido, sino que es flexible, pudiendo aparecer u obviarse elementos de la estructura a conveniencia[14];

2. **anidadas y jerarquizadas:** las estructuras de datos encerradas entre las llaves pueden ser anidadas entre sí según convenga a los objetivos de la estructura de datos.

Cell se sirve del primero de los principios de diseño de *JSON* para almacenar la información de adscripción de cada tesela, de forma que se cumpla el principio de asimetría propuesto para la definición de la rejilla.

Estas estructuras de datos *JSON* pueden ser almacenadas en una columna de una tabla de la base de datos. *PostgreSQL* ofrece dos tipos de datos para hacer esto: *JSON* y *JSONB*. El primero es *JSON* plano, en texto, y no es demasiado eficiente. El segundo, sin embargo, es *JSON* binario (de ahí la "B" final), y estas estructuras de datos son indexables. *PostgreSQL* además implementa el estándar *JSON Path*[15], por lo que ofrece muy buenas prestaciones para la búsqueda y el trabajo con estas estructuras de datos, a un nivel de rendimiento aceptable.

III.3.3. Modelo de datos para la plataforma *Cell*

La figura III.3.1 muestra el modelo de datos *PostGIS* que se usa para almacenar las información de la plataforma. A continuación pasamos a describir el propósito de cada tabla. La definición del modelo de datos se presenta en los repositorios de código como un *script*[16] *SQL* que se carga en el microservicio de la base de datos cuando ésta se crea. Exponemos en esta sección sólo las partes relevantes de dicho código, obviando las líneas que se utilizan para llevar a cabo acciones administrativas en la base de datos.

Tipo de dato: cell___cell

Como ya se ha indicado, la *PostgreSQL* es una base de datos orientada a objetos, es decir, que el usuario puede extender los tipos de datos que vienen por defecto con el sistema, creando tipos propios que se adapten a sus necesidades de modelado. La librería SQL de *Cell* crea un tipo de dato, *cell___cell*, orientado a la modelización de los datos de una tesela. Este tipo se utiliza en diversas partes de la librería para empaquetar de una

[13] Un *esquema*, en el contexto de las estructuras de datos, es un formato o reglas sintácticas y de composición a los que los datos deben adherirse para así cumplir con un estándar u otro de codificación de datos. El esquema determina los ítems de información de la estructura y los tipos de datos en los que estos ítems deben ser codificados.

[14] Obviamente, un programa que después vaya a trabajar leyendo esta estructura de datos debe estar preparado para encontrar omisiones o añadidos en la estructura.

[15] Digamos, haciendo una analogía, que *JSON Path* es el *SQL del JSON*.

[16] Un *script* es un, por lo general, bloque de código autónomo, no vinculado ni integrado en una aplicación, que se sirve del uso de *API* de librerías y otros servicios para implementar un flujo de trabajo destinado a producir un resultado. Es, por tanto, un medio muy utilizado para desarrollar pequeños programas muy autónomos que suelen ejecutarse en momentos puntuales o en sesiones de trabajo interactivas. Se usan mucho, por ejemplo, para desarrollar tareas administrativas en sistemas o crear flujos de tratamiento de datos reproducibles.

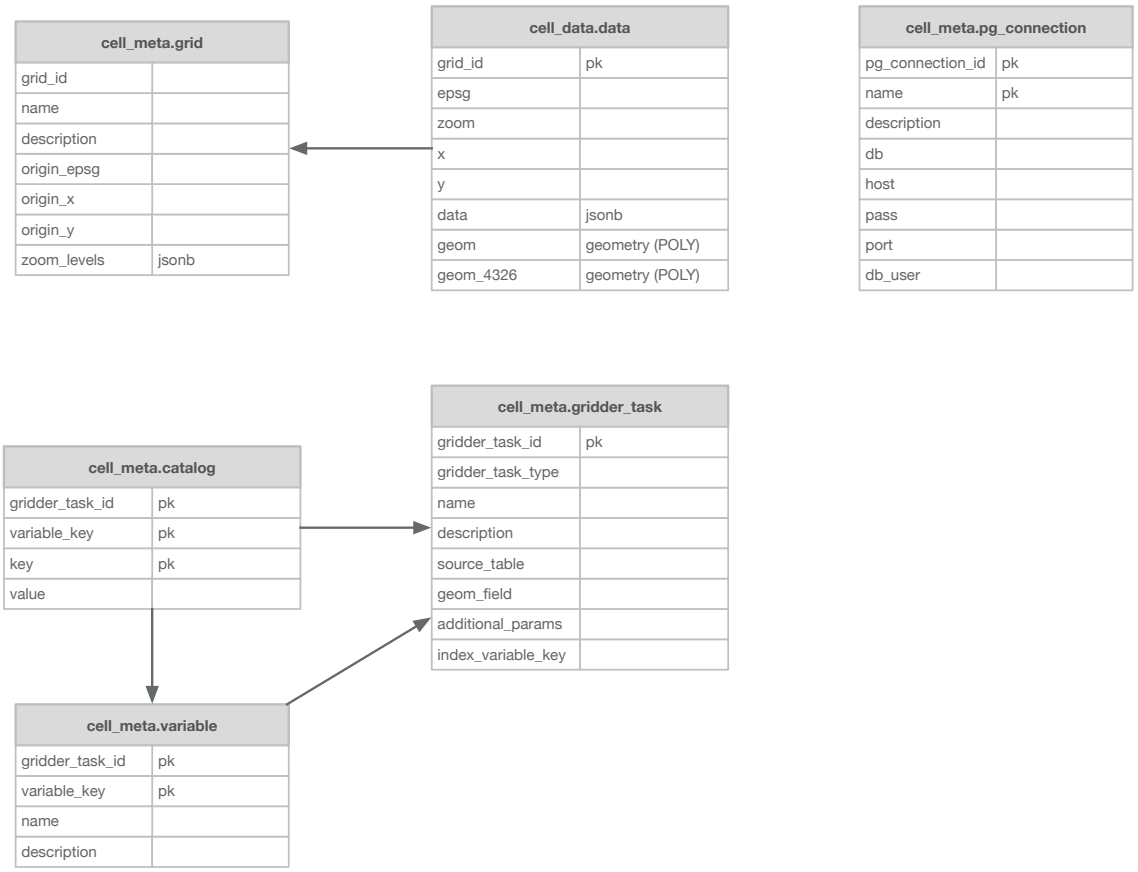

Figura III.3.1: Modelo de datos y relaciones de la base de datos de administración de la plataforma. Sólo se muestran las tablas relevantes.

forma clara e unívoca las propiedades de una tesela con la que se va a realizar diferentes operaciones.

El tipo se define como sigue:

```
1 create type cell__cell as (
2   grid_id varchar(100),
3   epsg integer,
4   zoom integer,
5   x bigint,
6   y bigint,
7   data jsonb
8 );
```

Código III.3.2: Tipo *SQL cell__cell*.

El tipo de compone de:

1. ***grid__id*:** el identificador de rejilla a la que pertenece esta tesela;

2. ***epsg*:** el código *EPSG*, es decir, el sistema de referencia, de la rejilla;
3. ***zoom*:** el nivel de resolución de la tesela en la citada rejilla;
4. ***x* e *y*:** las coordenadas (x,y) de la tesela en la rejilla en su nivel de resolución;
5. ***data*:** los datos de adscripción de la rejilla.

Nótese el uso del tipo de dato *JSONB* para almacenar los datos de adscripción de la tesela.

Tabla *grid*

La tabla ***grid*** almacena las características geométricas de las rejillas que se utilizan para la adscripción de la información. Su definición es:

```
1 create table grid(
2   grid_id varchar(64) primary key,
3   name varchar(150),
4   description text,
5   origin_epsg varchar(10),
6   origin_x float,
7   origin_y float,
8   zoom_levels jsonb[]
9 );
```

Código III.3.3: Tabla *grid*.

Las columnas son:

1. ***grid_id*:** el identificador único de la rejilla[17];
2. ***name*:** el nombre de la rejilla;
3. ***description*:** un descriptor para la rejilla;
4. ***origin_epsg*:** el código *EPSG* de la rejilla, es decir, el sistema de referencia en el que se define;
5. ***origin_x* y *origin_y*:** las coordenadas del origen de coordenadas de la rejilla, en unidades del sistema referenciado en la columna anterior;
6. ***zoom_levels*:** niveles de resolución definidos para la rejilla. Nótese que es de tipo *JSONB* con un par de corchetes detrás, lo que denota un tipo de dato matricial, es decir, una colección del tipo referenciado. Por lo tanto, los niveles de resolución se codifican a nivel de base de datos como una colección de *JSONB*.

[17] Un identificador único, comunmente llamado *clave primaria* (*primary key*), es un elemento fundamental en la metodología de modelado entidad-relación. La clave primaria de una tabla selecciona una o varias columnas que, en solitario o trabajando en ternas, permiten identificar de forma completamente unívoca a cada elemento (fila) de la tabla. Esta necesidad de identificación unívoca es lo que hace que las relaciones en el modelo sean posibles, fiables y con garantías de veracidad.

Los niveles de resolución de cada nivel de rejilla se definen de la siguiente manera:

```
{
  "name": "100 km",
  "size": 100000
}
```

Código III.3.4: Estructura *JSON* para la descripción de un nivel de resolución de una definición de rejilla.

Es decir, cada nivel de resolución tiene un nombre descriptivo y un tamaño de lado de tesela en las unidades del sistema de referencia de la rejilla.

Tabla *pg_connection*

Esta tabla cumple un papel meramente administrativo, sirviendo para almacenar los datos de conexión hacia las bases de datos *PostGIS* que contienen los datos de origen a teselar. Su definición es:

```
create table pg_connection(
  pg_connection_id varchar(64) primary key,
  name varchar(150),
  description text,
  application_name varchar(64),
  db varchar(64),
  host varchar(64),
  max_pool_size integer,
  min_pool_size integer,
  pass varchar(64),
  port integer,
  db_user varchar(64)
);
```

Código III.3.5: Tabla *pg_connection*.

donde ***pg_connection_id*** es un identificador único, ***name***, ***description*** y ***application_name*** son metadatos que identifican a la conexión y el resto de los parámetros son los parámetros necesarios para conectar a un servidor externo *PostgreSQL* y leer la información de origen.

Tabla *gridder_task*

Los *Gridder Task* son uno de los objetos de la plataforma más complejos y se explicarán en profundidad en posteriores capítulos. Por ahora, baste decir que un *Gridder Task* es un objeto que articula un proceso de adscripción de información de origen a la rejilla. Un *Gridder Task* define las características de los análisis de adscripción, generando en su ejecución la creación de una o varias variables de adscripción sobre la rejilla. Su definición es:

```
create table gridder_task(
  gridder_task_id varchar(64) primary key,
  gridder_task_type varchar(64),
  name varchar(150),
  description text,
  pg_connection_id varchar(64),
  source_table varchar(150),
  geom_field varchar(150),
  additional_params jsonb,
  index_variable_key varchar(64)
);
```

Código III.3.6: Tabla *gridder_task*.

donde:

1. ***gridder_task_id*:** es el identificador único del *task*;
2. ***gridder_task_type*:** hace referencia al tipo de análisis de adscripción;
3. ***name* y *description*:** son metadatos identificativos del *task*;
4. ***pg_connection_id*:** la conexión a los datos originales en una base de datos *PostgreSQL*, tal y como existen en la tabla ***pg_connection***;
5. ***source_table*:** hace referencia a la tabla en un origen de datos (definidos en la tabla ***pg_connection***, III.3.5) de la que se leeran los datos originales;
6. ***geom_field*:** identifica el campo geométrico dentro de la tabla referenciada en la columna anterior;
7. ***additional_params*:** parámetros adicionales de configuración del *task*. Nótese que es de tipo *JSONB*, lo que la hace ideal para contener un número variable de parámetros, dependiendo de las necesidades marcadas por el campo ***gridder_task_type***;
8. ***index_variable_key*:** la clave de la variable de adscripción índice.

Tabla *variable*

Esta tabla recoge el catálogo de variables de adscripción computadas para el conjunto de las teselas. Es el catálogo principal de información de adscripción del sistema, y sirve como base para la definición de los vectores temáticos de las teselas. Su definición es:

```
create table variable(
  gridder_task_id varchar(64) references cell_meta.gridder_task(
      gridder_task_id),
  variable_key varchar(64) unique,
  name text,
  description text,
  primary key (gridder_task_id, variable_key)
```

```
7 );
```

Código III.3.7: Tabla *variable*.

donde:

1. ***gridder_task_id*:** es el ID de la *Gridder Task* que generó esta variable;
2. ***variable_key*:** es la clave de la variable, es decir, la clave que se utilizará en el campo *JSON* temático de las teselas para almacenar la información de adscripción;
3. ***name* y *description*:** metadatos de identificación de la variable.

Tabla *catalog*

Esta tabla alberga información con estructura clave / valor para los catálogos de variables discretas o categóricas. El papel de este catálogo se explicará más adelante. Su definición es:

```
1 create table catalog(
2   gridder_task_id varchar(64),
3   variable_key varchar(64),
4   key varchar(64),
5   value varchar(500),
6   primary key (gridder_task_id, variable_key, key)
7 );
```

Código III.3.8: Tabla *catalog*.

donde:

1. ***gridder_task_id*:** identificador del *Gridder Task* que generó esa entrada de catálogo;
2. ***variable_key*:** clave de la variable a la que pertenece la entrada de catálogo;
3. ***key*:** clave de la entrada de catálogo;
4. ***value*:** valor de la entrada de catálogo.

Tabla *data*

Esta es la tabla más importante del sistema, puesto que guarda los resultados de adscripción. Extenderemos, a diferencia de las demás tablas, la descripción de su definición al conjunto de índices que se le han impuesto, de forma que se entiendan las razones en pos de la eficiencia y la velocidad en las consultas de recuperación de información teselada:

```
1 create table cell_data.data(
```

```
    grid_id varchar(100) references cell_meta.grid(grid_id),
    epsg integer,
    zoom integer,
    x bigint,
    y bigint,
    data jsonb,
    geom geometry(POLYGON),
    geom_4326 geometry(POLYGON),
    primary key(grid_id, zoom, x, y)
);

create index cell_data_grid_id_btree
on cell_data.data
using btree(grid_id);

create index cell_data_zoom_btree
on cell_data.data
using btree(zoom);

create index cell_data_x_btree
on cell_data.data
using btree(x);

create index cell_data_y_btree
on cell_data.data
using btree(y);

create index cell_data_data_gin
on cell_data.data
using gin(data);

create index cell_data_geom_gist
on cell_data.data
using gist(geom);

create index cell_data_geom_4326_gist
on cell_data.data
using gist(geom_4326);
```

Código III.3.9: Tabla *data*.

Cada uno de sus registros (filas) es la información de una tesela. Sus columnas son:

1. ***grid_id*:** el ID de la definición de rejilla que teseló la información;

2. ***EPSG*:** el sistema de referencia de la rejilla;

3. ***zoom*, *x* e *y*:** las coordenadas de la tesela, es decir, nivel de resolución y x e y dentro de ese nivel;

4. ***data*:** de tipo *JSONB* para almacenar la información del vector de adscripción. Todas las variables calculadas para la tesela estarán en formato *JSON* en este campo;

5. ***geom*:** la geometría de la tesela en formato *Simple Features* propio de *PostGIS*. Aunque como se ha remarcado muchas veces la definición de la tesela es matemática y su expresión geométrica no necesita de ser explícita, sin embargo, a la hora de desarrollar y depurar la plataforma, viene muy bien tenerla para visualizar y chequear la construcción de la rejilla en sistemas de información geográfica de escritorio como *QGIS*, además de tener la ventaja de disponer de los sofisticados índices de *PostGIS* sobre información geográfica para aumentar, en ciertas operaciones, la eficiencia de las consultas topológicas a la base de datos[18]. La tesela se guarda como polígono. Dada la simpleza de los mismos, el impacto a nivel de tamaño de la base de datos y rendimiento no es muy alto;

6. ***geom_4326*:** la geometría de la tesela, con las mismas propiedades de la anterior, pero reproyectada a WGS84 Geográficas. Este es el sistema de referencia por defecto de muchas librerías TIG, por lo que, a nivel de desarrollo, también merece la pena tener las teselas predefinidas en este sistema para aligerar ciertas consultas y procesos.

En cuanto a los índices, destacar que *PostgreSQL* ofrece distintos procedimientos de indexación. No son iguales los algoritmos para indexar información numérica, por ejemplo, que información geográfica, por lo tanto, *PostgreSQL* tiene que utilizar la metodología de indexación más apropiada para cada tipo de dato:

1. ***cell_data_grid_id_btree*:** este índice *BTREE* (apto para números y textos cortos) sobre el ID de la rejilla permite discriminar rápidamente entre teselas de distintas rejillas, en el caso de que en una misma instancia de *Cell* se mezclaran varias;

2. ***cell_data_zoom_btree*, *cell_data_zoom_x_btree* y *cell_data_zoom_y_btree*:** estos índices *BTREE* sobre las coordenadas de las teselas ayudan a encontrarlas rápidamente. Esto es fundamental a la hora de utilizar las posibilidades autoindexativas de la rejilla, ya que esta calcula relaciones topológicas inter-resolución para encontrar teselas padres e hijas en otros niveles de resolución. Una vez obtenidas, ser capaz de recuperarlas en esta tabla a una velocidad significativa es importante para el rendimiento del sistema. Después de todo, la recuperación de las teselas por sus coordenadas es una operación básica. En la recuperación de las teselas también se puede utilizar, con gran ventaja, el campo ***geom*** con las capacidades de indexación de *PostGIS*. Por ejemplo, recuperar todas las teselas que se muestran en un área de visualización en un visor a un nivel determinado de resolución;

3. ***cell_data_data_gin*:** uno de los índices más importantes de la base de datos y clave para el rendimiento del sistema en el aspecto alfanumérico de la base de datos. El método de indexación *GIN* (Smith, 2010) es un método relativamente reciente en *PostgreSQL* destinado a la indexación de información con estructuras laxas, como es el caso de *JSON*. Este índice genera un mapa de las claves presentes en el campo *JSON* y algunas de sus características temáticas, agilizando muchísimo la búsqueda

[18] Recordemos, no obstante, que una de las ventajas de la rejilla es su capacidad autoindexativa, por lo que esta ventaja también podría ser minimizada. No obstante, el coste/beneficio de guardar la geometría en sistemas de desarrollo y prueba es grande, por eso se mantiene.

de filas en esta tabla que contengan un conjunto de claves determinado. Las claves en los *JSON* contenidos en este campo identifican variables de adscripción, por lo que su presencia es fundamental para recuperar teselas con un determinado vector temático con eficiencia[19];

4. ***cell_data_geom_gist* y *cell_data_geom_4326_gist*:** estos dos índices de tipo *GIST* (ver figura III.3.2 son utilizados para la indexación de información geométrica por un procedimiento de árboles desbalanceados. Sin estos índices, el acceso a la información geométrica de la información geográfica se torna del todo inoperante. Estos índices utilizan un algoritmo muy ingenioso que detecta grupos de densidad variable de objetos geométricos y los agrupa. Con el índice, las operaciones topológicas marcadas por una opción de distancia se vuelven muy eficientes[20], por lo que son un gran apoyo adicional para calcular relaciones entre teselas y agilizar por ejemplo la recuperación de las mismas en ciertos escenarios. Además, como la información geométrica de la rejilla es tan compacta y regular, estos índices son especialmente eficientes y pequeños, a diferencia de los índices generados por un grupo de polígonos altamente irregulares en forma y tamaño (por ejemplo, una capa de usos del suelo o catastro).

III.3.4. La librería *SQL*

Este subsistema de base de datos para el almacenamiento de la información de gestión y de datos de adscripción no sólo lleva el modelo de datos comentado en sus tablas en las secciones anteriores, sino que también incorpora una librería de funciones de trabajo con la rejilla.

El desarrollo de una arquitectura de microservicios como esta plataforma se realiza por capas, a modo de cebolla, desde los microservicios más nucleares hacia el exterior, terminando en los componentes de visualización. En este caso, éste microservicio de base de datos sería la capa más interna, ya que alberga los datos y tiene que dar soporte a otros muchos microservicios, mientras que la capa más externa serían los visores para usuarios finales.

[19] Sin embargo, este no es el fuerte de *PostgreSQL* y hay sistemas más sofisticados para la indexación de este tipo de documento, especialmente ciertas bases de datos *NoSQL* como *MongoDB* (Jung et al., 2015).

[20] Por ejemplo, imaginemos una tabla con 10 millones de puntos. Si quisiéramos encontrar los puntos cercanos a otro en un radio de 100 metros, podríamos iterar los 9.999.999 puntos restantes calculándole la distancia al punto objetivo, y seleccionando aquellos que están por debajo de 100 metros. Esto es algo exasperantemente lento, ya que hay que chequear punto a punto, y el índice es inútil. Sin embargo, existe una función en *PostGIS*, *st_dwithin()*, que permite determinar con un sí/no si un punto está a menos de la distancia objetivo de otro. Esta operación sí utiliza el índice: si un grupo de un millón de puntos está, como conjunto, a más de 100 metros de distancia, es imposible que ninguno de sus puntos miembro esté a menos de 100 metros del objetivo. El sistema se acaba de ahorrar un millón de chequeos de distancia. Esto se va haciendo de una forma parecida a como está planteada la pirámide de resolución de la rejilla: con grupos cada vez más pequeños. Primero se descartan los grandes grupos de geometrías que están fuera de rango; para los que están dentro, se exploran sus subgrupos, y así se va descartando información hasta llegar a la solución final. La diferencia de usar los índices de forma eficiente puede ser de muchos minutos en la obtención del resultado en una consulta *SQL* espacial.

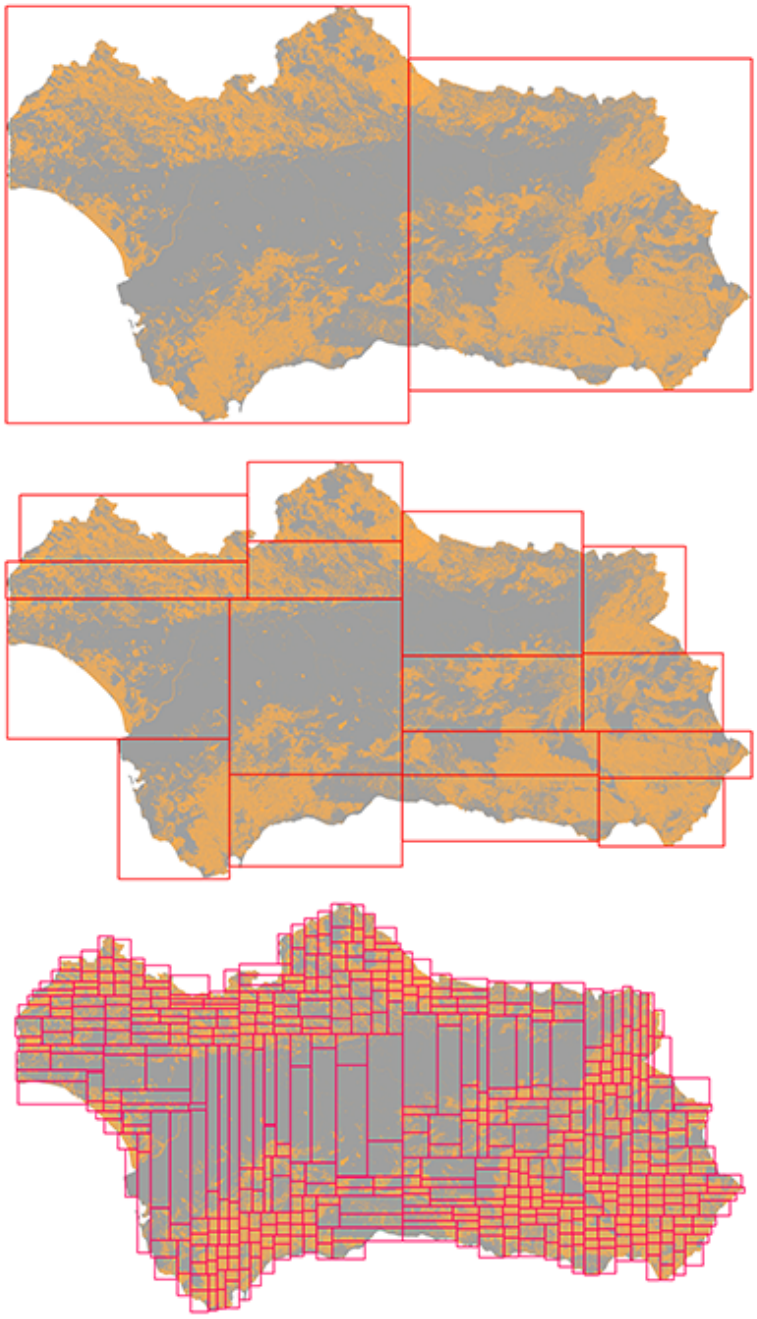

Figura III.3.2: Una aproximación a cómo indexa la información un índice de tipo *GIST*. Los datos mostrados son los de los Hábitats de Interés Comunitario, un conjunto de datos poligonales bastante denso. El índice *GIST* va subdividiendo el total de información en *clústers* de datos con aproximadamente la misma densidad de información en cada uno de ellos. En el primer nivel, el rectángulo de la izquierda tiene la misma densidad de datos que el de la derecha. Dentro de este primer nivel de separación se van haciendo, iterativamente, nuevos repartos igualitarios de densidad de información, creando en el proceso un árbol de relaciones entre cajas contenedoras-contenidas, de forma que todos los rectángulos del nivel final tienen la misma densidad aproximada. El índice guarda, para cada polígono, a qué rectángulo pertenece en cada nivel de índice. De esta manera, si buscáramos los polígonos a menos de X metros de distancia de un punto, primero se prueba la distancia a las cajas contenedoras, en orden descendiente de detalle. Si la distancia a la caja (una operación computacionalmente trivial) es mayor que la distancia objetivo, ningún polígono contenido en ellas puede estar a menos de la distancia. Como las cajas están anidadas, se van descartando niveles de indexación hasta llegar al único grupo de polígonos capaces de cumplir la condición de distancia. El incremento en rendimiento es abismal. Fuente: salida de un algoritmo de elaboración propia.

Además, la realización de pruebas se va complicando a medida que más y más microservicios se unen a la plataforma. La interacción entre los mismos, especialmente en entornos de computación paralela como éste, en donde están sucediendo muchas cosas en paralelo, aporta un extra de dificultad y confusión a la hora de enfrentarse a errores en la programación[21]. Por todo ello, la mejor forma de acercarse al desarrollo es hacerlo por capas, procurando siempre disponer, en cada una de ellas, de herramientas que permitan su depuración y testeo en un contexto aislado del máximo número de otros componentes o microservicios de la aplicación. Esto permite aislar los errores que son responsabilidad exclusiva de los procesos de la capa que está siendo desarrollada. También permite desarrollar nuevas pruebas de concepto más rápidamente.

La base de datos *PostgreSQL*, como base de datos extensible que es, permite al desarrollador no sólo implementar modelos entidad-relación, sino crear funciones que operan, dentro de la propia base de datos, con los datos que contiene. Aunque la inclusión de estas funciones en la base de datos es un tema controvertido a nivel de diseño de arquitectura de aplicaciones, como todo tiene sus pros y sus contras. Como contra, el hecho de que parte de la *lógica de la aplicación*[22] se encuentre dentro de la base de datos hace que el intercambio de un sistema de bases de datos por otro se dificulte, es decir, no es trivial volver a replicar dicha lógica en otro sistema de bases de datos distinto, perdiendo el sistema, por tanto, capacidad de migración a otras tecnologías. Como pro, el hecho de que algunos procesamientos críticos de datos se realice dentro de la propia base de datos, directamente junto al almacén de los mismos, y no por un microservicio externo que use la base de datos como mero reservorio de datos, puede imprimir rendimiento al sistema en algunos escenarios.

La principal razón de haber incluido esta librería es precisamente disponer de esas herramientas de testeo dentro del propio desarrollo del microservicio de la base de datos. La librería está creada casi exclusivamente con funciones *SQL*, que son las más ligeras y eficientes dentro de la oferta que ofrece *PostgreSQL* para la creación de este tipo de funciones de base de datos. Algunas que demandaban procesos un poco más complejos están escritas en el lenguaje *PL/PgSQL*[23].

Una función *SQL* tiene la siguiente sintáxis:

```
1 create or replace function cell__getgridorigin(
2   _grid_id varchar(100)
3 ) returns geometry as
4 $$
5   select st_setsrid(
6     st_makepoint(
7       origin_x,
8       origin_y
```

[21] Los famosos *bugs* de los programas. El término tiene una larga historia en ingeniería, no sólo en ingeniería de *software* como suele pensarse. Hay referencias a él en la década de 1870, en trabajos de Thomas Alva Edison (Shapiro, 2006).

[22] En ingeniería de *software*, se entiende por *lógica de la aplicación* aquellas partes del programa que operan con los datos para producir cálculos y resultados.

[23] *PostgreSQL* ofrece la posibilidad de escribir estas funciones en *Python*, *C* y *C++* e incluso *R*.

```
    ), origin_epsg::integer)
  from grid
  where grid_id = _grid_id
$$
language sql;
```

Código III.3.10: Función *SQL*.

Este código crea una función llamada ***cell___getgridorigin*** que va a devolver, como punto *PostGIS*, el origen de coordenadas de una rejilla almacenada en la tabla ***grid*** descrita anteriormente. Este ejemplo nos va a servir para definir el concepto de función en ingeniería de *software*, algo importante para entender posteriores capítulos.

Nótese como a continuación del nombre de la función se abre un cuerpo delimitado por paréntesis. Estos paréntesis (que en los casos más sencillos podrían estar vacíos, pero son necesarios para indicarle al interprete del código que se trata de la definición de una función), contiene los llamados **parámetros** de la función, es decir, los datos iniciales de trabajo con los que va a trabajar el código de la misma. Esta función en concreto acepta un único parámetro de tipo texto llamado ***__grid__id***, que se corresponde con el ID de la rejilla cuyo origen de coordenadas queremos obtener.

A continuación, después del cierre de paréntesis que delimita la definición de parámetros, viene la definición, tras la palabra reservada[24] **returns**, del tipo de dato de retorno de la función, es decir, el tipo de dato que la función va a devolver como resultado de su proceso sobre los datos iniciales proporcionados como parámetros. En este caso, esta función va a devolver un tipo *GEOMETRY*, propio de la extensión *PostGIS*, que define una geometría (en este caso será un punto).

En el bloque designado por el par de símbolos reservados ***$$*** se encuentra el código de la función propiamente dicho. En este caso, como es una función *SQL*, es una consulta en este lenguaje:

```
select st_setsrid(
  st_makepoint(
    origin_x,
    origin_y
  ), origin_epsg::integer)
from grid
where grid_id = _grid_id
```

Código III.3.11: Función *SQL*.

[24]En lenguajes informáticos, las *palabras y símbolos reservados* son las palabras y símbolos que conforman la sintáxis del lenguaje en cuestión y que son analizadas sintácticamente por el intérprete o compilador del lenguaje en un contexto sintáctico determinado para producir un efecto específico. La sintáxis de un lenguaje de programación define estas palabras y símbolos reservados, que no deben utilizarse fuera del contexto de sus reglas sintácticas, así como las reglas para combinarlas para producir los efectos de programación deseados.

Esta consulta extrae de la tabla ***grid*** (claúsula ***from***, línea 6) los datos de rejilla pertenecientes al ID que se le ha pasado como argumento a la función (claúsula ***where***, línea 7). De estos datos definitorios de la rejilla, ya comentados, se obtienen los datos ***origin_x*** y ***origin_y***, con los que se construye un punto geométrico (función de *PostGIS* ***st_makepoint()***, líneas 2-5), al que, finalmente, se le asigna el sistema de referencia[25] de la rejilla (función de *PostGIS* ***st_setsrid()***, líneas 1-5) a partir del campo de la tabla ***grid origin_epsg***. Este es el resultado final del procesamiento y es la geometría que se devuelve, tal y como se especificó en el ***returns*** de la cabecera de la función.

Este conjunto de funciones definidas junto al modelo de datos conforma una *API* de servicio de la propia base de datos. En la documentación de las *API* se listan precisamente los elementos reseñados anteriormente: la lista de las funciones que la conforman, explicando para cada una la naturaleza de sus parámetros y de su tipo de devolución, así como su función.

A continuación listamos las funciones de la *API* de la base de datos más importantes. El listado completo se puede obtener del análisis del código encontrado en los repositorios GitHub:

cell___bboxfromcorners(_bbox float[]) returns geometry

Toma como argumentos las coordenadas de una caja en el formato [coordenada x inferior, coordenada y inferior, coordenada x superior, coordenada y superior] y devuelve el polígono de la caja como tipo *geometry* de *PostGIS*.

cell___getgridorigin(_grid_id varchar(100)) returns geometry

Devuelve como punto geométrico el origen de la rejilla identicada con el parámetro *_grid_id*.

cell___getgridsrs(_grid_id varchar(20)) returns integer

Devuelve como número el código *EPSG* del sistema de referencia de la rejilla especificada con el parámetro *_grid_id*.

cell___getzoomlevelsize(_grid_id varchar, _zoomlevel integer) returns float

Devuelve el tamaño de lado de tesela del nivel de resolución ***_zoomlevel*** para la rejilla identificada por el ID ***_grid_id***.

cell___cellgeom(_cell cell___cell) returns geometry

Devuelve la geometría de una tesela definida por un tipo *cell___cell*.

cell___cellcenter(_cell cell___cell) returns geometry

Devuelve el centro como punto de la tesela definida por un tipo *cell___cell*.

[25] El estándar *Simple Features*, en el que se basa el funcionamiento de *PostGIS*, recomienda que las geometrías lleven siempre el código *EPSG* del sistema de referencia en el cual están expresadas sus coordenadas.

cell___cellgeom4326(_cell cell___cell) returns geometry

Devuelve la geometría de la tesela definida por un tipo *cell___cell* en sistema de coordenadas WGS84 geográficas.

cell___cellonpoint(_grid_id varchar(20), _zoom integer, _point geometry) returns cell___cell

Devuelve, como tipo *cell___cell*, la tesela en el nivel de resolución ***_zoom*** en la que cae una geometría puntual definida por el parámetro ***_point*** en la rejilla con ID ***_grid_id***.

cell___getbboxcoverage(_grid_id varchar(20), _zoom integer, _geom geometry) returns setof cell___cell

Devuelve todas las teselas (de ahí el modificador de retorno marcado con la palabra reservada ***setof***) mediante el tipo ***cell___cell*** que colisionan con la caja[26] de la geometría ***_geom***, teniendo el cuenta el nivel de resolución ***_zoom*** y la rejilla con ID ***_grid_id***.

cell___getcoverage(_grid_id varchar(20), _zoom integer, _geom geometry) returns setof cell___cell

Parecida a la función anterior, esta devuelve todas las teselas que recubren no la caja de una geometría, sino la geometría propiamente dicha. Existen las dos funciones porque la primera es infinitamente más rápida que esta (en varios órdenes de magnitud), ya que la anterior calcula, con la función *cell___cellonpoint()* en qué teselas caen las esquinas de la caja de la geometría y simplemente devuelve todas las teselas que se encuentran entre esas dos en las dos dimensiones, sin detectar colisiones con la geometría. Esta función, sin embargo, se vale de la anterior para adquirir todas las teselas que potencialmente colisionan con la geometría, pero después tiene que hacer chequeos sobre la colisión propiamente dicha. Digamos que la función anterior ejerce de índice para esta.

cell___setcell(_cell cell___cell) returns void

Esta es una de las funciones más criticas de la librería y de todo el sistema, ya que se encarga de actualizar o crear el vector de adscripción de una tesela. Tomando un tipo ***cell___cell***, esta función inserta la tesela en la tabla ***data*** si esta no existiera y escribe las variables de adscripción que pueda tener el dato de entrada a la función. Si la tesela existe, actualiza cualquier variable de adscripción que ya puediera existir en el vector temático de la tesela ya existente en la tabla ***data***, y crear cualquiera que no existiera aún.

cell___getcellsbyvarkeys(_grid_id varchar(20), _varkeys varchar[], _and boolean, _zoom integer, _geom geometry) returns setof cell___cell

Esta función devuelve las teselas cuyo vector de adscripción contiene las claves de variables de adscripción presentes en la lista ***_varkeys[]*** para la rejilla con ID ***_grid_id***. Por defecto, se devuelven las teselas que tienen todas las variables referenciadas[27], pero

[26] La caja (*bounding box*) de una geometría es el rectángulo mínimo que la incluye en su totalidad, designado por las coordenadas máximas y mínimas en x e y de todas los puntos que definen la geometría.

[27] Es decir, se usa una condición booleana ***AND***, para obtener sólo aquellas que tienen el vector completo.

se puede cambiar a una condición booleana ***OR*** poniendo el parámetro ***__and*** a falso. Se puede filtrar opcionalmente por el nivel de resolución con el parámetro ***__zoom*** y/o con una geometría que defina un ámbito geográfico con el parámetro ***__geom***.

cell___getvariablekeysbygriddertaskid(__griddertaskid varchar(64)) returns varchar[]

Devuelve todas las claves de las variables de adscripción generadas por un análisis de adscripción de ID ***__griddertaskid***.

cell___getindexvariablekeybygriddertaskid(__griddertaskid varchar(64)) returns varchar

Devuelve la clave de la variable de adscripción índice para el análisis de adscripción con ID ***__griddertaskid***.

III.4 Definicion de la rejilla

Como se explicó en el apartado III.3.3 (pág. 154) del capítulo anterior, la rejilla se almacena en la base de datos en la tabla *grid*. Esta tabla recoge las características de la o las rejillas a utilizar en la adscripción de la información original.

Cabe destacar que en el caso de bases de datos *Cell* que tienen más de una rejilla, las teselas que generan comparten la misma tabla *data* (III.3.9, pág. 158), diferenciándose entre sí por su campo *grid_id*.

Las rejillas se introducen en la base de datos con una instrucción *SQL* como la siguiente:

```
insert into grid
values (
  'eu-grid',
  'eu-grid',
  'A grid based on the official EU one',
  '3035',
  2700000,
  1500000,
  Array[
    '{"name": "100 km", "size": 100000}'::jsonb,
    '{"name": "50 km", "size": 50000}'::jsonb,
    '{"name": "10 km", "size": 10000}'::jsonb,
    '{"name": "5 km", "size": 5000}'::jsonb,
    '{"name": "1 km", "size": 1000}'::jsonb,
    '{"name": "500 m", "size": 500}'::jsonb,
    '{"name": "250 m", "size": 250}'::jsonb,
    '{"name": "125 m", "size": 125}'::jsonb,
    '{"name": "25 m", "size": 25}'::jsonb,
    '{"name": "5 m", "size": 5}'::jsonb
  ]
);
```

Código III.4.1: Inserción de una rejilla en la base de datos.

Ésta es la rejilla utilizada en las adscripciones realizadas en este trabajo:

1. su nombre (campo *name* de la tabla *grid*) e identificador (campo *grid_id* de la tabla *grid*) son "eu-grid", para destacar su compatibilidad con la oficial europea;

2. su sistema de referencia (campo *origin_epsg* de la tabla *grid*) es la proyección oficial

europea Lambert Azimutal Equiárea (*EPSG* 3035);

3. su origen de coordenadas (campos *origin_x* y *origin_y* de la tabla *grid*) está en el punto (2700000, 1500000) (en el Golfo de Cádiz);

4. dispone de 10 niveles de resolución (campo *zoom_levels* de la tabla *grid*): 100, 50, 10, 5 y 1 kilómetros, así como 500, 250, 125, 25 y 5 metros. Recordemos que los niveles de resolución no deben tener decimales y deben ser divisores perfectos de sus resoluciones antecesoras.

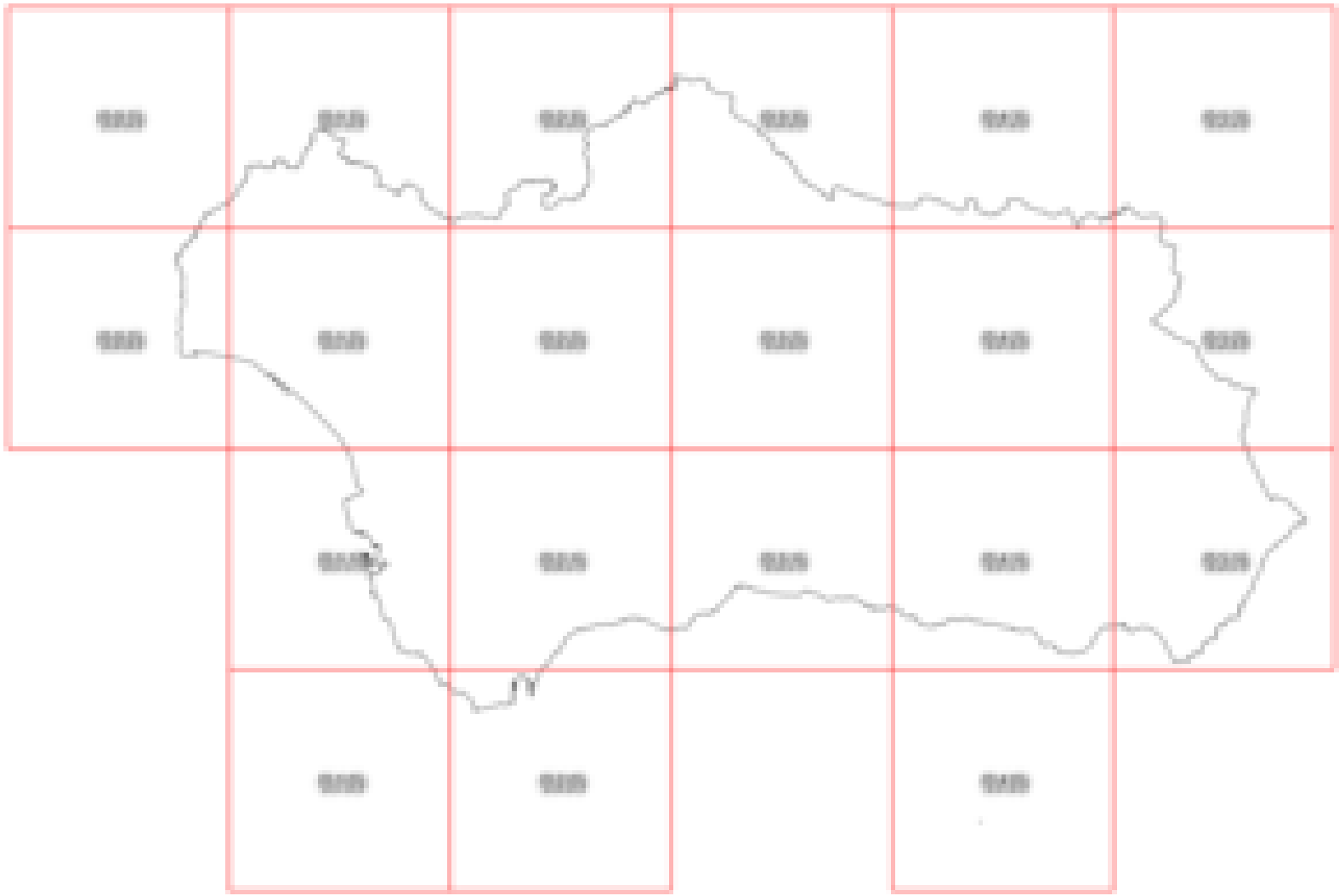

Figura III.4.1: La rejilla descrita en este capítulo, “eu-grid”, con las teselas de resolución 100 kilómetros colisionantes con Andalucía. Nótese las coordenadas [R,X,Y] de las mismas. Fuente: elaboración propia.

Figura III.4.2: Igual que la figura III.4.1, pero para el nivel de resolución 50 kilómetros. Fuente: elaboración propia.

Figura III.4.3: Igual que la figura III.4.1, pero para el nivel de resolución 10 kilómetros. Fuente: elaboración propia.

Figura III.4.4: Igual que la figura III.4.1, pero para el nivel de resolución 5 kilómetros. Fuente: elaboración propia.

Figura III.4.5: Igual que la figura III.4.1, pero para el nivel de resolución 1 kilómetro. Fuente: elaboración propia.

III.5 Datos originales

En este capítulo vamos a desarrollar las características y estructura de los datos originales utilizados en los procesos de adscripción. Se han utilizado en las pruebas de adscripción datos puntuales y poligonales, cuyas características ya han sido comentadas en la capítulo II.2 “Datos originales” (pag. 51).

III.5.1. Población

Los datos de población en rejilla publicados por el DERA (Instituto de Cartografía y Estadística de Andalucía (IECA), 2021a) se sirven publicados en formato *SHAPEFILE*, una por cada año en la serie de datos, siendo su geometría poligonal una tesela derivada de la oficial europea de 250 metros de lado y su información temática, como es normal en las *SHAPEFILE*, en una única tabla de datos. El sistema de referencia utilizado es ETRS89 UTM 30 Norte.

Figura III.5.1: Los datos originales de población, en forma de celdas de 250 metros de lado compatibles con la rejilla oficial europea, con el centroide de cada celda, que será el anclaje geométrico del proceso de adscripción. Fuente: elaboración propia.

Figura III.5.2: Cobertura autonómica de los datos originales de población. Fuente: elaboración propia.

La rejilla europea es compatible con la rejilla utilizada en las pruebas de adscripción. La información alfanumérica está bien normalizada gracias a la estructuración impuesta por la propia rejilla en la que se publica, existiendo valores nulos para el valor estadístico representados con "-1", dado que los *SHAPEFILE* no tienen la noción *NODATA*. La fuente de datos está excepcionalmente bien documentada, describiendo cada campo con detalle. La nomenclatura de los campos es clara y consistente, utilizándose una clave temática (por ejemplo, *ptot* para la población total) seguida de un numeral de dos cifras para describir el año del dato, conformando de esta manera un juego de datos para cada categoría temática, obteniéndose una línea temporal completa. Obviamente, al ser un formato monotabla, se incurre en desnormalizaciones propias de esta limitación, pero están bien acotadas y su procesamiento no implica ningún riesgo o trabajo adicional. Es una fuente, por tanto, de excelente calidad y en cuyo procesado no se ha encontrado ninguna dificultad ni discrepancia. Su estructura de datos se ajusta perfectamente a las necesidades del análisis de adscripción de nubes de puntos con información alfanumérica continua. Para llevarlo a cabo, se tomará como geometría de referencia para la adscripción el centroide de la tesela del polígono original. Es decir, aunque la capa sea originalmente poligonal, su tratamiento será puntual. Los datos tienen un recubrimiento muy parcial del área de estudio, limitándose a áreas con una cantidad de población que permita superar las limitaciones del secreto estadístico.

El tamaño del juego de datos completo, con toda la serie temporal, es de 310 *megabytes*. La capa contiene, teniendo en cuenta toda la serie histórica, 46.100 polígonos cuadrángulares distintos de 250 metros de lado. Si los tratamos como puntos, calculando el centroide de cada uno de ellos, que es como finalmente serán procesados durante la adscripción, se obtiene una densidad de 0,5 puntos de datos por kilómetro cuadrado para todo el área de estudio.

III.5.2. Modelo Digital del Terreno (MDT)

Esta fuente, como la anterior, también procede del DERA y está publicada en excelentes condiciones técnicas. La documentación es abundante y precisa, y el dato está muy bien tratado a nivel técnico. Su formato de publicación es un *GeoTIFF* con la cabecera[1] bien conformada, con información precisa de georreferenciación y con el tratamiento del valor nulo bien realizado y documentado. La resolución original es de 100 metros, en sistema de referencia ETRS89 UTM 30 Norte, y la unidad de la variable son metros. A esta capa, al igual que la anterior, se la transformará en una nube de puntos tomando el centroide del píxel del ráster, para así obtener una nube de puntos sobre la que aplicar un análisis de adscripción especialmente diseñado para adscribir este tipo de datos. Los datos tienen un recubrimiento completo del área de estudio. Remitimos al lector a las figuras mostradas en la parte de metodología II.2.2, II.2.3 y II.2.4 (pág. 55, 56 y 57, respectivamente).

En origen, esta capa tiene un tamaño, en *GeoTIFF*, de 67 *megabytes*. El formato *GeoTIFF* es capaz de realizar compresiones no destructivas con *LZW* o *PNG*, por lo que los datos, descomprimidos, tienen un peso real de 564 *megabytes*. Tiene 8.760.252 datos en la matriz, por lo que se obtienen los lógicos 100 puntos por kilómetro cuadrado en todo el área de estudio.

III.5.3. Hábitats de Interés Comunitario (HICS)

Esta fuente de datos está publicada en la Red de Información Ambiental de Andalucía (REDIAM) de la Agencia de Medio Ambiente y Agua (AMAYA) de la Junta de Andalucía (REDIAM, 2021) y tiene como base geométrica un recubrimiento no completo, pero sí bastante intrincado, de gran parte de la zona de estudio. El sistema de referencia es ETRS89 UTM 30 Norte.

Los datos originales vienen proporcionados en una *File Geodatabase* de *ESRI*. A pesar de ser un formato multitabla, la capa viene en una única tabla y, por tanto, su grado de normalización es bajo. La tabla principal de geometrías viene acompañada de una tabla de catálogo donde se recogen los códigos utilizados para codificar cada tipo de hábitat, con unos códigos estandarizados.

La tabla principal está fuertemente desnormalizada. Existe una relación de uno a muchos entre los polígonos y el catálogo de hábitats, por lo que se ha optado por la técnica de incluir varios datos atómicos dentro de un único campo de la tabla principal. De esta manera, si el polígono A tiene tres hábitats, se ha codificado la información de esta manera:

```
CODIGO_HABITAT_1_XX,CODIGO_HABITAT_2_YY,
CODIGO_HABITAT_3_ZZ
```

donde XX, YY y ZZ son el porcentaje estimado de preponderancia de ese hábitat en el polígono. Esto dificulta la transformación de esta fuente de datos. Esta fuente será trans-

[1] Las cabeceras de los *GeoTIFF* son fácilmente accesibles con el comando *gdalinfo* de la librería *GDAL*.

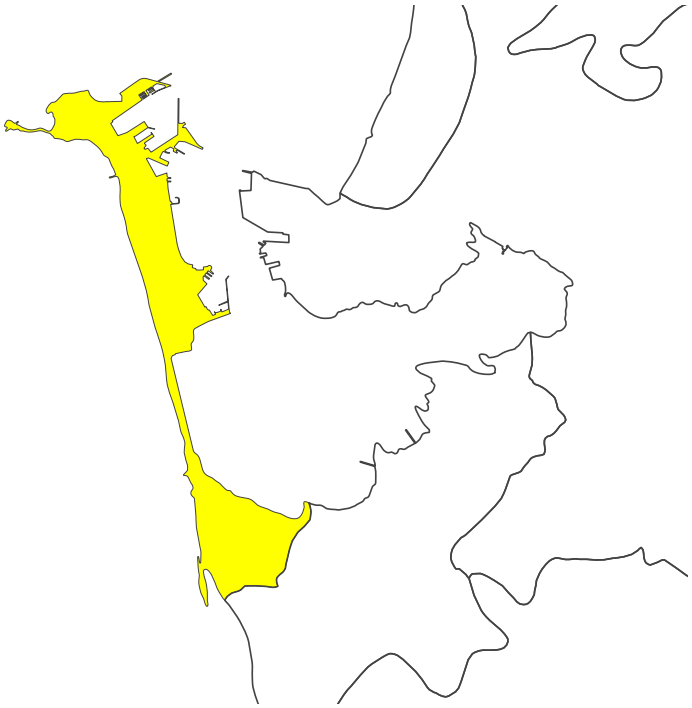

Figura III.5.3: El municipio de Cádiz es el menos esférico de Andalucía, con un valor de esfericidad de 0,22. Fuente: elaboración propia.

formada para ser utilizada con un análisis de adscripción para polígonos con categorías temáticas asociadas que genera el sumario de áreas de cobertura de cada una de dichas categorías en la tesela.

La capa tiene un peso original de 711 *megabytes*, y se compone de aproximadamente 760.543 polígonos, cuyos segmentos perimetrales tienen una longitud media de 8 metros, es decir, de media, encontramos una coordenada cada 8 metros, lo que atestigua la gran escala de significación de la capa. La media de área por polígono es de 4 hectáreas, es decir, son micropolígonos. La esfericidad media[2], sin embargo, es de 0,6, bastante alta para lo que cabría esperar por la forma que tienen.

III.5.4. Catastro

El catastro (Dirección General del Catastro, 2021) es la fuente de datos original más masiva adscrita y tratada, tanto en tamaño bruto como en complejidad de tratamiento. La capa recubre completamente el área de estudio.

El procedimiento completo de tratamiento de catastro puede encontrarse en Pérez Al-

[2]La esfericidad es un ratio geométrico que pone en relación el área de un polígono con el área de un círculo que posee el mismo perímetro. Dado que el círculo es la forma poligonal más compacta, es decir, la que alberga mayor área con el menor perímetro, este ratio, que se mueve entre 0 y 1, denota lo "redondo" o compacto que es un polígono. 0 sería nada compacto, mientras que 1 sólo lo alcanza el círculo. Su formulación matemática es $\frac{2\sqrt{\pi a}}{p}$, donde "a" es el área del polígono y "p" su perímetro.

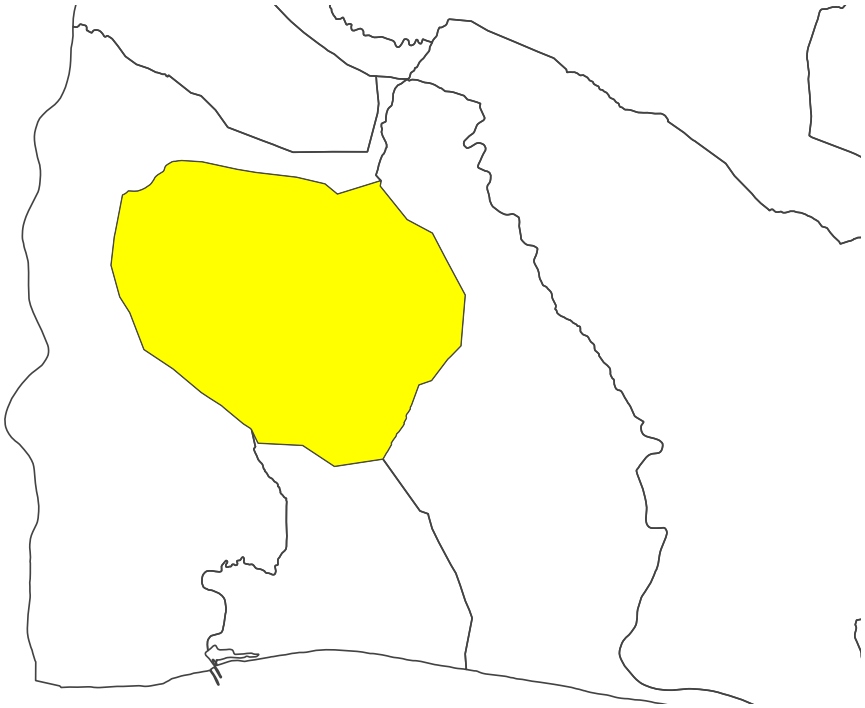

Figura III.5.4: El municipio de Villablanca (Huelva) es, por contra, el más esférico, con un valor de esfericidad de 0,91. Fuente: elaboración propia.

cántara, Díaz Cuevas et al., 2016 y Noguero Hernández et al., 2016. El resultado final de la integración del catastro sobre un único juego de datos en continuo en la Comunidad Andaluza ha tenido como objetivo la generación de un catálogo de indicadores catastrales enfocados a la determinación de la ocupación de las viviendas. Referimos al lector a la sección II.2.1, "Catastro", pág. 62, donde se hace una exposición detallada de los procedimientos que se describen a continuación.

La información vinculada al espacio residencial ha sido obtenida a partir del Catastro Inmobiliario, que se publica en formato *SHAPEFILE*, en dos grandes juegos de datos (*parcela* para las parcelas y *constru* para las construcciones) e información alfanumérica en un formato de longitud de línea fija, conocido como el formato catastral *CAT*. Dichas fuentes de datos son descargables desde la página *web* oficial de la Sede Electrónica del Catastro, en unidades municipales que después hay que poner en continuo en una única base de datos *PostgreSQL*.

El análisis en profundidad de la normalización del catastro es compleja y excede los objetivos de este trabajo, aunque se pueden consultar en las citadas referencias. Sin embargo, se puede decir que la casuística es grande, tanto a nivel alfanumérico como geométrico. Geométricamente, se producen incongruencias como parcelas catastrales solapantes o huecos en el tejido catastral. En el plano alfanumérico, las incidencias también son notables, como por ejemplo repeticiones de códigos catastrales, incongruencias en los códigos INE municipales, etc.

El problema principal, descrito en la bibliografía, de trabajo con esta fuente de datos para encontrar los bienes inmuebles residenciales es que éstos solo pueden asociarse espacialmente a la geometría de las parcelas, y no a las de las construcciones. Si bien esto es un problema menor en entornos de alta densidad urbana, donde la parcela y las construcciones son más o menos coincidentes en muchas de ellas, en ámbitos rurales, donde las parcelas pueden ser grandes y alojar diversas construcciones, se convierte en una dificultad. Los datos relativos a la superficie construida sobre rasante, el número de usos distintos o la antigüedad de los inmuebles de cada parcela han sido también calculados.

En Pérez Alcántara, Díaz Cuevas et al., 2016, partiendo de la hipótesis que "todo inmueble de uso residencial tiene que estar asociado a un espacio construido dentro de la parcela", se describe un método para zonas rurales donde se usa el *SHAPEFILE* de construcciones catastrales dentro de la parcela que tiene asociada la información alfanumérica de interés para ubicar el punto que agregue la información del número de inmuebles residenciales de dicha parcela, siempre que la construcción tenga una o más plantas. Posteriormente se aplica el algoritmo siguiente:

1. cuando existe un solo recinto geométrico en el *SHAPEFILE* de construcciones (con 1 o más planta) dentro de la parcela, la información se asigna al centroide de este elemento y no de la parcela;

2. cuando existen varios recintos, la información se asigna al centro de gravedad entre los mismos, ponderado por el volumen constructivo.

Tanto el proceso de descarga de la información geométrica y alfanumérica correspondiente a 2013, como el modelo de datos diseñado para la integración de todo este conjunto de datos en un sistema gestor de base de datos espaciales *PostgreSQL/PostGIS* que optimice las posibilidades de consulta, explotación y actualización, así como los diferentes procedimientos de explotación de la base de datos espacial creada, queda recogido en Noguero Hernández et al., 2016.

Para este trabajo, se han seleccionado aquellas parcelas con al menos un inmueble donde exista una construcción cuyo destino (uso) sea vivienda, extraído de la Tabla 14 (*constru*), incorporando el número de inmuebles con estas características como indicador de vivienda. Este indicador (denominado "pu029" en Pérez Alcántara, Díaz Cuevas et al., 2016) mostró mejores resultados que el basado exclusivamente en el número de inmuebles con uso dominante vivienda (extraído de la tabla 15 del formato *CAT*).

La información catastral se tratará para su adscripción mediante un análisis para nubes de puntos con variables continuas asociadas, puesto que se utilizarán los centroides de las construcciones, que tienen asignados los datos de indicadores.

En origen, los datos derivados del proyecto anteriormente citado ocupan 7,5 *gigabytes*. Como lo que se van a adscribir son los centroides constructivos volumétricos de las parcelas, daremos estadísticas de puntos, no de polígonos. La capa, considerando dichos puntos, posee un peso de 500 *megabytes*, estando compuesta por 2.263.375 puntos, alcanzando una densidad de 25 puntos por kilómetro cuadrado en la zona de estudio.

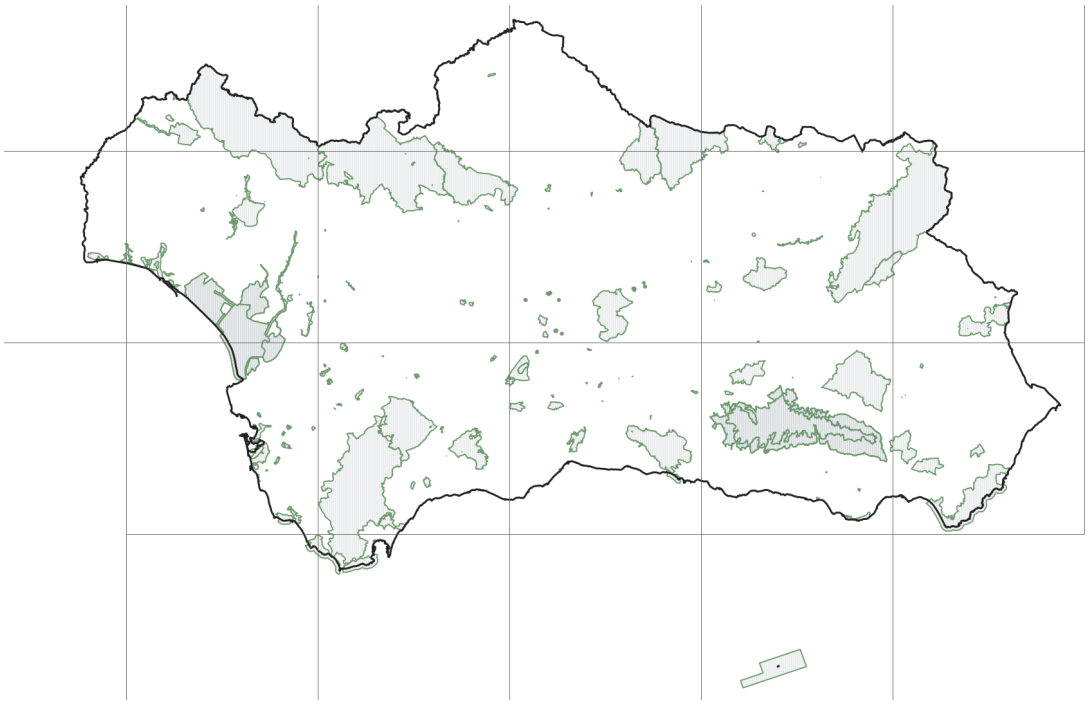

Figura III.5.5: Espacios Naturales Protegidos de Andalucía. Las teselas mostradas de referencia corresponden al nivel de resolución de 100 kilómetros. Elaboración propia.

III.5.5. Otra información de contexto

Aparte de estos cuatro conjuntos de datos, que forman el grueso de la aplicación temática objetivo de este trabajo, se han adscrito a la rejilla otras capas de información de contexto para servir de referencias, contraste a la hora de evaluar los análisis de adscripción y poder crear filtros territoriales posteriormente sobre los vectores de adscripción. Todos han sido extraídos del DERA, y todas son fuentes muy bien documentadas y normalizadas, publicados en formato *SHAPEFILE*:

1. **Espacios Naturales Protegidos:** capa poligonal, se le aplicará un análisis de tipo sumario de áreas de categorías asociados a polígonos;
2. **núcleos de población:** ídem;
3. **provincia:** ídem;
4. **sección censal:** ídem;
5. **municipio:** ídem, aunque además se le aplicó, ya que fue la primera fuente con la que se experimentó, el primer método de adscripción implementado, muy sencillo, que simplemente asigna a la tesela la categoría temática predominante de las presentes en la misma. En este caso, dicha categoría era el nombre del municipio.

Las estadísticas de estas capas son las siguientes:

Tabla III.5.1: Estadísticas de capas de contexto (poligonales). Fuente: elaboración propia.

Capa	Tamaño original (*MB*)	Polígonos	Segmento perimetral medio (m)	Área media (ha)	Esfericidad media
Provincia	3	8	73	1.094.926	0,51
Municipio	13	771	105	11.362	0,63
Núcleo población	22	13.348	38	18	0,53
EENNPP	6	205	94	9.878	0,59

III.5.6. *ETL* a *PostGIS*

Uno de los principales objetivos técnicos de esta metodología es asegurar la reproducibilidad de los procesos de análisis de datos. La reproducibilidad es una de las características más importantes de las técnicas de lo que se ha venido en llamar *ciencia de datos*, y se refiere a que los procesos sobre los datos han de ser posible reproducirlos en igualdad de condiciones sobre datos distintos (pero con el mismo esquema, lógicamente) y/o en entornos distintos. Esto obedece a la oficiosa *regla de los 2 minutos*: si en la máquina de desarrollo (un ordenador pequeño, como un portátil) un proceso dura más de 2 minutos mientras se está desarrollando o probando su viabilidad técnica, debe reducirse el volumen de los datos de prueba con el que se está trabajando. Una vez funcione en el entorno de desarrollo con los datos de prueba, el proceso se reproduce con el juego completo en una máquina *de producción* (un servidor o una torre de sobremesa), más potente. El procedimiento debe ser el mismo, sin modificaciones. Lo único que cambia es el contexto de ejecución, y esta reconfiguración del contexto desarrollo - producción se puede conseguir por diversos medios. Este autor lo hace con una herramienta de producción propia que utiliza entornos jerarquizados de variables de entorno que reconfiguran el contexto de ejecución.

ETL es un término que en proceso de datos significa *Extract, Transform, and Load*, es decir, *extraer, transformar y cargar* (Vassiliadis et al., 2002). Este término hace referencia a las distintas técnicas y *software* especializados en leer información de un sistema o formato fuente, transformarlas mediante técnicas de filtrado, control de calidad, etc. y volver a cargarlas en otro sistema o escribirlas en otro formato de salida. Existen muchos sistemas de este tipo, pero en TIG de *Software Libre* el más utilizado es la librería *GDAL / OGR*. Este sistema es capaz de leer muchísimos formatos distintos, así como escribir otros tantos[3]. Por ejemplo, con *GDAL* se puede leer fácilmente información en formato *SHAPEFILE* y escribirla en una base de datos *PostGIS*.

[3]El número de formatos en modo lectura, sin embargo, es mayor que el de escritura. Téngase en cuenta que muchos formatos de alto rendimiento, como por ejemplo los formatos de imágenes *MrSID* o *ECW*, con algoritmos de compresión muy sofisticados, están sujetos a licencias comerciales y no se proporcionan gratuítamente en modo escritura, pero sí en modo lectura. A las compañías desarrolladoras de estos algoritmos les suele interesar que la información codificada con ellos sea legible gratuítamente, no así su generación. Por lo tanto, siempre que existan *drivers* gratuítos (ojo, gratuítos no significa de *Software Libre*), *GDAL* es capaz de leer el formato, pero no escribirlo, a menos que se tenga una licencia comercial.

Aunque *GDAL / OGR* es una librería que está en el núcleo de muchos sistemas de TIG[4] para la gestión de la importación / exportación de distintos formatos, a nivel básico incluye una serie de sencillas utilidades de línea de comando[5] que permiten acceder a la funcionalidad de la librería. Esto permite hacer *scripts BASH*[6] (Newham et al., 2005), que garantizan la reproducibilidad de los procesos de importación y exportación que usan la librería. Una instalación de *GDAL / OGR*, por otra parte, es altamente modular, ya que el usuario puede seleccionar qué *drivers* avanzados instalar. Por ejemplo, los *drivers* de *PostGIS* son opcionales, por lo que puede haber instalaciones de la librería que no los tengan activos. Para garantizar la reproducibilidad de los procesos *ETL* sobre los datos originales se ha creado una imagen *Docker*[7] con una instalación de la librería. Es sobre esta imagen *Docker* sobre la que se lanzan los mencionados *scripts* para que se ejecuten siempre sobre el mismo entorno de instalación de la librería, asegurando así una altísima reproducibilidad del proceso en las condiciones originales (Boettiger, 2015).

El primer paso de los procesos *ETL* (la "E") es utilizar generalmente *GDAL* para leer el formato original y volcar la información bruta, sin procesamiento (es decir, sin entrar en la "T"), en la base de datos, en un esquema de importación, es decir, la "L" de *Load*. En puridad, el procedimiento aquí descrito es más bien un procedimiento *ELT* que *ETL*, ya que la transformación se hace en la base de datos, posteriormente a la carga.

Una vez los datos están en bruto en la base de datos se utiliza procesamiento *SQL* para transformarlos y escribirlos finalmente en su esquema de producción, desechando la información en bruto del esquema de importación que escribió *GDAL*.

Para organizar el trabajo con las fuentes de datos, se puede encontrar entre los repositorios *Git / DVC* de datos de las tesis una serie de repositorios dedicados al flujo de trabajo *ETL* de cada una de las fuentes de datos originales contempladas:

1. ***malkab-phd-data-cell_raw_data*:** este es el repositorio principal de datos, que crea la base de datos sobre la que van a trabajar el resto de subrepositorios de datos;

2. ***malkab-phd-data-context_data*:** *ETL* sobre datos de contexto (municipios, es-

[4] Y no sólo de *Software Libre*, ya que por ejemplo *ArcGIS* también utiliza *GDAL*.

[5] Por *línea de comando* se entiende en ingeniería de *software* la forma de interaccionar con una máquina no con un interfaz gráfico de ratón y ventanas, sino con una consola de texto mediante la cual se le envía a la máquina comandos con un formato muy específico que permite realizar acciones concretas. Aunque pueda parecer un método primitivo y lento de interaccionar con la máquina, a nivel profesional es un entorno tremendamente productivo, flexible y potente, siendo a menudo la única forma de interaccionar, por ejemplo, con máquinas remotas, como servidores.

[6] *BASH* (*Bourne Again SHell*) es una de las consolas de comandos más utilizadas en el mundo *MacOS*, *Linux* y derivados. Está dotada de un excelente catálogo de funcionalidad y es posible programar sofisticados *scripts* en ella, aunque la sintaxis del lenguaje es bastante extraña y contraintuitiva.

[7] *Docker* es, posiblemente, la tecnología más popular actualmente en el campo de la *contenerización*. Esta nueva metodología y herramienta del mundo de las *DevOps* está cambiando la forma de trabajo en muchos ámbitos de la ingeniería del *software*. Sería muy largo explicar en qué consiste y sus aplicaciones, así que baste decir que, en lo que a este trabajo atañe, permite una gestión de los microservicios mucho más limpia y encapsulada, un flujo desarrollo-producción reproducible y con infinitamente menos errores que metodologías anteriores y, sobre todo, una forma fantástica de hacer los procesos de tratamiento de datos altamente portables y reproducibles. Para este autor, y para casi todo el mundo en el sector de las *IT* que conoce, se ha convertido en una herramienta absolutamente insustituible.

pacios naturales protegidos, etc.) y el MDT;

3. ***malkab-phd-data-proyecto_nacional_data*:** *ETL* sobre los datos de catastro;
4. ***malkab-phd-data-hic*:** *ETL* sobre los Hábitats de Interés Comunitario;
5. ***malkab-phd-data-poblacion*:** *ETL* sobre los datos de población.

III.5.7. Creación de la base de datos principal para los datos de origen

Esta base de datos, que también es una *PostGIS*, no debe confundirse con la base de datos principal de la plataforma *Cell*. Esta base de datos no tiene nada que ver con la misma, y sencillamente es una instancia[8] de *PostGIS* destinada a contener y hacer operaciones *ETL* con las fuentes de datos originales. Esta instancia será la que utilice posteriormente la plataforma para leer los datos originales y adscribirlos, escribiendo el resultado final de los análisis de adscripción, las variables de adscripción, en la instancia *PostGIS* propia de la plataforma.

III.5.8. *ETL* de datos de contexto

Lo primero es importar a la base de datos los datos de contexto comentados anteriormente: provincias, municipios, núcleos de población y Espacios Naturales Protegidos, todos procedentes del DERA.

Todas estas fuentes de datos se proporcionan en formato *SHAPEFILE* y su esquema de datos es muy simple:

1. **provincias, municipios y núcleos de población:** geometría poligonal y un campo de nombre, es decir, un campo discreto categórico;
2. **Espacios Naturales Protegidos (EENNPP):** geometría poligonal y dos campos de catálogo, uno con el nombre del ENP y otro con su categoría.

Por lo tanto, lo único que se hace es el procedimiento de importación estándar:

1. lectura de la *SHAPEFILE* con *GDAL / OGR* y escritura a esquema de importación de la base de datos;
2. limpieza de campos no requeridos, renombramiento de los que se mantienen y reproyección de la geometría a Lambert[9];

[8] En desarrollo y operaciones (*Development & Operations*, acortado comunmente como *DevOps*), una rama de la ingeniería de *software* que se encarga de la administración de la infraestructura en el contexto de un proyecto de desarrollo de *software*, una *instancia* de un programa o sistema es una instalación concreta en un servidor de ese sistema. Por lo tanto, la plataforma *Cell* utiliza un microservicio con una instancia de *PostGIS* y otro con una instancia de *Redis*, pero las operaciones *ETL* descritas en este capítulo se realizan en *otra* instancia de *PostGIS*.

[9] Recordemos que uno de los requisitos de la plataforma es que la información de origen debe estar proyectada en el sistema de referencia de la rejilla.

3. escritura de los datos finales en un esquema definitivo.

III.5.9. *ETL* de MDT

El modelo digital del terreno se distribuye en el DERA en forma de *GeoTIFF*. El proceso ha sido el siguiente:

1. lectura del *GeoTIFF* con *GDAL / OGR* y escritura de sus datos matriciales en un fichero *CSV* con el formato *coordenada x,coordenada y,cota*;

2. lectura directa por parte de *PostGIS* del fichero *CSV*. *PostGIS* puede leer directamente información en este formato si ayuda de librerías terceras como *GDAL*;

3. a partir de las coordenadas, creación del punto geométrico en *PostGIS* con su cota y reproyección a Lambert.

III.5.10. *ETL* de datos de catastro

Los datos de catastro utilizados en este trabajo son el resultado de las publicaciones anteriormente citadas y presentes en la bibliografía. Dichos proyectos generaron todo el análisis de asignación de viviendas a las construcciones, por lo que se ha importado directamente la base de datos resultado de dichos análisis. Es un *dump*[10] de una base de datos *PostGIS* que se restaura sin más inconvenientes en la instancia de datos de origen que consumirá la plataforma. Estas copias de respando tienen un tamaño de 27 *gigabytes*.

III.5.11. *ETL* de los datos de Hábitats

Este es posiblemente el procesamiento *ETL* más complejo de todos[11]. Los datos originales están servidos en una *File GeoDatabase (FGDB)* de la empresa *ESRI*, un formato que *GDAL* es capaz de leer. Por lo tanto, los pasos del proceso son:

1. lectura de las tablas contenidas en la base de datos, que son una tabla maestra de polígonos con los hábitats y otra con el catálogo de códigos de los mismos;

2. desagregación de los polígonos en función de los hábitats encontrados en él, es decir, el objetivo era tener un polígono con un sólo dato de hábitat, de forma que los polígonos con varios hábitats asignados se repitieron tantas veces como hábitats presentes en él para que cada hábitat tuviera su propia copia del polígono para realizar la adscripción. Esto es necesario puesto que la forma de trabajar del análisis de adscripción que se le va a aplicar así lo exige en el modelo de datos de origen. Esto hizo que el cómputo total de polígonos pasara de los 760.543 originales a 1.492.511;

3. limpieza de códigos inexistentes, que los había.

[10] Un *dump* o *backup* es el nombre que suelen recibir en *DevOps* las copias de respaldo de una base de datos con objeto de preservarla o migrarla a otra instancia de base de datos.

[11] Si se obvia, por supuesto, el anterior, que necesitó meses de trabajo y un proyecto para él sólo.

III.5.12. *ETL* de datos de población

Como ya se comentó, los datos de población en rejilla publicados por el IECA vienen en un excelente estado de publicación y documentación, perfectamente listos para ser usados tal cual son publicados. Sin embargo, los datos originales se presentan en tantos paquetes *SHAPEFILE* como años hay de datos (años 2002, 2013, 2014, 2015, 2016, 2017 y 2018), y además tienen la marca "-1" como valor por defecto para las aplicaciones del secreto estadístico. Dado que para hacer más eficiente el análisis de teselado que se le va a aplicar a estos datos lo mejor es tener una tabla única, la transformación hecha a los datos consiste en:

1. lectura de las *SHAPEFILES* con *GDAL*, obteniendo 7 tablas en la *PostGIS* de destino con los esquemas originales[12];
2. reescritura de los "-1" como datos de *PostGIS* "null"[13], para todas las tablas;
3. creación de una única tabla con toda la serie de datos, uniendo las siete tablas originales por una relación topológica basada en los centroides de las celdas originales, transformando dicho centroide al sistema nativo de la rejilla (Lambert Azimutal Equiárea) y armonizando todos los nombres de campos para las variables demográficas en cada momento temporal;
4. eliminación de todas las filas que tenían a "null" todos los campos de datos. Esto es así porque los datos originales recogían la presencia de población aunque, por secreto estadístico, no se pudieran dar más detalles.

[12] Es decir, los mismos campos con los mismos tipos y los mismos nombres de campos.
[13] *null* es el nombre que tienen los datos *NODATA* en *PostGIS*.

III.6 La librería libcellbackend

El siguiente paso en la implementación de la plataforma, una vez el microservicio de base de datos está en marcha y se han hecho las pruebas de concepto en él[1], es desarrollar una librería independiente de cualquier microservicio que implemente la lógica de negocio de la plataforma. Esta librería recibe en *Cell* el nombre de ***libcellbackend***.

Como su nombre indica, es una librería destinada a dar servicios a los microservicios del *backend*, es decir, los que funcionan en el servidor. Más adelante veremos que existe una librería similar, ***libcellfrontend***, que comparte muchos conceptos con *libcellbackend*, orientada al trabajo en los clientes *web*.

Estas dos librerías no pueden ser las mismas, aunque estén programadas en el mismo lenguaje, ya que las librerías de *backend* tienen que lidiar con muchos microservicios que simplemente no existen en el *frontend*, como por ejemplo la base de datos. En el *frontend*, además, hay que trabajar con los servicios que ofrece el propio entorno de ejecución, es decir, el navegador *web*. Son, por lo tanto, demasiado específicas como para ser la misma. Se podría haber desarrollado, no obstante, una sublibrería con el núcleo común a ambas, pero se ha preferido duplicar la pequeña funcionalidad que comparten e integrarla con la naturaleza propia de cada librería en lugar de adaptar esa hipotética librería común.

libcellbackend debe cumplir los siguientes objetivos funcionales:

1. implementar la lógica de negocio de los grandes conceptos esbozados hasta ahora que gobiernan la lógica de la plataforma: los *Gridder Tasks*, que ejecutan los análisis de adscripción, y las *Variables*, que gestionan la información alfanumérica adscrita a las teselas;

2. debe permitir la administración general de la plataforma, gestionando la creación de información en la base de datos;

3. debe ser la base de lógica de aplicación para integrarla con otras librerías específicas de los distintos microservicios del *backend* (*API*, controlador y trabajadores) y de esta forma permitir a esos microservicios gestionar los conceptos citados anteriormente en su contexto funcional.

[1] Recordemos la metodología de desarrollo de tener siempre herramientas para llevar a cabo tests y pruebas de concepto dentro de cada microservicio, minimizando las pruebas inter-servicios todo lo posible, ya que son farragosas y propensas a fallos. Cada microservicio debe poder ser desarrollado individualmente y testeado en un entorno controlado en el que se minimicen las influencias de otros microservicios. Esto, sin embargo, no siempre es posible, pero hay que maximizar este efecto todo lo posible.

III.6.1. Elección del lenguaje de programación

Existen multitud de posibilidades de lenguajes de programación para programar plataformas de servicios *web*. Para elegir el lenguaje de programación debemos tener en cuenta algunas de las necesidades de la plataforma:

1. tiene que ser un lenguaje que sea multipropósito. *Fortran* es un gran lenguaje de programación para hacer investigación operativa matemática, pero no sería recomendable utilizarlo fuera de ese contexto. Sin embargo, la plataforma *.NET* de *Microsoft* es un entorno multipropósito para crear aplicaciones informáticas de amplio espectro, por poner sólo un ejemplo de los muchos disponibles;

2. tiene que tener una buena implementación y un buen entorno de desarrollo en el sistema operativo *Linux*. *Linux* es el sistema operativo con el que funcionan la mayoría de los servidores, y además, el entorno nativo de la plataforma de contenerización Docker, por lo que es la elección lógica. La plataforma *.NET* de *Microsoft*, por ejemplo, está perfectamente integrada en sistemas *Windows*, pero el soporte para *Linux* está mucho menos desarrollado;

3. tiene que ser de *Software Libre*. El *Software Libre* de calidad es una garantía de acceso al conocimiento y de escalabilidad de la plataforma, al permitir a otras personas, sin restricción, acceder al código, mejorarlo y utilizarlo;

4. tiene que tener un amplio ecosistema de paquetería. Los sistemas de gestión de paquetes, en los lenguajes de programación, hacen referencia a sistemas accesorios al propio lenguaje que permiten, de una forma fácil, descargar automáticamente desde repositorios en *Internet* las librerías de terceros que va a utilizar el programa para hacer partes del trabajo. Tener un amplia biblioteca de librerías multipropósito, con una comunidad de *Software Libre* detrás de buen tamaño, es una garantía de que el lenguaje va a ser versátil y va a poder cubrir un amplio espectro de funcionalidades;

5. tiene que tener un buen rendimiento en sistemas multiusuario. No todos los lenguajes están preparados para esto. Algunos lenguajes son muy buenos para crear aplicaciones de escritorio monousuario, o gestionar cómodos interfaces de ventanas y botones. Tiene que ser un lenguaje que haya nacido vinculado a las tecnologías de *Internet*;

6. tiene que ser fácil crear librerías para el lenguaje. El lenguaje debe venir acompañado de accesorios que permitan, de una forma cómoda, empaquetar código en forma de librería para que otros programas puedan utilizarlo. Esto es fundamental para el objetivo descrito en este capítulo, es decir, crear una librería base para todos los microservicios;

7. tiene que tener un mínimo de librerías y orientación hacia las TIG. El lenguaje debe contar, entre sus librerías, con librerías de calidad para trabajar con información geográfica;

Feb 2021	Feb 2020	Change	Programming Language	Ratings	Change
1	2	^	C	16.34%	-0.43%
2	1	v	Java	11.29%	-6.07%
3	3		Python	10.86%	+1.52%
4	4		C++	6.88%	+0.71%
5	5		C#	4.44%	-1.48%
6	6		Visual Basic	4.33%	-1.53%
7	7		JavaScript	2.27%	+0.21%
8	8		PHP	1.75%	-0.27%
9	9		SQL	1.72%	+0.20%
10	12	^	Assembly language	1.65%	+0.54%
11	13	^	R	1.56%	+0.55%
12	26	^^	Groovy	1.50%	+1.08%
13	11	v	Go	1.28%	+0.15%
14	15	^	Ruby	1.23%	+0.39%
15	10	vv	Swift	1.13%	-0.33%
16	16		MATLAB	1.06%	+0.27%
17	18	^	Delphi/Object Pascal	1.02%	+0.27%
18	22	^^	Classic Visual Basic	1.01%	+0.40%
19	19		Perl	0.93%	+0.23%
20	20		Objective-C	0.89%	+0.20%

Figura III.6.1: El ránking *Tiobe* de lenguajes de programación. Fuente: Tiobe Software BV, 2021.

8. debe tener, entre sus librerías, las que le permitan conectarse y trabajar con microservicios clave de la arquitectura, como la base de datos o la memoria compartida. Si no dispone de librerías[2] para conectarse con estos sistemas, directamente no vale;

9. debe tener librerías de *Machine Learning* para conseguir los objetivos marcados por la plataforma en este aspecto;

10. debe ser un lenguaje amigable con las tecnologías de *Internet*, es decir, que sirva para programar *backends*, como por ejemplo *APIs* de servicios. *C* y *C++* son excelentes lenguajes en cuanto a rendimiento, pero entre sus objetivos de diseño no está el ser amigable con la *web*. Simplemente no es su función[3].

Las opciones iniciales son las siguientes:

1. ***Python*:** un lenguaje multipropósito fácil de usar y versátil;

2. ***JavaScript / TypeScript*:** uno de los lenguajes más utilizados en aplicaciones de *Internet*, en alza constante. Es el lenguaje nativo de los navegadores *web* pero ha transcendido ese ámbito para saltar a un entorno de programación multipropósito;

3. ***Java*:** uno de los lenguajes más utilizados del mundo a nivel corporativo y muy presente en sistemas de *Internet*.

Como referencia, mencionemos tres encuestas o rankings realizados por las siguientes empresas:

[2] Este tipo de librerías de enlace con otros sistemas suelen llamarse *drivers*.

[3] Lo cual no quiere decir que ciertas partes críticas de una aplicación *web* no puedan estar programadas en estos lenguajes. Por ejemplo, en Sunntics, la empresa en la que actualmente el autor desarrolla su actividad profesional, los algoritmos de optimización matemática que son el núcleo de su producto *SaaS* están escritos en *Fortran*, controlados por una capa *Node.js*.

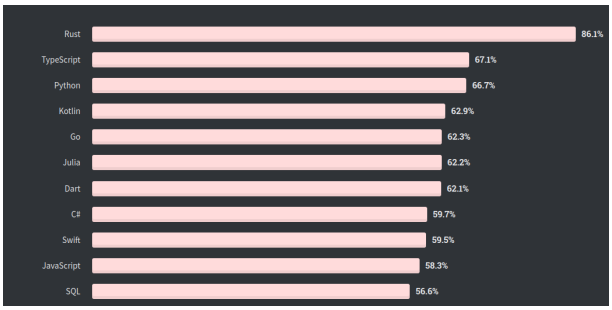

Figura III.6.2: El ránking de lenguajes de programación de la encuesta de *Stack Overflow*. Fuente: Stack Exchange Inc., 2021.

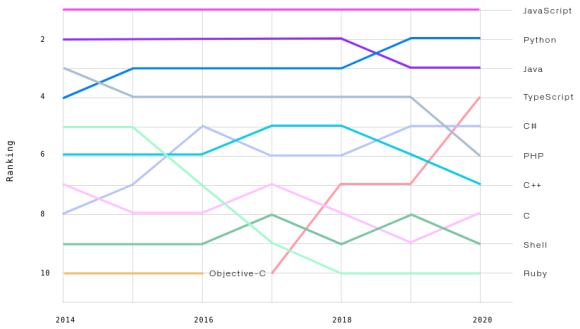

Figura III.6.3: El ránking de lenguajes de programación de *GitHub*. Fuente: GitHub Inc., 2021.

1. ***Tiobe Software BV*** es una empresa dedicada a certificar la calidad del *software* que publica un índice, el *Tiobe Index* (Tiobe Software BV, 2021), que mide el impacto, la demanda y el uso de cada lenguaje de programación. Este índice es de amplio espectro, ya que recoge el uso de los lenguajes de programación en un amplísimo campo de aplicaciones, desde sistemas embebidos industriales hasta la programación de sistemas *web*. No es una sorpresa, por lo tanto, que el lenguaje *C* lleve el primero desde hace décadas. El índice, a marzo de 2021, puede verse en la figura III.6.1;

2. ***Stack Overflow*** es una de las mayores comunidades de aclaración de dudas *peer-to-peer* en la industria del *software*. Todos los años hace una encuesta a sus miembros con diversas preguntas sobre su perfil profesional y el uso de las distintas tecnologías que utilizan a nivel profesional. También apoyan, como es lógico, sus estadísticas estudiando sus propios datos sobre preguntas y respuestas, categorizando las mismas en áreas temáticas. Todos los años publican los resultados de la encuesta en su *Developer Survey* (Stack Exchange Inc., 2021). Los resultados de la encuenta pueden verse en la figura III.6.2;

3. por último, una de las plataformas más utilizadas en la industria del desarrollo de *software* para alojar repositorios de código, especialmente de *Software Libre*, ***GitHub***, también utiliza su extenso acceso a las bases de código de miles de proyectos para extraer estadísticas anuales del uso y crecimiento de los distintos lenguajes y

tecnologías (GitHub Inc., 2021). El ránking para el año pasado se puede observar en la figura III.6.3.

A continuación discutimos sus pros y sus contras.

III.6.2. *Python*

Python es un lenguaje de programación de propósito general. Fue creado por Guido van Rossum[4] a finales de los 90 (Lutz, 2010) como lenguaje de *scripting* centrado en una alta legibilidad y expresividad del código y en ser sencillo de usar y aprender.

Python se ha convertido en uno de los lenguajes más utilizados, como puede verse en los ránkings. Su facilidad de uso y su alta expresividad[5] lo hacen un lenguaje perfecto para tareas que no precisan una alta velocidad ni eficiencia pero que son altamente experimentales e iterativas. Por ejemplo, la ciencia de datos: *Python* es un lenguaje muy utilizado en este campo, con excelentes librerías para el análisis y tratamiento de datos, incluido el *Machine Learning*. En resumen, *Python* es un lenguaje altamente productivo y expresión, a coste de la velocidad de ejecución.

Python cumple muchas de las necesidades citadas:

1. es un lenguaje marcadamente multipropósito;
2. tiene una buena implementación en *Linux*, al ser un lenguaje interpretado;
3. es *Software Libre* de contrastada calidad y estabilidad tanto en su desarrollo como en la gestión del proyecto;
4. el ecosistema de paquetería, denominado *PyPI*[6] (*Python Package Index*), es funcional y ubícuo, permitiendo la instalación fácil de librerías;
5. el rendimiento en sistemas multiusuarios no es el fuerte del lenguaje, puesto que no es uno de sus objetivos de diseño. Es una característica funcional, pero no está en el núcleo del lenguaje;
6. es fácil crear librerías con el lenguaje;

[4] La gestión de proyectos de *Software Libre* no es fácil y cada proyecto de éxito (como *Python*) acaba involucrando la acción coordinada de muchos organismos: particulares, academicos, empresas, agencias gubernamentales, etc. Son muchos los intereses y las formas de gobernanza de los proyectos toman formas distintas. Generalmente los proyectos tienen un comité ejecutivo electivo de expertos que deciden las líneas maestras del mismo, y existen fuentes de patronazgo y mecenazgo que financian el desarrollo y el mantenimiento del proyecto, muchas veces con recursos propios de las instituciones y empresas (por ejemplo, dedicando tiempo de los empleados a avanzar en el proyecto) o a programadores freelancers con un modelo similar al de los deportistas. El proyecto *Python* es curioso en tanto en cuanto reconoce a su fundador, el holandés Guido van Rossum, como una figura determinante en las decisiones del *Python Steering Counsil*. Tanto es así que ha ostentado el título de BDFL (*Benevolent Dictator for Life*) hasta su jubilación en 2019.

[5] *Alta expresividad* quiere decir que con pocas líneas de código se puede conseguir mucho. El mismo programa en *Python* tiene, por lo general, bastante menos líneas de código que el mismo programa en *C*. Obviamente, nada es gratis: el *Python* es lento.

[6] En el ecosistema *Python*, casi todo se llama *PyAlgo* o *AlgoPy*.

7. es un lenguaje muy orientado a las TIG, y uno de los más utilizados. Esto es así porque uno de los objetivos de diseño de *Python*, desde el principio, fue tener la capacidad de establecer puentes (*bindings*) al lenguaje *C* y *C++*, es decir, tener la capacidad de encapsular una librería escrita en estos potentes lenguajes en un envoltorio *Python* que lo haga fácil de usar. A partir de un programa o librería en *C* o *C++* la creación de estos *bindings* es prácticamente automática mediante el uso de ciertas herramientas del ecosistema *Python*. Esto permite usar desde *Python*, con la potencia y velocidad de *C* y *C++*, librerías en estos lenguajes como si de *Python* se tratara. Y todas las librerías de alto rendimiento TIG de *Software Libre*[7] están escritas en *C* y *C++*, con excelentes *bindings* a *Python*. Aparte de esta importante funcionalidad, son ya muchas las excelentes librerías de TIG escritas nativa o parcialmente en *Python*, como por ejemplo *Shapely*, *Fiona*, *Rasterio*, *GeoPandas* y un largo etc. Por lo tanto, en este aspecto, es el lenguaje ganador;

8. *Python*, por su extensivo uso, tiene librerías para conectarse a muchos servicios distintos, y desde luego los utilizados en la plataforma, *PostgreSQL* y *Redis*, no son una excepción. Para *PostgreSQL* está la excelente *PsycoPg*, y para *Redis* está *redis-py*;

9. en cuanto a la ciencia de datos y *Machine Learning*, *Python* destaca. El ecosistema para hacer ciencia de datos y *ML* para *Python* es potente y variado. Sirvan como ejemplo las excelentes librerías y *frameworks Numpy*, *Pandas*, *Scipy*, *Scikit-learn* o *Keras* (*TensorFlow*). La capacidad de *Python* de realizar rápidamente muchas iteraciones de un experimento lo convierten en un lenguaje muy atractivo en este campo;

10. en cuanto a la *web*, *Python* es un lenguaje moderadamente amigable. Existe un subecosistema en *Python* para crear aplicaciones en la nube, fundamentalmente centrada alrededor del *framework*[8] *Django*[9].

III.6.3. *JavaScript / TypeScript*

El lenguaje *JavaScript*[10] comenzó su andadura a comienzos de la era de *Internet*[11] siendo el lenguaje de programación que permitía al navegador *Netscape* incrustar pequeños

[7] *GDAL / OGR*, *GEOS*, *GRASS*, *PROJ*, etc.

[8] Un *Framework* en ingeniería de *software* es un conjunto de herramientas, procedimientos y estándares que ayudan a los programadores a desarrollar de forma estandarizada y eficientemente aplicaciones de un ámbito concreto. En este caso, *Django* es un *framework* que aporta al programador muchas de las piezas y unos estándares de proceso comunes a la programación de aplicaciones *web*. De esta forma el programador no tiene que estar constantemente reinventando la rueda, ya que se centra en desarrollar su aplicación con técnicas y patrones de diseño bien contrastados. Esto hace, además, que el concurso de varios programadores que conozcan el *framework* en un mismo proyecto sea más productivo, puesto que todos aplican las mismas ideas en la concepción general del programa.

[9] Que además tiene un derivado centrado en las TIG, *GeoDjango*.

[10] A pesar del nombre, *JavaScript* no tiene nada que ver con el otro lenguaje, *Java*.

[11] Aquella época se conoce como la *Guerra de los Navegadores*, por la encarnizada lucha entre *Microsoft Explorer* y *Netscape Navigator* por el dominio del mercado de los navegadores. Al final venció, por motivos comerciales, que no tanto por motivos de calidad, *Explorer*, y hoy en día el legado de *Netscape* sigue vigente en los productos *Mozilla*.

programas en las páginas *web* para hacer funcionar primitivos botones y formularios *online*. Como ya se discutió anteriormente, la tendencia es que los navegadores sean cada vez más capaces de hacer cosas por sí solos, hasta el punto de que se están convirtiendo en pequeños mini-sistemas operativos. El *JavaScript* es el lenguaje que sigue haciéndolo posible.

Los motores o interpretes *JavaScript* que incorporan navegadores como *Google Chrome*[12] son ciertamente sofisticados y están muy optimizados para la ejecución de programas multitarea. En un navegador están pasando muchísimas cosas simultáneamente y además, a diferencia de otros programas más sencillos, las interacciones que el usuario produce sobre una página *web* son de todo menos secuenciales. El estado de una página *web* puede cambiar de forma imprevisible, y debe reaccionar a muchos eventos que produce tanto el usuario como otros componentes de la página o los servidores a los que se conecta. Por ello, el *JavaScript* está diseñado para ejecutarse con un modelo de ejecución un tanto distinto al de otros lenguajes más tradicionales. Ciertamente, aprender *JavaScript* no es la mejor forma de iniciarse en la programación[13].

Este modelo de ejecución tiene sus ventajas (y sus inconvenientes). Ventajas lo suficientemente atractivas como para usarlo en otros entornos que no son los de un navegador *web*. En 2009, Ryan Dahl (Herron, 2020) extrae el motor *V8* de *Google* y lo adapta para que ejecute programas fuera el entorno del navegador, creando el entorno / *framework Node.js*, según el paradigma "*JavaScript everywhere*", es decir, programar una aplicación *web* en un único lenguaje, tanto las partes que son de *backend* en servidores como las partes *frontend* en navegadores. *Node.js*, por lo tanto, es un interprete *JavaScript* que puede ejecutar programas fuera del entorno de un navegador, con el consiguiente acceso a los servicios de un sistema operativo como *Linux*, *MacOS* o *Windows*.

TypeScript, por su parte, es un superconjunto de *JavaScript* desarrollado por *Microsoft*. Un superconjunto de un lenguaje de programación son una serie de añadidos y extensiones que mejoran las capacidades del lenguaje pero con el objetivo innegociable de no hacerlo incompatible con el lenguaje que pretende extender. Es decir, el lenguaje extendido debe ser capaz de traducirse sin problemas a una sintaxis que el lenguaje original interprete[14]. *TypeScript* añade a *JavaScript* una capacidad de los lenguajes de ordenador que es el ser fuertemente tipado, es decir, en *TypeScript* el programador tiene control sobre los tipos de datos que le da a cada dato que gestiona el programa, mientras que en un lenguaje debilmente tipado este control no se ejerce. Por ejemplo, en el programa *JavaScript*:

```
1 // Definimos un dato como número
2 let a = 5;
3
4 // Esto muestra por pantalla "6", ya que en
5 // JavaScript número + número es un número
6 console.log(a + 1);
7 // Esto mostraría 5, ya que número dividido
```

[12] Conocido como *V8*.

[13] Para el que quiera empezar, la recomendación del autor es clara: *Python*.

[14] Esta técnica recibe en ingeniería de *software* el nombre de *Sintactic Sugar*: son añadidos sintácticos que pueden ayudar al programador a orientarse o entender mejor el programa, pero que son de todo punto ignorados e inútiles para las funciones del lenguaje en sí cuando la máquina lo interprete.

```
8  // entre número también tiene sentido
9  console.log(10 / a);
10
11 // Ahora lo redefinimos como carácter, y el
12 // interprete de JavaScript no arroja ningún
13 // error ni advertencia
14 a = "a";
15
16 // Esto muestra por pantalla "a1", ya que
17 // letra + número en JavaScript es una unión
18 // de la letra con la grafía del número
19 console.log(a + 1);
20
21 // Pero esto resultaría en un error, ya que
22 // para JavaScript la división de un número
23 // por una letra no tiene sentido
24 console.log(10 / a);
```

Código III.6.1: Ejemplo de un programa escrito en *JavaScript*.

el programador tiene que ser muy consciente de qué tipo tiene en cada momento la variable "a" porque el interprete del lenguaje no le va a avisar de que la variable está cambiando de ser un número a una letra. Por consiguiente, si no es cuidadoso, el programador puede llegar a una parte del programa en la que intente hacer una operación entre tipos que no tengan sentido, como lo que sucede en la línea 24.

Sin embargo, el mismo programa en *TypeScript*:

```
1  // Definimos un dato explicitamente como número
2  let a: number = 5;
3
4  // Esto muestra por pantalla "6", ya que
5  // en TypeScript número + número es un número
6  console.log(a + 1);
7  // Esto mostraría 5, ya que número dividido
8  // entre número también tiene sentido
9  console.log(10 / a);
10
11 // Ahora lo redefinimos como carácter, y el
12 // preprocesador TypeScript daría inmediatamente
13 // un error, porque no podemos asignar una
14 // letra a una variable que ha sido especificada
15 // como número. El programa fallaría aquí y no
16 // nos permitiría continuar haciendo operaciones
17 // sin sentido.
18 a = "a";
```

Código III.6.2: El mismo programa en *TypeScript*.

Nótese que lo único que cambia es el operador ***:*** seguido del tipo de dato ***number*** de la línea 2. Ese es el designador de tipo que le explicita a *TypeScript* que la variable "a" está

destinada a contener números. Este código *TypeScript* pasa primero por un preprocesador sintáctico[15] que lo convierte en un programa 100 % *JavaScript*, pero controlando los tipos en el proceso. Si este proceso de preprocesado concluye sin detectar errores, el programa *JavaScript* creado es pasado a *Node.js* para su ejecución. En esa primera interpretación del código *TypeScript*, el interprete comprueba que el programador no comete errores de asignación de variables a tipos erróneos. Si se detectan, para y el programa no llega a *Node.js* como *JavaScript* reconvertido para ser ejecutado.

Estas dos formas de abordar los datos en un lenguaje para su posterior ejecución tienen sus ventajas y sus inconvenientes. Aunque parezca que la aproximación de *TypeScript* es más segura (y lo es), lo que hace *JavaScript* también permite aplicar técnicas de programación muy avanzadas y convenientes en algunos escenarios[16].

La evaluación del *TypeScript / JavaScript* en un entorno *Node.js* con respecto a los criterios buscados sería:

1. es un lenguaje multipropósito que, aunque con un fuerte sesgo a la programación de aplicaciones *web*, cada vez se usa en más ámbitos;
2. la implementación en *Linux* es impecable;
3. es *Software Libre* con garantías;
4. tiene un excelente sistema de paquetería que se llama *NPM* (*Node Package Manager*), con miles de librerías disponibles fácilmente instalables para realizar todo tipo de tareas;
5. el rendimiento en sistemas multiusuario es excelente, de hecho, es una de las bases de su diseño. Una aplicación *web* en *Node.js* puede servir con facilidad a muchos clientes simultáneamente gracias a su modelo de ejecución por eventos. El sistema nunca se para a esperar a que otra parte del sistema le conteste o algún dispositivo de *hardware* (como los lentos discos duros) respondan, siempre está haciendo algo. Deja la tarea en curso en una cola de eventos y ya cuando termine se encargará de ella;
6. la creación de librerías para el lenguaje es muy sencilla gracias al ecosistema del ya citado *NPM*;
7. en cuanto a las TIG, no es tan potente como *Python* ni mucho menos. Cada vez se ven más librerías que hacen cosas interesantes, como la excelente *Turf.js* que se usa en la plataforma, y gracias al incremento de popularidad del lenguaje, que está compitiendo con *Python* en ciertos nichos de utilidad, se espera que cada vez sean más las librerías y *ports*[17] de calidad de librerías escritas en otros lenguajes. Sin embargo, hay librerías de sobra para hacer lo básico;

[15] Como no es un interprete ni un compilador, lo llaman *transpiler*, y a lo que hace, *transpilation*.

[16] Este es un debate eterno entre los programadores. *Python*, por ejemplo, es un lenguaje debilmente tipado, como *JavaScript*. *C*, *C++* y *Java*, por su parte, son lenguajes extraordinariamente tipados. Personalmente, prefiero los lenguajes fuertemente tipados.

[17] Un *port*, en ingeniería de *software*, es una reescritura de una librería escrita en un lenguaje de programación a otro lenguaje distinto. Por ejemplo, el motor topológico *GEOS*, escrito en *C* y *C++* y utilizado

8. posee librerías de *drivers* buenas para conectarse tanto a *PostgreSQL* como a *Redis*;

9. es un lenguaje muy amigable con las tecnologías *web*, no en vano, es su principal dominio de aplicación. Muchos *backends* de productos conocidos están escritos en este lenguaje.

III.6.4. *Java*

Java es, posiblemente, el lenguaje de programación más utilizado a nivel corporativo, como se puede ver en los índices. Es un lenguaje creado para ser portable entre sistemas distintos por la empresa *Sun Microsystems* a mediados de los 90. Es un lenguaje a tener en consideración en prácticamente todos los proyectos de *software* debido a su popularidad.

Java es un lenguaje fuertemente tipado y orientado a objetos, además de ser portable entre distintos sistemas operativos. Es un lenguaje muy protocolarizado y con innumerables *frameworks* para cubrir todo tipo de casos de uso. *Java* no es sólo un lenguaje, es un sistema completo de prácticas estándar de ingeniería de *software*.

La evaluación de este lenguaje con respecto a los objetivos propuestos es:

1. es un lenguaje multipropósito, para todo tipo de aplicaciones, desde móviles (es el lenguaje base de la plataforma *Android* de *Google*, por ejemplo) hasta aplicaciones *web*[18];

2. *Java* es multiplataforma, por lo que su implementación en *Linux* es perfecta;

3. resulta complejo saber hasta que punto *Java* es *Software Libre*, sólo gratuíto o qué. Teóricamente existen implementaciones de referencia que están en el dominio del *Software Libre*, como es la *OpenJDK*, pero dado que *Sun Microsystems* (que sí que tenía vocación de mantener a *Java* en el ámbito del *Software Libre*) fue comprada por *Oracle Corporation*, posiblemente una de las empresas más hostiles al *Software Libre*, y que *Oracle* ya ha demandado a *Google* por cantidades millonarias por supuestos usos fuera de licencia del lenguaje en sus productos, el asunto desde luego el que esto escribe no lo tiene nada claro;

4. posee un evolucionado sistema de paquetería y ensamblaje de programas llamado *Maven*;

5. en cuanto a rendimiento en aplicaciones multiusuario, su utilización en el campo de las aplicaciones *web* demuestra su aptitud en este área;

6. la creación de librerías para el lenguaje es fácil;

---

en todo el *Software Libre* TIG, es un *port* de una librería originalmente escrita en *Java* llamada *Java Topology Suite* (*JTS*).

[18] Siempre se dice que *Java* no tiene el rendimiento suficiente para la creación de videojuegos, y en parte es verdad, dado que los videojuegos de alto rendimiento se programan en *C* y *C++*. Sin embargo, se da la paradoja de que el videojuego más vendido de la historia, *Minecraft*, está escrito precisamente en *Java*.

7. aunque no tiene tantas librerías como *C* y *C++*, que es el lenguaje en el que se suelen escribir las aplicaciones de TIG (y, por consiguiente, *Python*), existen en *Java* suficientes librerías para hacer lo básico;

8. *Java* tiene librerías de *drivers* para conectarse a cualquier sistema mínimamente popular;

9. *Java* posee también una extensa selección de librerías y *frameworks* para *Machine Learning*. Podemos destacar el *framework Apache Spark*, *DL4J* (*Deep Learning for Java*, *ELKI* o *Java-ML*;

10. *Java* es definitivamente un lenguaje muy amigable para la *web* y su capacidad para desarrollar aplicaciones cliente-servidor es notable.

III.6.5. Evaluación del lenguaje

Finalmente, el lenguaje elegido para la creación de las librerías y microservicios ha sido *TypeScript*. En la elección han pesado los siguientes factores:

1. se puede aprovechar el lenguaje tanto para el *backend* como para el *frontend*;

2. su altísimo rendimiento en tareas computacionalmente pesadas y lanzadas en paralelo, como es el teselado;

3. cubre bien expediente en todas las áreas, aunque en *Machine Learning* está un poco falto de librerías en comparación con *Python*. Sin embargo, dado que se ha optado por una arquitectura de microservicios, no existe problema en coger lo mejor de ambos lenguajes y dejar las tareas de *ML* en manos de microservicios *Python*;

4. la existencia de la librería *Turf.js*, que implementa una gran cantidad de algoritmia geoespacial básica necesaria para el tratamiento de la vertiente geométrica de la tesela tanto en cliente como en servidor, es una ventaja;

5. en cuanto a *Java*, la alta capacitación en ingeniería de *software* necesaria para utilizarlo eficientemente excede las capacidades del presente trabajo. *TypeScript* es más accesible para comenzar a experimentar y obtener un buen resultado.

III.6.6. Dependencias de la librería *libcellbackend*

Esta librería va a ser el núcleo de la lógica de negocio de gestión y adscripción de la información original a las teselas. En esta librería se van a concentrar los objetos que modelizan todos los conceptos explicados hasta ahora, y que será heredada por los microservicios *API*, controlador y trabajadores para tratar la información y los procesos sobre la misma bajo los mismos parámetros operativos. La librería se escribe y se testea de forma autónoma, para posteriormente ser incorporada a los programas que gobiernan los microservicios citados. Esto garantiza una estructura de la información y un diseño de las *API* de los microservicios común a todos ellos, lo que agiliza y estructura el trabajo.

Esta librería es el siguiente nivel de evolución de la plataforma tras haber formalizado la base de datos. Al igual que la base de datos incluía una librería escrita en *SQL* y *PL/PgSQL* para probar los conceptos que desarrolla, una librería en *TypeScript* también se prepara para ser funcionalmente autónoma y ser capaz, operándola de forma independiente a ningún microservicio, de gestionar los procesos básicos que debe realizar la plataforma. Es el núcleo del sistema, al que se le irán añadiendo capas posteriores en forma de microservicios que harán la plataforma usable. Pero el principal componente es esta librería.

Esta librería, además, usa un conjunto de sublibrerías que hacen trabajos muy concretos no exclusivos de esta plataforma, sino reutilizables en muchos programas que las precisen. Las librerías que usa un programa u otra librería son conocidos en ingeniería de *software* como *dependencias*, y en los lenguajes y entornos de programación comentados, como ya hemos visto en los tres casos revisados, dichas dependencias son almacenadas en un repositorio en *Internet* que programas accesorios al lenguaje de programación son capaces de descargar y enlazar con el programa o librería que van a hacer uso de ellas. Este proceso de montaje de dependencias permite que el programa anfitrión se conecte a las *API* de las librerías dependencia para hacer uso de sus funciones.

Las dependencias, por tanto, de esta librería, son:

1. ***node-logger*:** una librería *Node.js* escrita para la plataforma que, utilizando la librería *Node.js Winston*, escribe los registros de ejecución de lo que va sucediendo dentro de la librería[19] en ficheros de texto *CSV*[20];

2. ***rxpg*:** otra librería escrita ex-professo para la plataforma. Se encarga de encapsular el *driver Node.js* de la base de datos *PostgreSQL* en una capa externa de *programación reactiva* para facilitar su uso en el entorno de ejecución por eventos de *Node.js*;

3. ***Turf.js*:** *Turf.js* (Agafonkin et al., 2021b) es una librería de análisis topológico y geoespacial desarrollada por varios ingenieros de la empresa *Mapbox* (Mapbox, 2021a), que permite hacer, entre muchas otras cosas, operaciones de procesamiento de polígonos;

4. ***proj4js*:** *PROJ* es una de las librerías más veteranas del *Software Libre* TIG y su *port* a *JavaScript*, *proj4js* (Adair et al., 2021), se encarga en la plataforma de realizar las conversiones de sistemas de referencia;

5. ***mathjs*:** esta librería (Jong et al., 2021) implementa funciones matemáticas básicas, como redondeos de varios tipos. Aunque el propio lenguaje *JavaScript* trae muchas

[19] En ingeniería, estos registros son conocidos como *logs*. Todo sistema mínimamente complejo debe implementar esta funcionalidad, mediante la cual se va dejando un registro, en una base de datos o en ficheros de texto, de la actividad del sistema. Su propósito final es la depuración de posibles errores y la acumulación de información sobre la actividad del sistema que lleve a su mejora o a descubrir patrones de uso por parte de los usuarios.

[20] *Comma Separated Values*, *Valores separados por comas*. Una sencilla forma de estructurar la información en ficheros de texto en los que cada ítem de datos está separado por una coma u otro carácter identificable como separador. Este formato es muy frecuente en el intercambio de datos, ya que es fácilmente transformable en estructuras de datos más complejas.

funciones matemáticas de serie[21], no cubren muchas veces las necesidades de todas las aplicaciones. *mathjs* se usa muy poco en la plataforma, pero es necesaria;

6. ***simple-statistics*:** esta librería (MacWright et al., 2021), en la línea de la anterior *mathjs*, aporta funcionalidad de base estadística. Implementa funciones estadísticas como distintas medidas de tendencia central y de dispersión sobre una serie de datos, así como clasificadores bayesianos y regresiones lineales. Se utiliza en la plataforma para hacer estadísticas sobre los datos de adscripción;

7. ***lodash*:** esta es una librería (Dalton et al., 2021) multipropósito de *JavaScript* que aporta una gran cantidad de funciones para trabajar con matrices y listas, fechas, colecciones de datos, matemáticas y para lidiar con ciertos aspectos del lenguaje *JavaScript* que hacen complejas ciertas tareas de programación[22]. Se usa en la plataforma, fundamentalmente, por su excelente colección de funciones para hacer operaciones con listas y matrices de datos.

III.6.7. Clases base de la librería *libcellbackend*

Otra de las ventajas de *TypeScript* es que permite programar en *JavaScript* con un estilo orientado a objetos bien definido[23]. Toda la plataforma está programada según dicho paradigma, por lo que vamos a describir las capacidades de la librería en función de sus clases y sus funciones.

Primero abordaremos las clases más sencillas de la librería. Estas clases proporcionan soporte a los procesos de adscripción y de generación de variables de adscripción, que son los procesos más complejos de la misma, y que se explicarán en posteriores apartados.

Clase *Coordinate*

La clase *Coordinate* modela una coordenada, un punto sobre un sistema de referencia determinado. Como tal, es una clase relativamente simple.

Miembros de la clase *Coordinate*

***EPSG*: string**

Código *EPSG* del sistema de referencia al que se refieren las coordenadas.

[21] Los lenguajes de programación suelen venir con una serie de funciones de serie, lo que se suele llamar la *librería estándar* del lenguaje. Algunos, como *C*, tiene como principio de diseño minimizarla al máximo, hasta hacerla casi inexistente, dejando este trabajo para desarrolladores que no pertenecen al núcleo del desarrollo del lenguaje en sí, mientras que otros, como *Python*, crean deliberadamente una librería estándar gigantesca para incluir como un paquete, con el lenguaje, una gran cantidad de funcionalidad de base.

[22] Derivadas de las desventajas de ser un lenguaje debilmente tipado, generalmente. La mayoría de estos problemas se evitan usando *TypeScript*.

[23] No es que no se pueda programar con orientación a objetos en *JavaScript* ni mucho menos, lo que pasa es que el lenguaje en sí no formaliza la forma de hacerlo, y existen varios *dialectos* del lenguaje, con sus propias ventajas e inconvenientes, para practicar este estilo de programación. *TypeScript* lo formaliza, de forma que siempre se hace igual.

x: number* e *y: number

Las coordenadas propiamente dichas.

geojson: turf.Feature<turf.Point>

Devuelve la coordenada como un punto *GeoJSON* para ser usado en la librería *Turf.js*.

pggeojson: any

Devuelve la coordenada en un formato compatible con la definición de puntos que hace *PostGIS*.

Métodos de la clase *Coordinate*

reproject(epsg: string): Coordinate

Reproyecta la coordenada al sistema dado por el parámetro "epsg". Devuelve otro objeto *Coordinate* con la coordenada transformada. La transformación la realiza la librería *proj4js*.

fromGeoJSON(geojson: any)

Convierte un punto *GeoJSON* en un objeto *Coordinate*.

euclideanDistance(coordinate: Coordinate): number

Devuelve la distancia euclídea[24] entre este objeto *Coordinate* y otro objeto *Coordinate* proporcionado como parámetro.

euclideanDistanceFromXY(coordinates: [number, number]): number

Igual que el anterior, pero en vez de proporcionar un objeto *Coordinate*, se proporciona como parámetro directamente un par de coordenadas.

Clase *ZoomLevel*

Esta pequeña clase describe un nivel de resolución de la rejilla. Se compone de dos miembros muy simples: un miembro "name", cadena de caracteres, que contiene el nombre del nivel de resolución (p.e., "100 kilómetros") y otro llamado "size", numérico, que guarda el tamaño del lado de la tesela.

Clase *Bbox*

Esta clase modeliza una *bounding box*, es decir, una caja de coordenadas.

Miembros de la clase *Bbox*

lowerLeft: Coordinate

[24] La distancia euclídea es la distancia del día a día, la del Teorema de Pitágoras.

Devuelve la esquina inferior izquierda de la caja como un objeto *Coordinate*.

upperRight: Coordinate

Devuelve la esquina superior derecha de la caja como un objeto *Coordinate*.

epsg: string

El código *EPSG* del sistema de referencia en el que están expresadas las coordenadas de la caja.

geojson: Feature<Polygon>

Devuelve la caja como un polígono *GeoJSON* para la librería *Turf.js*.

pggeojson: any

Devuelve la caja como geometría compatible con el sistema de la base de datos *PostGIS*.

Métodos de la clase *Bbox*

coordinateInBbox(coordinate: Coordinate): boolean

Devuelve verdadero si la coordenada *Coordinate* está dentro de la caja.

Clase *Polygon*

Esta clase modela un polígono para la librería *Turf.js*.

Miembros de la clase *Polygon*

area: number

Devuelve el área del polígono.

Métodos de la clase *Polygon*

fromGeoJson(geojson: any): any

Construye un polígono a partir de una definición poligonal *GeoJSON*.

intersection(polygon: Polygon): Polygon

Intersecciona este polígono con otro polígono definido por "polygon". Devuelve el polígono interseccionado.

Clase *Cell*

Esta clase modela una tesela de una rejilla determinada. Esta clase se comunica con la base de datos, siendo capaz de escribir y leer información desde la tabla *data*.

Miembros de la clase *Cell*

epsg: string

El código *EPSG* del sistema de referencia de la tesela, heredado de su rejilla madre.

x: number* e *y: number

Las coordenadas de la tesela en su rejilla madre.

grid: Grid

Ver clase *Grid* más adelante. Guarda una referencia al objeto *Grid* que modela su rejilla madre.

zoom: number

El nivel de resolución en la rejilla madre a la que pertenece esta tesela.

sideSize: number

El tamaño de lado, en unidades del sistema de referencia de la rejilla madre, que mide esta tesela.

area: number

El área, en unidades de área del sistema de referencia de la rejilla madre, que ocupa esta tesela.

lowerLeft: Coordinate

Un objeto *Coordinate* con las coordenadas de la esquina inferior izquierda de la tesela.

upperRight: Coordinate

Un objeto *Coordinate* con las coordenadas de la esquina superior derecha de la tesela.

center: Coordinate

Un objeto *Coordinate* con las coordenadas del centro de la tesela.

data: any

El vector temático de la tesela, con las variables de adscripción.

offset: number

El *offset* de la tesela, es decir, un valor positivo o negativo (figura III.6.4) que se añade al tamaño de la tesela para hacerla más grande o más pequeña. Esta funcionalidad es utilizada por algunos análisis de adscripción que precisan considerar para su resolución un área alrededor de la tesela mayor que su propia extensión, ya que la adscripción se va a ver influenciada por la presencia de geometrías a esa distancia de influencia.

bbox: Bbox

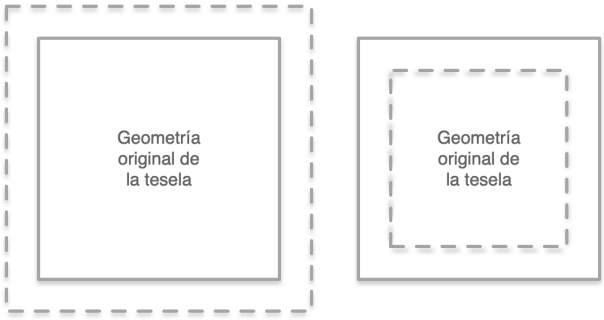

Figura III.6.4: *Offset* positivo (izquierda) y negativo (derecha) de una tesela. En trazo continuo se representa la geometría original de la misma, mientras que el trazo discontinuo es la geometría modificada por el *offset*.

Devuelve la *bounding box* de la tesela en un objeto de clase *Bbox*.

ewkt: string

Devuelve la representación *EWKT*[25] de la tesela, para *PostGIS*.

geojson: any

Devuelve la representación de la tesela en formato *GeoJSON*, para la librería *Turf.js*.

gridId: string

El ID de la rejilla madre.

corners(): { lowerLeft: Coordinate, upperLeft: Coordinate, upperRight: Coordinate, lowerRight: Coordinate }

Devuelve una lista de objetos *Coordinate* con las esquinas de la tesela.

scaleFactor: number

Devuelve el cociente entre el área de la tesela medida sobre el elipsoide WGS84 por la librería *Turf.js* y la medición proyectada en el sistema de referencia nativo de la tesela. Se expresa como 1 %.

Métodos de la clase *Cell*

getSubCells(zoom: number): Cell[]

Devuelve las teselas hijas de esta tesela en el nivel de resolución "zoom".

coordinateInCell(coordinate: Coordinate): boolean

Devuelve verdadero si la coordenada "coordinate" está dentro de la tesela.

intersectPolygon(polygon: Polygon): any

[25] *Extended Well-Known Text*, una forma de representar geometrías del estándar *Simple Features* con la que trabaja *PostGIS*.

Intersecciona un poligono con el polígono de la tesela. Realizada por la librería *Turf.js*.

drillDownClone$(pg: RxPg, targetZoom: number): rx.Observable<Cell>)

Esta función clona las variables de adscripción fruto de calcular un análisis de adscripción sobre la tesela objetivo al conjunto de sus teselas descendientes en la pirámide de resolución de la rejilla hasta alcanzar el nivel de resolución "targetZoom". Este proceso de copia masiva de información adscrita es muy útil para agilizar los análisis de adscripción en los que ésta operación tiene sentido (no en todos los tipos de análisis es posible sacarle partido a esta funcionalidad. Estas variables de adscripción clonadas son escritas en la base de datos en las teselas correspondientes en la tabla *data*.

getGrid$(pg: RxPg): rx.Observable<Cell>

Carga desde la base de datos las características de la rejilla madre de esta tesela.

clone(): Cell

Crea una copia de esta tesela.

Clase *Grid*

La clase *Grid* gestiona la definición y las operaciones con una rejilla. Si un objeto tiene la capacidad de guardar sus datos en el microservicio de la base de datos *PostgreSQL*, es el propio objeto el que sabe cómo guardar sus propios datos o recuperarlos desde la base de datos. Este objeto gestiona los datos de la tabla *grid* de la base de datos.

El objeto reproduce el esquema de datos de la rejilla ya comentado en el capítulo dedicado a la base de datos, e implementa una serie de métodos para trabajar con la rejilla.

Miembros de la clase *Grid*

gridId: string

El identificador de la rejilla.

epsg: string

El código *EPSG* del sistema de referencia de la rejilla.

zoomLevels: ZoomLevel[]

La definición de los niveles de resolución, con el mismo formato ya visto en la sección de base de datos. Se compone de una lista de objetos de la clase *ZoomLevel*.

origin: Coordinate

Un objeto *Coordinate* que marca el origen de coordenadas de la rejilla.

name: string

El nombre de la rejilla.

description: string

La descripción de la rejilla.

Métodos de la clase *Grid*

coordinateInCell(coordinate: Coordinate, zoomLevel: number): Cell

Devuelve un objeto *Cell* describiendo la tesela en la que cae la coordenada "coordinate" en el nivel de resolución número "zoomLevel".

getBboxCellCoverage(bbox: Bbox, zoomLevel: number): Cell[]

Devuelve todas las teselas del nivel de resolución "zoomLevel" que recubren la caja descrita por "bbox".

getSquareCellsInBbox(bbox: Bbox, zoom: number): { lowerLeft: Cell, upperRight: Cell }

Devuelve las teselas del nivel de resolución "zoom" en las que caen las esquinas inferior izquierda y superior derecha de la caja definida por "bbox". Estas dos teselas sirven para averiguar de forma rápida el conjunto de celdas que solapa una caja.

numChildCells(topLevel: number, bottomLevel: number): number

Devuelve el total de teselas descencientes, hasta el nivel de resolución "bottomLevel", que tiene una tesela de nivel de resolución "topLevel", es decir, la suma de todas las teselas hijas en cada nivel de resolución hasta alcanzar "bottomLevel".

III.6.8. Objetivos funcionales de los *Gridder Tasks*

Antes de describir las clases más importantes de la librería *libcellbackend*, la clase base *GridderTask* y la familia de clases hijas que genera, debemos discutir minimamente cuáles son los objetivos funcionales de estas importantes clases que se van a encargar de la gestión de los análisis de teselado.

Como ya se describió en la sección de metodología, el principal medio mediante el que se articula la adscripción de la información original a la tesela es el denominado *análisis de adscripción*. En la plataforma, estos análisis de adscripción tienen el nombre técnico de *Gridder Tasks*.

Este concepto ya apareció en el capítulo de la base de datos, ya que tienen su propia tabla para almacenar su información y controlarla. Un *Gridder Task* adscribe a las teselas definidas por una rejilla un conjunto de datos originales mediante un método u otro de adscripción.

Estos *Gridder Tasks* pueden ser creados por un programador y ser incorporados a la plataforma. El programador que vaya a programar un nuevo *Gridder Task* debe seguir

una metodología de diseño que permite su integración en la plataforma. Básicamente, un *Gridder Task*, para poder ser integrado, debe cumplir una serie de hitos funcionales que marcan el flujo de información entre los datos originales y la información final adscrita a la rejilla.

El fin último de un *Gridder Task* es la definición, en el vector temático o de adscripción de cada tesela, de una serie de *variables de adscripción*. Estas variables tienen el nombre técnico, en la implementación de la plataforma, simplemente de *Variable*. Cada variable, por tanto, es exclusiva de un único *Gridder Task* que la ha generado. Cada *Gridder Task* es identificado unívocamente en el sistema con un identificador. Asimismo, cada *Variable* también tiene un identificador único a nivel de plataforma.

Un *Gridder Task* debe cumplir los siguientes objetivos funcionales:

1. debe aplicar una lógica de adscripción sobre unos datos originales que se definen por su tipo de geometría (datos puntuales, lineales o poligonales) y que llevan en su vertiente alfanumérica asociada variables que también se caracterizarán por ser discretas o continuas;

2. generará una o más variables de adscripción como fruto de dicho análisis. Estas variables de adscripción están unívocamente identificadas y poseen una clave que será la que se inserte, junto con su valor, en el campo *data* de la tabla *data*, que contiene la información alfanumérica de la tesela procesada (el vector de adscripción);

3. el *Gridder Task* se aplicará tesela a tesela, por lo que es un trabajo paralelizable por varios trabajadores para aumentar la velocidad del trabajo global de adscripción;

4. el *Gridder Task* debe ser diseñado de forma que pueda aprovechar al máximo, si es posible, y teniendo en cuenta las características concretas del algoritmo de adscripción implementado, las relaciones topológicas inter-resolución y de autoindexación que ofrece la estructura de la rejilla, para procesar las teselas descendientes de la tesela objeto del análisis sin tener que aplicar el análisis completo a las mismas. Esto no siempre es posible con algunos algoritmos de adscripción;

5. para agilizar el proceso de recuperación de teselas con presencia de variables generadas por un *Gridder Task* concreto, dado que hay *Gridder Tasks* que pueden generar cientos de variables, el *Gridder Task* debe generar también, a partir de los datos originales, una variable especial llamada *variable índice*, o *Index Variable*, que es añadida también al vector de adscripción de la tesela. El valor de esta variable índice es un conjunto de valores que sumarizan el proceso de adscripción, pero su principal papel es indiciario de la presencia de información del *Gridder Task* en la tesela.

III.6.9. Clases para realizar análisis de adscripción en *libcellbackend*

La metodología definida en el capítulo III.7 “Teselado” (pag. 235) se lleva a cabo en el seno de la librería *libcellbackend* por una serie de clases que forman el núcleo duro de

su funcionalidad.

Estas clases son *Catalog*, *Variable* y *GridderTask*. Esta última clase es la más importante, ya que define el análisis de teselado. Dicha clase es una superclase, es decir, es el fundamento para crear clases específicas, especializadas, que implementan la lógica de cada uno de los tipos de análisis de teselado que la plataforma puede llegar a realizar. La clase *Variable* gestiona las variables de adscripción generadas por los análisis, mientras que *Catalog* sólo es utilizada por las variables de adscripción de tipo categórico o discreto.

Los detalles del funcionamiento de estas clases se discuten más adelante en este mismo capítulo. Vamos a describir aquí, no obstante, los distintos tipos de clases derivadas de *GridderTask* en los que se han implementado los análisis de adscripción.

A la hora de planificar, diseñar e implementar un nuevo algoritmo de teselación, hay que tener en cuenta una serie de cuestiones principales, como siempre, derivadas de la dualidad de la información geográfica:

1. **tipo de geometría:** cada *GridderTask* va a actuar sobre un tipo de geometría concreto. No se diseña igual una *GridderTask* cuya tarea va a ser teselar nubes de puntos que otra que va a particionar polígonos, o medir distancias de líneas;

2. **el tipo de operación geométrica a realizar sobre las geometrías originales:** evidentemente, es un punto muy dependiente del anterior, y viene muy marcado por las características topológicas de las distintas geometrías. Con los puntos las opciones son más o menos reducidas: recuento por contención en tesela, distancias, interpolaciones, etc. Sin embargo, con los polígonos son muchas más, como intersecciones y colisiones con la tesela o entre ellos, análisis de área, de contacto, etc.;

3. **la transformación de la información temática vinculada a las geometrías:** la teselación se hace para agregar por un método u otro la información temática de las geometrías originales en el contexto de la tesela. Por lo tanto, hay que tener muy claro qué operaciones se va a imponer sobre qué información temática original. Para ello, van a haber grandes diferencias en las posibilidades de transformación y operación dependiendo de la naturaleza del dato original:

    1. **dependiente del tipo de dato:** las operaciones disponibles para variables numéricas son distintas que para la información textual, o de fecha, por ejemplo;
    2. **dependiente de la naturaleza continua o discreta del dato:** en datos numéricos, las opciones disponibles también dependerán de si la variable en cuestión es continua o discreta.

Las *GridderTask* son objetos complejos con muchas más sutilezas que las referenciadas en este breve apartado, la mayoría de ellas orientadas al incremento de la eficiencia en su trabajo. En base a estas premisas de diseño, a continuación hacemos una primera y breve presentación de las *GridderTask* implementadas en el prototipo *Cell*. Cada una de estas clases genera un análisis de adscripción que son conocidos en la plataforma con un identificador que también se detalla:

1. **clase *PointAggregationsGridderTask*:** esta *GridderTask*, como su propio nombre indica, está diseñada para hacer agregaciones de puntos. La información original debe ser una nube de puntos, y dichos puntos pueden llevar cualquier dotación de información temática. La *GridderTask* permite definir las operaciones de agregación que se quieren hacer sobre cada uno de los campos temáticos de los puntos. Así, por ejemplo, se podría calcular el total agregado sobre una variable, mientras que sobre otra (o sobre la anterior, por qué no) se calcula la media. Los análisis de adscripción realizados con esta clase tienen el tipo interno en la librería ***POINTAGGREGATIONS***;

2. **clase *PointIdwGridderTask*:** esta *GridderTask* realiza una interpolación puntual en función del algoritmo *IDW* sobre una nube de puntos. Tomando como punto de cálculo para el *IDW* el centro de la tesela a procesar, se buscan los vecinos más próximos en la nube de puntos, se toma una variable numérica continua de los mismos, y se interpola, asignando a la tesela el valor de la interpolación. El tipo interno es ***POINTIDW***;

3. **clase *MdtProcessingGridderTask*:** este *GridderTask* es un ejemplo de versatilidad en la definición de este concepto. El algoritmo está destinado al procesamiento de un Modelo Digital del Terreno, aunque es aplicable a cualquier nube de puntos con una variable temática continua. El algoritmo se bifurca: si el nivel de resolución de la tesela se encuentra por encima del nivel de resolución de la nube de puntos, se calcula la media aritmética de la variable continua con los puntos que caen en la tesela, pero si está por debajo, ésta adquiere como valor final el valor de una interpolación *IDW* frente a la nube de puntos. Su tipo es ***MDTPROCESSING***;

4. **clase *DiscretePolyTopAreaGridderTask*:** este *GridderTask* es muy sencillo, ya que simplemente toma una capa original poligonal con una variable temática discreta[26], corta los polígonos que colisionan con la tesela y encuentra la categoría con más área. Esta categoría es asignada a la tesela en su conjunto. Su tipo es ***DISCRETEPOLYTOPAREA***;

5. **clase *DiscretePolyAreaSummaryGridderTask*:** este *GridderTask* es similar al anterior, aunque más elaborado. Funciona también para teselaciones de polígonos con variables temáticas discretas, pero en lugar de quedarse sólo con la categoría ganadora guarda el sumario completo de ocupación de la tesela por cada categoría presente. Su tipo es ***DISCRETEPOLYAREASUMMARY***.

Los pasos y consideraciones que tendría que tener en cuenta un programador para crear una nueva *GridderTask* se detallan más adelante en este mismo capítulo.

III.6.10. Implementación de *GridderTasks* en *libcellbackend*

A continuación vamos a describir la implementación de las clases *GridderTask* en la librería *libcellbackend*.

[26] Por tanto, pueden ser categorías textuales, como nombres de divisiones administrativas, por ejemplo.

La librería *libcellbackend* contiene una clase *GridderTask* que es una clase base de la cual deben heredar todas las clases derivadas que implementan cada análisis de adscripción en concreto. Esta clase base, como todas las clases base, proporciona métodos y miembros comunes a todos los análisis de adscripción, definiendo una interfaz común de clase. Esto es importante ya que los trabajadores de teselado deben tratar a todos los análisis de adscripción con unas reglas comunes, con un comportamiento uniforme. Si no, cada análisis de adscripción sería radicalmente distinto del resto, necesitando por tanto un tratamiento especial y rompiéndose uno de los objetivos de la ingeniería de *software*: la generalización de los procesos[27]. Esta clase base impone un marco de referencia a un programador para diseñar sus propias clases *GridderTask*.

Las clases derivadas *GridderTask* implementadas en el prototipo son:

1. ***DiscretePolyAreaSummaryGridderTask*:** adscripción poligonal generando variables de adscripción categóricas con un catálogo adjunto;

2. ***DiscretePolyTopAreaGridderTask*:** adscripción poligonal generando una única variable de adscripción categórica;

3. ***MdtProcessingGridderTask*:** adscripción de nube de puntos con interpolación de un modelo digital del terreno;

4. ***PointAggregationsGridderTask*:** adscripción de nubes de puntos con variables continuas;

5. ***PointIdwGridderTask*:** adscripción de nubes de puntos con interpolación por medio de Inverse Distance Weight.

III.6.11. Diseño de una clase de la familia *GridderTask*

Recordemos que la finalidad última de una clase *GridderTask* es la generación, en el vector de adscripción de cada tesela que colisione con datos originales, de una o varias variables de adscripción. Estas variables de adscripción son controladas por la clase *Variable*.

Cuando se diseña un *GridderTask*, hay que tener claros los siguientes elementos:

1. el tipo de geometría original a teselar: línea, punto, polígono;

2. el tipo de variable temática que lleva asociada, y qué tratamiento se le va a dar a la misma. Las opciones y metodologías para adscribir variables originales discretas o

[27] En ingeniería de *software* se intenta evitar siempre "lo especial". Cuando algo es especial, requiere un tratamiento acorde. Eso significa más código, pero, sobre todo, documentar casos especiales, recordar que este caso en concreto hace las cosas ligeramente distintas, etc. El código corre el riesgo de convertirse en código nada claro, farragoso, el llamado *código spaghetti*, lleno de excepciones, bloques de bifurcación que chequean condiciones especiales, variables *flag*, etc. En definitiva, un código difícil de mantener, extender y reutilizar. En información espacial, las fuentes de datos vectoriales no topológicas, como las *SHAPEFILES*, también se llaman *geometrías spaghetti* exactamente por las mismas razones.

categóricas no son las mismas que para variables continuas. Por ejemplo, las primeras generarán catálogos con las categorías presentes en el dominio de la variable;

3. cualquier característica de configuración que pueda tener el tipo de teselado que se vaya a hacer, como por ejemplo redondeo de decimales en las variables, etc.;

4. cualquier posible estrategia y/o condición de teselado en cascada de teselas descendientes al terminar la adscripción de la tesela en curso.

III.6.12. Variables y catálogos

Hay que tener en cuenta que uno de los objetivos de la plataforma es hacer los datos que van al vector de adscripción de la tesela lo más pequeños posibles. Una de las razones es un límite de diseño del sistema *PostgreSQL*, que no permite guardar tipos *JSONB* de más de 255 megabytes de tamaño. Por tanto, para una única tesela, el máximo de datos en su vector de adscripción no puede superar los 255 *megabytes*, que es bastante, pero no infinito.

Otra de las razones para hacer el tamaño del vector de adscripción lo más pequeño posible es el tránsito de la información de las teselas desde el servidor hasta el cliente. Recordemos que, gracias a la propiedad de la rejilla de estar definida matemáticamente, el tránsito de esta información se ahorra el transporte de la geometría, pero aún así hay que minimizar el tamaño del vector de adscripción. Por ello, las clases *GridderTask* deben mantener el tamaño de las variables de adscripción generadas lo más pequeño posible.

Las clases *GridderTask* generan una o más variables de adscripción. Estas pueden ser de dos tipos: discretas (categóricas) o continuas. Las variables continuas no suelen ser un problema puesto que son números enteros o con un número acotado de decimales (es buena práctica acotarlos a una dimensión razonable). Pero las variables discretas sí lo son. Por ejemplo, supongamos que queremos añadir una variable de adscripción que codifique el municipio que ocupa más área de la tesela. Según los datos del DERA del IECA, el nombre de municipio más largo de Andalucía es “San Sebastián de los Ballesteros” (provincia de Córdoba), con 32 caracteres. Recordemos que el formato *JSON* es un formato clave - valor, por lo que si le añadimos el nombre de la variable, por pequeño que sea, nos quedaría un vector de adscripción con este aspecto:

```
{ municipiomaxarea: "San Sebastián de los Ballesteros" }
```

lo cual corre peligrosamente en contra del objetivo de generar vectores de adscripción pequeños.

Con este objeto, la plataforma, cada vez que se crea un objeto de una clase *GridderTask* para realizar un análisis de adscripción, crea primero las variables en la tabla “variable” de la base de datos. Cada variable, en dicha tabla, tiene un código único para ese *GridderTask* en la columna *variable_key*, siendo este código el que se utilizará para identificar el valor de la variable en el vector de adscripción. Este proceso conlleva los siguientes pasos:

1. lectura desde la base de datos de todos los códigos de variables existentes;
2. generación de un nuevo código abreviado por un algoritmo de *hashing* tomando como semilla de *hash* el nombre de la variable, que también es único;
3. recorte del código *hash* generado al mínimo de caracteres posibles evitando colisiones de nombre con los códigos ya existentes.

El objetivo es tener una colección de códigos lo más cortos posibles. Un algoritmo de *hashing* es un complejo algoritmo matemático que genera, a partir de una semilla, un código alfanumérico de longitud fija que tiene la propiedad de que:

1. siempre es el mismo para la misma semilla;
2. las probabilidades de colisión con el código generado por una semilla distinta son mínimos, es decir, no es imposible que se repitan, pero las probabilidades son absurdamente bajas.

Existen muchos algoritmos de este tipo y su desarrollo pertenece al ámbito de la criptografía, siendo una rama de investigación muy activa puesto que en ella se basa todo el sistema criptográfico y de ciberseguridad que gobierna la red. Las comunicaciones con los bancos, los certificados digitales, etc., todo lo que tenga que ver con criptografía tiene un algoritmo de este tipo detrás.

Por ejemplo, supongamos que vamos a configurar y lanzar en la plataforma un análisis de adscripción del tipo *DISCRETEPOLYTOPAREA*, gobernado por la clase *DiscretePolyTopAreaGridderTask*, que lo que hace es adscribir en el vector una categoría temática presente en los datos de origen según ocupe más área en la tesela. Sigamos con el ejemplo de adscribir el municipio predominante en la tesela. La variable de origen a adscribir es el nombre del municipio ("San Sebastián..."). Hay aproximadamente 770 municipios en Andalucía, que llenan toda su extensión, y con nombres demasiado largos. Es un gasto de espacio inasumible.

Por lo tanto, la plataforma primero lee todos los códigos de variables que ya existan en la misma y estén recogidas ya posiblemente en los vectores de adscripción de las teselas. Supongamos que tenemos estos códigos:

```
e  > Espacio Natural Protegido predominante
3  > Total población
f  > Total población hombres
f2 > Total población mujeres
```

El nombre de la variable se ha determinado que sea "Municipio predominante". Esta es la semilla para el algoritmo de *hashing*, que devuelve un *hash* que es, supongamos, "e8323b4832a923d"[28].

[28] Los *hash* suelen codificarse en uno de los sistemas numéricos que manejan los ordenadores. Como es bien sabido, el sistema numérico de los ordenadores se basa en el binario, donde todos los números se

Como ya existe en el sistema una variable que ha cogido el código "e" (puesto que su *hash*, cuando se creó, empezaba por este caracter), la nueva variable adquiere el código definitivo "e8". Así, los códigos se mantienen lo más pequeños posible.

Ahora tendríamos un vector de adscripción así:

```
{ e8: "San Sebastián de los Ballesteros" }
```

Mejor, pero hay que solucionar el problema del nombre del municipio. El principio es el mismo, ya que estamos ante una variable categórica: haremos una conversión entre el nombre de la categoría, "San Sebastián...", y un *mini-hash* como el descrito para el nombre de la variable. El proceso es similar, solo que esta vez los códigos pueden ser aún más pequeños porque el dominio de las categorías se procesa dentro de una misma variable, es decir, las variables no comparten catálogos. De esta manera, el sistema cogería los 770 largos nombres de los municipios y les generaría un *mini-hash* a cada uno, evitando colisiones entre ellos. Por ejemplo:

```
7   > Abla (Almería)
e   > Abrucena (Almería)
7e0 > Adamuz (Córdoba)
1   > Adra (Almería)
494 > Agrón (Granada)
353 > Aguadulce (Sevilla)
5a  > Aguilar de la Frontera (Córdoba)
cb8 > Alájar (Huelva)
0fa > Alameda (Málaga)
6c2 > Alamedilla (Granada)
dba > Alanís (Sevilla)
4bf > Albaida del Aljarafe (Sevilla)
9   > Albánchez (Almería)
303 > Albánchez de Mágina (Jaén)
97  > Alboloduy (Almería)
6cf > Albolote (Granada)
45  > Albondón (Granada)
6   > Albox (Almería)
9bf > Albuñán (Granada)
2e  > Albuñol (Granada)
    [...]
```

representan con las grafías 0 y 1. Menos conocido es que los valores 0 y 1 suelen organizarse en grupos de 8, un *byte* u *octeto*, por lo que también es útil en informática manejarse con el sistema numérico *octal*, es decir, los números se forman con las grafías del 0 al 7. Y por último también se usa mucho, sobre todo en las representaciones de memoria, el sistema *hexadecimal*, el doble del octal, con lo que se precisan 16 grafías para representar los números. Se usan las grafías del 0 al 9 más las letras "A", "B", "C", "D", "E" y "F". Esto es fácil de ver, por ejemplo, en los códigos de color de los sistemas de información geográfica a la hora de diseñar semiología cartográfica. En hexadecimal, "10" se escribe "A", y "FF" es, en decimal, "255". Los números mágicos de las máquinas, las potencias de 2.

de forma que nuestras teselas en las que Abla es el municipio con más área tendría una variable de adscripción de este tipo en el vector de las teselas:

```
{ e8: 7 }
```

que es muy ventajoso a la hora de transportar la información y aprovechar el espacio en la base de datos. Lógicamente, cuantas más variables tenga el sistema, más largos se harán los *mini-hashes*. Por ejemplo, en el proceso de adscripción con el método *DISCRETEPOLYTOPAREA*, que usa este sistema, de los municipios, el *mini-hash* más largo tiene 4 caracteres.

El sistema es capaz de proveer a los clientes con estos catálogos para que puedan interpretar los datos del vector de adscripción, en los dos sentidos.

III.6.13. Variables de indexación

Un problema de rendimiento detectado en la *PostgreSQL* a la hora de gestionar los vectores de adscripción, en las primeras versiones de la plataforma, es el filtrado de claves dentro de los *JSON*. Si bien la *PostgreSQL* provee de funciones de sobra para el tratamiento y filtrado de este tipo de datos, cuando los datos se hacen muy grandes y se busca la presencia dentro del *JSON* de múltiples claves del mismo, el rendimiento tiende a caer. Por ejemplo, con varios miles de teselas en el sistema, y con varios cientos de variables potenciales en los vectores de adscripción, como veremos en la discusión siguiente de los tipos de *GridderTask* implementados, buscar las teselas que tienen en su vector de adscripción un determinado número de claves *JSON* se enlentece mucho.

Pongamos por ejemplo, adelantándonos un poco, el funcionamiento del *GridderTask DISCRETEPOLYAREASUMMARY*, que crea un sumario de la presencia de área de un grupo de categorías poligonales. Por ejemplo, cuando se aplica este análisis de teselado a los Espacios Naturales Protegidos (EENNPP) obtendremos una variable de adscripción para el total de ocupación de área de cada EENNPP en cada tesela de, digamos, un kilómetro de resolución. De esta manera, dicho análisis generará una variable de adscripción que sería "Área del Parque Natural de los Alcornocales", otra que sería "Área del Paraje Natural Acantilados de Maro-Cerro Gordo", y así para todo el catálogo de más de 200 EENNPP andaluces. Imaginemos ahora que queremos obtener un determinado vector de adscripción, con un número X de variables de adscripción de nuestro interés, exclusivamente en teselas que forman parte de algún EENNPP. Las variables de adscripción del análisis que procesó los EENNPP, esas más de 200 variables, nos permitiría actuar de filtro para filtrar la presencia de cualquier EENNPP. Por lo tanto, el filtro que pasaríamos a la plataforma, en pseudocódigo[29]:

[29]Por *pseudocódigo* se hace referencia en ingeniería de *software* a la expresión técnica de algoritmos, estructuras de datos, etc. cuya finalidad no es la implementación de dicho código en un lenguaje de programación en concreto, ya que la sintaxis varía mucho entre lenguaje y lenguaje. Con el uso del pseudocódigo, los ingenieros pueden transmitir la idea de un algoritmo complejo sin pararse en los detalles y sutilezas de la implementación del mismo en un lenguaje de programación determinado. Con la lectura del pseudocó-

```
filtro =
  [ presencia de la variable de adscripción "Área del
    Parque Natural de los Alcornocales ]

    Ó

  [ presencia de la variable de adscripción "Área del
    Paraje Natural Acantilados de Maro-Cerro Gordo ]

    Ó

  [ ... ]

    Ó

  [ presencia de la variable de adscripción "Área del
    EENNPP número 200" ]
```

es decir, para encontrar las teselas sobre un EENNPP tenemos que testear si existe presencia de *cualquiera* de los EENNPP existentes.

Estos filtros con tantos *Ó* lógicos es lo que es un serio perjuicio para el rendimiento de los mecanismos de indexación de *PostgreSQL* sobre tipos de datos *JSON*. Recordemos que esta funcionalidad es relativamente reciente en *PostgreSQL* y que *PostgreSQL* es ante todo una base de datos relacional, por lo que no es tan potente en este área como sistemas más especializados. Buscar la presencia opcional (claúsula *Ó*) de 200 variables en *JSON* en potencialmente cientos o miles de teselas es demasiado lento.

Para evitar este problema, los análisis de teselación implementados con la clase *GridderTask* generan, aparte de las variables de adscripción que le son propias, y que se describen en los siguientes apartados, una variable extra especial que recibe el nombre de *variable de indexación*. Dicha variable le indica a la plataforma que un vector de adscripción de una tesela posee *alguna* o *algunas* de las variables de adscripción de una familia determinada. En el caso de los EENNPP anteriormente descrito, aparte de las 200 variables de adscripción de área de cada uno de los EENNPP, el análisis de adscripción de tipo *DISCRETEPOLYAREASUMMARY* escribirá esta variable de indexación para marcar la presencia de un EENNPP en la tesela, por lo que los filtros de tipo "buscar todas las teselas que tienen un EENNPP" se agilizan mucho porque ya sólo hay que buscar una variable de adscripción, no 200.

Esta técnica también ayuda a agilizar la búsqueda de un grupo de variables de adscripción, al filtrar previamente las teselas que pueden tener los objetos buscados. Por ejemplo, en el caso de buscar una muestra de 100 de los 200 EENNPP, la condición de claúsulas *Ó* encadenadas descritas anteriormente sigue siendo bastante pesada, pero al menos la

digo, un ingeniero puede entender la idea e implementarla en el lenguaje de su elección. Es, por tanto, una importante herramienta de comunicación.

variable de indexación ha hecho un filtrado previo de todas las teselas donde no existen EENNPP y ya la búsqueda se circunscribe a aquellas que se sabe tienen alguno, lo que agiliza también el proceso.

Aún así, éste es uno de los puntos clave de mejora de la implementación *Cell* propuesta y descrita en este trabajo.

III.6.14. La clase base *GridderTask*

Todas las clases *GridderTask* comparten los miembros y métodos de su clase base, *GridderTask*. Eso quiere decir que en la definición de una nueva *GridderTask* va a haber elementos comunes con el resto de sus compañeras:

1. todas tienen que tener un tipo de análisis, un nombre que identique el tipo de análisis de teselado que llevan a cabo. Para cada clase de *GridderTask* este tipo es distinto;

2. todas tienen un nombre y una descripción del análisis de teselado que hacen. Este nombre y descripción es de cada análisis que se vaya a hacer con la clase, no de la clase en sí. Por ejemplo, con la clase *DiscretePolyTopAreaGridderTask* se podría lanzar un análisis sobre los municipios (de nombre, por ejemplo, "Municipio preponderante") y otro distinto sobre las provincias, con otro nombre. La clase utilizada es la misma, pero son dos análisis de adscripción distintos;

3. todas tienen una tabla de origen de datos, "sourceTable", donde se encuentran los datos de origen a teselar;

4. dentro de esa tabla, todas tienen el nombre del campo que contiene la geometría, "geomField";

5. todas tienen el código de su variable índice.

El resto de parámetros de configuración de una clase de análisis en concreto diferirán según la aproximación y la lógica de la adscripción que implementan.

A continuación se describe el interfaz de la clase base *GridderTask*.

Miembros de la clase *GridderTask*

gridderTaskType: EGRIDDERTASKTYPE

El código del tipo de análisis que implementa la clase, sobre todo para identificarlo cuando el objeto guarde sus datos en la base de datos. Por ejemplo, "POINTAGGREGATIONS" para la clase *PointAggregationsGridderTask*.

gridderTaskTypeName: string

El nombre del análisis en sí, no de un análisis concreto.

gridderTaskTypeDescription: string

La descripción del análisis en sí, no de un análisis concreto.

sourceTable: string

La tabla de los datos de origen en la *PostgreSQL* que los alberga.

gridderTaskId: string

El ID del análisis concreto que un objeto de la clase está definiendo. Por ejemplo, "Municipio predominante".

name: string

El nombre del análisis concreto.

description: string

La descripción del análisis concreto.

geomField: string

El nombre de la columna de *sourceTable* que alberga la geometría original.

indexVariableKey: string

El *mini-hash* de la variable índice del análisis concreto.

indexVariable: Variable

Un objeto *Variable* con la variable índice.

variables: Variable[]

Una lista con objetos *Variable* con todas las variables del análisis concreto.

Métodos de la clase *GridderTask*

computeCell$(sourcePg: RxPg, cellPg: RxPg, cell: Cell, minZoom: number): rx.Observable<Cell

>

Analiza la adscripción de una tesela. Para ello, el método precisa de una conexión a la base de datos de origen (*sourcePg*), otra conexión a la base de datos de la plataforma (*cellPg*), la tesela a adscribir, como objecto *Cell* y el nivel de resolución mínimo en la pirámide (máximo en resolución) hasta el que se quiere teselar (*minZoom*). Devuelve las teselas adscritas (puede adscribir teselas hijas en el proceso, depende del análisis) como una lista. Este método es exclusivo de cada clase hija del tipo *GridderTask*. La clase base lo implementa para determinar el interfaz común de la familia de clases pero arroja un error si no está reimplementada en la clase heredada.

setup$(sourcePg: RxPg, cellPg: RxPg): rx.Observable<GridderTask>

Prepara el análisis antes de comenzar a adscribir teselas. Esta función, también propia de cada clase hija y, por tanto, necesitada de ser reimplementada en ellas, crea las variables y los catálogos necesarios en la base de datos de la plataforma, si los hubiera, para apoyar el resultado de la adscripción. Tan sólo precisa de conexiones tanto a los datos de origen como a la base de datos de la plataforma.

setIndexVariableKey$(cellPg: RxPg, indexVariableKey?: string): rx.Observable<GridderTask>

Este método establece la clave de la variable índice.

getVariables$(cellPg: RxPg): rx.Observable<GridderTask>

Carga las información de variables de la *GridderTask* en el miembro *variables*. También carga la variable índice.

computeCellsBatch$(sourcePg: RxPg, cellPg: RxPg, cells: Cell[], targetZoom: number): rx.Observable<any>

Esta función analiza la adscripción de un grupo de teselas dadas por el parámetro *cells*. Precisa de conexión a las bases de datos de origen y de la plataforma y del nivel de resolución máximo que se pretende alcanzar. Este método no está pensado para que lo usen los trabajadores, que adscriben tesela a tesela, sino que está pensado para lanzar manualmente la adscripción de un grupo de teselas en modo local, para pruebas y pequeños trabajos de adscripción.

III.6.15. Clases de la familia *GridderTask*

A continuación describimos las clases derivadas de la clase base *GridderTask* que constituyen la oferta de análisis de adscripción implementados en el prototipo, de la más sencilla a la más compleja.

Se muestra, junto a la descripción de cada una, el *JSON* que permite configurar el análisis para ver qué propiedades tiene dicha configuración. Esta configuración es la que inicializa un objeto de la clase, es decir, es la que determina, dentro del marco del interfaz de la misma, qué características concretas tiene el análisis con el que se va a trabajar.

Clase *DiscretePolyTopAreaGridderTask*

Esta clase rige los análisis de teselado de tipo "DISCRETEPOLYTOPAREA". Geométricamente, trabaja sobre polígonos, y, alfanuméricamente, sobre una variable categórica de los mismos. El objetivo es encontrar, entre los polígonos, qué categoría ocupa más área de la tesela.

Este sencillo *GridderTask* genera una única variable de tipo categórico con un catálogo asociado. Dicha única variable, además, es la variable índice del análisis. Su *JSON* de configuración es el siguiente:

```
{
  "gridderTask": {
    "gridderTaskId": "municipioDiscretePolyTopArea",
    "name": "Municipio máxima área",
    "description": "Teselado de municipios con sus
       provincias por máxima área usando el algoritmo
       DiscretePolyTopAreaGridderTask.",
    "sourceTable": "context.municipio",
    "geomField": "geom",
    "discreteFields": [ "provincia", "municipio" ],
    "variableName": "Municipio: máxima área",
    "variableDescription": "Nombre del municipio y su
       provincia del municipio que ocupa la mayor área de
        la tesela.",
    "categoryTemplate": "{{{municipio}}} ({{{provincia
       }}})"
  }
}
```

Código III.6.3: Configuración *JSON* de un *GridderTask* de tipo "DISCRETEPOLYTOPAREA".

donde encontramos elementos de configuración comunes a todas las clases *GridderTask* que vamos a comentar:

1. ***gridderTaskId***, ***name*** y ***description*** son el identificador, el nombre y la descripción del análisis que se está configurando, respectivamente;
2. ***sourceTable*** es el nombre de la tabla de origen de datos, en la base de datos de origen, que contiene los datos originales a teselar;
3. ***geomField*** es la columna de datos que contiene la geometría a teselar.

Y también encontramos elementos específicos para la lógica de teselación de esta clase en concreto:

1. ***discreteFields*** es una lista de campos de *sourceTable* que van a componer, conjuntamente, la categoría de cada uno de los polígonos a adscribir. En este caso, estamos generando una categoría en base a dos campos discretos. Por un lado, el nombre del municipio y, por otro, la provincia a la que pertenece;
2. ***variableName*** y ***varibleDescription*** es el nombre y la descripción, respectivamente, de la única variable que va a generar este *GridderTask*;
3. ***categoryTemplate*** es la plantilla para la creación de la categoría en el catálogo para cada polígono. En este caso, las categorías de crearán componiendo descriptores del tipo "Nombre del municipio (provincia)". Nótese el uso de los símbolos "{{{" y "}}}" como marcas de substitución de las dos columnas designadas como descriptores categóricos en el dato de configuración *discreteFields*.

Todos los análisis, antes de estar listos para teselar, deben ejecutar su método *setup$*, que los prepara, realizando una serie de tareas administrativas previas. Durante su preparación, este análisis va a crear una única variable en el sistema que se llamará, en el caso del ejemplo, "Municipio: máxima área", con la descripción pasada en *variableDescription*. A esta variable el sistema le asignará un *mini-hash* que no colisione con los existentes, por supuesto.

A continuación, el método *setup$*, dado que este análisis genera una variable categórica, ha de construir e insertar en el catálogo de la plataforma todas las posibles categorías temáticas que se encuentran en el dominio de la variable, según los campos descritos en *discreteFields*. Para ello, toma la plantilla de categoría presentada en *categoryTemplate* y lee todos los datos diferentes que existan en esos campos, componiendo la plantilla. De esta manera, si los datos originales eran (extracto):

municipio	provincia
Abla	Almería
Abrucena	Almería
Adamuz	Córdoba
Adra	Almería
Agrón	Granada
Aguadulce	Sevilla
Aguilar de la Frontera	Córdoba
Alájar	Huelva
Alameda	Málaga
Alamedilla	Granada
Alanís	Sevilla
Albaida del Aljarafe	Sevilla
Albánchez	Almería
Albánchez de Mágina	Jaén
Alboloduy	Almería
Albolote	Granada
Albondón	Granada
Albox	Almería
Albuñán	Granada
Albuñol	Granada
Albuñuelas	Granada
Alcalá de Guadaira	Sevilla
Alcalá de los Gazules	Cádiz
Alcalá del Río	Sevilla
Alcalá del Valle	Cádiz
Alcalá la Real	Jaén
Alcaracejos	Córdoba
Alcaucín	Málaga
Alcaudete	Jaén
Alcolea	Almería
Alcolea del Río	Sevilla

municipio	provincia
Alcóntar	Almería
Alcudia de Monteagud	Almería
Aldeaquemada	Jaén
Aldeire	Granada
Alfacar	Granada
Alfarnate	Málaga
Alfarnatejo	Málaga
Algámitas	Sevilla

el análisis creará, por composición y substitución de los campos *discreteFields* en la plantilla *categoryTemplate*:

key	value
7	Abla (Almería)
e	Abrucena (Almería)
7e0	Adamuz (Córdoba)
1	Adra (Almería)
494	Agrón (Granada)
353	Aguadulce (Sevilla)
5a	Aguilar de la Frontera (Córdoba)
cb8	Alájar (Huelva)
0fa	Alameda (Málaga)
6c2	Alamedilla (Granada)
dba	Alanís (Sevilla)
4bf	Albaida del Aljarafe (Sevilla)
9	Albánchez (Almería)
303	Albánchez de Mágina (Jaén)
97	Alboloduy (Almería)
6cf	Albolote (Granada)
45	Albondón (Granada)
6	Albox (Almería)
9bf	Albuñán (Granada)
2e	Albuñol (Granada)
e24	Albuñuelas (Granada)
67a	Alcalá de Guadaira (Sevilla)
f3	Alcalá de los Gazules (Cádiz)
0a7	Alcalá del Río (Sevilla)
84	Alcalá del Valle (Cádiz)
68a	Alcalá la Real (Jaén)
910	Alcaracejos (Córdoba)
895	Alcaucín (Málaga)
0ac	Alcaudete (Jaén)
c	Alcolea (Almería)

key	value
898	Alcolea del Río (Sevilla)
f	Alcóntar (Almería)
2	Alcudia de Monteagud (Almería)
9e0	Aldeaquemada (Jaén)
0ea	Aldeire (Granada)
6ab	Alfacar (Granada)
eee	Alfarnatejo (Málaga)
ed	Alfarnate (Málaga)
a30	Algámitas (Sevilla)

cada una, obviamente, con su *mini-hash*. Estos pares clave - valor son los que se insertan en el catálogo de la plataforma. Esto concluye la configuración previa del análisis por parte del método *setup$*.

Una vez hecho el *setup$*, el análisis ya está listo para teselar. El proceso de teselación se define a continuación:

1. **determinación de geometrías originales colisionantes:** los polígonos que solapan la geometría de la tesela son candidatos a la adscripción;
2. **carga de información original colisionante:** se cargan los datos de los polígonos colisionantes de las columnas *discreteFields* y se compone con ellos la descripción de la categoría según el campo *categoryTemplate*. Ésta categoría se busca en el catálogo para encontrar su *mini-hash*;
3. **análisis geométrico de la información original colisionante:** los polígonos colisionantes se interseccionan con la geometría de la tesela, dejando su área de afectación sobre la misma. Se calcula el área de las intersecciones, agrupando aquellas que posean la misma categoría según *categoryTemplate*, para encontrar el total por categoría;
4. **análisis alfanumérico de la información original colisionante:** de las áreas totales por categoría, se coge la que más área ocupa. Su *mini-hash* será el valor de la variable para este análisis para la tesela en curso;
5. **análisis de la herencia inter-resolución:** en este caso, el análisis de la herencia se basa en una sencilla regla, ya que si la categoría ganadora tiene el 100 % del área de la tesela, todas sus descendientes la tendrán también al 100 %. Por lo tanto, en este caso, todas las teselas descendientes hasta el nivel de resolución objetivo se quedan con el valor de la tesela en curso. Si esta condición no se da, el análisis devuelve las teselas hijas en el siguiente nivel de resolución para su adscripción.

Clase *DiscretePolyAreaSummaryGridderTask*

Esta clase es similar, geométricamente hablando, a la anterior, pero, en lugar de quedarse con la categoría ganadora en área en la tesela, describe el sumario completo de

ocupación de cada una de las categorías presentes en la misma. Por ello, en lugar de generar una única variable de adscripción, este análisis va a generar tantas variables como categorías haya en el dominio de los datos originales. Así, en el ejemplo dado, sobre el desglose de área de cada municipio de Andalucía, el análisis generará durante la ejecución de su método ***setup$*** tantas variables de adscripción como municipios haya, es decir, aproximadamente 770, más una variable de indexación adicional. Su *JSON* de configuración es el siguiente:

```
{
  "gridderTask": {
    "gridderTaskId": "municipioDiscreteAreaSummary",
    "gridderTaskType": "DISCRETEPOLYAREASUMMARY",
    "name": "Desglose de área de municipios",
    "description": "Área de cada municipio en la tesela,
       incluyendo su provincia.",
    "sourceTable": "context.municipio",
    "geomField": "geom",
    "discreteFields": [ "provincia", "municipio" ],
    "variableNameTemplate": "Área {{{municipio}}} ({{{
       provincia}}})",
    "variableDescriptionTemplate": "Área del municipio
       {{{municipio}}}, provincia {{{provincia}}}."
  }
}
```

Código III.6.4: Configuración de un *GridderTask* de tipo "DISCRETEPOLYAREASUMMARY".

En él se pueden observar los campos de configuración comunes a todas las *GridderTask*, ya descritos en la sección anterior. Como parámetros de configuración propios, esta *GridderTask* tiene:

1. ***discreteFields*:** al igual que el caso anterior, dado que es una adscripción también de variable categórica, el campo ***discreteFields*** identifica a los campos de origen que sirven para componer la categoría;

2. ***variableNameTemplate*:** es la plantilla para conformar, con los campos dados en ***discreteFields***, el nombre de la variable para cada categoría temática;

3. ***variableDescriptionTemplate*:** es la plantilla para generar, igual que el parámetro de configuración anterior, la descripción de la variable de adscripción para cada categoría temática.

Supongamos, pues, que tenemos los mismos datos de origen que en el análisis anterior. Este análisis, durante su preparación, generaría las siguientes variables de adscripción en la tabla "variable" de la base de datos:

variable_key	name
5	Área Abla (Almería)
2	Área Abrucena (Almería)
ef	Área Adamuz (Córdoba)
1	Área Adra (Almería)
a3	Área Agrón (Granada)
3e4	Área Aguadulce (Sevilla)
71e	Área Aguilar de la Frontera (Córdoba)
2eb	Área Alájar (Huelva)
1cd	Área Alameda (Málaga)
52	Área Alamedilla (Granada)
ab3	Área Alanís (Sevilla)
1f2	Área Albaida del Aljarafe (Sevilla)
2e	Área Albánchez (Almería)
d43	Área Albánchez de Mágina (Jaén)
6	Área Alboloduy (Almería)
56	Área Albolote (Granada)
bf4	Área Albondón (Granada)
59	Área Albox (Almería)
94	Área Albuñán (Granada)
fb5	Área Albuñol (Granada)
e50	Área Albuñuelas (Granada)
5a3	Área Alcalá de Guadaira (Sevilla)
39	Área Alcalá de los Gazules (Cádiz)
d32	Área Alcalá del Río (Sevilla)
7f	Área Alcalá del Valle (Cádiz)
37d	Área Alcalá la Real (Jaén)
c7	Área Alcaracejos (Córdoba)
c83	Área Alcaucín (Málaga)
24c	Área Alcaudete (Jaén)
d	Área Alcolea (Almería)
b4b	Área Alcolea del Río (Sevilla)
5e	Área Alcóntar (Almería)
58	Área Alcudia de Monteagud (Almería)
6e	Área Aldeaquemada (Jaén)
f8	Área Aldeire (Granada)
90	Área Alfacar (Granada)
4f40	Área Alfarnatejo (Málaga)
220	Área Alfarnate (Málaga)
2ad0	Área Algámitas (Sevilla)

Es decir, una tesela con presencia de la variable con clave “b4b” estaría denotando la presencia de parte del área del municipio “Alcolea del Río”. Las descripciones, que por

espacio no se muestran, seguirían un patrón similar, marcado por el ítem de configuración ***variableDesciptionTemplate***. Dado que el valor de cada una de estas variables de adscripción de área es un número, este análisis no genera entradas de catálogo.

El proceso de teselación ya puede aplicarse a teselas:

1. **determinación de geometrías originales colisionantes:** al igual que el anterior, los polígonos que solapan la geometría de la tesela son candidatos a la adscripción;

2. **carga de información original colisionante:** igual que el anterior, se cargan los datos de los polígonos colisionantes de las columnas ***discreteFields*** y se compone con ellos el nombre de la categoría según el campo ***variableNameTemplate***. Con este valor se busca en el conjunto de variables del sistema y se encuentra el *mini-hash* que corresponde a cada una de las variables que controlan el área de cada uno de los municipios presentes;

3. **análisis geométrico de la información original colisionante:** igual que el anterior. Los polígonos se interseccionan y se calcula el área de las intersecciones, agrupando aquellas que posean la misma categoría según los campos de ***discreteFields***, para encontrar el total por categoría;

4. **análisis alfanumérico de la información original colisionante:** se cogen las áreas de cada categoría y se busca el *mini-hash* de su variable. Se escriben en el vector de adscripción de la tesela todas ellas, con el área de solape interpretada con un valor porcentual del 0 al 1, redondeada a dos decimales;

5. **análisis de la herencia inter-resolución:** este caso es similar al anterior. Si sólo una categoría está presente y con el 100 % de área, todas sus descendientes la tendrán también al 100 %. Si no, hay que devolver las teselas hijas y aplicarles la adscripción a cada una de ellas.

Clase *PointIdwGridderTask*

Esta clase trabaja con nubes de puntos para interpolar teselas en las que no existen datos. Supongamos que tenemos una nube de puntos, ya sea regular, como por ejemplo la que proporciona un ráster, o irregular, como la que proporciona una *TIN*[30], y que queremos interpolar a partir de dicha nube sobre una resolución de rejilla inferior a la propia resolución de la nube. Obviamente, aquellas teselas que colisionen con puntos con datos están de suerte, ya que tienen una relación topológica directa con los mismos, pero aquellas que no tienen colisión por inclusión deben contentarse con una interpolación gracias a los puntos más cercanos.

[30] Una *Triangular Irregular Network*, una forma de discretizar un ámbito geográfico en función de puntos de muestreo irregular a los que se le aplica un algoritmo que genera una cobertura triangular del área de recubrimiento. La propiedad de dichos triángulos es ser lo más pequeños posible. Esta técnica se usa, por ejemplo, en la creación de modelos digitales del terreno, pero se usa muchísimo en TIG y procesamiento de inteligencia artificial de superficies. Y también mucho en gráficos 3D, donde estas estructuras de datos forman la malla que dan forma a los objetos tridimiensionales.

Este análisis, por lo tanto, trabaja con geometrías originales que no tienen por qué estar dentro de la propia tesela, es decir, que trabajan con un ámbito exterior a la misma. Como metodo de interpolación, que existen muchos, se ha optado por el clásico *Inverse Distance Weigth*, por su sencillez de implementación.

El *JSON* de configuración de este análisis de adscripción es el siguiente:

```
{
  "gridderTask": {
    "gridderTaskId": "gridderTaskMdtIdwProcessing",
    "gridderTaskType": "POINTIDW",
    "name": "Interpolación IDW de datos MDT",
    "description": "Test de interpolación IDW sobre los
       datos del MDT.",
    "sourceTable": "mdt.mdt",
    "geomField": "geom",
    "maxDistance": 200,
    "numberOfPoints": 16,
    "valueField": "h",
    "round": 1,
    "power": 2,
    "variableName": "Procesamiento IDW MDT",
    "variableDescription": "Procesamiento de datos del
       MDT por medio de interpolación IDW."
  }
}
```

Código III.6.5: Configuración de un *GridderTask* de tipo "POINTIDW".

Las variables de configuración específicas de este análisis son las siguientes:

1. ***maxDistance*:** distancia máxima de búsqueda de puntos de datos para alimentar al algoritmo *IDW*. Más allá de esta distancia máxima la influencia de los puntos es tan baja por la caida del modelo gravitatorio del *IDW* que es despreciable. Por supuesto, este parámetro depende muchísimo de la resolución de los datos originales, en el caso de mallas regulares. Este valor es adecuado para 100 metros de resolución, que es lo que tiene el DERA del IECA considerado;

2. ***numberOfPoints*:** el número de puntos más cercanos para la convolución *IDW*, es decir, se cogen los puntos que van a aportar más a la interpolación por su distancia al punto objetivo;

3. ***valueField*:** el campo de la tabla de datos original que posee la variable de interpolación (una cota, temperatura, precipitación, etc.);

4. ***round*:** decimales de redondeo de la interpolación;

5. ***power*:** factor de caída de la influencia del punto por potencia de la distancia;

6. ***variableName* y *variableDescription*:** nombre y descripción de la variable a generar por el análisis de adscripción.

Este análisis de adscripción genera una única variable de adscripción para almacenar el resultado de la interpolación durante la ejecución de su método ***setup$***.

El análisis funciona de la siguiente manera:

1. **determinación de geometrías originales colisionantes:** se toma el centro de la tesela objetivo, que será el punto significativo para la interpolación de la tesela. Se encuentran todos los puntos que estén a menos de ***maxDistance*** de este punto objetivo;

2. **carga de información original colisionante:** se cargan los valores ***valueField*** de cada punto de los anteriores;

3. **análisis geométrico de la información original colisionante:** se mide la distancia de cada punto seleccionado, seleccionando los ***numberOfPoints*** más cercanos;

4. **análisis alfanumérico de la información original colisionante:** se toman los valores ***valueField*** de los puntos más cercanos seleccionados en el punto anterior y, junto a las distancia encontradas para los mismos, se calcula el *IDW* del centroide de la tesela objetivo;

5. **análisis de la herencia inter-resolución:** en este caso no se puede aplicar ninguna lógica de herencia inter-resolución, ya que el valor calculado para la tesela objetivo no sirve para ninguna de las celdas hijas, y ni siquiera la presencia de puntos dentro de ***maxDistance*** está garantizada para las teselas hijas que se encuentran en los límites de la nube de puntos.

Clase *PointAggregationsGridderTask*

Esta clase rige los análisis de teselado de tipo "POINTAGGREGATIONS". Este análisis trabaja sobre fuentes de datos puntuales, agregando los valores temáticos mediante una función de agrupación de datos que será asignada a tantas variables como expresiones se incluyan en el cálculo del análisis de adscripción.

Las funciones de agrupación de datos actuan sobre un conjunto de datos recolectados de varios registros de una tabla. Por ejemplo, si tuviéramos una nube de puntos con esta estructura temática:

p_total	p_mujeres	p_hombres	geom
299	149	150	[punto]
601	302	299	[punto]
98	52	46	[punto]
526	250	276	[punto]
51	24	27	[punto]

p_total	p_mujeres	p_hombres	geom
455	223	232	[punto]
14	7	7	[punto]
14	5	9	[punto]
11	5	6	[punto]
243	118	125	[punto]
17	8	9	[punto]
82	41	41	[punto]
11	5	6	[punto]
709	361	348	[punto]
1039	499	540	[punto]
991	519	472	[punto]
18	5	13	[punto]

donde los campos, respectivamente, indican población total, de mujeres y de hombres, y donde "geom" es una geometría puntual, este análisis de teselado buscaría qué puntos caen en la tesela objetivo y permitiría operar con cada columna aplicándole a las mismas una función de agrupación estadística. Si en la tesela objetivo cayeran los 5 primeros puntos, las expresiones de agrupación:

1. ***sum(p_total)*** haría el sumatorio de la población total, arrojando 1575;
2. ***round(avg(p_mujeres)::numeric, 2)*** haría la media de la población de mujeres, redondeándola a dos decimales, arrojando 155;
3. ***round(sum(p_mujeres)/sum(p_hombres), 2)*** calcularía el ratio entre mujeres y hombres, redondeado a dos decimales, arrojando 0,97.

Este análisis de teselado creará tantas variables de adscripción como expresiones o fórmulas se apliquen sobre los datos de los puntos que caigan en la tesela, pudiendo, de esta manera, aprovechar un único análisis de teselado para calcular todas las variables que fueran necesarias. Por supuesto, también crea una variable de indexación.

El *JSON* de configuración de este tipo de análisis de teselado es:

```
{
  "gridderTask": {
    "gridderTaskId": "
       gridderTaskPointAggregationsPoblacion",
    "gridderTaskType": "POINTAGGREGATIONS",
    "name": "Estadísticas de población",
    "description": "Estadísticas de población",
    "sourceTable": "poblacion.poblacion",
    "geomField": "geom",
    "variableDefinitions": [
```

```
10        {
11          "name": "Población total 2002",
12          "description": "Población total del año 2002.",
13          "expression": "sum(ptot02)"
14        },
15        {
16          "name": "Población hombres 2002",
17          "description": "Población de hombres del año
              2002.",
18          "expression": "sum(ph02)"
19        },
20        {
21          "name": "Población mujeres 2002",
22          "description": "Población de mujeres del año
              2002.",
23          "expression": "sum(pm02)"
24        },
25        {
26          "name": "Población edad 0-15 2002",
27          "description": "Población de edades entre 0 y 15
              años del año 2002.",
28          "expression": "sum(e001502)"
29        },
30        {
31          "name": "Población edad 16-64 2002",
32          "description": "Población de edades entre 16 y 64
               años del año 2002.",
33          "expression": "sum(e166402)"
34        },
35        {
36          "name": "Población edad mayor 64 2002",
37          "description": "Población de edad mayor de 64 añ
              os del año 2002.",
38          "expression": "sum(e6502)"
39        },
40        {
41          "name": "Población española 2002",
42          "description": "Población española del año
              2002.",
43          "expression": "sum(esp02)"
44        },
45        {
46          "name": "Población UE Schengen 2002",
47          "description": "Población de origen Unión Europea
               (espacio Schengen) del año 2002.",
48          "expression": "sum(ue1502)"
```

```
49        },
50        {
51          "name": "Población magrebí 2002",
52          "description": "Población de origen magrebí del a
              ño 2002.",
53          "expression": "sum(mag02)"
54        },
55        {
56          "name": "Población americana 2002",
57          "description": "Población de origen americano del
                año 2002.",
58          "expression": "sum(ams02)"
59        },
60        {
61          "name": "Población otros 2002",
62          "description": "Población de otros origenes del a
              ño 2002.",
63          "expression": "sum(otr02)"
64        },
65        {
66          "name": "Índice de dependencia total 2002",
67          "description": "Índice de dependencia total de
              2002. El índice de dependencia total es el
              cociente de la suma de población joven y
              anciana entre el total de adultos.",
68          "expression": "round(((sum(e001502)::float + sum(
              e6502)::float) / (nullif(sum(e166402), 0))::
              float)::numeric, 2)"
69        },
70        {
71          "name": "Índice de dependencia infantil 2002",
72          "description": "Índice de dependencia infantil de
                2002. El índice de dependencia infantil es el
                cociente de la suma de población joven entre
              el total de adultos.",
73          "expression": "round(((sum(e001502)::float) / (
              nullif(sum(e166402), 0))::float)::numeric, 2)"
74        },
75        {
76          "name": "Índice de dependencia anciana 2002",
77          "description": "Índice de dependencia anciana de
              2002. El índice de dependencia anciana es el
              cociente de la suma de población anciana entre
                el total de adultos.",
78          "expression": "round(((sum(e6502)::float) / (
              nullif(sum(e166402), 0))::float)::numeric, 2)"
```

```
79          }
80       ]
81    }
82 }
```

Código III.6.6: Configuración de un *GridderTask* de tipo "POINTAGGREGATIONS".

Las variables de configuración específicas de este análisis es sólo una, *variableDefinitions*, donde, en forma de lista[31], vienen recogidas las características de cada una de las variables de adscripción a calcular sobre la tabla original de datos. Las características que definen estas variables son:

1. ***name*:** un nombre para la variable;
2. ***description*:** una descripción para la variable;
3. ***expression*:** la expresión de cálculo de la variable.

El ejemplo mostrado calcula varios sumatorios totales (funciones ***sum()***) de los datos que tienen asociados los puntos, cuyo significado es fácil de entender a partir de las descripciones de las variables, más el cálculo de tres índices de dependencia, que son cocientes entre los sumatorios de algunas variables:

1. ***índice de dependencia total*:** cociente entre los totales de población dependiente (entre 0 y 15 años y mayores de 64) entre la población entre 16 y 64;
2. ***índice de dependencia infantil*:** cociente entre los totales de población entre 0 y 15 años y la que está entre 16 y 64;
3. ***índice de dependencia anciana*:** cociente entre los totales de población mayores de 64 años y el que está entre 16 y 64.

Estas variables se calculan para toda la serie temporal disponible, que incluye datos para los años 2002, 2013, 2014, 2015, 2016, 2017 y 2018, aunque no se muestran todos por ser breves en el código.

Durante la ejecución del método ***setup$*** de configuración de este análisis de teselado, la plataforma crea una variable de adscripción por cada elemento encontrado en la variable de configuración ***variableDefinitions***, además de crear la variable de indexación.

El análisis funciona según estos pasos:

1. **determinación de geometrías originales colisionantes:** se encuentran todos los puntos de la tabla original que caen dentro de la tesela;
2. **carga de información original colisionante:** se cargan todos los datos asociados a esos puntos;

[31] Recordemos que, en *JSON*, las listas están encerradas entre corchetes([]).

3. **análisis geométrico de la información original colisionante:** en este paso no se realiza ningun análisis adicional. La condición de inclusión realizada en el paso 1 aloja toda la lógica geométrica del análisis;
4. **análisis alfanumérico de la información original colisionante:** sobre las columnas de datos del conjunto de puntos colisionantes se aplican cada una de las ***expression*** de la definición de las variables de ***variableDefinitions***, guardándose el valor de cada cálculo en el vector de adscripción de la tesela teniendo en cuenta la clave de la variable generada durante la ejecución de ***setup$***;
5. **análisis de la herencia inter-resolución:** no es posible ni seguro implementar en este análisis una lógica de herencia inter-resolución, ya que la aplicación de las fórmulas de las ***expression*** depende en cada momento de los datos seleccionados por colisión para la agrupación. Por lo tanto, lo único que se hace es devolver nuevas peticiones de análisis de adscripción para todas las teselas hijas. Este proceso se interrumpe en cuanto una tesela no contenga puntos, ya que ninguna de sus descendientes los contendrá igualmente.

Clase *MDTPROCESSING*

Este es un ejemplo de análisis de teselado en el que se mezclan técnicas mixtas dependiendo de una condición, en este caso topológica. La libertad que tiene un programador de desarrollar cualquier lógica de adscripción es total, siempre y cuando, claro está, se ciña a los criterios de diseño de clases *GridderTask*. Estos nuevos *GridderTask* pueden ser muy generalistas, como el anterior, soportando un buen número de casos de uso muy comunes, o muy específicos. Como se verá en este caso, este *GridderTask* es un ejemplo de análisis de estos últimos, ya que está específicamente diseñado para adscribir modelos digitales del terreno configurados en una rejilla regular de una determinada resolución, es decir, con puntos de cota separados a intervalos regulares. Por ejemplo, el juego de datos adscrito en este trabajo es uno de los modelos digitales del terreno oficiales del IECA, con una resolución de 100 metros. El MDT original debe estar expresado en forma de nube de puntos.

Para adscribir este modelo de datos de origen se utiliza una técnica mixta:

1. si la resolución del nivel de rejilla actual es inferior a la resolución del MDT de origen (por ejemplo, teselas de 1 kilómetro de resolución frente a un MDT de 100 metros), se calculan los puntos que colisionan con la tesela (como en el *GridderTask* ***POINTAGGREGATIONS***) y se le hace la media a las alturas;
2. si la resolución, por el contrario, es superior (teselas de 25 metros frente a MDT de 100), entonces se hace una interpolación *IDW* como la descrita en el *GridderTask* ***POINTIDW***, para crear una interpolación en aquellas teselas que no tienen colisiones o suavizar el valor de aquellas en las que sólo cae uno.

Este análisis de teselado creará una única variable de adscripción, que funciona también como variable índice.

El *JSON* de configuración es el siguiente:

```
{
  "gridderTask": {
    "gridderTaskId": "gridderTaskMdtProcessing",
    "gridderTaskType": "MDTPROCESSING",
    "name": "Interpolación MDT por media de alturas e IDW
      ",
    "description": "Interpolación del Modelo Digital del
      Terreno (MDT) mediante el método de media de
      alturas si la densidad de puntos de alturas es lo
      suficientemente alta en la tesela o por
      interpolación Inverse Distance Weighting si no.",
    "sourceTable": "mdt.mdt",
    "geomField": "geom",
    "mdtResolution": 100,
    "maxDistance": 200,
    "numberOfPoints": 16,
    "heightField": "h",
    "round": 1,
    "power": 2,
    "variableName": "Procesamiento MDT",
    "variableDescription": "Procesamiento del MDT de 100
      metros mediante media / interpolación IDW"
  }
}
```

Código III.6.7: Configuración de un *GridderTask* de tipo "MDTPROCESSING".

Se pueden observar muchas variables de configuración pertenecientes al *GridderTask* de tipo ***POINTIDW***:

1. ***mdtResolution*:** la resolución del MDT para decidir, en función de la resolución de la tesela objetivo, por qué método de cálculo optar;

2. ***maxDistance*:** al igual que en ***POINTIDW***, la distancia máxima a la que se buscarán puntos para la convolución *IDW*;

3. ***numberOfPoints*:** el número de puntos que se usará en la convolución *IDW*;

4. ***heightField*:** campo de los datos puntuales que alberga la cota;

5. ***round*:** redondeo del valor medio de altura o de la interpolación *IDW*;

6. ***power*:** la fricción por distancia para la convolución *IDW*;

7. ***variableName* y *variableDescription*:** nombre y descripción de la variable a generar.

El análisis, cuando se configura con su método ***setup$***, crea una única variable de adscripción con el nombre y la descripción dadas en los dos últimos parámetros anteriores. Una vez configurado, la adscripción de una tesela sigue los siguientes pasos:

1. **determinación de geometrías originales colisionantes:** en caso de estar por encima de ***mdtResolution***, se buscan todos los puntos colisionantes con la tesela. Si se está por debajo, se buscan todos los puntos que estén a menos de ***maxDistance*** de la tesela, utilizando la propiedad ***offset*** de la misma;

2. **carga de información original colisionante:** se carga el valor del campo ***heightField*** para todos los puntos seleccionados en el punto anterior, por el método que sea;

3. **análisis geométrico de la información original colisionante:** en el caso de estar en el caso por encima de ***mdtResolution***, no se hace nada. En caso de estar por debajo, se miden las distancias de los puntos seleccionados al centro de la tesela, descartando todos menos los ***numberOfPoints*** primeros, los más cercanos;

4. **análisis alfanumérico de la información original colisionante:** según ***mdtResolution***, o bien se hace la media de las cotas de los puntos colisionantes o bien se calcula el *IDW* de los puntos seleccionados en el punto anterior, redondeando el valor resultante y almacenándolo en la variable creada durante ***setup$***;

5. **análisis de la herencia inter-resolución:** al igual que en el caso de ***POINTIDW***, si la tesela no tiene ni colisiones ni puntos a menos de ***maxDistance***, el proceso se acaba. En cualquier otro caso, es imprescindible volver a calcular la adscripción de todas sus teselas hijas, sin posibilidades de aprovechar los cálculos de la tesela madre para ningún descendiente.

III.7 Teselado

Una vez descrita la librería base del sistema en el capítulo anterior, en éste vamos a ver cómo se pone en funcionamiento, en dos modalidades distintas, con el objetivo de llevar a cabo el cálculo de las adscripciones de los datos originales.

La primera modalidad será el uso de sencillas herramientas *CLI* (*Command Line Interface*), que de una forma sencilla son capaces de llevar a cabo trabajos de adscripción a la tesela sin necesidad de desplegar la arquitectura de microservicios de controlador y trabajadores.

La segunda modalidad descrita incluye el despliegue de la infraestructura de microservicios completa para automatizar y paralelizar el proceso de adscripción en una infraestructura de servidores que puede constar de una o varias máquinas, y que es capaz de escalar horizontalmente.

III.7.1. Teselación con utilidades *CLI*

El desarrollo de este tipo de aplicaciones sencillas de línea de comando es una práctica habitual porque permiten probar procesos bien definidos y codificados en librerías ya implementadas, como es el caso de *libcellbackend*, en un entorno muy controlado. Se considera una buena práctica seguir este proceso de complicación incremental en la arquitectura puesto que, en el caso de probar una nueva característica de la librería directamente con la arquitectura de microservicios al completo, los posibles fallos que ésta pudiera tener son más difíciles de depurar y detectar, puesto que es un entorno de ejecución de tareas computacionales muchísimo más complejo. Al lanzar los procesos en un entorno de un sólo microservicio, sin complicaciones adicionales que puedan provocar las interacciones con otros, la puesta a prueba de correcciones o nuevas funcionalidades en la librería objeto de desarrollo se simplifica mucho.

Se han creado dos de estas utilidades de línea de comando para ejecutarlas sobre la base de datos de datos de origen y la base de datos de la plataforma descrita en el capítulo III.3 “Implementación de la base de datos de gestión *Cell*” (pág. 145). Estas dos herramientas están escritas en el lenguaje *TypeScript* para la plataforma *Node.js*.

La herramienta *CLI griddersetup*

Esta herramienta permite preparar los objetos necesarios para llevar a cabo un análisis de teselado en la base de datos de la plataforma. Mediante un *JSON* de configuración que

se le proporciona como parámetro de entrada a la herramienta, ésta configura los siguientes elementos en la plataforma:

1. ***cellPg*:** detalles de conexión a la base de datos de la plataforma *Cell*;
2. ***sourcePg*:** detalles de conexión a la base de datos de origen;
3. ***grid*:** definición de la rejilla a utilizar en el teselado;
4. ***gridderTask*:** definición del análisis de teselado a realizar.

Con esta configuración, la herramienta crea e introduce en la base de datos de la plataforma los elementos descritos y prepara el análisis de telesado descrito ejecutando su método de inicialización y configuración ***setup$***. Recordemos que dicho método realiza importantes tareas de preparación del análisis, como la creación de variables de adscripción y/o catálogos, si lo precisa el tipo de análisis seleccionado. Un ejemplo del *JSON* de configuración de esta herramienta se muestra a continuación:

```
{

  "cellPg": {
    "db": "cell",
    "host": "servidor",
    "pass": "contrasena",
    "port": "5432",
    "user": "usuario"
  },

  "sourcePg": {
    "db": "datos_origen",
    "host": "servidor",
    "pass": "contrasena",
    "port": "5432",
    "user": "usuario"
  },

  "grid": {
    "description": "A grid based on the official EU one",
    "gridId": "eu-grid",
    "name": "eu-grid",
    "originEpsg": "3035",
    "originX": 2700000,
    "originY": 1500000,
    "zoomLevels": [
      {"name": "100 km", "size": 100000},
      {"name": "50 km", "size": 50000},
      {"name": "10 km", "size": 10000},
```

```
30        {"name": "5 km", "size": 5000},
31        {"name": "1 km", "size": 1000},
32        {"name": "500 m", "size": 500},
33        {"name": "250 m", "size": 250},
34        {"name": "125 m", "size": 125},
35        {"name": "25 m", "size": 25},
36        {"name": "5 m", "size": 5}
37      ]
38    },
39
40    "gridderTask": {
41      "gridderTaskId": "gridderTaskMdtProcessing",
42      "gridderTaskType": "MDTPROCESSING",
43      "name": "Interpolación MDT por media de alturas e IDW
          ",
44      "description": "Interpolación del Modelo Digital del
          Terreno (MDT) mediante el método de media de
          alturas si la densidad de puntos de alturas es lo
          suficientemente alta en la tesela o por
          interpolación Inverse Distance Weighting si no.",
45      "sourceTable": "mdt.mdt",
46      "geomField": "geom",
47      "mdtResolution": 100,
48      "maxDistance": 200,
49      "numberOfPoints": 16,
50      "heightField": "h",
51      "round": 1,
52      "power": 2,
53      "variableName": "Procesamiento MDT",
54      "variableDescription": "Procesamiento del MDT de 100
          metros mediante media / interpolación IDW"
55    },
56
57    "verbose": false
58
59  }
```

Código III.7.1: Ejemplo de *JSON* de configuración de la herramienta *CLI griddersetup*.

En esta configuración se le está indicando a la herramienta *CLI* que:

1. la base de datos de la plataforma (***cellPg***) se encuentra en un servidor *PostgreSQL* ubicado en el servidor ***host***, puerto ***port***, y que se accede a él con las credenciales ***user*** / ***pass***. En dicho servidor de base de datos, existe una base de datos llamada ***cell*** (parámetro ***db***);

2. que la base de datos de datos de origen se encuentra, de forma análoga al anterior objeto, en la ubicación descrita;

3. que la rejilla a crear en la plataforma y a utilizar en el teselado es la descrita en el objeto ***grid***;

4. que el análisis a llevar a cabo y a configurar por la herramienta (método ***setup$***) es el descrito en el objeto ***gridderTask***;

5. que se quiere que el proceso de ejecución de la herramienta se desarrolle sin dar muchos detalles del mismo por consola (parámetro ***verbose***).

Con toda esta información, la herramienta *griddersetup* prepara el entorno de ejecución de análisis de teselado con el *GridderTask* definido en ***gridderTask*** sobre la rejilla definida en ***grid***. Esta herramienta *CLI* sólo tiene que ejecutarse una vez para dejar el entorno listo para próxima herramienta.

La herramienta *CLI gridder*

Una vez el entorno de ejecución está preparado por la herramienta *griddersetup*, la herramienta *gridder* es la encargada de llevarlos a cabo sobre un grupo de teselas concreto. A diferencia de la herramienta anterior, esta se puede ejecutar tantas veces como haga falta para adscribir distintos grupos de teselas. Para ello, al igual que la herramienta anterior, se le proporciona un *JSON* de configuración con las características del trabajo de teselado que se desea realizar. Este *JSON* es:

```
{

  "cellPg": {
    "db": "cell",
    "host": "servidor",
    "pass": "contrasena",
    "port": "5432",
    "user": "usuario"
  },

  "sourcePg": {
    "db": "datos_origen",
    "host": "servidor",
    "pass": "contrasena",
    "port": "5432",
    "user": "usuario"
  },

  "grid": {
    "description": "A grid based on the official EU one",
    "gridId": "eu-grid",
```

```
    "name": "eu-grid",
    "originEpsg": "3035",
    "originX": 2700000,
    "originY": 1500000,
    "zoomLevels": [
      {"name": "100 km", "size": 100000},
      {"name": "50 km", "size": 50000},
      {"name": "10 km", "size": 10000},
      {"name": "5 km", "size": 5000},
      {"name": "1 km", "size": 1000},
      {"name": "500 m", "size": 500},
      {"name": "250 m", "size": 250},
      {"name": "125 m", "size": 125},
      {"name": "25 m", "size": 25},
      {"name": "5 m", "size": 5}
    ]
  },

  "gridderTask": {
    "gridderTaskId": "gridderTaskMdtProcessing",
    "gridderTaskType": "MDTPROCESSING",
    "name": "Interpolación MDT por media de alturas e IDW
      ",
    "description": "Interpolación del Modelo Digital del
      Terreno (MDT) mediante el método de media de
      alturas si la densidad de puntos de alturas es lo
      suficientemente alta en la tesela o por
      interpolación Inverse Distance Weighting si no.",
    "sourceTable": "mdt.mdt",
    "geomField": "geom",
    "mdtResolution": 100,
    "maxDistance": 200,
    "numberOfPoints": 16,
    "heightField": "h",
    "round": 1,
    "power": 2,
    "variableName": "Procesamiento MDT",
    "variableDescription": "Procesamiento del MDT de 100
      metros mediante media / interpolación IDW"
  },

  "cells": [
    { "gridId": "eu-grid", "zoom": 1, "x": 8, "y": 3 },
    { "gridId": "eu-grid", "zoom": 1, "x": 8, "y": 4 },
    { "gridId": "eu-grid", "zoom": 1, "x": 8, "y": 5 }
  ],
```

```
62
63    "targetZoom": 7,
64
65    "verbose": false
66
67 }
```

Código III.7.2: Ejemplo de *JSON* de configuración de la herramienta *CLI gridder*.

donde:

1. los objetos ***cellPg***, ***sourcePg***, ***grid*** y ***gridderTask*** tienen la misma función que en el *JSON* de configuración anterior. En este caso, sirven para certificar que dichos objetos existen acordes a las especificaciones referidas en este *JSON*. Es un método un tanto redundante de seguridad;

2. ***cells*** es una lista de las teselas a procesar con el entorno definido anteriormente. Las teselas son referenciadas indicando el ID de la rejilla a la que pertenecen (***gridId***), su nivel de resolución (coordenada R, ***zoom***) y sus coordenadas X e Y dentro del mismo (***x*** e ***y***);

3. ***targetZoom*** le indica a la herramienta el nivel de resolución objetivo al que tiene que parar el proceso de adscripción de las teselas citadas. En este caso, se le está indicando que el proceso debe parar al alcanzar el nivel 7, es decir, según la definición de la rejilla proporcionada, y sabiendo que el primer elemento de la lista, en *JSON*, no es el 1 sino el 0, esto corresponde al nivel de 125 metros, éste incluido;

4. ***verbose*** cumple la misma función que en la herramienta anterior.

Lo normal se darle a la herramienta un grupo de teselas de baja resolución (en este caso, tres de resolución 50 km) y hacer bajar el análisis de adscripción hasta el nivel deseado. De esta manera, se maximiza el efecto de las relaciones inter-resolución de los tipos de análisis de adscripción que los tienen, agilizando mucho el proceso.

III.7.2. Configuración final de *GridderTask* para la adscripción de las fuentes originales

En este paso, con las dos herramientas *CLI* ya preparadas, se pueden probar y depurar el funcionamiento de los diferentes tipos de *GridderTask* a aplica a cada una de las fuentes de datos originales. Dada la extensión de algunos de dichos ficheros de configuración, remitimos al lector a los repositorios ***GitHub*** descritos en el anexo V.2 “Repositorios GitHub” (pág. 349), describiendo en este apartado la configuración de los análisis de una forma más literal.

Datos de contexto: Provincia

Como primer nivel de datos de contexto se han adscrito las provincias, procedentes del DERA, por medio de un análisis de tipo "DISCRETEPOLYAREASUMMARY" sobre el campo del dato original "provincia", lo que ha creado 9 variables de adscripción, una por provincia:

Tabla III.7.1: Variables de adscripción generadas durante la adscripción de los datos de contexto de polígonos de provincias. Fuente: elaboración propia.

key	name	description
35aa	Área Almería	Área de la provincia Almería.
dc19e	Área Cádiz	Área de la provincia Cádiz.
413ae9	Área Córdoba	Área de la provincia Córdoba.
fb97	Área Granada	Área de la provincia Granada.
919f	Área Huelva	Área de la provincia Huelva.
7215	Área Jaén	Área de la provincia Jaén.
809b	Área Málaga	Área de la provincia Málaga.
44f5	Área Sevilla	Área de la provincia Sevilla.
00000	Index var provinciaDiscrete-AreaSummary	Index variable for GridderTask provinciaDiscreteAreaSummary

Nótese la creación de la variable índice "00000". Esta fuente de datos originales se ha adscrito hasta el nivel de resolución de 125 metros.

Datos de contexto: municipios

Segundo nivel de contexto territorial, para el que se han utilizado dos metodologías de adscripción distintas:

1. un método de tipo "DISCRETEPOLYTOPAREA" para consignar para cada tesela el municipio mayoritario;
2. un método de tipo "DISCRETEPOLYAREASUMMARY" para ver el desglose de áreas de los municipios en la tesela.

El primero crea una única variable de adscripción, llamada "Municipio: máxima área", con la clave interna "9", con un catálogo con los nombres y provincias del municipio, del cual se da una muestra en la tabla III.7.2.

Tabla III.7.2: Extracto del catálogo generado por un análisis de teselado de tipo "DISCRETEPOLYTO-PAREA" sobre los municipios de Andalucía, formando el valor del catálogo con el nombre del municipio y su provincia entre paréntesis. Fuente: elaboración propia.

variable_key	key	value
9	7	Abla (Almería)
9	e	Abrucena (Almería)

variable_key	key	value
9	7e0	Adamuz (Córdoba)
9	1	Adra (Almería)
9	494	Agrón (Granada)
9	353	Aguadulce (Sevilla)
9	5a	Aguilar de la Frontera (Córdoba)
9	cb8	Alájar (Huelva)
9	0fa	Alameda (Málaga)
9	6c2	Alamedilla (Granada)
9	dba	Alanís (Sevilla)
9	4bf	Albaida del Aljarafe (Sevilla)
9	9	Albánchez (Almería)
9	303	Albánchez de Mágina (Jaén)
9	97	Alboloduy (Almería)
9	6cf	Albolote (Granada)

El segundo crea 772 variables, una para consignar el área de cada municipio en las teselas donde existe, de las que se puede ver una muestra en la tabla III.7.3.

Tabla III.7.3: Extracto del conjunto de variables generadas por un análisis de teselado de tipo "DISCRETEPOLYAREASUMMARY" sobre los municipios de Andalucía, formando nombre y descripción de las mismas con el nombre y la provincia del municipio. Fuente: elaboración propia.

key	name	description
5	Área Abla (Almería)	Área del municipio Abla, provincia Almería
2	Área Abrucena (Almería)	Área del municipio Abrucena, provincia Almería
ef	Área Adamuz (Córdoba)	Área del municipio Adamuz, provincia Córdoba
1	Área Adra (Almería)	Área del municipio Adra, provincia Almería
a3	Área Agrón (Granada)	Área del municipio Agrón, provincia Granada
3e4	Área Aguadulce (Sevilla)	Área del municipio Aguadulce, provincia Sevilla
71e	Área Aguilar de la Frontera (Córdoba)	Área del municipio Aguilar de la Frontera, provincia Córdoba
2eb	Área Alájar (Huelva)	Área del municipio Alájar, provincia Huelva
1cd	Área Alameda (Málaga)	Área del municipio Alameda, provincia Málaga
52	Área Alamedilla (Granada)	Área del municipio Alamedilla, provincia Granada
ab3	Área Alanís (Sevilla)	Área del municipio Alanís, provincia Sevilla
1f2	Área Albaida del Aljarafe (Sevilla)	Área del municipio Albaida del Aljarafe, provincia Sevilla

key	name	description
2e	Área Albánchez (Almería)	Área del municipio Albánchez, provincia Almería
d43	Área Albánchez de Mágina (Jaén)	Área del municipio Albánchez de Mágina, provincia Jaén
6	Área Alboloduy (Almería)	Área del municipio Alboloduy, provincia Almería
56	Área Albolote (Granada)	Área del municipio Albolote, provincia Granada

Datos de contexto: secciones censales

Las secciones censales constituyen el siguiente nivel de contexto territorial incorporado a la base de datos de adscripción. Se han teselado 5940 secciones censales por medio de un análisis de tipo "DISCRETEPOLYAREASUMMARY", lo que ha generado un total de 5941 variables de adscripción, es decir, una por código de sección censal existente más la variable de indexación. Se puede ver un extracto del conjunto de variables creadas en la tabla III.7.4.

Tabla III.7.4: Extracto del conjunto de variables generadas por un análisis de tipo "DISCRETEPOLYAREASUMMARY" sobre las áreas censales de Andalucía, formando nombre y descripción con el código de la sección. Fuente: elaboración propia.

key	name	description
7728	Área 0400101001	Área de la sección censal 0400101001.
d9a9	Área 0400201001	Área de la sección censal 0400201001.
e912	Área 0400301001	Área de la sección censal 0400301001.
8f73	Área 0400301002	Área de la sección censal 0400301002.
1fc7	Área 0400301003	Área de la sección censal 0400301003.
baa0	Área 0400301004	Área de la sección censal 0400301004.
eadf	Área 0400301005	Área de la sección censal 0400301005.
d333	Área 0400301006	Área de la sección censal 0400301006.
15e34	Área 0400301007	Área de la sección censal 0400301007.
6e88	Área 0400302001	Área de la sección censal 0400302001.
a593	Área 0400302002	Área de la sección censal 0400302002.
55db	Área 0400302003	Área de la sección censal 0400302003.
3d49	Área 0400303001	Área de la sección censal 0400303001.
3359	Área 0400303002	Área de la sección censal 0400303002.
1323	Área 0400303003	Área de la sección censal 0400303003.

Datos de contexto: núcleos de población

Otro dato de contexto adscrito ha sido el catálogo de núcleos de población poligonales del DERA del IECA. Se ha utilizado para ello un análisis de tipo "DISCRETEPOLYA-

REASUMMARY" utilizando como claves de adscripción el nombre y su nivel (disperso, seccionado o cabecera). Se han adscrito 13.348 polígonos de núcleos de población, que al tener una cobertura muy escasa de la zona de estudio ha sido computacionalmente hablando un esfuerzo liviano. En la tabla III.7.5 se puede ver un extracto de las variables generadas durante el proceso de adscripción. Se han generado 10.446 variables de adscripción.

Tabla III.7.5: Extracto del conjunto de variables generadas por un análisis de tipo "DISCRETEPOLYA-REASUMMARY" sobre los núcleos de población de Andalucía, utilizando el nombre del núcleo y su nivel para componer nombre y descripción. Fuente: elaboración propia.

key	name	description
1de	Área Abadía del Sacromonte (Disperso)	Área del núcleo de población Abadía del Sacromonte, nivel Disperso.
e523	Área Abajo (Seccionado)	Área del núcleo de población Abajo, nivel Seccionado.
046	Área Abejorreras (Disperso)	Área del núcleo de población Abejorreras, nivel Disperso.
977	Área Aben Humeya y Valle de Bartodano (Disperso)	Área del núcleo de población Aben Humeya y Valle de Bartodano, nivel Disperso.
589	Área Abla (Cabecera)	Área del núcleo de población Abla, nivel Cabecera.
4a7	Área Abragen (Disperso)	Área del núcleo de población Abragen, nivel Disperso.
aa0	Área Abriojal (Disperso)	Área del núcleo de población Abriojal, nivel Disperso.
f605	Área Abrucena (Cabecera)	Área del núcleo de población Abrucena, nivel Cabecera.
60c	Área Acebuchal (Disperso)	Área del núcleo de población Acebuchal, nivel Disperso.
706	Área Acebuchal (Seccionado)	Área del núcleo de población Acebuchal, nivel Seccionado.
6b6	Área Acebuche (Seccionado)	Área del núcleo de población Acebuche, nivel Seccionado.
efc	Área Acebuches (Seccionado)	Área del núcleo de población Acebuches, nivel Seccionado.
3ba	Área Aceites Cazorla (Disperso)	Área del núcleo de población Aceites Cazorla, nivel Disperso.
ed6	Área Aceites Torresur (Disperso)	Área del núcleo de población Aceites Torresur, nivel Disperso.
ecc	Área Aceites Zarate (Disperso)	Área del núcleo de población Aceites Zarate, nivel Disperso.
cb6	Área Acequia de la Playa (Disperso)	Área del núcleo de población Acequia de la Playa, nivel Disperso.
394	Área Acequia del Aral (Disperso)	Área del núcleo de población Acequia del Aral, nivel Disperso.

key	name	description
1b1	Área Acequias (Seccionado)	Área del núcleo de población Acequias, nivel Seccionado.
688	Área Acosta (Disperso)	Área del núcleo de población Acosta, nivel Disperso.

Datos de contexto: Espacios Naturales Protegidos

Como último ejemplo de datos de contexto se han adscrito los polígonos del catálogo de Espacios Naturales Protegidos (EENNPP) de Andalucía. Los datos originales constan de 205 polígonos que no recubren completamente el área de estudio, la mayoría de un buen tamaño, lo que facilita el uso de las relaciones inter-resolución del análisis de adscripción elegido, que es también el "DISCRETEPOLYAREASUMMARY". Para la agrupación categórica se han utilizado los campos nombre y categoría del espacio natural. El análisis ha generado 206 variables de adscripción, de las que se puede ver un extracto en la tabla III.7.6.

Tabla III.7.6: Extracto del conjunto de variables generadas por un análisis de tipo "DISCRETEPOLYAREASUMMARY" sobre los Espacios Naturales Protegidos de Andalucía, utilizando el nombre y la categoría del espacio para su nombre y descripción. Fuente: elaboración propia.

key	name	description
e0e	Área ACANTILADO DEL ASPERILLO (Monumento Natural)	Área del EENNPP ACANTILADO DEL ASPERILLO, figura de protección Monumento Natural.
ede	Área ACANTILADOS DE MARO-CERRO GORDO (Paraje Natural)	Área del EENNPP ACANTILADOS DE MARO-CERRO GORDO, figura de protección Paraje Natural.
c92d	Área ACEBUCHE DE EL ESPINILLO (Monumento Natural)	Área del EENNPP ACEBUCHE DE EL ESPINILLO, figura de protección Monumento Natural.
6dc	Área ACEBUCHES DE EL ROCÍO (Monumento Natural)	Área del EENNPP ACEBUCHES DE EL ROCÍO, figura de protección Monumento Natural.
dd1	Área ALBORÁN (Paraje Natural)	Área del EENNPP ALBORÁN, figura de protección Paraje Natural.
0f2	Área ALBUFERA DE ADRA (Reserva Natural)	Área del EENNPP ALBUFERA DE ADRA, figura de protección Reserva Natural.
8b1	Área ALBUFERA DE ADRA (Zona de protección de la Reserva Natural)	Área del EENNPP ALBUFERA DE ADRA, figura de protección Zona de protección de la Reserva Natural.
d7e5	Área ALTO GUADALQUIVIR (Paraje Natural)	Área del EENNPP ALTO GUADALQUIVIR, figura de protección Paraje Natural.

key	name	description
180	Área ARRECIFE BARRERA DE POSIDONIA (Monumento Natural)	Área del EENNPP ARRECIFE BARRERA DE POSIDONIA, figura de protección Monumento Natural.
9b8	Área BAHÍA DE CÁDIZ (Parque Natural)	Área del EENNPP BAHÍA DE CÁDIZ, figura de protección Parque Natural.
4ce	Área BOSQUE DE LA BAÑIZUELA (Monumento Natural)	Área del EENNPP BOSQUE DE LA BAÑIZUELA, figura de protección Monumento Natural.
e30	Área BRAZO DEL ESTE (Paraje Natural)	Área del EENNPP BRAZO DEL ESTE, figura de protección Paraje Natural.
837	Área CABO DE GATA-NÍJAR (Parque Natural)	Área del EENNPP CABO DE GATA-NÍJAR, figura de protección Parque Natural.
27e	Área CAÑADA DE LOS PÁJAROS (Reserva Natural Concertada)	Área del EENNPP CAÑADA DE LOS PÁJAROS, figura de protección Reserva Natural Concertada.

Hábitats de Interés Comunitario

Esta fuente de datos original poligonal se ha sometido a un análisis de teselado de tipo "DISCRETEPOLYAREASUMMARY" para obtener la distribución de cada tipo de hábitat en las teselas. Esta ha sido una de las fuentes de datos originales más costosas de adscribir debido a los siguientes factores:

1. el alto grado de sus categorías temáticas, 101;

2. el pequeño tamaño de los polígonos, que impiden el aprovechamiento efectivo de las relaciones inter-resolución de la rejilla, por lo que hay que computar cada tesela hasta el nivel de resolución deseado de 125 metros en muchos casos;

3. aunque los polígonos pueden parecer muy intrincados, como se vió en el capítulo de datos de origen su esfericidad media era de 0,6. Sin embargo, muchos de ellos tienen esfericidades extraordinariamente bajas. El hecho también de que sus segmentos perimetrales tengan una longitud media de 8 metros da una idea del detalle de la digitalización de información de esta capa;

4. el nivel de recubrimiento de la capa a nivel del área de estudio, sin ser completo, si es bastante extenso, llegando a niveles de densidad muy altos en zonas serranas;

5. el enorme número de geometrías poligonales involucradas, ya que, en la fuente de datos original, existía una relación uno a muchos entre los tipos de hábitats y los polígonos. Como el análisis de adscripción "DISCRETEPOLYAREASUMMARY" no puede trabajar con relaciones uno a muchos entre la información geométrica y

la componente temática asociada, ha habido que duplicar los polígonos por cada hábitat asociado a los mismos. Es decir, si un polígono recogía la presencia de 7 hábitats distintos, para adaptar la fuente original de datos a los requerimientos del análisis de tipo "DISCRETEPOLYAREASUMMARY" ha habido que replicar dicho polígono 7 veces, uno para cada categoría de hábitat presente. Por lo tanto, el número final de polígonos asciende a 1.492.511, prácticamente el doble de los 760.543 originales.

Esta combinación de tantos polígonos con una altísima complejidad morfológica hace que cada tesela tenga que ser computada independientemente. Teniendo en cuenta que el recorte de polígonos es una actividad muy estresante computacionalmente hablando, lo normal es que esta capa sea costosa de procesar.

Como es norma en este tipo de análisis de teselado, se han generado 102 variables de adscripción (recordemos siempre la variable índice), de las que un extracto se puede observar en la tabla III.7.7.

Tabla III.7.7: Extracto del conjunto de variables generadas por un análisis de tipo "DISCRETEPOLYAREASUMMARY" sobre los Hábitats de Interés Comunitario de Andalucía, utilizando el nombre del hábitat para su nombre y descripción. Fuente: elaboración propia.

key	name	description
9a9	Área HIC Abetales de Abies pinsapo (+)	Área del Hábitat de Interés Comunitario Abetales de Abies pinsapo (+)
c75	Área HIC Acantilados con vegetación de las costas atlánticas y bálticas	Área del Hábitat de Interés Comunitario Acantilados con vegetación de las costas atlánticas y bálticas
ae1	Área HIC Acantilados con vegetación de las costas mediterráneas con Limonium spp endémicos	Área del Hábitat de Interés Comunitario Acantilados con vegetación de las costas mediterráneas con Limonium spp endémicos
a6	Área HIC Aguas oligomesotróficas calcáreas con vegetación béntica de Chara spp. Subtipo: Aguas oligomesotróficas calcáreas con vegetación béntica de Chara spp.	Área del Hábitat de Interés Comunitario Aguas oligomesotróficas calcáreas con vegetación béntica de Chara spp. Subtipo: Aguas oligomesotróficas calcáreas con vegetación béntica de Chara spp.

Población

La población, cuya fuente original es la rejilla de población publicada por el IECA con una resolución de 250 metros, viene originalmente en formato poligonal, pero para

su adscripción se ha transformado en una malla regular de puntos de igual resolución, tomando como referencia el centro de las teselas originales del IECA.

Sobre dicha nube de puntos se ha ejecutado un análisis de adscripción de tipo "POINTAGGREGATIONS", defininiendo sobre las teselas varias variables de adscripción. En total, se han generado 99 variables, un total de 14 variables demográficas por 7 años de datos disponibles, más la variable de indexación. El total de puntos de la rejilla original es de 46.160, con una distribución espacial en la zona de estudio muy poco densificada. Un extracto de estas variables de adscripción para un año de la serie de datos se puede observar en la tabla III.7.8.

Tabla III.7.8: Extracto del conjunto de variables generadas por un análisis de tipo "POINTAGGREGATIONS" sobre la rejilla de población del IECA, simplificándola de polígono a punto mediante el uso de los centroides de las teselas originales, definiendo un conjunto de 14 variables demográficas para los 7 años de datos presentes en los datos originales. Fuente: elaboración propia.

key	name	description
e6b8	Index var gridderTaskPointAggregationsPoblacion	Index variable for GridderTask gridderTaskPointAggregationsPoblacion.
9416	Índice de dependencia anciana 2002	Índice de dependencia anciana de 2002. El índice de dependencia anciana es el cociente de la suma de población anciana entre el total de adultos.
8fd3d	Índice de dependencia infantil 2002	Índice de dependencia infantil de 2002. El índice de dependencia infantil es el cociente de la suma de población joven entre el total de adultos.
52ed	Índice de dependencia total 2002	Índice de dependencia total de 2002. El índice de dependencia total es el cociente de la suma de población joven y anciana entre el total de adultos.
efc7	Población americana 2002	Población de origen americano del año 2002.
d0ad	Población edad 0-15 2002	Población de edades entre 0 y 15 años del año 2002.
3eb7	Población edad 16-64 2002	Población de edades entre 16 y 64 años del año 2002.
2555	Población edad mayor 64 2002	Población de edad mayor de 64 años del año 2002.
53dea	Población española 2002	Población española del año 2002.
3d28	Población hombres 2002	Población de hombres del año 2002.
2951	Población magrebí 2002	Población de origen magrebí del año 2002.
8d73	Población mujeres 2002	Población de mujeres del año 2002.
57c7	Población otros 2002	Población de otros origenes del año 2002.
5383f	Población total 2002	Población total del año 2002.

key	name	description
2fa5d	Población UE Schengen 2002	Población de origen Unión Europea (espacio Schengen) del año 2002.

Modelo Digital del Terreno

El modelo digital del terreno del IECA con resolución 100 metros se publica como ráster. Este ráster ha sido transformado en una rejilla de puntos regular de la misma resolución para ser procesado por un análisis de tipo "MDTPROCESSING", teniendo esos 100 metros de resolución como distancia de transición entre la media de las alturas colisionantes en la tesela o una interpolación de tipo *IDW* de los datos. Se ha hecho una prueba de teselación a resolución de 25 metros en un área limitada del área de estudio para probar la interpolación. Este tipo de análisis sólo genera una variable de adscripción con el resultado del procesamiento de las alturas. Ver figura III.7.1, pág 251.

Catastro

El caso del catastro es similar al de la población. La información, originalmente en formato poligonal, ha sido convertida a punto tomando como referencia un punto interior a las construcciones y parcelas. Sobre esta capa se ha realizado un análisis de adscripción de tipo "POINTAGGREGATIONS" que ha generado 32 variables de adscripción, de las que se da una muestra en la tabla III.7.9.

Tabla III.7.9: Extracto del conjunto de variables generadas por un análisis de tipo "POINTAGGREGATIONS" sobre los datos de catastro. Fuente: elaboración propia.

key	name	description
d0a4	Total superficie fincas con construcciones	Total de superficie de fincas con construcciones (df_1).
3607	Total de superficie construida	Superficie total construida. Será igual a la suma de las superficies de los elementos constructivos de la finca (incluidos porches y terrazas) (df_2).
954d	Superficie construida sobre rasante	Total de superficie construida sobre rasante (df_3).
be32	Superficie construida bajo rasante	Total de superficie construida bajo rasante (df_4).
6824	Número de bienes inmuebles	Total de bienes inmuebles (pu001).
b91c	Bien inmueble más antiguo	Bien inmueble más antiguo (pu002).
e2fa	Bien inmueble más moderno	Bien inmueble más moderno (pu003).
2f7	Bienes inmuebles uso principal Almacén - Estacionamiento	Total de bienes inmuebles con uso principal 'Almacén - Estacionamiento' (pu004).
e01a	Bienes inmuebles uso principal Residencial	Total de bienes inmuebles con uso principal 'Residencial' (pu005).
5b67	Bienes inmuebles uso principal Industrial	Total de bienes inmuebles con uso principal 'Industrial' (pu006).

key	name	description
c231	Bienes inmuebles uso principal Oficina	Total de bienes inmuebles con uso principal 'Oficina' (pu007).
eb4f	Bienes inmuebles uso principal Comercial	Total de bienes inmuebles con uso principal 'Comercial' (pu008).
d4c3	Bienes inmuebles uso principal Deportivo	Total de bienes inmuebles con uso principal 'Deportivo' (pu009).
f01a	Bienes inmuebles uso principal Espectáculos	Total de bienes inmuebles con uso principal 'Espectáculos' (pu010).

III.7.3. Teselación con la arquitectura de microservicios

Una vez los análisis de adscripción están desarrollados e implementados en la librería ***libcellbackend*** y probados y depurados gracias a las utilidades *CLI* descritas en la sección anterior, el siguiente paso es pasar al teselado en un entorno de computación distribuida[1] utilizando todos los microservicios. Este sistema distribuido ya no se circunscribe necesariamente una sóla máquina, sino a un conjunto de ellas, pasando de una concepción más o menos monolítica del modo de ejecución de la herramienta adoptada por las herramientas *CLI* al modelo de ejecución basado en microservicios.

Como se recordará, esta arquitectura de microservicios se compone de varios de ellos:

1. la ***API***, que constituye la puerta de entrada al sistema;
2. el **controlador**, que controla el proceso distribuido de teselado y análisis de *Machine Learning*;
3. los **trabajadores**, que son los que se encargan de realizar las tareas específicas.

Estos microservicios están apoyados por el microservicio[2] de la base de datos *PostgreSQL* de la plataforma y el de la memoria compartida *Redis*.

La *API*

Como ya se ha dicho, la *API* de un sistema es la interfaz pública de interacción de un usuario externo con la plataforma. La *API* escuda los componentes sensibles de la arquitectura a la interacción directa exterior, como por ejemplo el acceso a la base de datos. La *API* regula lo que es posible hacer en la plataforma y la forma de hacerlo, así como las comunicaciones y el acceso a la información generada en la misma desde el exterior.

[1] En terminología actual, es la hora de llevar "a la nube" el sistema.

[2] Recordemos que el término *microservicio* hace referencia a una filosofía de diseño de arquitecturas de grandes sistema informáticos. El nombre puede parecer confuso, haciendo pensar que estos microservicios son en realidad pequeños. La base de datos principal de la plataforma, por ejemplo, no es para nada pequeña en términos de consumo de recursos computacionales. En un escenario de recursos computacionales abundantes, de hecho, lo óptimo sería que tuviera para ella sola un gran y potente servidor para rendir al máximo de sus posibilidades y no convertirse en un potencial cuello de botella del sistema.

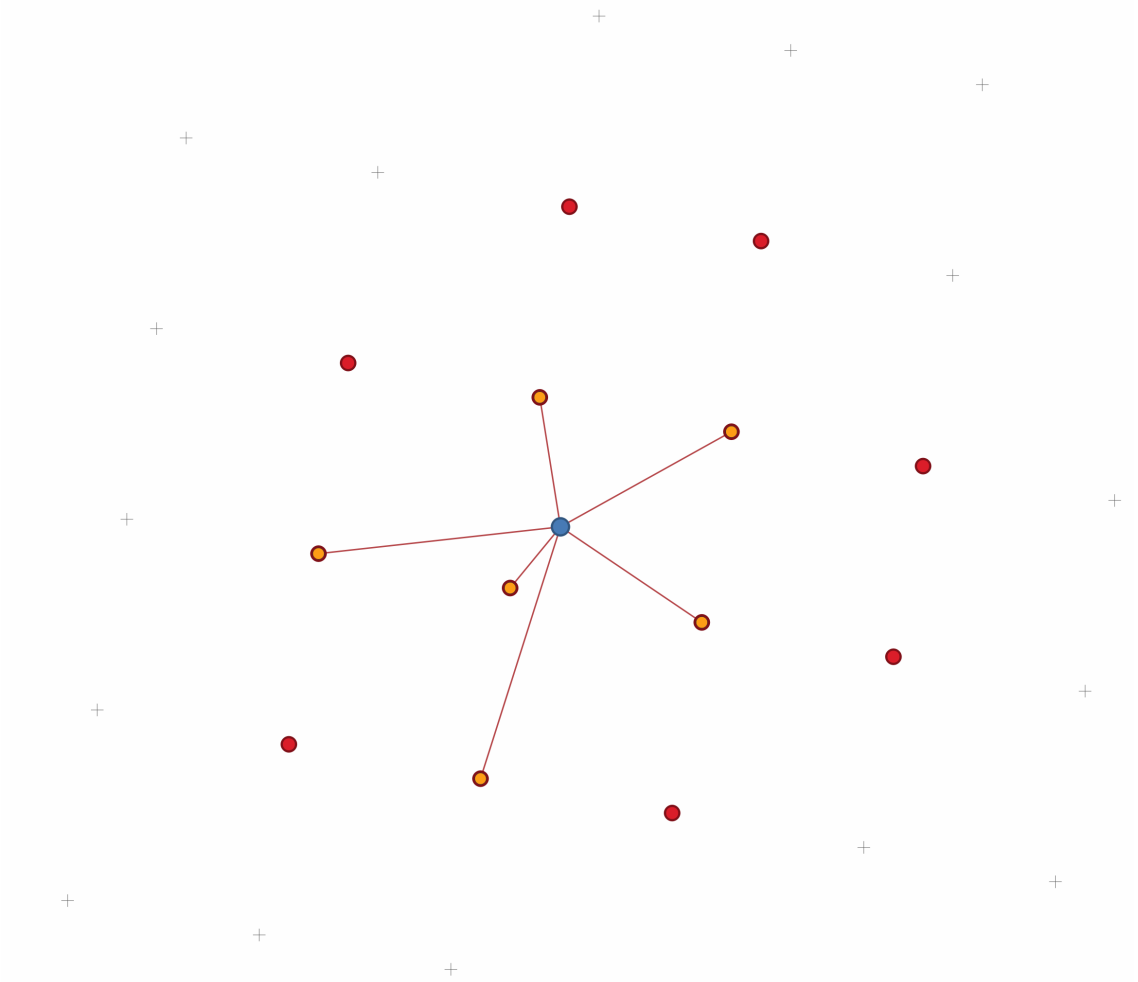

Figura III.7.1: La interpolación *IDW* realizada sobre el MDT cuando éste está por debajo de la resolución nativa de los datos originales. El centro de la tesela (en azul) es interpolado por fricción por distancia a partir de los seis puntos más cercanos (unidos por líneas) seleccionados entre los que se encuentran dentro de la distancia objetivo (en rojo). Fuente: elaboración propia.

Es más, la *API* es la base de los productos *Software as a Service* (*SaaS*), es decir, el "*software* como servicio". Este concepto ya se revisó en la introducción, en el apartado I.1.14 "La *web* moderna: hacia un nuevo paradigma en la relación con el *software*" (pág. 33).

Hay muchos tipos de *API*, metodologías y estándares con las que construirlas. No olvidemos que si la *API* no trabaja con un estándar de comunicaciones bien conocido y soportado por la comunidad de ingeniería de *software*, la creación de sistemas clientes que sean capaces de comunicarse con ella será muy costosa y la plataforma acabará, por ello, por no utilizarse[3]. La *API* de *Cell*, como ya se comentó, está diseñada con una arquitectura

[3]En ciertos sectores de la ingeniería de *software* esto, la no estandarización, es algo buscado. Por ejemplo, en la industria de los videojuegos *online*. Las compañías que diseñan y fabrican estos sistemas no lo hacen con estándares bien conocidos por el resto de la industria, más bien al contrario: diseñan

REST, que utiliza los métodos estándar del protocolo de comunicaciones *HTTP* (el más utilizado en *Internet*) para la gestión de recursos en un servidor.

Desde un punto de vista de la implementación, la *API* es un microservicio que corre en su propio *Docker* y que está implementado con las siguientes tecnologías:

1. está escrito en el lenguaje *TypeScript* sobre la plataforma *Node.js*;
2. utiliza el *framework Express* para generar la *API* de una forma estándar y rápida;
3. utiliza, sobre *Express*, una librería de implementación propia, *Appian*, que ayuda a crear respuestas estándar desde la *API* y ofrece una serie de servicios comunes para desarrollarla de una forma aún más rápida y estándar;
4. utiliza las librerías de creación propia *RxPg* y *RxRedis* como clientes a los sistemas *PostgreSQL* y *Redis* (la base de datos y la memoria compartida) bajo la perspectiva de la programación reactiva impulsada por la propuesta de *API ReactiveX*;
5. utiliza una serie de librerías de creación propia y de terceros para diversos fines: *lodash* para la manipulación de matrices, *morgan* para el seguimiento de peticiones *HTTP*, *node-logger*, de creación propia y basada en la librería *winston*, para la recopilación y seguimiento de la actividad de la *API* y, finalmente, *node-utils*, una colección de funciones de uso general para entornos *Node.js*, también de creación propia, que ayuda a tratar de una forma estándar con el sistema de ficheros del servidor y realizar operaciones con formatos de intercambio de información como *JSON* o *CSV*.

Servicios ofrecidos por la *API*

Como *API*, el microservicio implementado con las tecnologías descritas anteriormente tiene que ofrecer una serie de métodos para la interacción de los clientes con la plataforma. A continuación se describen estos métodos, indicando en cada uno la operación *REST* utilizada para llevarla a cabo. Los parámetros en las *URL* de las entradas *API* están precedidas por dobles puntos.

La interacción con una *API* de servicios *REST* se articula, como ya se ha mencionado, alrededor de las llamadas estándar del protocolo *HTTP* (*HyperText Transfer Protocol*). Como ya se apuntó en el apartado I.1.14 de la introducción, el *HTTP* es un protocolo de comunicaciones que funciona sobre la base genérica de la *Internet* como medio de transporte de datos. Este tipo de protocolo de comunicaciones, que ya se centra en resolver una tarea concreta (en este caso, la transferencia de documentos de hipertexto), toman

sus propios protocolos de *API* que sólo conocen ellos. Esto es así porque los sistemas clientes de estas plataformas (los juegos que los usuarios se instalan en sus ordenadores) están diseñados e implementados por la propia compañía. *API*, cliente y servidor tienen el mismo fabricante. La ventaja de esto es que, por ejemplo, un *cracker* (un *hacker* con intenciones aviesas) no lo tiene fácil para crear un cliente que se comunique con los servidores del juego y que le permita, por ejemplo, hacer trampas al mismo. Las compañías orientadas a los productos *SaaS*, como *Google*, por ejemplo, sí utilizan estándares conocidos para permitir que alrededor de su amplio catálogo de servicios otras empresas puedan crear productos que se integren e interaccionen con ellos.

por ello el nombre de *protocolo de capa de aplicación*. Y, dada su generalizado uso e implantación, estos mismos protocolos, a su vez, pueden ser utilizados para crear una *API* (que, en definitiva, no es más que otro protocolo de comunicaciones) para poder tratar de un tema más específico todavía. Todo funciona como una cebolla, en capas.

Pongamos una analogía biológica. Si nos remitiéramos a la comunicación humana, la *Internet* sería nuestra dotación biológica, nuestro "*hardware*", que nos permite comunicarnos, es decir, el conjunto sentidos más aparato fonador más cerebro. Esto nos permite a todos los seres humanos establecer una base biológica igualitaria para la comunicación: emitimos sonidos y los reconocemos como tales. El *HTTP* sería una lengua humana concreta. Si yo me quiero comunicar con un hablante de una lengua que no es la mía, primero tengo que aprender sus rudimentos. Con sus rudimentos aprendidos, sentaría las bases de una forma de comunicación que está reglada de forma que me permite construir unidades de comunicación entendibles y eficaces: sé dónde colocar el sujeto, como manejar los tiempos verbales, etc. La *API* sería ya tener un dominio específico en ese idioma sobre un tema concreto. Conocer, en definitiva, el vocabulario y los conceptos de una materia en el otro idioma. Puedo saber chino, puedo saber de ordenación del territorio, pero sin un dominio específico del vocabulario especializado de la disciplina en chino, la comunicación se dificulta. Eso es lo que hace una *API*.

En la metodología de construcción de *APIs REST* se opta por aprovechar al máximo las capacidades aportadas por el estándar *HTTP* para construir *APIs* de dominio de aplicación específicos. El estándar *HTTP* establece que la comunicación entre cliente y servidor se hará con el intercambio de peticiones hechas por el procedimiento de lanzar desde el cliente hacia el servidor una serie de *URLs* (*Uniform Resource Locators*, localizadores uniformes de recursos) que sirven para indicarle al servidor que queremos realizar una acción sobre unos datos alojados en él.

El uso por defecto del *HTTP* es el servicio de páginas *web*, es decir, de ficheros de hipertexto, que suele ser información escrita con el lenguaje de marcas *HTML* (*HyperText Mark-Up Language*, lenguaje de marcas de hipertexto). Cuando abrimos un navegador y escribimos en su barra de direcciones una dirección, lo que estamos haciendo es solicitarle al servidor que se aloja tras esa dirección el documento de hipertexto por defecto, es decir, lo que se llama comunmente en diseño *web* la *home page*. En realidad, lo que el navegador ha hecho es lanzar una petición *HTTP* de tipo *GET* sobre el servidor *HTTP* que se encuentra sirviendo recursos en el servidor.

El protocolo *HTTP* define, por tanto, una serie de peticiones para acceder y modificar[4] recursos (es decir, datos y ficheros) en el servidor. Las más usuales son las siguientes:

- ***GET*:** requerimiento de un recurso, ya sea una página *HTML* o cualquier otro fichero o dato que el servidor aloje;
- ***POST*:** requerimiento de subida de un recurso, un fichero o un grupo de datos. A una petición de este tipo puede ir asociada información de muy distinta forma, ya sea en la cabecera de la petición o como fichero adjunto. Esto es lo que solicita un

[4]Modificar si se tiene permiso para hacerlo, claro.

navegador al servidor cuando subimos un documento, por ejemplo, a través de una página *web*;

- ***PATCH*:** requerimiento de modificación de un recurso existente. *POST* y *PATCH* son requerimientos separados para que quede claro que en el primero se pretende crear nueva información (y, por tanto, puede fallar en caso de que ya exista) y en el segundo lo que se quiere es modificar información existente;
- ***DELETE*:** requerimiento para borrar un recurso existente.

Con esto quedan definidas las operaciones *CRUD* (*Create, Read, Update, Delete*, es decir, "crear, leer, modificar y borrar") que definen las operaciones básicas de un sistema de información. De esta manera, una *API* bien diseñada permite el trabajo remoto con la información alojada en el servidor.

Estos cuatro requerimientos, además, han de ser acompañados de parámetros que permitan al servidor y a la *API* describir sobre qué recursos en concreto se quiere actuar. En la *API* descrita a continuación se usan con profusión los llamados *URL Parameters* (parámetros *URL*), ya que estos identificadores van incrustados en las *URL* que se le mandan al servidor.

Por ejemplo, la orden (segunda descrita en los siguientes apartados) ***GET /grid/:gridId*** significa que si enviamos al servidor, desde la misma barra de direcciones de un navegador (que es un cliente *HTTP*, no lo olvidemos), una *URL* como la siguiente:

```
http://unservidorcellcualquiera.es/grid/eu-grid
```

el servidor *HTTP*[5] ubicado en *unservidorcellcualquiera.es* recibirá una petición *GET*[6] a la función ***grid*** con el parámetro ***eu-grid***. Ante tal petición, la *API* de servicios *Cell* busca en la base de datos discutida en el capítulo III.3, sección III.3.3, "Tabla *grid*" (pág. 154), la definición matemática de una rejilla llamada ***eu-grid***, que será devuelta como documento *JSON* tal y como se describe a continuación.

Métodos de la *API* de servicios

A continuación se detallan los métodos de la *API* de servicios de la plataforma.

GET /apiinfo

Un simple método para comprobar que la plataforma está activa y accesible. Devuelve un *JSON* detallando la configuración de la plataforma, que está constituida por la descripción de una serie de variables de entorno que rigen su funcionamiento.

GET /grid/:gridId

[5] Eso es lo que significa el *http://* inicial: le estamos indicando al servidor el protocolo de comunicación a utilizar.

[6] Esto es así porque las barras de direcciones de los navegadores siempre mandan requerimientos *GET*, ya que su función es la descarga o petición de información.

Devuelve un *JSON* con la definición de la rejilla con ID ":gridId". El *JSON* devuelto es:

```
{
  "grid": {
    "description": "A grid based on the official EU one",
    "gridId": "eu-grid",
    "name": "eu-grid",
    "originEpsg": "3035",
    "originX": 2700000,
    "originY": 1500000,
    "zoomLevels": [
      {"name": "100 km", "size": 100000},
      {"name": "50 km", "size": 50000},
      {"name": "10 km", "size": 10000},
      {"name": "5 km", "size": 5000},
      {"name": "1 km", "size": 1000},
      {"name": "500 m", "size": 500},
      {"name": "250 m", "size": 250},
      {"name": "125 m", "size": 125},
      {"name": "25 m", "size": 25},
      {"name": "5 m", "size": 5}
    ]
  }
}
```

Código III.7.3: *JSON* con las características de una rejilla para las entradas *API GET*, *POST* y *PATCH* de la familia de entradas */grid*.

POST /grid

Permite subir a la plataforma un *JSON* como el anterior de definición de rejilla para su creación y registro en la plataforma para futuros usos.

PATCH /grid/:gridId

Permite la modificación de los parámetros de una rejilla ya creada en el sistema, identificada con el ID ":gridId", y a partir de un *JSON* como el anterior de definición de rejilla que modifica sus características. Si la rejilla ya ha generado teselas, sin embargo, este sólo permite modificar elementos de metadatos de la rejilla como el nombre y la descripción, ya que sería catastrófico, por ejemplo, cambiar el origen de coordenadas de la rejilla con información ya adscrita. Su uso, por lo tanto, es muy limitado.

DEL /grid/:gridId

Borra la rejilla con ID ":gridId". Al igual que el anterior, este método fallará si la rejilla ya tiene información adscrita a teselas.

GET /sourcepg/:sourcePgId

Devuelve el origen de datos *PostgreSQL* identicado por el ID ":sourcePgId" con el siguiente *JSON*:

```
{
  "sourcePg": {
    "id": "sourcePgId",
    "name": "Nombre de la fuente de datos",
    "description": "Descripción de la fuente de datos",
    "db": "cell_raw_data",
    "host": "localhost",
    "pass": "contrasena",
    "port": "5432",
    "user": "usuario"
  }
}
```

Código III.7.4: *JSON* con las características de un origen de datos *PostgreSQL* para las entradas *API GET*, *POST* y *PATCH* de la familia de entradas */sourcepg*.

POST /sourcepg/:sourcePgId

Permite subir a la plataforma un *JSON* como el del método anterior con la descripción de la ubicación de un origen de datos originales para teselar.

PATCH /sourcepg/:sourcePgId

Permite modificar un origen de datos identificado por el ID *:sourcePgId* subiendo a la plataforma un *JSON* como el descrito anteriormente. Si esta fuente de datos ya ha participado en la configuración de cualquier análisis de teselado (ver más adelante), sólo se podran modificar elementos como el nombre o la descripción.

DEL /sourcepg/:sourcePgId

Permite borrar el origen de datos *:sourcePgId* si este aún no ha sido utilizado en la configuración de ningún análisis de teselado (ver más adelante).

GET /griddertask/:gridderTaskId

Devuelve un *JSON* con la definición de un determinado análisis de adscripción, identificado por su ID *:gridderTaskId*, según los *JSON* de configuración.

```
{
  "gridderTask": {
    "gridderTaskId": "gridderTaskMdtProcessing",
    "gridderTaskType": "MDTPROCESSING",
```

```
5       "name": "Interpolación MDT por media de alturas e IDW
            ",
6       "description": "Interpolación del Modelo Digital del
            Terreno (MDT) mediante el método de media de
            alturas si la densidad de puntos de alturas es lo
            suficientemente alta en la tesela o por
            interpolación Inverse Distance Weighting si no.",
7       "sourceTable": "mdt.mdt",
8       "geomField": "geom",
9       "mdtResolution": 100,
10      "maxDistance": 200,
11      "numberOfPoints": 16,
12      "heightField": "h",
13      "round": 1,
14      "power": 2,
15      "variableName": "Procesamiento MDT",
16      "variableDescription": "Procesamiento del MDT de 100
            metros mediante media / interpolación IDW",
17      "status": 0
18    }
19  }
```

Código III.7.5: *JSON* con las características de un análisis de adscripción para las entradas *API GET*, *POST* y *PATCH* de la familia de entradas */griddertask*.

La mayoría de los elementos de este *JSON* ya han sido descritos. Sin embargo, el elemento *status* es nuevo, y determina el estado actual del análisis de teselado, que puede ser uno de los siguientes:

1. estado 0, ***NOTREADY*:** indica que el análisis está definido en la plataforma, pero no se ha ejecutado su método *setup$* de configuración. Aún se está a tiempo de realizar cambios en el mismo;
2. estado 1, ***READY*:** indica que el análisis está definido en la plataforma y ha sido configurado con su método *setup$*. Ya no se pueden hacer cambios profundos en el mismo y el análisis está listo para aceptar requerimientos de procesamiento de teselas;
3. estado 2, ***ONHOLD*:** indica que el análisis ha recibido un requerimiento de procesamiento de teselas pero todos los trabajadores están ocupados en otra tareas, por lo que el análisis está a la espera de que se liberen trabajadores para avanzar en el teselado;
4. estado 3, ***RUNNING*:** indica que el análisis tiene requerimientos de teselación en curso y que al menos un trabajador está trabajando en ellos;
5. estado 5, ***ERROR*:** indica que el análisis tiene al menos un error de teselación que debe ser investigado.

Cuando un requerimiento de teselado se termina con éxito, el análisis vuelve al estado *READY*. Estos códigos de estado están predeterminados por la librería de creación propia *rewhitt*. El hecho de que no exista un estado 4 no es un error, es que ese estado, *POST-PROCESSING*, es un estado descrito por la librería *rewhitt* que la plataforma *Cell* no utiliza. El hecho de que tampoco exista un estado que denote que el análisis está terminado (que sí que existe en la librería *rewhitt*) es porque los análisis tienen un potencial de ser reutilizados constantemente en nuevas tareas de adscripción. Ver la familia de entradas */gridderjob* para más detalles.

POST /griddertask

Permite subir a la plataforma un *JSON* como el anterior de definición de un nuevo análisis de adscripción para su creación y registro en la plataforma para futuros usos. Esta entrada *API* no lanza el método *setup$* del análisis para su preparación.

PATCH /griddertask/:gridderTaskId

Permite la modificación del análisis de teselado identificado por *:gridderTaskId* mediante la subida de un *JSON* como el anterior de configuración del mismo. Al igual que la rejilla, si el análisis ya ha sido configurado (con la ejecución del método *API setup*, ver más adelante), esta entrada *API* sólo permite cambiar elementos del análisis como el nombre o la descripción.

DEL /griddertask/:gridderTaskId

Permite el borrado de un análisis de adscripción identificado por *:gridderTaskId*. Esto conlleva el borrado de todas las variables de adscripción del mismo presentes en los vectores de adscripción de las teselas, con el potencial descarte de información que puede haber tardado muchas horas en ser computada. Los clientes harían bien en solicitar una confirmación bien explícita al usuario antes de lanzar este requerimiento a la *API*.

GET /griddertask/setup/:gridderTaskId

Indica al análisis de adscripción identificado por *:gridderTaskId* que ejecute su método *setup$*, creando todas las variables de adscripción y/o catálogos necesarios para ejecutar adscripciones en el futuro. Si el análisis ya ha sido configurado, arroja un error.

GET /griddertask/variables/:gridderTaskId

Devuelve las variables generadas por un análisis de teselado.

GET /gridderjob/:gridderJobId

Devuelve un *JSON* describiendo una sesión de adscripción para un análisis determinado, sobre una fuente de datos originales concreta, y para un conjunto de teselas, con una estructura similar a la utilizada en la herramienta *CLI gridder*.

```
1 {
2   "gridderJob": {
```

```
3
4       "gridderJobId": "gridderJobId",
5
6       "sourcePgId": "sourcePgId",
7
8       "gridderTaskId": "gridderTaskId",
9
10      "targetZoom": 7,
11
12      "status": 0,
13
14      "cells": [
15        { "gridId": "eu-grid", "zoom": 1, "x": 8, "y": 3 },
16        { "gridId": "eu-grid", "zoom": 1, "x": 8, "y": 4 },
17        { "gridId": "eu-grid", "zoom": 1, "x": 8, "y": 5 }
18      ]
19
20    }
21  }
```

Código III.7.6: *JSON* con una petición de procesamiento de teselas para un análisis de teselado para la entrada *API GET /gridderjob/:gridderJobId*.

donde *gridderJobId* es el ID de la sesión de adscripción, *sourcePgId* es el ID del origen de datos originales a adscribir, *gridderTaskId* es el ID del análisis de teselado a ejecutar y *targetZoom* es el nivel de resolución objetivo. El *status* hace referencia al estado de ejecución de la sesión de adscripción, que es similar a la comentada para la familia de entradas *API /griddertask*, con las siguientes variantes:

1. estado 1, ***READY*:** indica que la sesión está creada en la plataforma y está lista para ser ejecutada, pero aún no se ha dado dicha orden;
2. estado 2, ***ONHOLD*:** indica que la sesión está en marcha pero todos los trabajadores están ocupados en otras tareas, por lo que la sesión está a la espera de que se liberen trabajadores para avanzar en el teselado;
3. estado 3, ***RUNNING*:** indica que la sesión tiene al menos un trabajador trabajando en ella;
4. estado 5, ***ERROR*:** indica que la sesión tiene al menos un error de teselación que debe ser investigado;
5. estado 6, ***COMPLETED*:** indica que la sesión ha finalizado su teselación con éxito.

POST /gridderjob

Sube a la plataforma un *JSON* como el anterior (sin el ítem *status*, que sería ignorado en cualquier caso) para definir una sesión de teselación. Esta entrada no arranca la sesion.

PATCH /gridderjob/:gridderJobId

Modifica una sesión de teselado subiendo un *JSON* como el anterior si ésta está aún en estado *READY*. Más allá de dicho estado este método devuelve un error.

DEL /gridderjob/:gridderJobId

Borra una sesión de teselado identificada por *:gridderJobId*. Esta entrada *API* no borra información presente en los vectores de adscripción de las teselas originada por la ejecución del *GridderJob* borrado.

GET /gridderjob/run/:gridderJobId

Pone en cola en la memoria compartida la sesión de teselado identificada por *:gridderJobId* para su ejecución.

GET /variable/:variableId

Devuelve las características de la variable de adscripción identificada por *:variableId* en forma del *JSON* mostrado en el listado III.7.7.

```
{
  "variable": {
    "gridderTaskId": "gridderTaskId",
    "variableKey": "variableKey",
    "name": "Nombre de la variable",
    "description": "Descripción de la variable."
  }
}
```

Código III.7.7: *JSON* con las características de una variable de adscripción entradas *API* de la familia */variable*.

donde *gridderTaskId* es el ID del análisis de teselado al que pertenece la variable, *variableKey* es la clave de la variable tal y como es utilizada para almacenar la información en el vector de adscripción de las teselas y *name* y *description* son el nombre y la descripción de la variable.

GET /variable/list

Devuelve un *JSON* con la lista de las variables presentes en el sistema, con el mismo formato que el anterior.

GET /catalog/:gridderTaskId/:variableKey

Devuelve un *JSON* con el catálogo del análisis de teselado identificado con *:gridderTaskId* para la variable identificada con *:variableKey*, como se puede ver en el *JSON* III.7.8.

```
{
  "variable": {
    "gridderTaskId": "gridderTaskId",
    "variableKey": "variableKey",
    "catalog": [
      {
        "key": 3,
        "value": "Huercal-Overa (Almería)"
      },
      {
        "key": b7d,
        "value": "Benaocaz (Cádiz)"
      },
      {
        "key": 42,
        "value": "Fernan Núñez (Córdoba)"
      }
    ]
  }
}
```

Código III.7.8: *JSON* con un catálogo tal y como lo devuelve la entrada *API /catalog/:gridderTaskId/:variableKey*.

GET /mljob/:mlJobId

Devuelve un *JSON* describiendo una sesión de aplicación de procedimiento de *Machine Learning* sobre un vector de adscripción determinado:

```
{
  "mlJob": {

    "mlJobId": "mlJobId",

    "mlJobType": "KMEANS",

    "vector": [ "varKey0", "varKey1",
      "varKey2" ],

    "startingZoom": 3,

    "targetZoom": 7,

    "status": 0,

```

```
17      "sample": 15000,
18
19      "clusters": 5
20
21    }
22  }
```

Código III.7.9: *JSON* con una petición de procesamiento de *Machine Learning* para un vector de adscripción la familia de entradas *API /mljob/:mlJobId*.

donde *mlJobId* es el ID de la sesión de *Machine Learning*, *mlJobType* es el tipo de algoritmo de *Machine Learning* a utilizar, *vector* es el vector de adscripción sobre el que hacer el procedimiento *ML*, con las *keys* de las variables de adscripción con las que componer el vector, *startingZoom* es el nivel de resolución inicial, *targetZoom* el final y *status* es el estado de la sesión (con iguales valores posibles que para el caso de la entrada */gridderjob/:gridderJobId*). *sample* y *clusters* pertenecen a la configuración de un tipo de procedimiento *ML* "KMEANS".

POST /mljob

Sube a la plataforma un *JSON* como el anterior (sin el ítem *status*, que sería ignorado en cualquier caso) para definir una sesión de *Machine Learning*. Esta entrada no arranca la sesion.

PATCH /mljob/:mlJobId

Modifica una sesión de *Machine Learning* subiendo un *JSON* como el anterior si ésta está aún en estado *READY*. Más allá de dicho estado este método devuelve un error.

DEL /mljob/:mlJobId

Borra una sesión de *Machine Learning* identificada por *:mlJobId*. Esta entrada *API* no borra información presente en los vectores de adscripción de las teselas.

GET /mljob/run/:mlJobId

Pone en cola en la memoria compartida la sesión de teselado identificada por :gridder-JobId} para su ejecución.

POST /query

Sube un *JSON* de petición de teselas. La petición tiene la siguiente sintaxis:

```
1  {
2    "query": {
3
4      "bbox": [
5        "bottomRight": [ x, y ],
6        "upperLeft": [ x, y ]
```

```
    ],

    "vector": {
      "and": {
        "or": {
          "keys": [ "variableKey0",
            "variableKey1" ]
        },
        "and": {
          "keys": [ "variableKey2",
            "variableKey3" ]
        }
      }
    },

    "targetZoom": 7

  }
}
```

Código III.7.10: *JSON* con una petición de teselas para la entrada *API GET /query*.

donde:

1. *bbox* es la caja de coordenadas, la extensión, a examinar para encontrar los vectores de adscripción deseados, definida por su coordenada de esquina inferior izquierda y esquina superior derecha (*bottomRight* y *upperLeft*), en coordenadas WGS84;

2. *vector* define los posibles vectores de adscripción que deben tener las teselas para ser seleccinadas y devueltas en la petición. Utiliza un filtro de condiciones *Y* (*and*) y *O* (*or*) que se pueden encadenar. El ejemplo propuesto devolvería las teselas que tuvieran alguno de los siguientes vectores de adscripción:

   ```
   (variableKey0 Ó variableKey1) Y (variableKey2 Y variableKey3)
   ```

   por lo que los posibles vectores de adscripción candidatos son:

   - [variableKey0, variableKey2, variableKey3]
   - [variableKey1, variableKey2, variableKey3]

Esta entrada *API* busca en la base de datos las teselas que poseen alguno de estos vectores de adscripción y devuelve otro *JSON* de respuesta:

```
{
  "queryResults": [
```

```
    {
      "gridId": "eu-grid",
      "zoom": 7, "x": 8, "y": 3,
      "data": {
        "variableKey0": "variableValue0",
        "variableKey2": "variableValue2",
        "variableKey3": "variableValue3"
      }
    },
    {
      "gridId": "eu-grid",
      "zoom": 7, "x": 8, "y": 2,
      "data": {
        "variableKey1": "variableValue1",
        "variableKey2": "variableValue2",
        "variableKey3": "variableValue3"
      }
    }
  ]
}
```

Código III.7.11: *JSON* de respuesta de la entrada *API GET /query*.

Implementación de *CellController* y *CellWorker*

Los microservicios *CellController* y *CellWorker* controlan el teselado y los trabajos de *Machine Learning* en la plataforma, coordinándose gracias al microservicio de memoria compartida *Redis* y conectándose a la base de datos principal *PostgreSQL* para leer, procesar y escribir datos.

La coordinación de los trabajos de teselado y *ML* está en manos de una librería de creación propia llamada *rewhitt*. Esta librería proporciona un *framework* para la creación de microservicios controladores y trabajadores. Pasamos a describir ambos microservicios.

El microservicio *CellController*

El controlador tiene las siguientes atribuciones:

1. poner en la cola de mensajes de la memoria compartida los lotes de trabajo de teselado y *Machine Learning* (respectivamente, los ya comentados *GridderJob* y *MlJob*) para que los trabajadores los vayan adquiriendo para su resolución según vayan terminando los trabajos que están en curso;

2. controlar la actividad de los trabajadores mediante el estudio de los *latidos de corazón* (*heartbeats*) que emiten periódicamente;

3. llevar el control del estado de los lotes de trabajo;

4. periódicamente, borrar la cola de mensajes y volver a llenarla de lotes de trabajo para intentar volver a hacer trabajos fallidos o reconfigurar las prioridades de computación de la plataforma.

Los microservicios trabajadores *CellWorker*

Los trabajadores tienen como única tarea el llevar a cabo los lotes de trabajo de teselado y *Machine Learning* que encuentran en la cola de mensajes de la memoria compartida. Reportan a la cola de mensajes del controlador, también en la memoria compartida, acerca de su actividad a intervalos regulares, los llamados *heartbeat*.

Interacciones entre la *API*, el controlador y los trabajadores

Como ya se ha discutido en el capítulo anterior, la *API* es la puerta de entrada al sistema para los clientes. Los clientes definen, gracias a los métodos de entrada de la *API*, diferentes objetos en la plataforma (rejillas, análisis de teselado, etc.).

Cuando la *API* define y ejecuta una sesión de teselado (*GridderJob*) o de *Machine Learning* (*MlJob*), sucede el siguiente proceso:

1. la *API* pone en la cola de mensajes de la memoria compartida *API*-Controlador un mensaje especificando la petición de inicio del trabajo;

2. el controlador recibe dicho mensaje, con lo que prepara los lotes de trabajo. Usualmente, esto implica dividir el trabajo entre las teselas del nivel de resolución más bajo solicitado, creando varios paquetes de trabajo *GridderJobBatch* y *MlJobBatch*, registrándolos en la base de datos de control de trabajos y poniéndolos, en la memoria compartida, en la cola de mensajes Controlador-Trabajadores;

3. los controladores, cuando se quedan sin trabajo, están constantemente pendientes de que lleguen nuevos en la cola controlador-trabajadores. Cuando ésto sucede, cogen el trabajo y lo realizan. Cuando lo han terminado, ponen el resultado final en la cola de mensajes de la memoria compartida trabajadores-controlador;

4. el controlador recibe los mensajes de los trabajadores, los procesa y guarda en la base de datos los resultados. Además, dichos trabajos pueden, a su vez, haber generado nuevos trabajos en las teselas hijas, por lo que estos entran otra vez en el flujo de trabajo siendo publicados de nuevo en la cola controlador-trabajadores.

Además, los trabajadores están, a intervalos prefijados, emitiendo su *heartbeat*, que van a una cola de mensajes especial de la memoria compartida, donde el controlador los recibe.

A intervalos también prefijados, el controlador realiza tareas de mantenimiento, que incluyen:

1. **revisión del estado de los trabajadores:** el controlador revisa los últimos *heartbeats* de los trabajadores para ver si alguno no da señales de actividad desde hace un tiempo prudencial. Si esto es así, significa que el trabajador ha sufrido algún error

no controlado en su ejecución y se ha caido. El sistema *Docker* habrá levantado de nuevo otro trabajador automáticamente para substituirlo, pero el trabajo que estuviera realizando el primero se ha perdido. El controlador toma nota del número de veces que se ha intentado ese trabajo. Si excede un número de veces predeterminado, el trabajo es marcado definitivamente como erróneo y hay que revisarlo. Si el trabajador falló por otras causas (una caida generalizada del sistema, por ejemplo), el trabajo se vuelve a poner en la cola para que otro trabajador intente llevarlo a cabo;

2. **el controlador gestiona la cola de trabajos controlador-trabajadores:** la vacía y examina todos los trabajos registrados en la base de datos que estén aún pendientes de realización, para volverlos a meter en la cola controlador-trabajadores. Este proceso le da la oportunidad al controlador de aplicar alguna lógica de prioridad en este proceso de re-publicación de trabajos, ya que puede poner por delante los trabajos pertenecientes a una determinada sesión de teselado o *Machine Learning* para darle prioridad. Por defecto los trabajos de *Machine Learning* se ponen por delante de los de teselado.

III.7.4. Evaluación del proceso de teselado

A continuación examinamos cómo ha sido el proceso de teselado, presentando un resumen de los aspectos más relevantes del mismo.

Las pruebas de teselado se han llevado a cabo en una máquina *Linux* con 8 núcleos de procesamiento a 4.3 *GHz* y 64 *GB* de memoria *RAM*. Los tiempos de trabajo mostrados en la tabla III.7.10 corresponden al mayor tiempo de proceso para una tesela de 50 kilómetros, es decir, en el peor escenario de densidad de información, y realizado sin paralelizar por un único proceso de adscripción. Esto quiere decir que el procesado ha sido hecho en bloque por un único trabajador, con tres trabajadores trabajando a la vez en teselas de 50 kilómetros, para probar posibles cuellos de botella en el sistema. El nivel de resolución objetivo para las pruebas de integración de datos ha sido de 125 metros, aunque, para probar las capacidades de interpolación del método "MDTPROCESSING" sobre los datos del MDT, se ha hecho una prueba piloto de teselado a 5 metros en un área muy limitada de 10 kilómetros cuadrados.

Tabla III.7.10: Conjunto de análisis de teselado llevados a cabo, indicando el nivel de resolución alcanzado y el total de variables creadas por cada uno de ellos. Especialmente importante es el resultado de la columna "Tiempo proceso", que recoge el tiempo máximo de proceso de una tesela de 50 kilómetros de lado para el análisis en cuestión. La teselación del MDT a 5 metros (*) ha sido una prueba piloto en una tesela de 10 kilómetros de resolución, siendo el tiempo de calculo previsto una extrapolación a la tesela de 50 kilómetros utilizado en los otros análisis. Fuente: elaboración propia.

Datos originales	Método de teselado	Tiempo proceso	Nivel	Total variables
Catastro	POINTAGGREGATIONS	50 min	125 m	32
Población	POINTAGGREGATIONS	5 min	125 m	99
MDT	MDTPROCESSING	1,75 h	125 m	1
HICS	DISCRETEPOLYAREASUMMARY	14,5 h	125 m	102
Provincia	DISCRETEPOLYAREASUMMARY	45 min	125 m	9
Municipio	DISCRETEPOLYAREASUMMARY	1 h	125 m	772

Datos originales	Método de teselado	Tiempo proceso	Nivel	Total variables
Municipio	DISCRETEPOLYTOPAREA	30 min	125 m	1
Núcleos de población	DISCRETEPOLYAREASUMMARY	10 min	125 m	10445
Sección censal	DISCRETEPOLYAREASUMMARY	1 h	125 m	5941
EENNPP	DISCRETEPOLYAREASUMMARY	50 min	125 m	206
MDT (*)	MDTPROCESSING	43 días	5 m	1

El total de teselas generadas en la tabla *data* es de aproximadamente **12 millones**, con un tamaño total del juego de datos adscrito de **6.3*GB***. En la tabla III.7.11 puede consultarse el total teórico de teselas para el área de estudio, tanto en datos acumulados para toda la jerarquía de resolución como individualmente.

Tabla III.7.11: Número potencial de teselas para una cobertura completa del área de estudio en cada nivel de resolución. La columna "Teselas" indica el total de teselas para el nivel de resolución indicado, mientras que la columna "Total acumulado" muestra el número total de teselas inscritas en la tabla *data* en el caso de cubrir todos los niveles de resolución de forma jerárquica. Fuente: elaboración propia.

Resolución	Teselas	Total acumulado
100 km	9	9
50 km	35	43
10 km	873	916
5 km	3.491	4407
1 km	87.268	91.675
500 m	349.072	440.747
250 m	1.396.288	1.837.035
125 m	5.585.152	7.422.187
25 m	139.628.800	147.050.987
5 m	3.490.720.000	3.637.770.987

De estos totales potenciales, los análisis de teselado llevados a cabo han generado el número de teselas listados en la tabla III.7.12, siendo la prueba piloto de adscripción del MDT a 5 metros la que ha generado, para una celda de 10 kilómetros, un conjunto muy importante de las mismas: 4.370.000. Este nivel de interpolación a tanta resolución, con una duración estimada para una tesela de 50 kilómetros de 43 días en un sólo trabajador, es claramente objeto de mejora y de paralelización. La teselación a esos niveles de todo el área de estudio alcanzaría el nada despreciable volumen de aproximadamente 3.500 millones de teselas.

Tabla III.7.12: Resultados de teselación para cada análisis contemplado. Se recogen el número de teselas resultante para cada uno de ellos en cada nivel de resolución. Fuente: elaboración propia.

Datos originales	100 km	50 km	10 km	5 km	1 km	500 m	250 m
Catastro	19	53	940	3.400	37.900	78.400	149.100
Población	19	53	802	2.100	11.300	22.200	46.100
MDT	19	53	973	3.700	88.600	352.400	1.404.000
HICS	19	53	965	3500	67.500	237.400	828.300
Provincia	19	53	974	3.700	88.700	352.800	1.406.600
Municipio	20	54	977	3.700	88.700	352.800	1.406.800
Núcleos de población	19	53	861	2.500	16.800	39.600	102.500

Datos originales	100 km	50 km	10 km	5 km	1 km	500 m	250 m
Sección censal	19	53	974	3.700	88.700	352.800	1.406.600
EENNPP	19	51	507	1.300	20.600	76.000	290.300

El total de variables creadas por los análisis de adscripción es de 17.610, la mayoría correspondiente a procesos del tipo "DISCRETEPOLYAREASUMMARY", que generan una variable por cada categoría temática encontrada en la tesela. El número de variables generadas por cada análisis se puede consultar en la tabla III.7.10, y el desglose de cómo se reparten estas variables en las teselas en la tabla III.7.13. En ella se puede observar cómo la densidad de variables en una tesela decrece, lógicamente, a medida que aumenta la escala. Es interesante ver cómo la asimetría en la información de adscripción se expresa según el tipo de ámbito territorial escogido. Así, en núcleos urbanos densos, la media de variables por tesela es muy superior a la de ámbitos rurales.

Tabla III.7.13: Total de teselas y estadísticas de variables por cada una de ellas en los distintos niveles de resolución. "Total teselas" representa el número total de teselas presentes en el juego final de datos adscritos, "Máximo variables", "Mínimo variables" y "Media variables" son el número máximo, mínimo y de media de variables encontradas para el conjunto de teselas del nivel de resolución, mientras que "Media Sevilla", "Media sierra" y "Media campiña" corresponde a la media de variables por tesela en tres ámbitos distintos: un centro urbano denso, un medio rural serrano y otro de la Campiña del Valle del Guadalquivir. Fuente: elaboración propia.

Resolución	Total teselas	Maximo variables	Mínimo variables	Media variables	Media Sevilla	Media sierra	Media campiña
100 km	20	3285	161	2030			
50 km	55	1644	161	566			
10 km	987	652	51	168	391	91	111
5 km	3753	422	11	115	281	53	63
1 km	89.500	185	2	40	160	17	16
500 m	355.500	165	2	25	142	13	10
250 m	1.416.000	158	2	17	123	12	8
125 m	5.657.000	155	1	12	57	11	8
25 m (*)	170.000						
5 m (*)	4.200.000						

Factores en el rendimiento del teselado

Como se puede observar en la tabla III.7.10, los tiempos de teselado son, para la mayoría de los análisis, razonables. El único que se sale un poco de la escala es el análisis de los HICS: la tesela más densamente poblada de datos ha tardado en procesarse 14,5 horas. En las tablas III.7.14 y III.7.15, se pueden observar las estadísticas de complejidad de las distintas capas de origen. Las razones para el incremento tan dramático en tiempo de proceso para los HICS son dos:

- **su número de polígonos y su cobertura:** es una capa con una densidad de datos muy grande, como atestiguan el gran número de polígonos implicados y su segmentación perimetral media, aunque no tanto así su esfericidad;
- **el escaso tamaño promedio de los polígonos:** con sólo 4 ha de promedio, son con

mucho los polígonos más pequeños del conjunto de datos. Los núcleos de población también lo son, pero con una cobertura de un 2 % escaso, el procesamiento es rápido.

Es sin duda el tamaño promedio el factor más limitante del proceso, puesto que impide el aprovechamiento de las relaciones topológicas inter-resolución.

Tabla III.7.14: Estadísticas de complejidad de los datos de origen con tratamiento poligonal. Se muestra el tamaño original del dato, el número de polígonos presentes tras el procesamiento *ETL* de adaptación para los requerimientos del análisis de teselado, "SP" corresponde a la media del segmento perimetral, "Área" es la media de área de los polígonos y, por último, "Esf." es la media de esfericidad de los mismos. "Cobertura" muestra, en porcentaje, el recubrimiento del área de estudio. Fuente: elaboración propia.

Datos originales	Tam. datos (MB)	Tiempo proceso	Polígonos	SP (m)	Área (ha)	Esf.	Cobertura (%)
HICS	711	14,5 h	1.492.511	8	4	0,6	68
Provincia	3	45 min	8	73	1.094.926	0,51	100
Municipio	13	1 h	771	105	11.362	0,63	100
Núcleos de población	22	10 min	13.348	38	18	0,53	2
Sección censal	33	1 h	6.023	53	1.454	0,69	100
EENNPP	6	50 min	205	94	9.878	0,59	23

Tabla III.7.15: Estadísticas de complejidad de los datos de origen con tratamiento puntual. Se muestra el tamaño original del dato, el número de puntos y su densidad en el área de estudio. Fuente: elaboración propia.

Datos originales	Tamaño datos (MB)	Puntos	Densidad (puntos / km^2)
MDT	564	8.760.252	100
Población	310	46.100	0,5
Catastro	7.5GB	2.263.375	26

El efecto de las relaciones topológicas inter-resolución

La clave de la eficiencia en el teselado, sobre todo poligonal, es el aprovechamiento de este efecto de "profundización" en la relación topológica inter-resolución de la que se pueden valer algunos análisis de adscripción. El método "DISCRETEPOLYAREASUMMARY" es capaz de aprovecharlo cuando una única categoría temática ocupa el 100 % de la tesela.

En la figura III.7.2 se muestra gráficamente el efecto que tienen las relaciones inter-resolución en teselas madre - descendientes.

La secuencia de figuras III.7.3 - III.7.9 muestra el avance en la secuencia de teselación de una tesela de 50 km de resolución, en este caso de los municipios. Nótese como el proceso comienza teselando la resolución de 10 km pero cómo inmediatamente que una de ellas (fig. III.7.3) alcanza el 100 % se crean en cascada todas las teselas descendientes hasta el nivel de resolución objetivo del análisis. Esto va sucediendo en la secuencia con todas las teselas de los distintos niveles de resolución, hasta que finalmente las únicas teselas

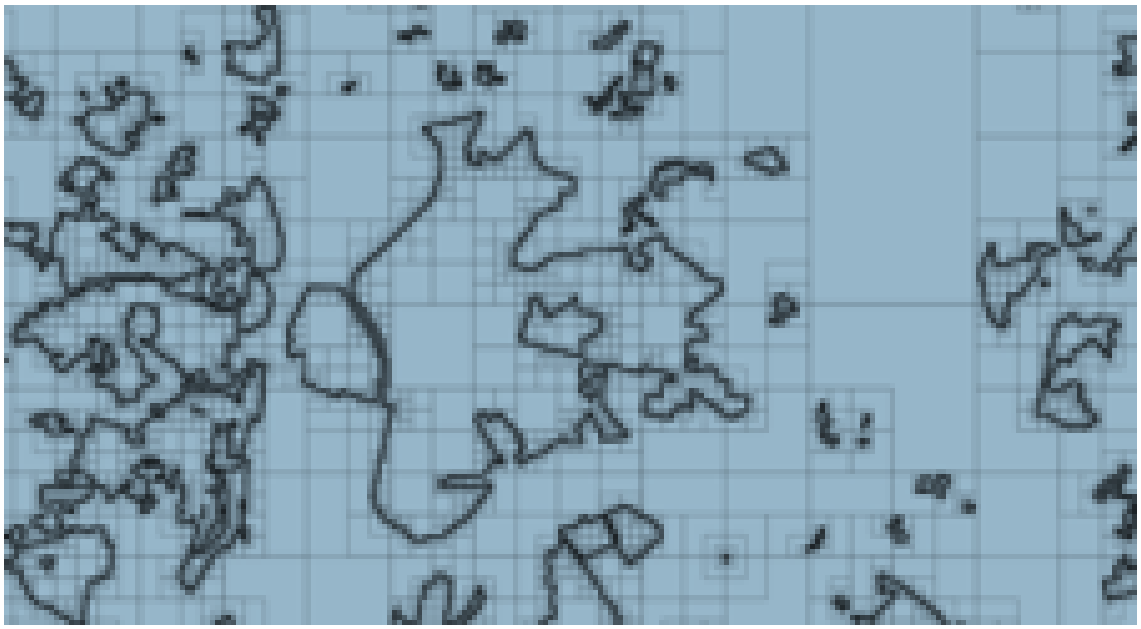

Figura III.7.2: Esta figura muestra el efecto de herencia inter-resolución de un conjunto de datos, en este caso, los núcleos de población. Nótese como en el interior de polígonos especialmente grandes se pueden acomodar grandes teselas que no tocan los límites del mismo. En estas teselas se alcanza el 100 % de ocupación de la categoría temática, por lo que a todas las teselas descendientes les es asignada, sin procesamiento de ningún tipo, la misma variable de adscripción que la tesela madre que alcanzó el 100 % originalmente, acelerando enormemente el proceso en su conjunto. Nótese como la resolución teselar se intensifica en los límites de los polígonos, donde el área de ocupación de la variable temática categórica no llega al 100 % y, por lo tanto, hay que seguir procesando las teselas descendientes individualmente. Fuente: elaboración propia.

que quedan por calcularse mediante la aplicación del análisis, y no por herencia, son las del nivel de resolución objetivo, que son precisamente aquellas que delimitan las fronteras entre los polígonos y, por tanto, entre clasificaciones categóricas de la variable.

Computación paralela

Otro resultado destacable del proceso de teselación es la computación paralela entre los distintos microservicios trabajadores, mostrado en la secuencia de figuras III.7.10 - III.7.16. En la secuencia, cada color corresponde al proceso realizado un por trabajador en concreto. A efectos de prueba de carga del sistema, se ha ido aumentando el número de trabajadores de 3 a 5.

Interpolación MDT

Un análisis aparte merece el análisis de adscripción “MDTPROCESSING”, orientado a la adscripción de modelos digitales del terreno. Como se recordará, este análisis aplica una técnica de adscripción mixta:

- si la resolución de la tesela está por encima de la resolución nativa del MDT, se aplica como dato a la tesela la media de los puntos que caen dentro de la misma;

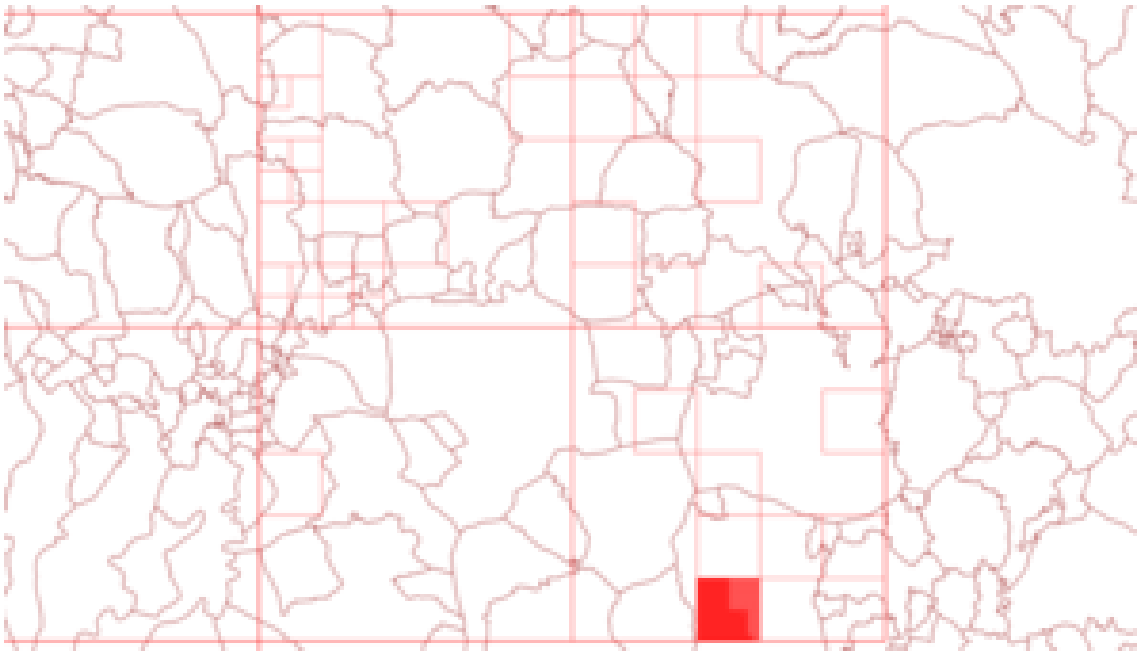

Figura III.7.3: Secuencia de teselado por herencia, paso 1. Fuente: elaboración propia.

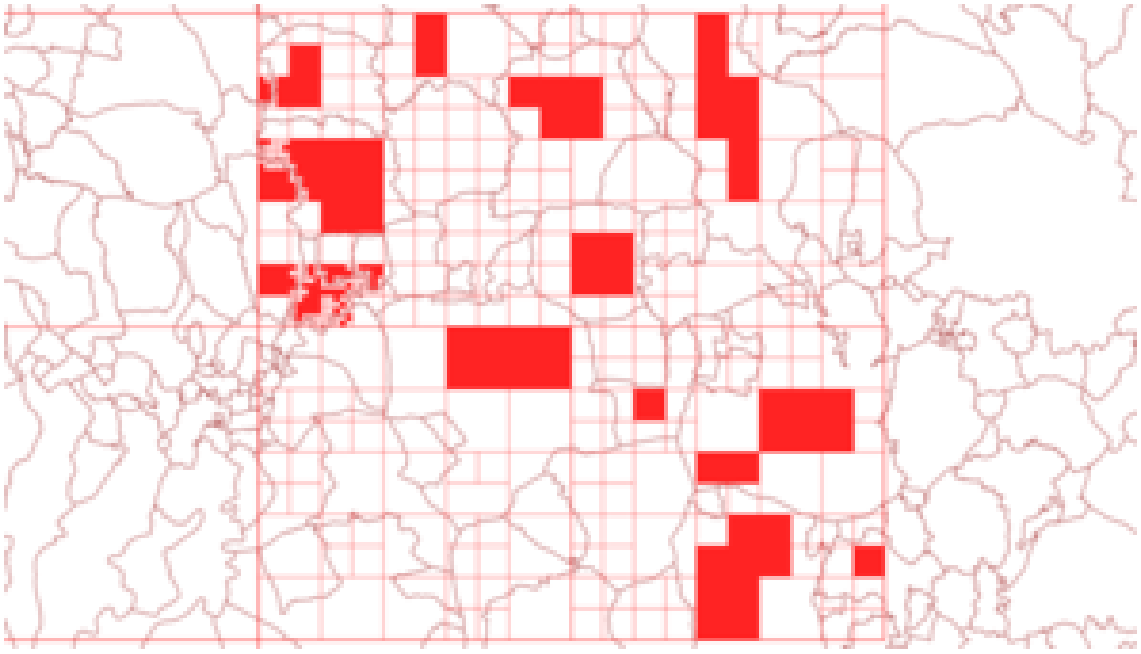

Figura III.7.4: Secuencia de teselado por herencia, paso 2. Fuente: elaboración propia.

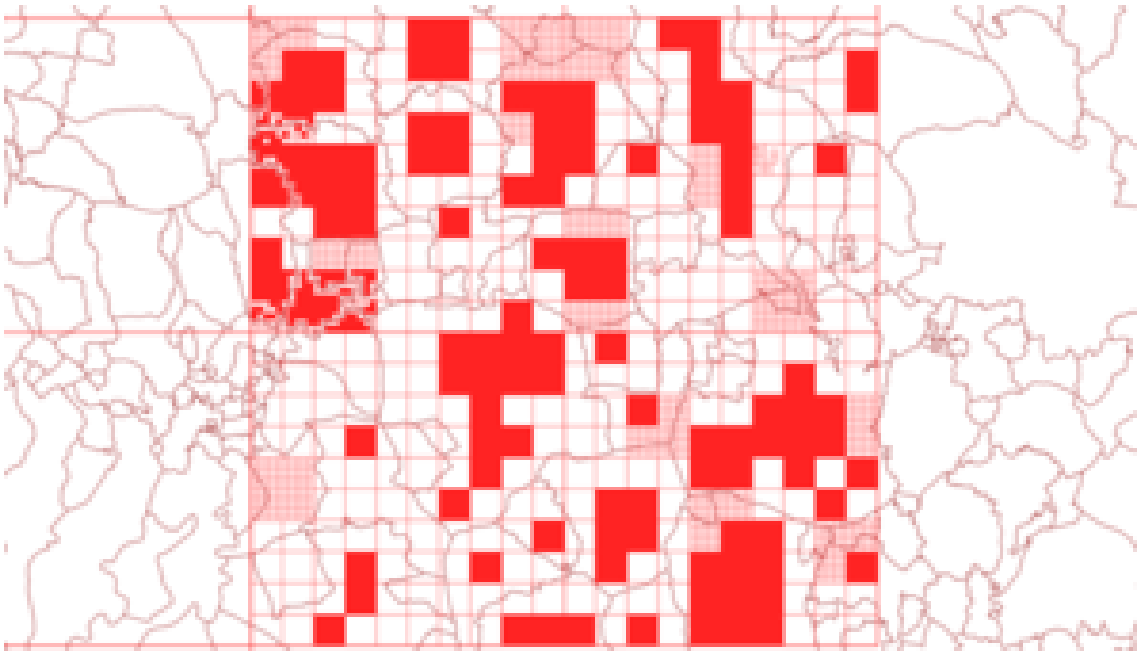

Figura III.7.5: Secuencia de teselado por herencia, paso 3. Fuente: elaboración propia.

Figura III.7.6: Secuencia de teselado por herencia, paso 4. Fuente: elaboración propia.

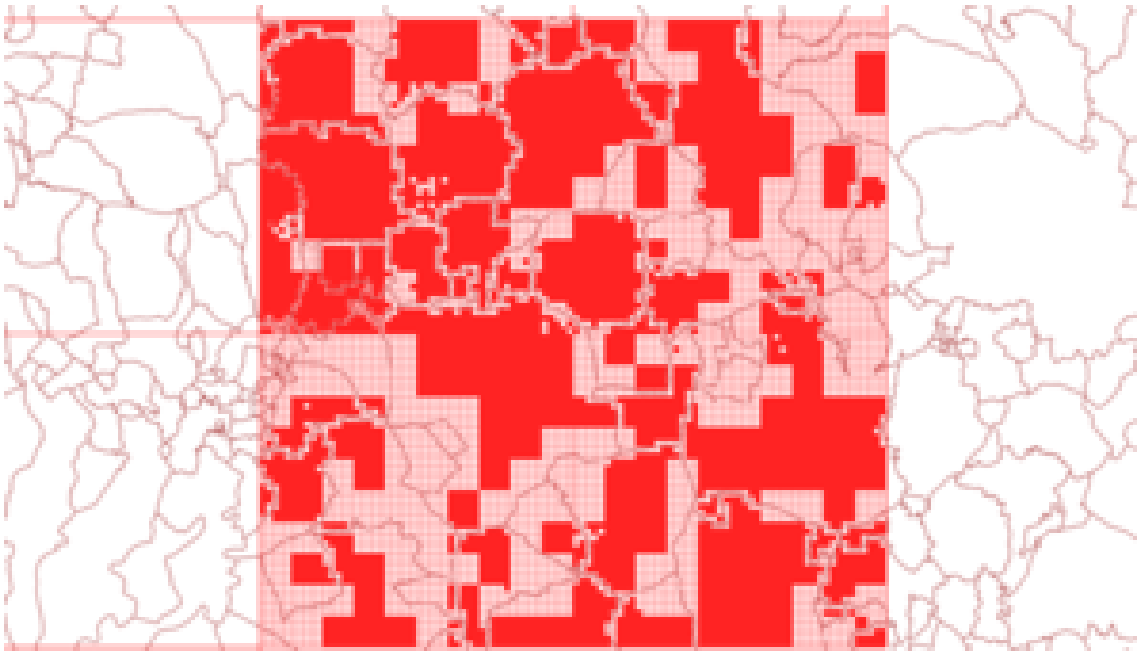

Figura III.7.7: Secuencia de teselado por herencia, paso 5. Fuente: elaboración propia.

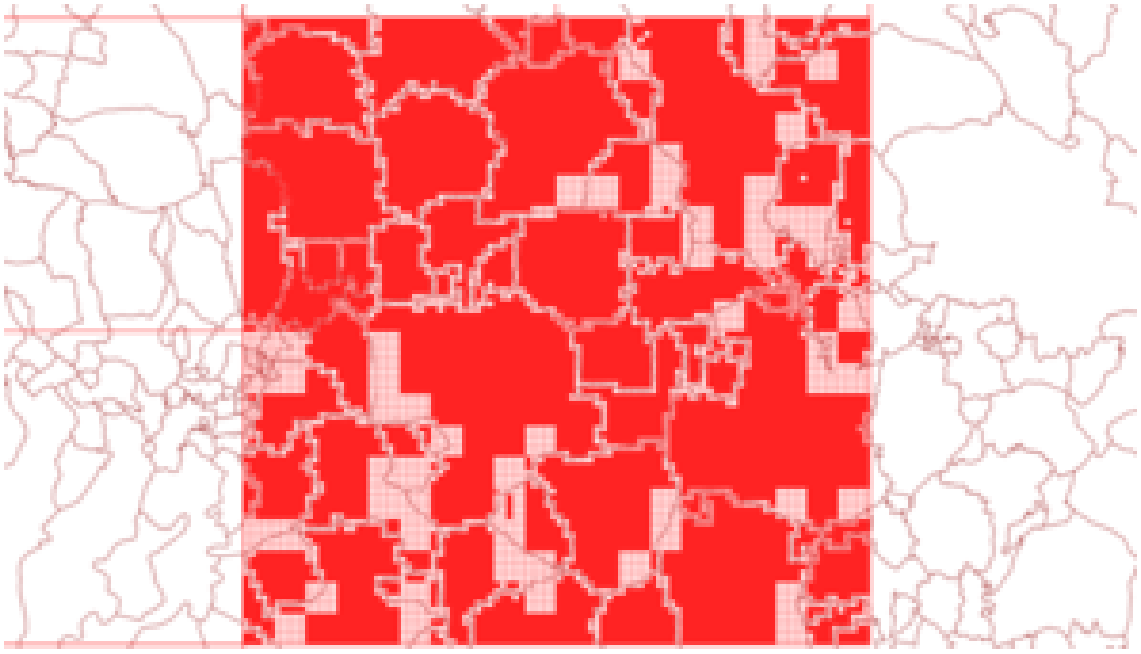

Figura III.7.8: Secuencia de teselado por herencia, paso 6. Fuente: elaboración propia.

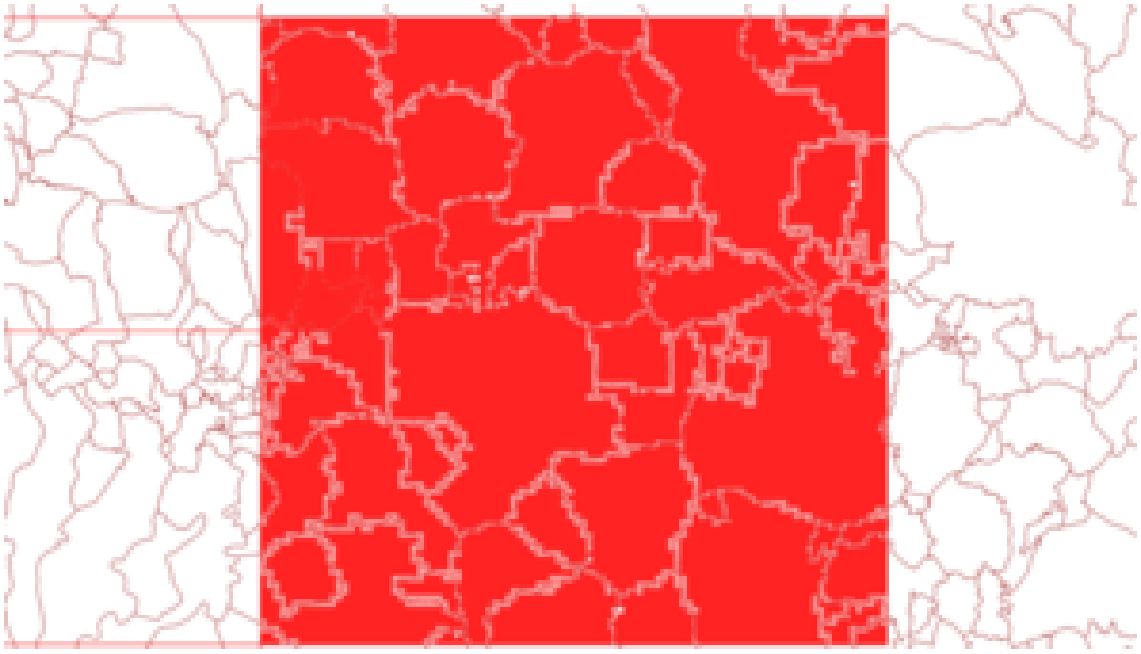

Figura III.7.9: Secuencia de teselado por herencia, paso 7. Fuente: elaboración propia.

Figura III.7.10: Reparto del procesamiento de los HIC entre los microservicios trabajadores en la resolución de 50 km. Cada color corresponde al trabajo llevado a cabo por un trabajador distinto. Fuente: elaboración propia.

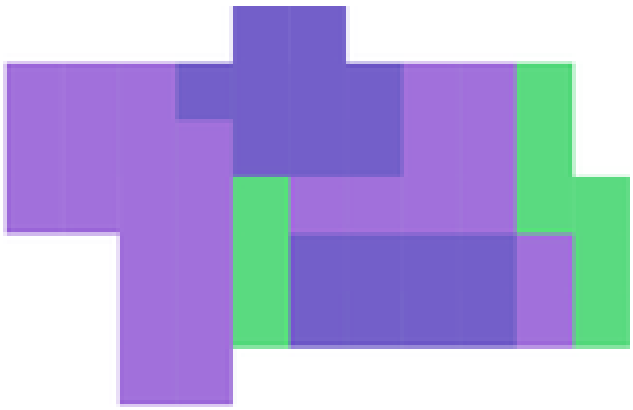

Figura III.7.11: Reparto del procesamiento de los HIC entre los microservicios trabajadores en la resolución de 10 km. Cada color corresponde al trabajo llevado a cabo por un trabajador distinto. Fuente: elaboración propia.

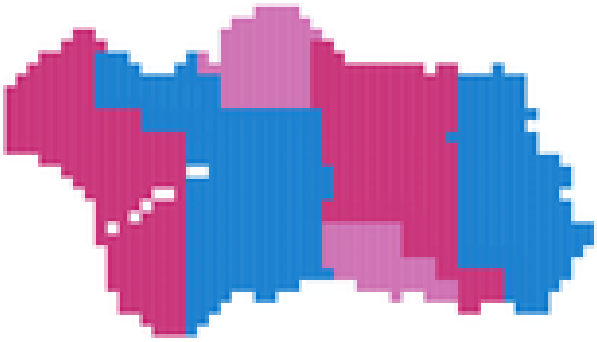

Figura III.7.12: Reparto del procesamiento de los HIC entre los microservicios trabajadores en la resolución de 5 km. Cada color corresponde al trabajo llevado a cabo por un trabajador distinto. Fuente: elaboración propia.

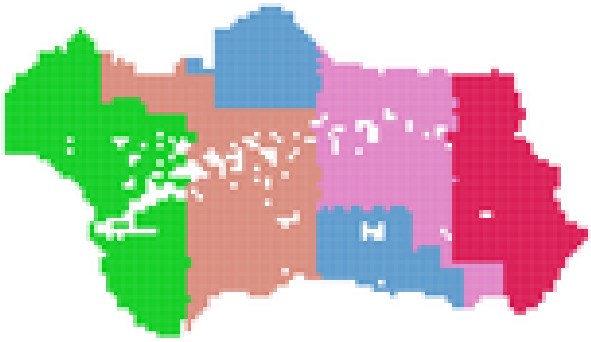

Figura III.7.13: Reparto del procesamiento de los HIC entre los microservicios trabajadores en la resolución de 1 km. Cada color corresponde al trabajo llevado a cabo por un trabajador distinto. Fuente: elaboración propia.

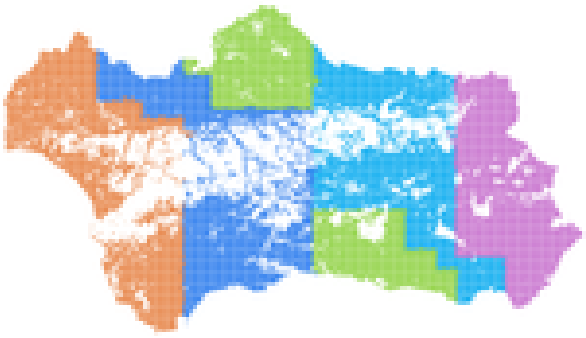

Figura III.7.14: Reparto del procesamiento de los HIC entre los microservicios trabajadores en la resolución de 500 m. Cada color corresponde al trabajo llevado a cabo por un trabajador distinto. Fuente: elaboración propia.

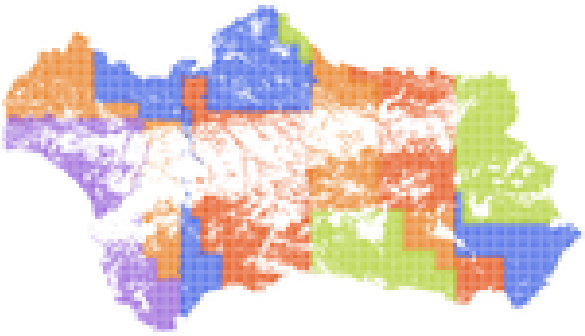

Figura III.7.15: Reparto del procesamiento de los HIC entre los microservicios trabajadores en la resolución de 250 m. Cada color corresponde al trabajo llevado a cabo por un trabajador distinto. Fuente: elaboración propia.

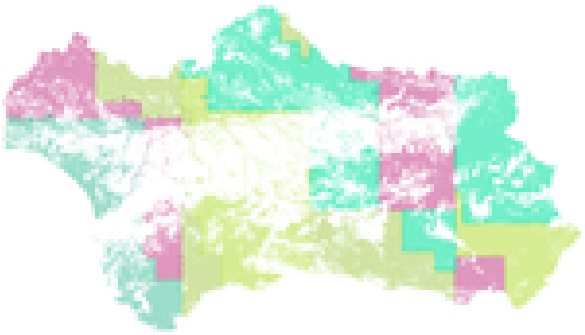

Figura III.7.16: Reparto del procesamiento de los HIC entre los microservicios trabajadores en la resolución de 125 m. Cada color corresponde al trabajo llevado a cabo por un trabajador distinto. Fuente: elaboración propia.

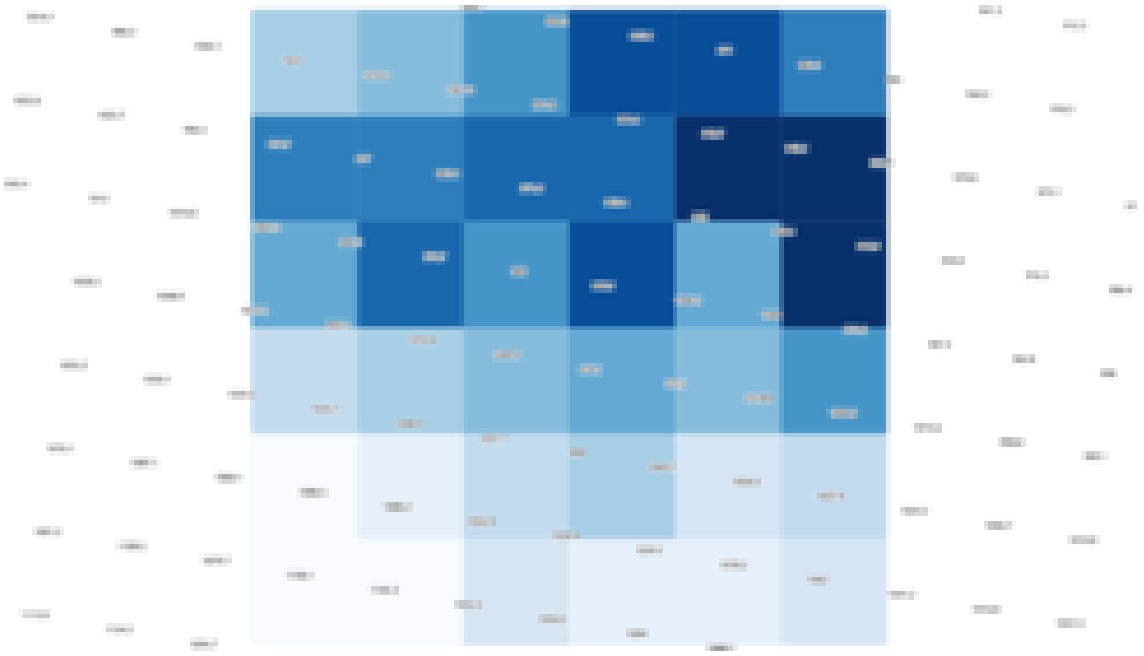

Figura III.7.17: Adscripción de los datos del MDT a 125 metros. Por estar por encima de la resolución nativa del juego de datos (100 metros), el análisis de teselado "MDTPROCESSING" adscribe los datos originales, señalados por los valores de cota, por media de los puntos que caen dentro de la tesela. Fuente: elaboración propia.

- si, por el contrario, la tesela está por debajo de la resolución del MDT, se aplica la interpolación *IDW*.

Las figuras III.7.17, III.7.18 y III.7.19 muestran los resultados de dicho análisis de adscripción a 125 (por encima de la resolución del MDT), 25 y t metros, estas últimas dos resoluciones por debajo.

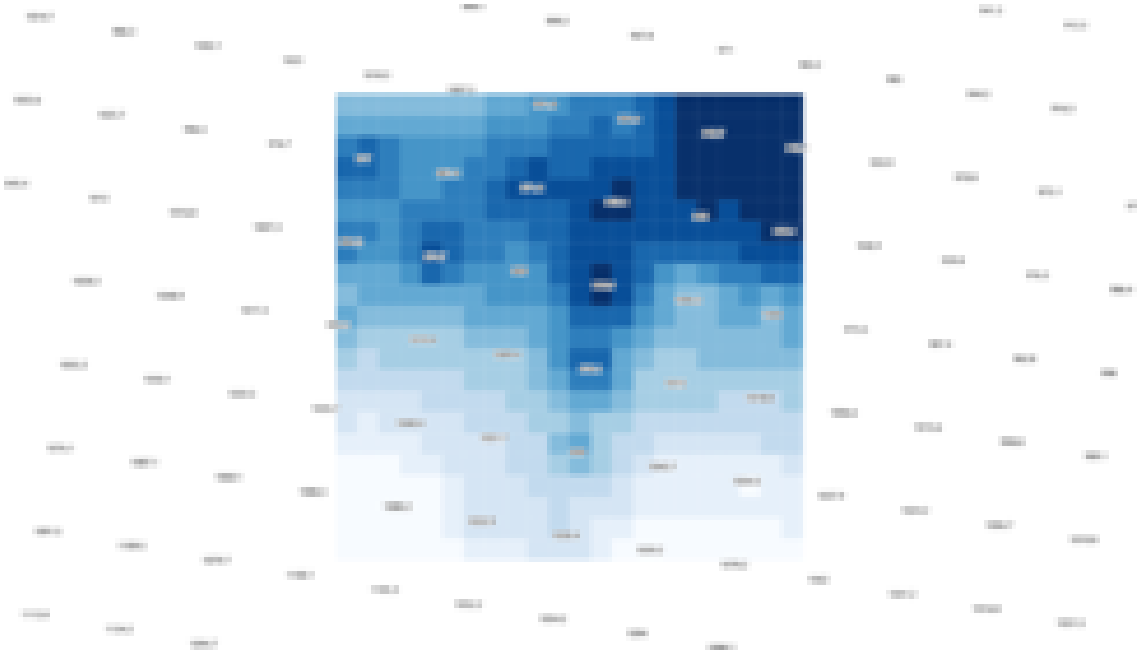

Figura III.7.18: Adscripción de los datos del MDT a 25 metros. Por estar por debajo de la resolución nativa del juego de datos (100 metros), el análisis de teselado "MDTPROCESSING" adscribe los datos originales, señalados por los valores de cota, por interpolación *IDW*. Fuente: elaboración propia.

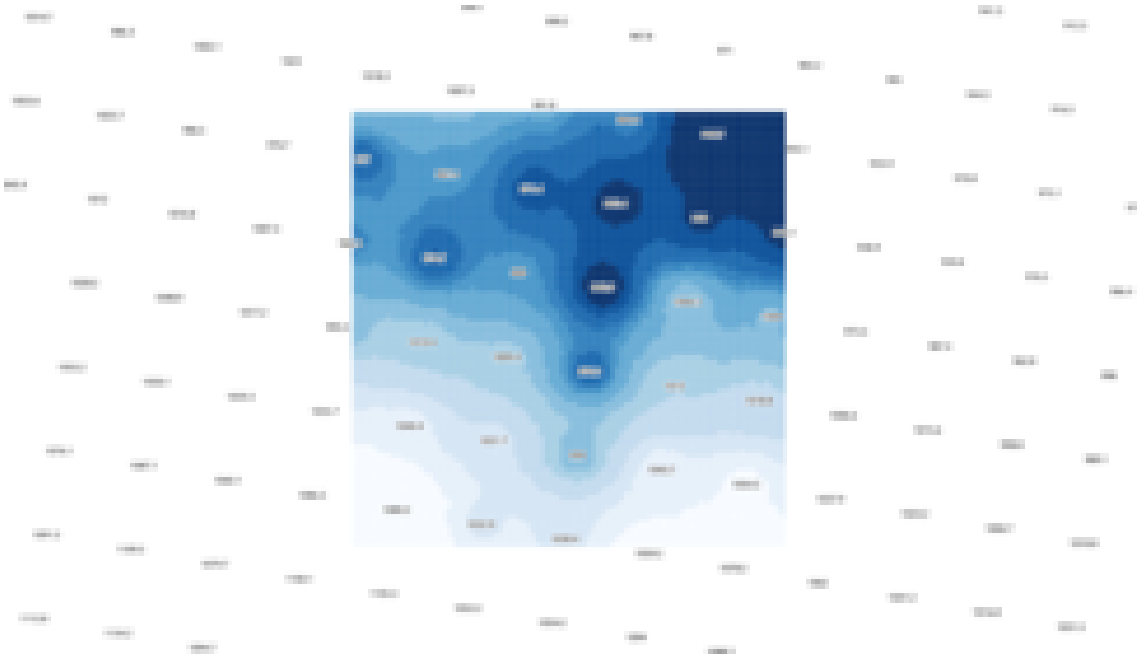

Figura III.7.19: Adscripción de los datos del MDT a 5 metros. Por estar por debajo de la resolución nativa del juego de datos (100 metros), el análisis de teselado "MDTPROCESSING" adscribe los datos originales, señalados por los valores de cota, por interpolación *IDW*. Fuente: elaboración propia.

III.7.5. Formato final de la información adscrita

Para cerrar este capítulo sobre la adscripción haremos un repaso al formato final que tiene la información adscrita. Para ello, se han seleccionado dos teselas de control del nivel de resolución de un kilómetro en los alrededores de la ciudad de Granada, una en entorno urbano (4/411/176) y otra en entorno rural (4/424/182).

Para entender el producto de datos de los análisis de adscripción se proporcionan los *JSON* de datos de ambas teselas (listados III.7.12 y III.7.13). La tesela urbana, como ya se vió, tiene más datos, especialmente por los datos de población: 150 variables codificadas. La rural, 19. Para interpretar los resultados se proporciona también la tabla III.7.16, que contrasta los valores de las variables, comunes a las dos teselas o no, indicando la variable en cuestión y algunas notas aclaratorias. Finalmente, esta interpretación debe apoyarse también en las figuras III.7.21, III.7.22, III.7.23, III.7.24, III.7.25, III.7.26, III.7.27, III.7.28 y III.7.29 para entender el contexto geométrico de las distintas adscripciones realizadas.

```
1 {"9": "03e", "192": 0.41, "2f7": 3, "a11": 1, "c7a": 0.23, "05a2": 0,
    "0670": 542, "0a7e": 0.21, "0beb": 0.13, "1041": 535, "1111": {
    area: 1}, "1810": 0.21, "1db1": 359, "1f35": 85, "1fe8": 13, "204a
    ": 582, "21da": 0, "2222": {area: 0.41}, "2555": 77, "26f8": 302,
    "2951": 0, "3547": 331, "3607": 231560, "3a72": 0, "3b20": 128, "3
    c08": 1, "3cd1": 291, "3d21": 146, "3d28": 302, "3eb7": 354,
    "4302": 0, "4445": 0, "4523": 0.36, "4892": 591, "4a29": 160, "52
    ed": 0.64, "5555": {area: 0.35}, "5654": 0.56, "567c": 300, "57c7
    ": 0, "57ea": 0.57, "5807": 270, "5b67": 13, "5c96": 0.37, "5fdb":
     2, "6108": 0.22, "630c": 0, "667d": 9, "6824": 426, "6ae0": 75,
    "6e73": 0, "6fdf": 562, "71dc": 181, "75d3": 304, "75fe": 0.22,
    "796d": 0.41, "7a5d": 1, "7c8c": 0.4, "7f2f": 686, "8071": 480,
    "8081": 1, "8186": 586, "871c": 0.61, "87d1": 0, "8932": 2,
    "8956": 0, "8ac5": 7, "8b66": 365, "8d73": 278, "8e93": 80, "8f9e
    ": 76, "90e1": 0, "9416": 0.22, "954d": 150078, "95f4": 131,
    "9789": 0.63, "9892": 572, "9932": 3, "9fb0": 0.36, "a3d6": 0.68,
    "a461": 9, "a55f": 5, "a57c": 576, "a60d": 373, "a621": 299, "b426
    ": 0.46, "b82c": 359, "b91c": 1910, "be32": 81482, "bebf": 78, "
    c02a": 414, "c096": 1, "c206": {nPoints: "179"}, "c231": 1, "c416
    ": 5, "cd33": 167, "cdd2": 328, "cf13": 568, "cf1f": 0.21, "d0a4":
     364399, "d0ad": 148, "d4ce": 350, "d819": 0.4, "d97e": 628, "df3b
    ": 0, "dfba": 6, "e01a": 369, "e13a": 1132.3, "e2fa": 2012, "e589
    ": 557, "e659": 2, "e6b8": {nPoints: "7"}, "eb4f": 22, "eda4": 13,
     "ef77": 0, "efc7": 0, "f48d": 136, "fb97": 1, "fe2f": 0.58,
    "00000": {area: 1}, "0e061": 5, "0f55d": 0, "24e0e": 9, "29e1e":
    0, "2e47b": 5, "2fa5d": 0, "37eec": 78, "44444": {area: 1}, "5383f
    ": 585, "53dea": 573, "64a78": 0.59, "667d6": 279, "8fd3d": 0.42,
    "9382c": 354, "a1abc": 815.7, "ab008": 0.04, "ccea1": 0, "e0762":
    264, "e3f8f": 0.22, "e8dc1": 272, "e9579": 0, "f4805": 668, "ffa2b
    ": 273}
```

Código III.7.12: Datos de adscripción de la tesela de control urbana 4/411/176.

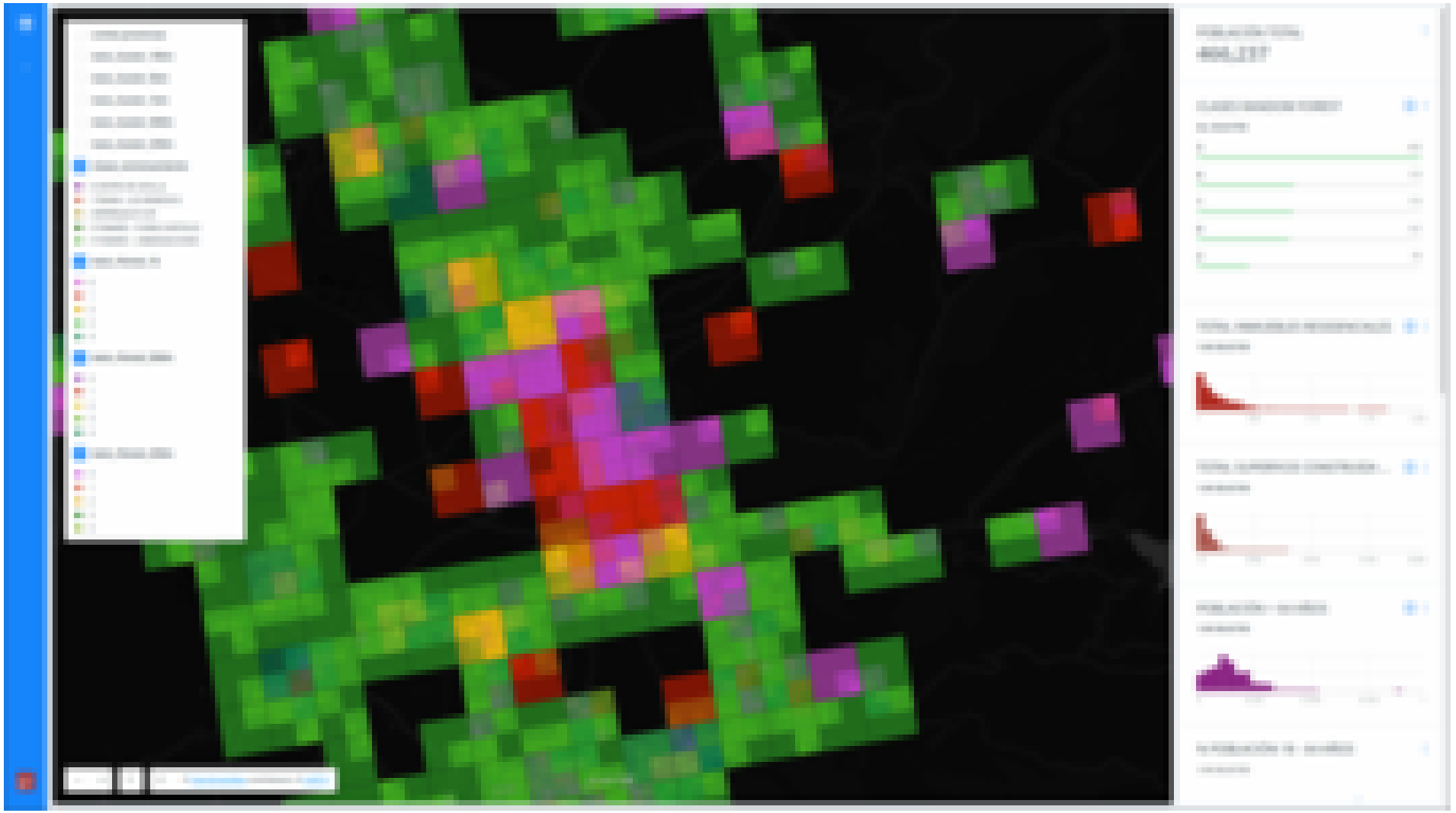

Figura III.7.20: Teselación en el área metropolitana de Granada, combinando las teselas semitransparentes de 1 kilómetro, 500 y 250 metros, con los datos de un resultado de *Machine Learning* con *Random forest* (ver próximo capítulo).

```
{"9": "10", "4f": 0.27, "9e": 0.87, "ec": 0.73, "3da": 0.14, "447":
    0.7, "7bf": 0.34, "a82": 1, "c7a": 0.98, "e46": 0.05, "1111": {
    area: 1}, "5555": {area: 3.08}, "7dea": 0.73, "f8c7": 0.27, "fb97
    ": 1, "00000": {area: 1}, "44444": {area: 1}, "a1abc": 1519.7}
```

Código III.7.13: Datos de adscripción de la tesela de control rural 4/424/182.

Tabla III.7.16: Comparativa de las variables de adscripción encontradas en los datos de las dos teselas de control de un kilómetro, de coordenadas 4/411/176 (entorno urbano) y 4/424/182 (entorno rural). Fuente: elaboración propia.

Clave variable	4/411/176	4/424/182	name
00000	{area: 1}	{area: 1}	Provincia: variable índice, "area: 1" representa recubrimiento completo de teselas con polígonos de provincias
05a2	0		Población: Población americana 2016
0670	542		Población: Población española 2017
0a7e	0.21		Población: Índice de dependencia anciana 2017
0beb	0.13		HICS: Área HIC Matorrales áridos y semiáridos (Matorrales termomediterráneos pre-estépicos). Subtipo: Matorrales de sustitución termófilos, con endemismos
0e061	5		Población: Población UE Schengen 2013
0f55d	0		Población: Población americana 2015
1041	535		Población: Población española 2018
1111	{area: 1}	{area: 1}	Municipios: variable índice, recubrimiento completo de teselas
1810	0.21		Población: Índice de dependencia anciana 2016
192	0.41		Núcleos población: Área Granada (Cabecera)
1db1	359		Población: Población edad 18-64 2018
1f35	85		Población: Población edad mayor 64 2013
1fe8	13		Población: Población UE Schengen 2018
204a	582		Población: Población total 2016
21da	0		Población: Población otros 2017
2222	{area: 0.41}		Núcleos población: variable índice, indica una cobertura del 41 % de la primera tesela por combinación de núcleos de población
24e0e	9		Población: Población americana 2014
2555	77		Población: Población edad mayor 64 2002
26f8	302		Población: Población hombres 2017
2951	0		Población: Población magrebí 2002
29e1e	0		Población: Población magrebí 2013
2e47b	5		Población: Población americana 2013
2f7	3		Catastro: Bienes inmuebles uso principal Almacén - Estacionamiento
2fa5d	0		Población: Población UE Schengen 2002
33333		{area: 1}	EENNPP: variable índice, recubrimiento completo de la segunda tesela
3547	331		Población: Población hombres 2013
3607	231560		Catastro: Total de superficie construida
37eec	78		Población: Población edad mayor 64 2014
3a72	0		Población: Población magrebí 2018
3b20	128		Población: Población edad 0-17 2017
3c08	1		Catastro: Bienes inmuebles uso principal Ocio y Hostelería
3cd1	291		Población: Población hombres 2018
3d21	146		Población: Población edad 0-15 2015
3d28	302		Población: Población hombres 2002
3da		0.14	HICS: Área HIC Zonas subestépicas de gramíneas y anuales del Thero-Brachypodietea. Subtipo: Majadales de Poa bulbosa (Poetea bulbosae) (*-)
3eb7	354		Población: Población edad 16-64 2002
4302	0		Población: Población magrebí 2016
44444	{area: 1}	{area: 1}	Secciones censales: variable índice, recubrimiento completo de las dos teselas por secciones censales
4445	0		Secciones censales: Área 1808703020

Clave variable	4/411/176	4/424/182	name
447		0.7	HICS: Área HIC Formaciones estables xerotermófilas de Buxus sempervirens en pendientes rocosas (Berberidion p p). Subtipo: Espinares y orlas húmedas (Rhamno-Prunetalia)
4523	0.36		Población: Índice de dependencia infantil 2018
4892	591		Población: Población total 2014
4a29	160		Población: Población edad 0-15 2014
4f		0.27	Municipios: Área Beas de Granada (Granada), las segunda tesela está entre dos municipios
52ed	0.64		Población: Índice de dependencia total 2002
5383f	585		Población: Población total 2002
53dea	573		Población: Población española 2002
5555	{area: 0.35}	{area: 3.08}	HICS: variable índice, indica recubrimiento de las teselas. La segunda tiene un 300 % de recubrimiento debido a que los polígonos de HICS se repiten debido a la relación uno a muchos entre polígonos y HICS
5654	0.56		Secciones censales: Área 1808708006
567c	300		Población: Población hombres 2016
57c7	0		Población: Población otros 2002
57ea	0.57		Población: Índice de dependencia total 2017
5807	270		Población: Población mujeres 2017
5b67	13		Catastro: Bienes inmuebles uso principal Industrial
5c96	0.37		Población: Índice de dependencia infantil 2016
5fdb	2		Catastro: Bienes inmuebles uso principal Religioso
6108	0.22		Población: Índice de dependencia anciana 2015
630c	0		Población: Población UE Schengen 2016
64a78	0.59		Población: Índice de dependencia total 2018
667d	9		Población: Población otros 2013
667d6	279		Población: Población mujeres 2018
6824	426		Catastro: Número de bienes inmuebles
6ae0	75		Población: Población edad mayor 64 2016
6e73	0		Población: Población UE Schengen 2014
6fdf	562		Población: Población española 2016
71dc	181		Catastro: Total de direcciones postales
75d3	304		Población: Población hombres 2015
75fe	0.22		Población: Índice de dependencia anciana 2018
796d	0.41		Población: Índice de dependencia infantil 2015
7a5d	1		Catastro: Bienes inmuebles uso principal Edificio Singular
7bf		0.34	HICS: Área HIC Dehesas perennifolias de Quercus spp.
7c8c	0.4		Población: Índice de dependencia infantil 2013
7dea		0.73	Secciones censales: Área 1809901001
7f2f	686		Catastro: Total unidades constructivas
8071	480		Catastro: Total de unidades constructivas de uso residencial
8081	1		Catastro: Media de usos diferentes por parcela
8186	586		Población: Población total 2015
871c	0.61		Población: Índice de dependencia total 2013
87d1	0		Población: Población magrebí 2015
8932	2		*Machine Learning*: Clúster KMeans estructura población
8956	0		Población: Población magrebí 2017
8ac5	7		Población: Población otros 2015
8b66	365		Población: Población edad 16-64 2016
8d73	278		Población: Población mujeres 2002
8e93	80		Población: Población edad mayor 64 2018
8f9e	76		Población: Población edad mayor 64 2017
8fd3d	0.42		Población: Índice de dependencia infantil 2002
9	03e	10	Municipio: máxima área. “03e” es el código de “Granada” y “10” el de “Huétor-Santillán”
90e1	0		Población: Población americana 2017
9382c	354		Población: Población edad 16-64 2015
9416	0.22		Población: Índice de dependencia anciana 2002
954d	150078		Catastro: Superficie construida sobre rasante
95f4	131		Población: Población edad 0-18 2018
9789	0.63		Población: Índice de dependencia total 2015

Clave variable	4/411/176	4/424/182	name
9892	572		Población: Población total 2017
9932	3		*Machine Learning*: Clase Random forest población + residencial + densidad
9e		0.87	HICS: Área HIC Matorrales pulvinulares orófilos europeos meridionales. Subtipo: Matorrales almohadillados de media montaña, meso-supramediterráneos, endémicos
9fb0	0.36		Población: Índice de dependencia infantil 2017
a11	1		Municipios: Área Granada (Granada)
a1abc	815.7	1519.7	MDT: Procesamiento MDT
a3d6	0.68		Población: Índice de dependencia total 2014
a461	9		Población: Población UE Schengen 2017
a55f	5		Población: Población otros 2018
a57c	576		Población: Población total 2018
a60d	373		Catastro: Total de inmuebles con al menos una construcción con destino residencial
a621	299		Población: Población hombres 2014
a82		1	EENNPP: Área SIERRA DE HUÉTOR (Parque Natural)
ab008	0.04		Secciones censales: Área 1808708004
b426	0.46		Población: Índice de dependencia infantil 2014
b82c	359		Población: Población edad 17-64 2017
b91c	1910		Catastro: Bien inmueble más antiguo
be32	81482		Catastro: Superficie construida bajo rasante
bebf	78		Población: Población edad mayor 64 2015
c02a	414		Población: Población edad 16-64 2013
c096	1		Catastro: Bienes inmuebles uso principal Sanidad y Beneficencia
c206	{nPoints: 179}		Catastro: variable índice, indica que en la tesela se han encontrado 179 puntos de la nube de catastro
c231	1		Catastro: Bienes inmuebles uso principal Oficina
c416	5		Población: Población americana 2018
c7a	0.23	0.98	HICS: Área HIC Zonas subestépicas de gramíneas y anuales del Thero-Brachypodietea. Subtipo: Pastizales vivaces neutro-basófilos mediterráneos (Lygeo-Stipetea) (*-)
ccea1	0		Población: Población magrebí 2014
cd33	167		Población: Población edad 0-15 2013
cdd2	328		Población: Población mujeres 2013
cf13	568		Población: Población española 2014
cf1f	0.21		Población: Índice de dependencia anciana 2013
d0a4	364399		Catastro: Total superficie fincas con construcciones
d0ad	148		Población: Población edad 0-15 2002
d4ce	350		Población: Población edad 16-64 2014
d819	0.4		Secciones censales: Área 1808703002
d97e	628		Población: Población española 2013
df3b	0		Población: Población UE Schengen 2015
dfba	6		Población: Población otros 2016
e01a	369		Catastro: Bienes inmuebles uso principal Residencial
e0762	264		Población: Población mujeres 2016
e13a	1132.3		Catastro: Superficie media de inmuebles
e2fa	2012		Catastro: Bien inmueble más moderno
e3f8f	0.22		Población: Índice de dependencia anciana 2014
e46		0.05	HICS: Área HIC Bosques de Quercus ilex y Quercus rotundifolia
e589	557		Población: Población española 2015
e659	2		Catastro: Construcciones reformadas
e6b8	{nPoints: 7}		Población: variable índice, indica que se han encontrado 7 puntos de población en la tesela
e8dc1	272		Población: Población mujeres 2014
e9579	0		Población: Población otros 2014
eb4f	22		Catastro: Bienes inmuebles uso principal Comercial
ec		0.73	Municipios: Área Huétor-Santillán (Granada)
eda4	13		Catastro: Bienes inmuebles uso principal Cultura
ef77	0		Secciones censales: Área 1808703021
efc7	0		Población: Población americana 2002

Clave variable	4/411/176	4/424/182	name
f4805	668		Población: Población total 2013
f48d	136		Población: Población edad 0-16 2016
f8c7		0.27	Secciones censales: Área 1802401001
fb97	1	1	Municipios: Área Granada
fe2f	0.58		Población: Índice de dependencia total 2016
ffa2b	273		Población: Población mujeres 2015

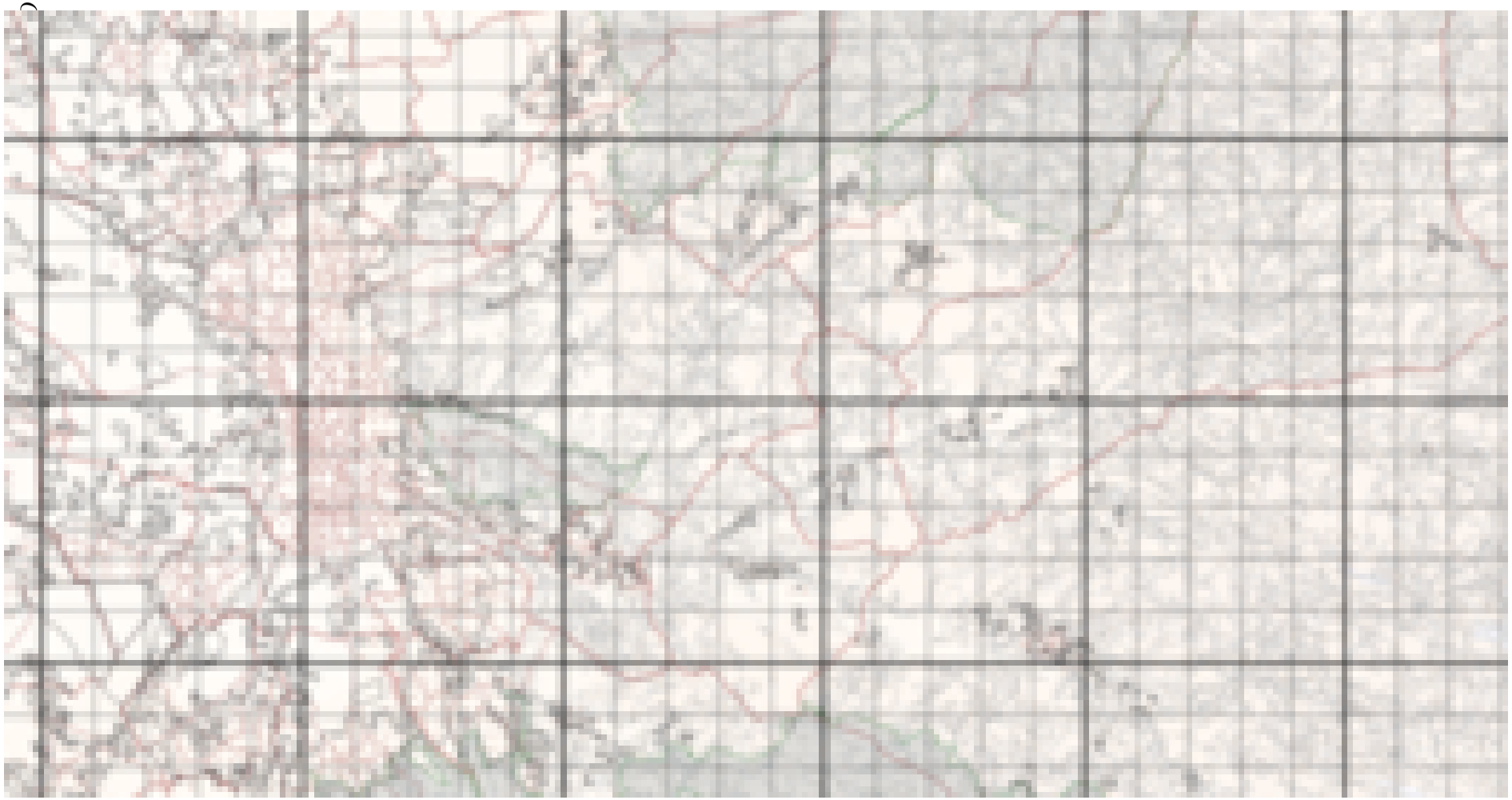

Figura III.7.21: La ciudad de Granada y alrededores. Las celdas en oscuro son de 5 km, mientras que las grises son de 1 km. Se pueden observar los datos de municipios, secciones censales, EENNPP, MDT, Hábitats de Interés Comunitario, núcleos de población y población. Para más detalles, ver las siguientes figuras. Estas imágenes complementan a la tabla III.7.16. Fuente: elaboración propia.

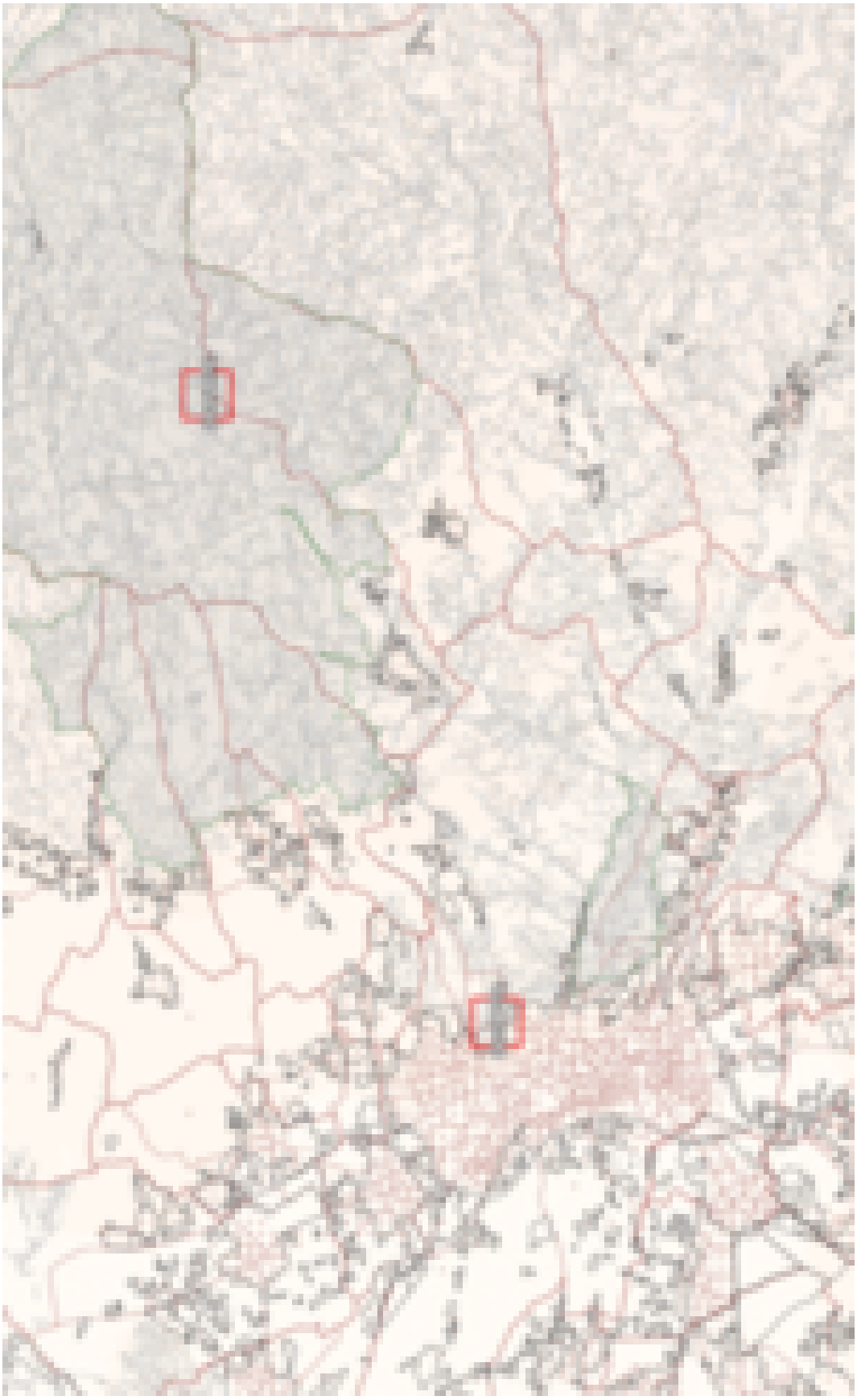

Figura III.7.22: Selección de las dos teselas de 1 kilómetro de control, una en un entorno rural y otra en entorno urbano. Fuente: elaboración propia.

Figura III.7.23: Colisiones de las tesela de control con los municipios. Nótese como la tesela de control 424/182 (oriental) colisiona con dos municipios.
Fuente: elaboración propia.

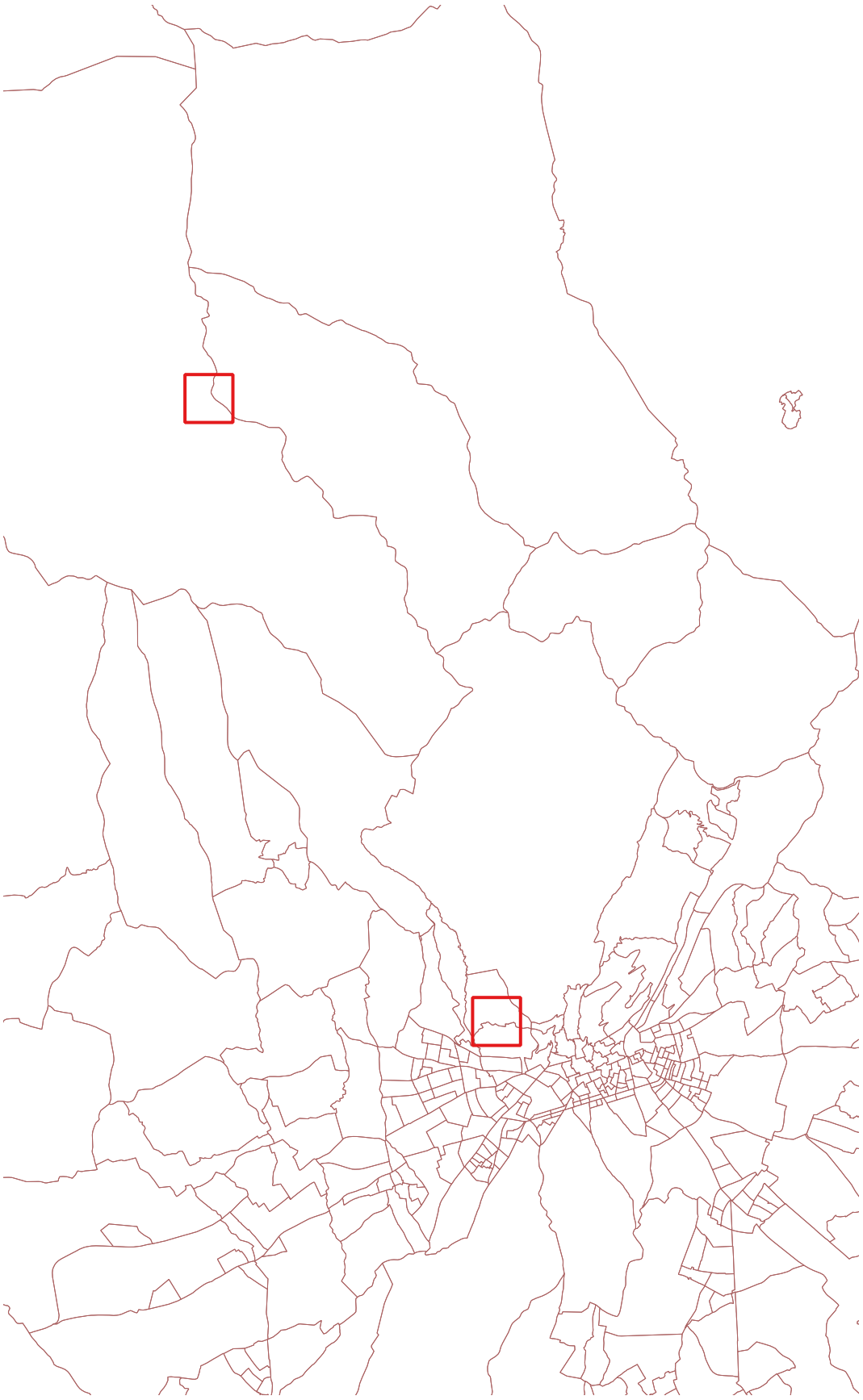

Figura III.7.24: Colisiones de las teselas de control con las secciones censales. La oriental colisiona con dos, pero la occidental con 5, aunque no se aprecie por la escala. Fuente: elaboración propia.

Figura III.7.25: Colisión de las teselas de control con los núcleos de población. Fuente: elaboración propia.

Figura III.7.26: Colisión de las teselas de control con los puntos de población. Fuente: elaboración propia.

Figura III.7.27: Colisiones de las teselas de control con los Espacios Naturales Protegidos. Fuente: elaboración propia.

Figura III.7.28: Colisiones de las teselas de control con los Hábitats de Interés Comunitario. La occidental colisiona parcialmente con 2, alcanzando el conjunto de los HICS a solapar con el 35 % de la tesela (como se puede observar en la tabla III.7.16) en la variable índice de los HICS. La oriental colisiona con 6, para un total de recubrimiento del 300 %, debido a la replicación de polígonos ya comentada. Fuente: elaboración propia.

Figura III.7.29: Colisión de las teselas de control con la malla regular de 100 metros de la nube de puntos del MDT. Fuente: elaboración propia.

III.8 Machine Learning

Para gestionar los trabajos de *Machine Learning*, se ha creado una librería llamada *PyCell*. No es tan completa ni forma el núcleo de la funcionalidad de la misma, como lo hace la librería *libcellbackend* descrita en el capítulo anterior, ya que se encarga de los procesos de *Machine Learning*.

Al igual que en la librería *libcellbackend* existe una clase genérica *GridderTask* para marcarle la pauta a un programador de cómo implementar un nuevo análisis de teselado, la librería *PyCell* crea una clase genérica *MlTask* para hacer lo propio con los métodos de *Machine Learning*.

A diferencia de *libcellbackend*, *PyCell*, como denota su nombre, está escrita en *Python*. La razón de este cambio de lenguaje son dos:

1. el conjunto de librerías y *frameworks* para hacer *Machine Learning* en *Python* supera con mucho al presente actualmente en *JavaScript*/*TypeScript*. Aunque *JavaScript* está tomando en los últimos años cierto empuje en este área y lo normal es que siga creciendo[1], actualmente *Python* es una opción más razonable para este campo;

2. como ya se discutió al hablar de la arquitectura de microservicios, una de las ventajas es que éstos están muy desacoplados, son bastante independientes, los unos de los otros. Los trajadores sólo necesitan conectarse a la memoria compartida *Redis* o, en algunos casos, a la base de datos de datos de origen o a la de *Cell*, para realizar su trabajo. Por ello, dado que *Python* es un lenguaje que tiene buenos *drivers* para *PostGIS* y *Redis*, es perfectamente factible hacer trabajar juntos microservicios escritos en lenguajes distintos. Esta es una de las grandes ventajas de este tipo de arquitecturas.

III.8.1. La clase *MlTask*

La clase *MlTask* modela un proceso de *Machine Learning* en la plataforma. Describimos sus miembros y sus métodos. Los tipos se incluyen como guía, ya que *Python* es un lenguaje debilmente tipado y en realidad no los usa.

[1] No en vano, *JavaScript* ya tiene su propio *port* de *TensorFlow*, una de las librerías más importantes de *Deep Learning*.

Miembros de la clase *MlTask*

A continuación se describen los miembros de la clase *MlTask*.

mlTaskType: EMLTASKTYPE

El tipo de procedimiento de *Machine Learning*. Actualmente, los dos ya discutidos previamente: "KMEANS" y "RANDOMFOREST", para el *K-Means* y el *Random forest*, respectivamente.

name: string

Nombre del procedimiento de *Machine Learning*.

description: string

Descripción del procedimiento de *Machine Learning*.

vector: string[]

Conjunto de las *keys* de las variables de adscripción que determinan el vector sobre el que hacer el procedimiento *ML*. Recordamos aquí que una tesela, para poder entrar en el procedimiento *ML*, debe tener todas las variables de adscripción solicitadas en su vector.

startingZoom: number y targetZoom: number

El nivel de resolución al que comenzar y terminar el proceso de *Machine Learning*.

Métodos de la clase *MlTask*

A continuación se describen los métodos de la clase *MlTask*.

train(zoomLevel: number): any

El procedimiento de *Machine Learning* entrena el modelo según las especificaciones de cada uno de los procedimientos, que difieren por ser asistidos y desasistidos. Hay que tener en cuenta que este entrenamiento debe hacerse para cada nivel de resolución que se encuentra entre *startingZoom* y *targetZoom*, ya que los dominios de las variables de adscripción pueden variar significativamente entre ellos.

predict(zoomLevel: number, vector: any): any

Aplica el modelo entrenado en *train()* al vector *vector*, devolviendo la predicción del mismo. Al igual que en el caso del método *train()*, la predicción se hace con el modelo del nivel de resolución al que pertenezca la tesela que proporciona el vector, determinado por el parámetro *zoomLevel*.

III.8.2. Clases de la familia *MlTask*

A continuación describimos las particularidades de las clases derivadas de *MlTask* que implementan los detalles de los procedimientos de *ML K-Means* y *Random forest*.

Clase *KMeansMlTask*

Esta clase derivada implementa el procedimiento del *K-Means*. Recordemos que *K-Means* es un método de *Machine Learning* no supervisado, es decir, que realiza la clasificación de los vectores en base a un entrenamiento no supervisado por un conocimiento previo del fenómeno. El miembro *mlTaskType* de esta clase es "KMEANS".

A continuación detallamos los miembros adicionales que implementa esta clase. Recordemos que una clase derivada de otra tiene todos los miembros de la clase base, pudiéndo ampliar o reimplementar éstos para especializarse:

1. ***sample*:** el número de teselas con el vector solicitado en el nivel de resolución que se está procesando que tomar como muestra para el entrenamiento. El modelo se entrenará con los valores vectoriales de dichas teselas, seleccionadas al azar;

2. ***clusters*:** el número de *clusters* o grupos a discriminar en los datos.

Clase *RandomForestMlTask*

Esta clase derivada implementa el procedimiento del *Random forest*. *Random forest* es un método supervisado, por lo que precisa de áreas de entrenamiento para entrenar el modelo para discriminarlas en la predicción futura de datos. El miembro *mlTaskType* de esta clase es "RANDOMFOREST".

Esta clase solo implementa un miembro adicional que se llama *trainingZones*. Este miembro guarda una lista de polígonos de entrenamiento junto con la clase de discriminación[2] a la que representan, con la siguiente estructura:

```
{
  "trainingZones": [
    {
      "className": "Núcleo histórico",
      "polygons": [
        { [ polígono 0 en formato GeoJSON ] },
        { [ polígono 1 en formato GeoJSON ] },
        [...],
        { [ polígono N en formato GeoJSON ] }
      ]
    },
    {
      "className": "Nuevos barrios periféricos",
      "polygons": [
        { [ polígono 0 en formato GeoJSON ] },
        { [ polígono 1 en formato GeoJSON ] },
        [...],
        { [ polígono N en formato GeoJSON ] }
```

[2]No confundir esta acepción del concepto *clase* con el *clase* de la programación orientada a objetos.

```
19          ]
20      },
21    ]
22 }
```

Código III.8.1: *JSON* que representa las zonas de entrenamiento para la clase *RandomForestMlTask*.

En el listado III.8.1 se puede ver un ejemplo de creación de dos clases de entrenamiento, "Núcleo histórico" y "Nuevos barrios periféricos", que tienen una lista de polígonos representativos de la clase en su miembro *polygons*. Estos polígonos están expresados según el estándar *GeoJSON* en el sistema de referencia WGS84 Geográficas (el del *GPS*).

Con estas zonas de entrenamiento, para cada nivel de resolución entre los seleccionados, la clase *RandomForestMlTask* extrae las teselas que colisionan con ellas y entrena con los vectores de adscripción de las mismas el modelo. Este modelo podrá ser aplicado posteriormente a los vectores de adscripción de otras teselas no pertenecientes a las zonas de entrenamiento.

III.8.3. Otras clases de la librería

Esta librería, como está tan orientada al *Machine Learning*, no necesita replicar todas las clases de la librería *libcellbackend*. Sin embargo, si precisa de implementar parte de algunas de ellas. Entre ellas, se reimplementan en *Python* las clases *Grid*, *Cell*, *Polygon* y *ZoomLevel*, con identica funcionalidad, aunque parcialmente debido al menor enfoque de esta librería, que la ya discutida en la sección III.6.7 (pag. 199).

III.8.4. Evaluación del proceso de *Machine Learning*

La plataforma cumple perfectamente con el objetivo de producir vectores de datos compatibles con los requerimientos de los algoritmos de *Machine Learning* objeto de la prueba. Aunque sólo se hayan cogido dos, la inmensa mayoría de algoritmos de este tipo[3] funcionan consumiendo la información tanto de aprendizaje como de inferencia en el formato que hemos venido en llamar *vectores de adscripción*. Gracias al uso, sobre todo, de las variables índice de los análisis de adscripción, el prototipo de la plataforma es razonablemente eficiente, aún no siendo éste el fuerte de una base de datos *SQL* como *PostGIS*, extrayendo vectores de adscripción de la información de las teselas.

Con el objetivo de demostrar el funcionamiento de la plataforma en este ámbito se han realizado dos ejercicios, uno para el *K-Means* (no supervisado) y otro para el *Random forest* (supervisado) que pasamos a comentar aquí. La librería *Python* utilizada para llevar a cabo estos análisis es *Scikit-learn* (Pedregosa et al., 2011).

[3] Y los de programación lineal y otras metodologías muy interesantes en el campo de la optimización matemática.

K-Means

El *K-Means* es un método de *Machine Learning* no supervisado, es decir, que opera sobre los datos sin entrenamiento previo proporcionado por un experto[4]. Es un algoritmo de clasificación, y se basa en encontrar una serie de *clústers* o grupos de datos en un espacio multidimensional. Las entradas al algoritmo son un grupo de datos de entrenamiento, es decir, un subconjunto de los datos que se considere significativo estadísticamente hablando (o tomados al azar). También se puede alimentar al algoritmo directamente con todos los datos a clasificar. En cualquier caso, ya sea usando una población estadísticamente significativa o tomada al azar o bien el juego completo de datos, el algoritmo aprende del grupo de entrenamiento suministrado para, posteriormente a tener el modelo, clasificar a nuevos vectores en base a lo aprendido de la muestra. El algoritmo también necesita saber el número de *clústers* a buscar.

Proporcionar el número apropiado de *clústers*, aparte de tener un set de entrenamiento significativo, obviamente, es el reto del *K-Means*. Pocos clústers provocarán un efecto de extrema generalización en el aprendizaje, con lo que las futuras clasificaciones carecerán de matices. El caso extremo es usar un único clúster: todos los vectores serán percibidos como pertenecientes al mismo, único, grupo.

El caso contrario también es peligroso. Proporcionar más *clústers* de la cuenta lleva al algoritmo a sobreajustar el modelo, es decir, a estar excesivamente condicionado por las características de la muestra. La técnica con el *K-Means* suele ser comenzar con un grupo bajo de *clústers*, estudiar la distribución intra-*clúster* de los valores clasificados y ver si se ha producido o una excesiva generalización o un sobreajuste, probando de nuevo el algoritmo aumentando o disminuyendo el número de *clústers*. En cualquier caso, en este proceso de ajuste se aprende, por lo general, bastante de los datos.

Para probar el funcionamiento del prototipo de la plataforma con este algoritmo se han tomado los vectores de las rejillas formados por las siguientes variables modificadas:

- primero, se obtiene la suma de la población joven, adulta y mayor del año 2018;
- estas tres variables se dividen por la suma anterior para obtener ratios de los grupos de edad en función del total de la población.

Este es el vector de adscripción a utilizar para entrenar al algoritmo. Se ha hecho para los niveles de resolución entre 100 kilómetros y 250 metros. Las conclusiones son las siguientes:

- en los niveles de 100 a 5 kilómetros el algoritmo, con 3 clústeres, identificaba perfectamente un patrón consistente en los niveles de resolución citados (figuras **??**, **??**, III.8.1 y III.8.2):
    - un *clúster* para las celdas con una dependencia equilibrada entre la infantil y la anciana;

[4]Sin embargo, el experto debe poner a posteriori la interpretación estadística de los datos para validar la utilidad de los resultados y, por tanto, del modelo.

    - otro con un desequilibrio hacia la dependencia anciana severa;
    - otro con una dependencia infantil ligeramente superior a la anciana;

- sin embargo, al llegar al nivel de 1 kilómetro, los 3 *clústers* anteriores dejaron de ser significativos, puesto que el algoritmo ponía su atención en casos extremos de dependencia anciana y otros con una dependencia prácticamente nula (sólo adultos), quedando un enorme *clúster* donde acababan clasificados casi todos los demás casos, sin matices. El algoritmo, por tanto, estaba incurriendo en un problema de generalización. Al subir a 5 *clústers* se volvieron a obtener resultados significativos:
    - uno para el caso extremo una elevadísima dependencia anciana (sólo población mayor);
    - otro para casos de dependencia prácticamente nula (ni niños ni mayores). Estos son los dos casos extremos de la ejecución con tres *clústers*. Los casos extremos seguían reservándose grupos minoritarios
    - el *clúster* extremadamente generalista que se creaba cuando eran tres se expandía ahora para aportar detalle, generando:
        - uno para las dependencias un tanto bajas, pero equilibradas;
        - otro para dependencias en rangos normales, pero desbalanceada a la anciana;
        - otro para dependencias en el mismo rango, pero desbalanceada a la infantil.

Este patrón se ha repetido consistentemente en los niveles de 1 kilómetro hasta 250 metros.

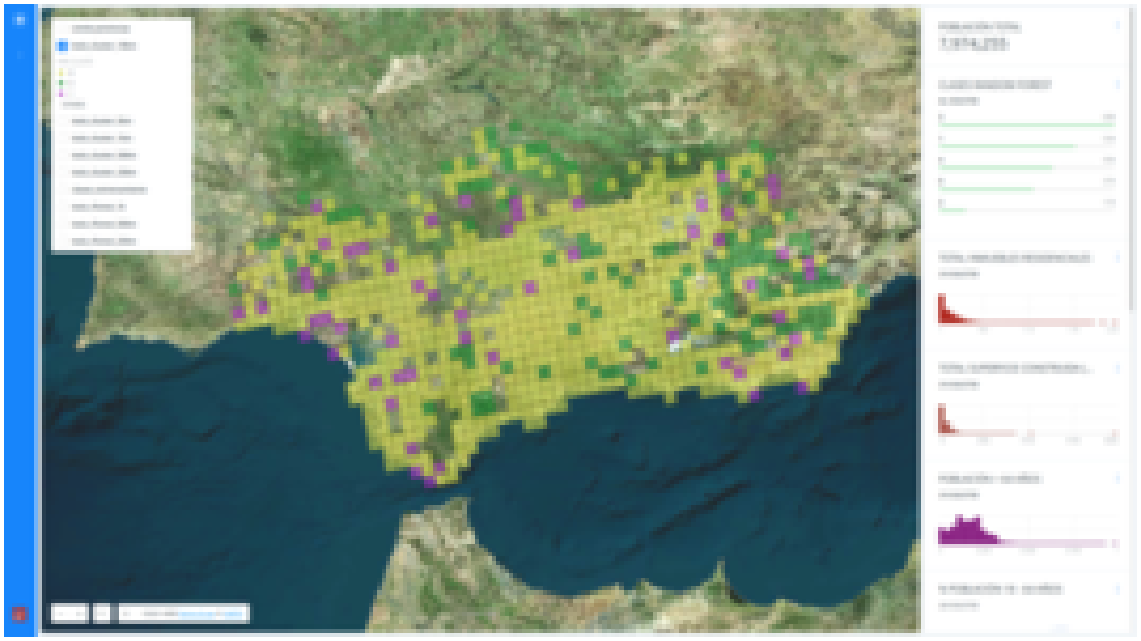

Figura III.8.1: K-Means sobre la estructura de la población a 10 km. Las teselas verdes representan un desequilibrio severo en la dependencia hacia la dependencia anciana, las amarillas tienen una dependencia equilibrada infantil / anciana y las moradas tienen un desequilibrio hacia la infantil. Fuente: elaboración propia.

Figura III.8.2: K-Means sobre la estructura de la población a 5 km. Las teselas amarillas tiene las dependencias infantil y anciana equilibradas, las moradas tienen un desequilibrio severo hacia la dependencia anciana, mientras que las verdes tienen un ligero desequilibrio hacia la infantil. Fuente: elaboración propia.

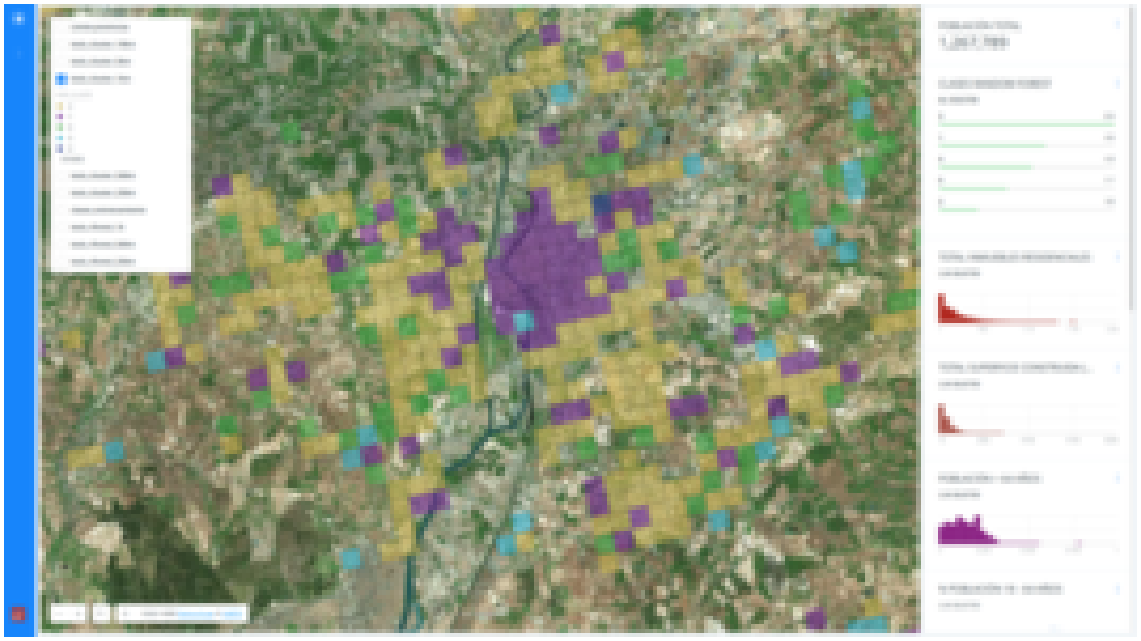

Figura III.8.3: K-Means sobre la estructura de la población a 1 km. A este nivel la clasificación de tres *clúster* sufría de sobregeneralización por lo que se pasó a 5. Detalle de la aglomeración urbana de Sevilla. Las teselas amarillas corresponden a un ligero desequilibrio hacia la dependencia infantil, las verdes a dependencias bajas pero equilibradas y las moradas a dependencias ligeramente desequilibradas a la anciana, las cyan son extremos de dependencia nula y las azul oscuro de extrema dependencia anciana. Fuente: elaboración propia.

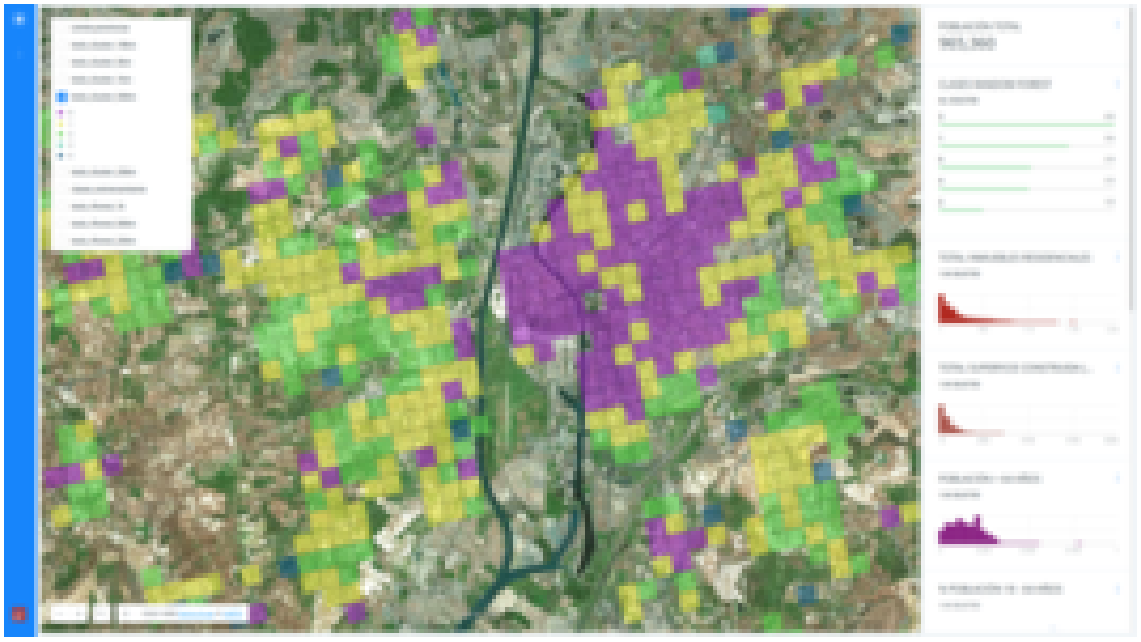

Figura III.8.4: K-Means sobre la estructura de la población a 500 m. Las teselas verdes corresponden a un ligero desequilibrio hacia la dependencia infantil, las amarillas a dependencias bajas pero equilibradas y las moradas a dependencias ligeramente desequilibradas a la anciana, las cyan son extremos de dependencia nula y las azul oscuro de extrema dependencia anciana. Fuente: elaboración propia.

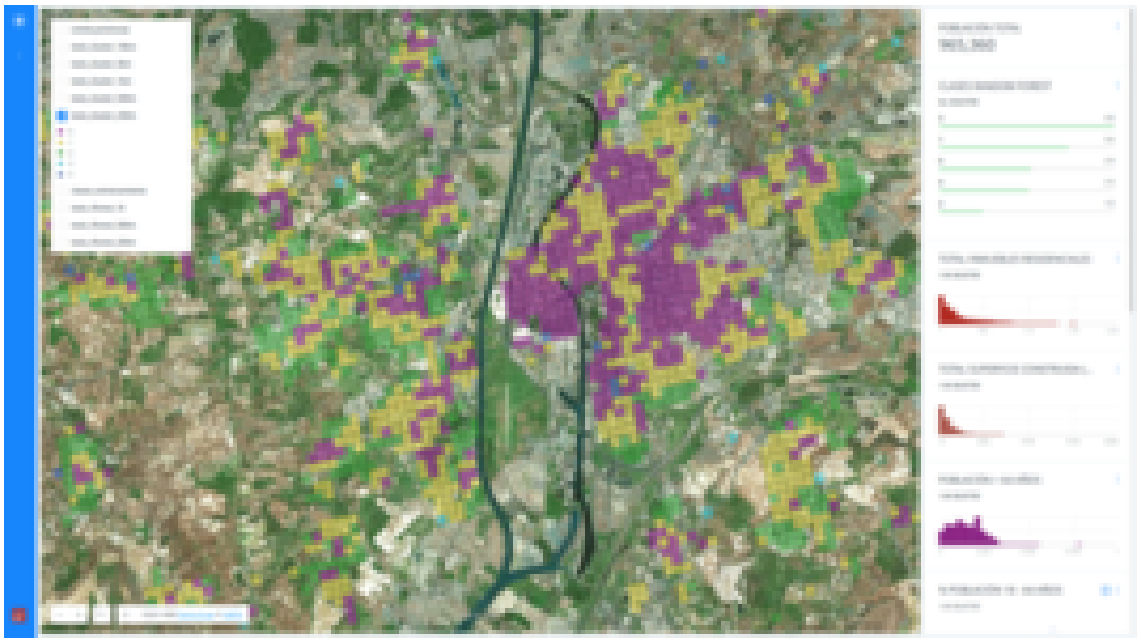

Figura III.8.5: K-Means sobre la estructura de la población a 250 m. Las teselas verdes corresponden a un ligero desequilibrio hacia la dependencia infantil, las amarillas a dependencias bajas pero equilibradas y las moradas a dependencias ligeramente desequilibradas a la anciana, las cyan son extremos de dependencia nula y las azul oscuro de extrema dependencia anciana. Fuente: elaboración propia.

Random Forest

El *Random forest*, en contraposición al *K-Means*, es un algoritmo de clasificación supervisado. Esto quiere decir que debe ser entrenado con información previamente seleccionada por un experto. Al igual que el *K-Means*, su objetivo es entrenar un modelo que sea capaz de clasificar nuevos vectores en clases aprendidas mediante sets de entrenamiento avalados por un criterio experto.

El *Random forest* es un algoritmo de tipo *ensemble* (conjuntado) porque la clasificación se produce por un voto mayoritario entre un grupo de sub-modelos entrenados. *Random forest*, internamente, usa un grupo de árboles de decisión que han sido entrenados con los mismos datos pero que abordan la clasificación desde puntos de vista distintos y, muy importante, no correlacionados. Cuando se presenta al modelo un nuevo vector multivariante para su clasificación, éste es clasificado por todos los árboles y la opinión mayoritaria de éstos provoca la clasificación final.

Para probar este algoritmo en el prototipo se han tomado las siguientes variables:

- las tres proporciones de población ya comentadas en el *K-Means*;
- dos variables catastrales: el total de usos residenciales y la superficie total construida.

Las zonas de entrenamiento seleccionadas se pueden ver en las figuras III.8.6, III.8.7, III.8.8, III.8.9:

- Casco Antiguo de Sevilla: tramado urbano muy denso y población ligeramente envejecida;
- Barrio de Triana y Los Remedios: tramado urbano también muy denso y población aún más envejecida;
- Los Bermejales: tramado urbano de densidad alta-media y población joven;
- nuevos barrios de urbanizaciones unifamiliares de Tomares: urbanismo de densidad media-baja y población joven;
- pueblo antiguo de Tomares: urbanismo más denso que el anterior, sin llegar a la densidad de una ciudad como las zonas de entrenamiento de Sevilla, y con población más envejecida que la anterior.

Al contrario que en el caso del *K-Means*, aquí no se puede entrenar al algoritmo en cualquier nivel de resolución, puesto que una tesela muy grande (100, 50 kilómetros) englobaría a todas las zonas de entrenamiento y el proceso no tendría sentido. Por lo tanto, se ha ejecutado el entrenamiento sobre los niveles de 1 kilómetro, 500 y 250 metros. Los resultados se comentan en las figuras III.8.10, III.8.11 y III.8.12.

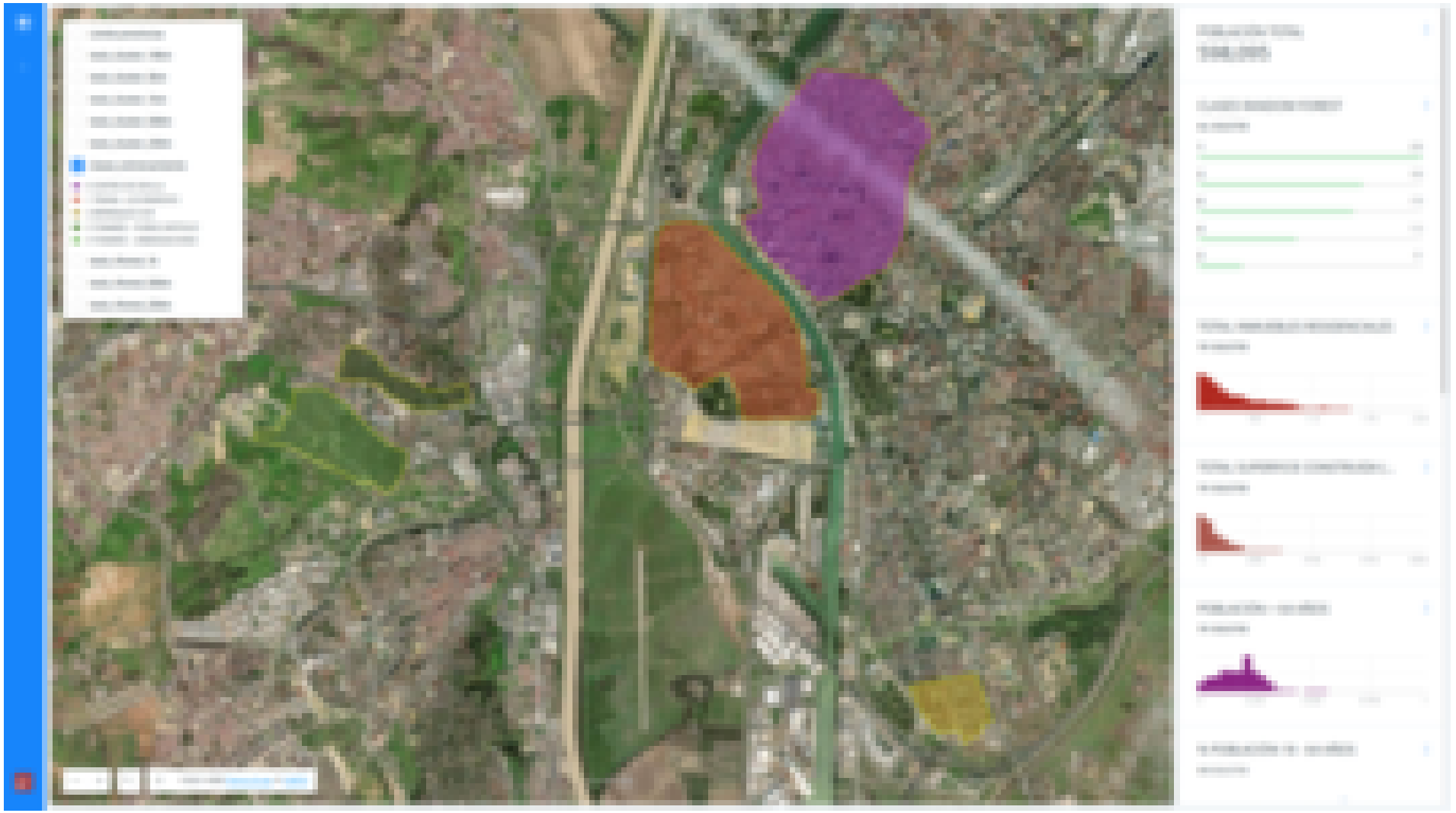

Figura III.8.6: Zonas de entrenamiento para *Random forest*: morado: Casco Antiguo de Sevilla, naranja: Barrio de Triana y Los Remedios, amarillo, al sur: Barrio de los Bermejales, al oeste, inferior: nuevos barrios de urbanizaciones de Tomares, al oeste, superior: pueblo antiguo de Tomares.

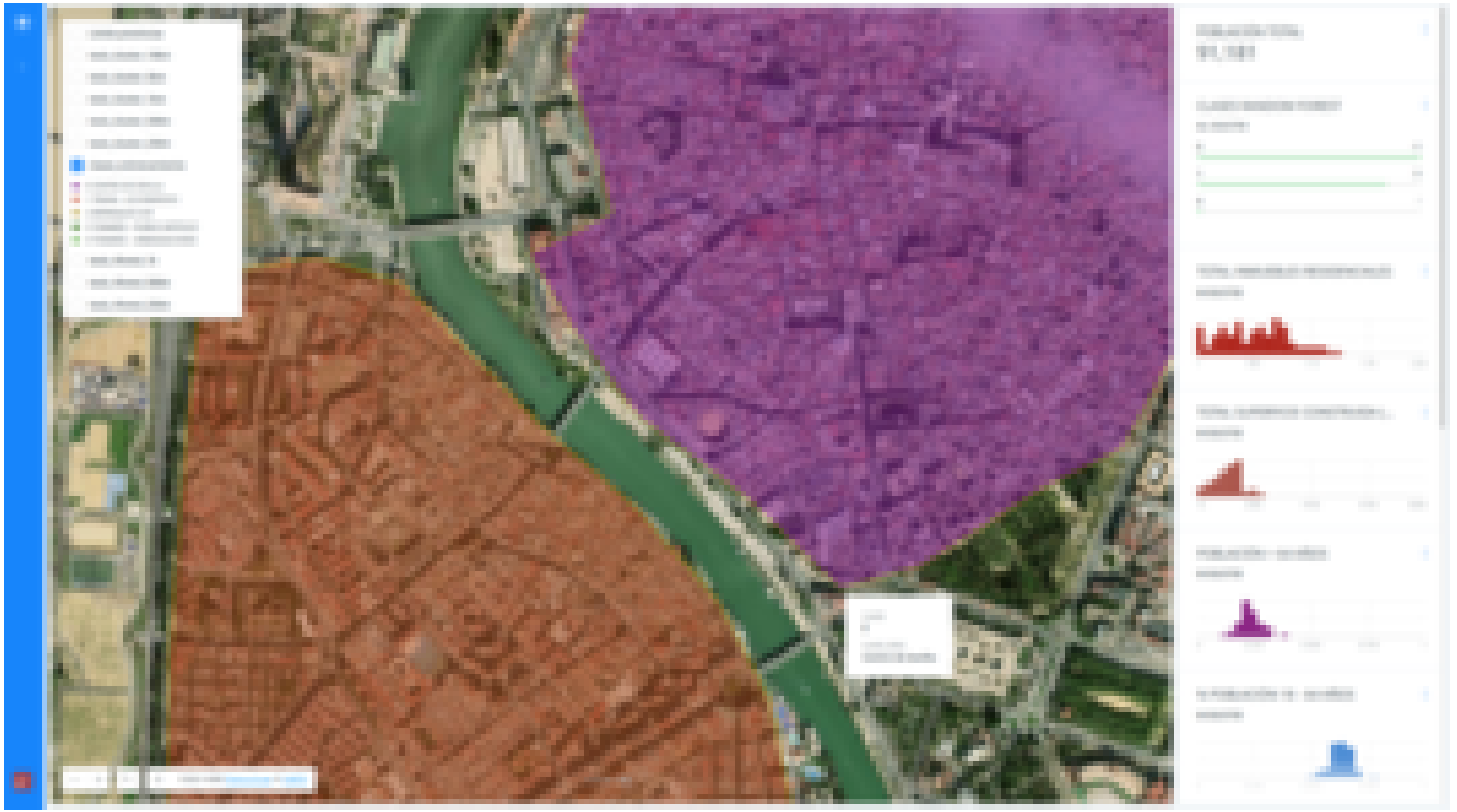

Figura III.8.7: Casco Antiguo y Triana y Los Remedios: estas zonas de entrenamiento se caracterizan por su alta densidad urbana y su población ligeramente envejecida.

Figura III.8.8: Los Bermejales: esta zona de entrenamiento se caracteriza por tener una población joven y una densidad urbana alta-media.

Figura III.8.9: Tomares, Sevilla: zonas de entrenamiento en nuevos barrios de viviendas unifamiliares, con urbanismo más extensivo y población joven, y el pueblo antiguo de Tomares, con un urbanismo más intensivo y una población más envejecida.

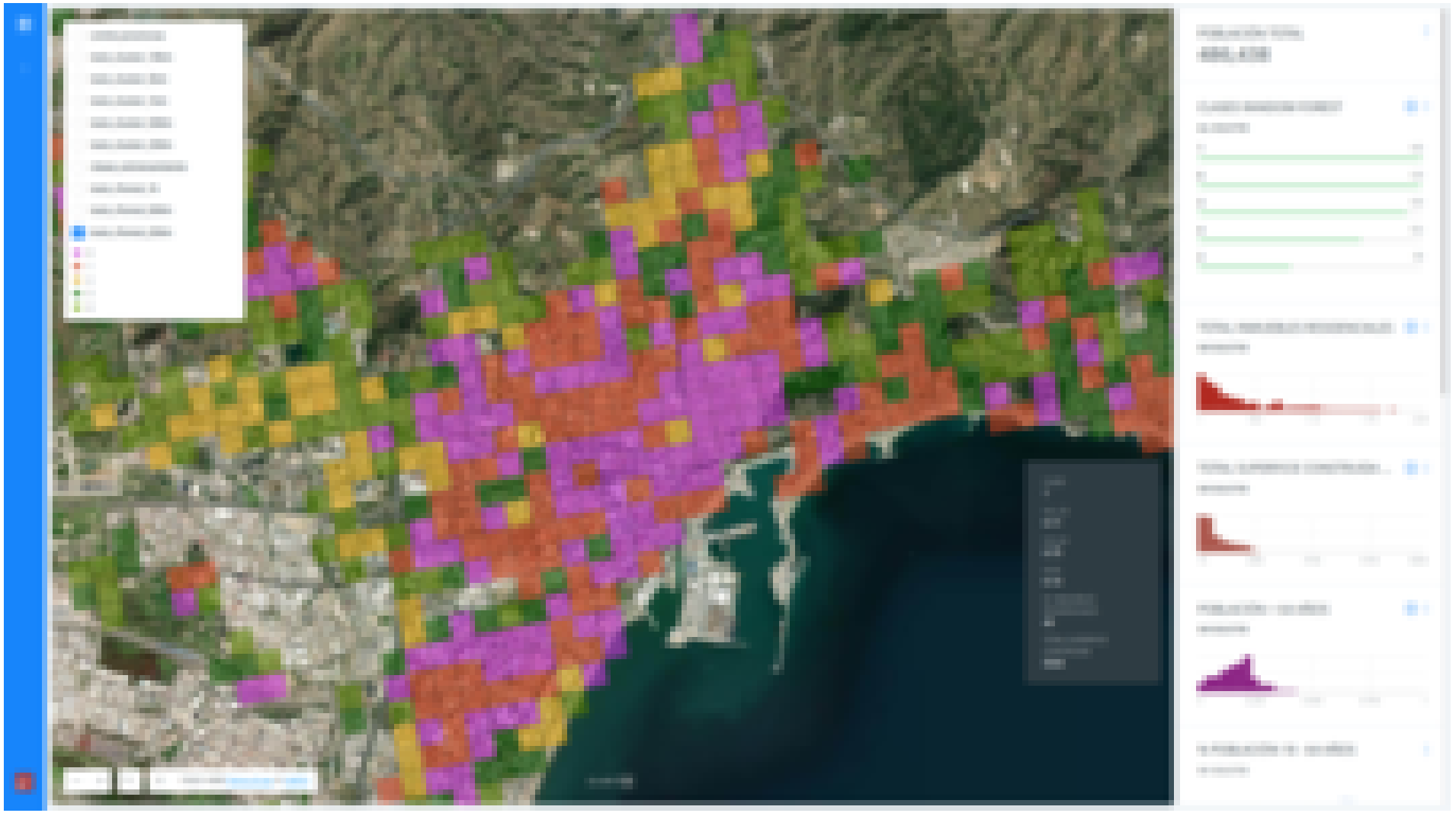

Figura III.8.10: El modelo *Random forest* aplicado a las teselas de 250 metros de Málaga. El morado corresponde a la clase “Centro de Sevilla”, el rojo a “Triana - Los Remedios”, el amarillo a “Bermejales”, el verde claro a “Tomares - Urbanizaciones” y el verde oscuro a “Tomares - Pueblo antiguo”.

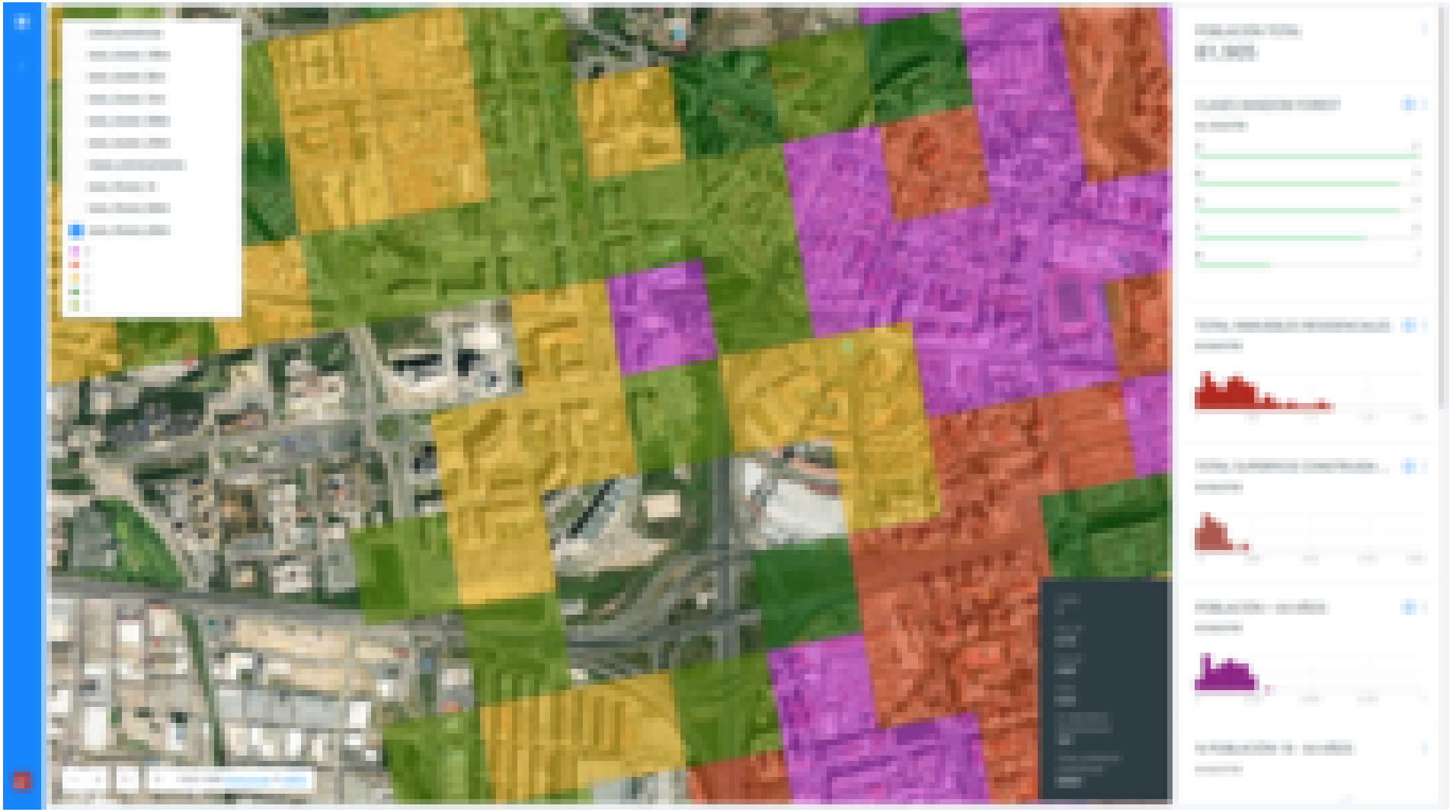

Figura III.8.11: Detalle de la aplicación del modelo *Random forest* a barrios de Málaga. Misma interpretación que en figura III.8.10.

Figura III.8.12: Modelo *Random forest* aplicado a Lora del Río, Sevilla. Misma interpretación que en figura III.8.10.

III.9 Visualización

Con la descripción de la librería *PyCell* terminamos la descripción del *backend*, por lo que en este capítulo describimos el *frontend*, que en el prototipo creado es un visor *web* cartográfico. Éste se va a comunicar con el *backend* a través de las entradas de la *API REST* descritas en la sección III.7.3 (pág. 250).

En este capítulo describimos la creación de un cliente *web* específico para trabajar con la plataforma, un cliente que comprende el estándar de comunicación que la *API* utiliza para la transmisión de la información de la rejilla definida matemáticamente y que, por tanto, es capaz de recrear en el navegador *web*, sobre una base cartográfica, la geometría de las teselas.

Las principales características de este cliente son las siguientes:

1. utiliza una librería, cuya arquitectura está calcada de la librería *libcellbackend*, llamada *libcellfrontend*, que reproduce la funcionalidad necesaria de la anterior para trabajar en el cliente, es decir, en el navegador *web*;

2. utiliza un componente de la plataforma *SaaS* de gestión de datos geográficos *Mapbox*, llamado *Mapbox GL*, que proporciona un visor *web* y cartografía base de altas prestaciones sobre la que construir el recreador de teselas;

3. está escrito con el *framework* de *Google* para la creación de aplicaciones *web* altamente dinámicas *Angular*;

4. utiliza la librería *Proj4js* para la transformación de sistemas de referencia en las teselas;

5. utiliza la librería *Turf.js*, también de la empresa *Mapbox*, para la realización de geoprocesos;

6. utiliza la librería de visualización *C3* y, por tanto, su librería base *D3*, para la visualización de histogramas de frecuencias de los dominios de las variables de adscripción coordinados con el mapa.

III.9.1. La librería *libcellfrontend*

Como ya se ha indicado, esta librería es la "versión recortada" de *libcellbackend*, en el sentido que reproduce parte de la funcionalidad de la misma. Aunque ambas librerías

están escritas en el mismo lenguaje, *TypeScript*, no se puede utilizar la librería del *backend* servidor directamente para el *frontend* cliente porque ésta posee muchísimas llamadas a librerías y funcionalidades que sólo existen en el entorno de ejecución de *JavaScript Node.js* y en ningún caso en el entorno de ejecución de *JavaScript* de un navegador *web*. Ejemplos de funcionalidades que no existen en este último entorno son los enlaces a la base de datos *PostgreSQL* o a la memoria compartida *Redis*, por poner sólo dos ejemplos muy evidentes.

Por lo tanto, *libcellfrontend* copia y adapta la funcionalidad de las siguientes clases del *backend*:

1. ***Coordinate*:** necesaria para la gestión de coordenadas;
2. ***ZoomLevel*:** gestión de los niveles de resolución de una rejilla;
3. ***Bbox*:** modelización de cajas de coordenadas;
4. ***Polygon*:** modelización de polígonos para la librería *Turf.js*;
5. ***Cell*:** modelización de una tesela;
6. ***Grid*:** modelización de la rejilla de adscripción;
7. ***Catalog*:** modelización de catálogos de variables de adscripción que lo usen;
8. ***Variable*:** modelo de las variables de adscripción;
9. ***GridderTask*:** modelo de los análisis de adscripción;
10. ***MlTask*:** modelo de los análisis de *Machine Learning* de la librería *PyCell*;
11. ***GridderJob*:** modelo de las sesiones de teselación;
12. ***MlJob*:** modelo de las sesiones de *Machine Learning*.

Hasta la clase *Cell*, la implementación es prácticamente la misma que en el *backend*. Sin embargo, el resto de las clases, aún conservando mucha funcionalidad de sus contrapartidas *backend*, substituyen toda la gestión de sus datos en la base de datos *PostgreSQL* por métodos que les permiten recuperarla desde las entradas *API* descritas en la página 250.

III.9.2. *Mapbox GL*

Mapbox GL es, en el ecosistema que la empresa *SaaS Mapbox* ofrece en su cartera de productos, el componente que permite a los desarrolladores incorporar a los clientes *web* un potente visor cartográfico. Aunque existen alternativas libres como el visor *Leaflet* que pueden alcanzar prestaciones equiparables en algunos campos, la elección de un componente no de *Software Libre* como *Mapbox GL* se basa en los siguientes puntos:

1. el margen de uso gratuíto que proporciona una cuenta de *Mapbox* es más que suficiente para trabajar un prototipo de forma rápida sin incurrir en gastos de licencia por uso. El llamado *Free Usage Tier*[1] de la licencia de *Mapbox* es más que generoso para este caso;

2. uno de los objetivos para la creación del visor era medir hasta que punto aguantaba el mismo suministrándole muchas teselas de rangos de resolución altos, como por ejemplo 1 kilómetro o 500 metros a nivel de toda Andalucía. Eso son muchas teselas visualizándose en pantalla. Los primeros prototipos del visor utilizaban el *DOM* del navegador[2] para representar, con *SVG* y la librería *D3*, las teselas. Esto no tenía un rendimiento demasiado alto, pudiéndose dibujar a la vez no más de 10.000 teselas. La idea era probar el siguiente salto tecnológico en los navegadores que permite dibujar incluso escenas en 3D. Esta tecnología se denomina *WebGL* (por eso el producto de *Mapbox* se llama *Mapbox GL*), una tecnología bastante compleja cuyo estudio y aplicación quedaba fuera de las posibilidades de este trabajo. Por lo tanto, se ha recurrido a una implementación comercial de la misma para probar el rendimiento. Es posible replicar estos resultados, con bastante esfuerzo, con unas dependencias 100 % *Software Libre*.

El componente *Mapbox GL* ofrece, como todos los componentes de *software*, una *API* para manipular su comportamiento y proporcionarle datos. Entre las funciones básicas de esta *API*, que es bastante amplia, encontramos varias áreas funcionales:

1. **movimientos en el mapa:** la interacción básica con un mapa, como son los desplazamientos, los zooms, etc.;

2. **carga de capas cartográficas:** otro de los productos de la plataforma *Mapbox* es *Mapbox Studio*, una aplicación 100 % *online* para el diseño de cartografía. La cartografía diseñada con este producto es directamente utilizable en un visor *Mapbox GL*. La propia empresa *Mapbox* cuenta con un departamento de cartógrafos que publican, cada mes, diversos estilos de mapas adaptados a aplicaciones concretas o influenciados por distintos estilos artísticos. Estos mapas son utilizables también de forma gratuíta;

3. **dibujado de nuevas geometrías sobre el mapa en tiempo real mediante tecnología *WebGL*:** esta es la parte más interesante para el prototipo de visor, ya

[1] Aquí vendría una extensa discusión sobre modelos de negocio de plataformas *SaaS* (*Software as a Service*) que no viene al caso, ya que cada empresa tiene el suyo. Sin embargo, dado que el patrón de licencia predominante en estos servicios es un modelo *pay as you go* (pago por uso), es decir, cuanto más uso de la plataforma, más pago a fin de mes, una característica común a muchas de ellas es permitir un uso gratuíto dentro de unos límites modestos de uso. Por ejemplo, *Mapbox*, con su *Mapbox GL*, permite 50.000 visualizaciones de mapa (es decir, 50.000 clientes distintos que utilicen un mapa *Mapbox GL* incrustado en una página *web*) antes de comenzar a facturar por uso. Un límite más que interesante para experimentar con la plataforma sin incurrir en gastos. El *Free Tier* tiene como finalidad fomentar el prototipado, el uso y la prueba de la tecnología antes de lanzar un proyecto con muchos miles de potenciales usuarios.

[2] El *DOM* del navegador es una representación en memoria de todos los contenidos de una página *web*, que son bastantes más de los que son visibles a primera vista. Gracias a esta representación, en forma de árbol, de los elementos de una página *web* en memoria el navegador puede manipular las características de cada objeto en tiempo real, en respuesta, por ejemplo, a las acciones de un usuario sobre la página. Es muy cómodo y eficiente siempre que la página no tenga mucha información.

que permite cargar sobre el mapa, aplicando una semiología cartográfica determinada, una gran cantidad de geometrías vectoriales.

El visor recibe, desde la *API*, las teselas con sus vectores de adscripción sin geometría alguna, sólo con sus coordenadas [R,X,Y]. El visor recrea las geometrías de las teselas con el siguiente procesamiento:

1. la librería *libcellfrontend* lee la definición de la rejilla en la que están expresadas las rejillas y recrea la geometría, que puede ser el propio polígono de la tesela o su centroide;
2. la librería *Proj4js* reproyecta la tesela al sistema de coordenadas *WGS84* geográficas, identificado con el código *EPSG* 4326, que es el sistema en el que tratan los datos casi todos los visores *web*, incluido *Mapbox GL*;
3. *Mapbox GL* utiliza su *API* de semiología cartográfica para darle una representación de escala de color y/o tamaño al símbolo elegido de la tesela, representándolo sobre el mapa.

III.9.3. Histograma de frecuencias de las variables de adscripción con *C3 / D3*

La librería *D3* es sin duda una de las librerías más potentes para crear visualizaciones de datos en un entorno *web*. El nombre de la librería, *Data Driven Documents* (documentos orientados a datos) da una idea muy precisa de la filosofía de la librería en su conjunto: los datos determinan la visualización y, si estos cambian, la visualización muta. De esta manera, la filosofía de trabajo de *D3* es crear una serie de reglas visuales que se ajustan a una estructura de datos *JSON* como las que se han visto en este trabajo. Una vez dichas reglas visuales, que controlan forma, color y posición, entre otros muchos parámetros, están implementadas, éstas se aplican consistentemente a los datos que se les vayan proporcionando, generando, en el proceso de suministro de un nuevo paquete de datos, precisas transiciones entre el antiguo estado de los datos y el nuevo.

El poder de la flexibilidad de *D3* reside en su concepción como librería de bajo nivel[3], lo que le confiere una extraordinaria flexibilidad al afrontar diseños novedosos de visualizaciones, al coste de una complejidad no trivial.

[3] *Bajo nivel*, en ingeniería de *software*, no es un término peyorativo, aunque lo parezca. Por *bajo* y *alto* nivel, al hablar de un sistema informático, nos estamos refiriendo a cuánto de pegado está el sistema a la máquina que lo ejecuta (bajo nivel) o si está dotado de una buena cantidad de capas de abstracción para acercarlo, en facilidad de comprensión, a un usuario humano. Se usa mucho, por ejemplo, al referirse a lenguajes de programación. *C* y *C++* son considerados lenguajes de programación de *bajo nivel* porque están muy pegados a la máquina: tienen un acceso y un control privilegiado de los recursos físicos de la máquina (la memoria, por ejemplo). Esto hace que sean lenguajes extraordinariamente potentes, ya que un programador competente puede manipular la memoria del ordenador, por ejemplo, con una sutileza no disponible en otros lenguajes. Esto viene acompañado de contrapartidas, por supuesto: son lenguajes muy complejos y pueden potencialmente llevar a la máquina al desastre por esa misma capacidad de acceder a sistemas delicados de la misma. Son lenguajes que generan una gran cantidad de código para cualquier tarea, por nímia que parezca, ya que entran en tanto detalle, tanto control de lo que sucede en la máquina, que hay que escribir mucho, generalmente, para hacer cosas sencillas. Por supuesto, el estar tan pegados

Sin embargo, obviamente la visualización de datos estadísticos recurre a una serie de gráficos más o menos estándar de probada eficacia y que se utilizan en muchos contextos distintos. Uno de estos gráficos son los histogramas de frecuencias. Para estos casos de gráficos de uso común hay librerías de componentes que, utilizando *D3* como base, desarrollan los mismos, proporcionando componentes gráficos fáciles de usar, de calidad y altamente interactivos. Una de estas librerías en *C3*.

El histograma que acompaña al mapa funciona de forma coordinada con las teselas visualizadas en el mismo. Dado que cada desplazamiento en el mapa genera una nueva petición de información de teselas al servidor, con objeto de actualizar la zona mostrada actualmente, los vectores de adscripción de las mismas son pasados al componente del histograma, que se actualiza en consecuencia, sincronizando de esta manera visualización cartográfica y de datos.

III.9.4. Resultados con *Mapbox GL*

El visor Mapbox creado se ha centrado en la exploración y prototipado de la capacidad de la plataforma de orientarse por entero a los datos, sin tomar, por parte del *backend*, ninguna predeterminación en cuanto a la visualización de la información. Inspirado por la filosofía de la librería *D3*, según la cual las visualizaciones deben basarse en reglas que son aplicadas a los datos, pero poniendo siempre el foco en éstos últimos, este prototipo de visualizador recrea en el cliente la vertiente geométrica de la información teselada en función de la definición matemática de la rejilla y los datos de los vectores de adscripción de las propias teselas.

El uso del visor *Mapbox GL* permite un acceso fácil a la tecnología *Web GL*. Los primeros prototipos con *DOM + SVG + D3* con *HTML5* tenían un límite de dibujado de 10000 teselas, a partir de las cuales el rendimiento bajaba muchísimo, hasta el punto de hacer el visor inoperable. La manipulación de información en el *DOM* tiene muchísimas ventajas, especialmente si se combina con *D3*, pero el objetivo era mostrar más teselas en pantalla. Por ello, se consideró y se hicieron algunas pruebas con tecnología *HTML5 Canvas*, pero las aplicaciones de Tecnologías de la Información Geográfica no estaban muy desarrolladas en ese ámbito instrumental y el esfuerzo de poner en pié desde cero un sistema de esa magnitud ponían en peligro otras partes más importantes de este trabajo. Por lo tanto, se decidió utilizar *Mapbox GL*, que permitía una acceso rápido y fácil, ya

a la máquina los convierte en lenguajes extraordinariamente rápidos. En el otro extremo encontramos a los lenguajes de *alto nivel*. Un ejemplo claro es *Python*: está diseñado para ser altamente expresivo, es decir, que con pocas líneas de código se puedan hacer cosas de cierta complejidad. *Python* consigue esto poniendo, entre la máquina y el programador, una enorme cantidad de capas de abstracción, una especie de *pilotos automáticos*, que descargan del programador la responsabilidad de tener que gestionar de forma precisa los recursos de la máquina. Un programador de *Python*, por ejemplo, no tiene que preocuparse de la gestión de la memoria de su programa, ya que esta se realiza automáticamente por un subsistema del lenguaje. La contrapartida, obviamente, es la delegación en el lenguaje de muchas tareas, por lo que se pierde control y el lenguaje, en definitiva, permite hacer menos cosas, pero más fácilmente. Decir que permite hacer menos cosas es una forma de hablar, claro. Permite hacer prácticamente de todo, lo que pasa es que con *C* o *C++* se puede hacer practicamente *cualquier cosa* de lo que sea capaz la circuitería de la máquina.

que su *API* es de una gran calidad, a un motor de dibujado que podía abordar una gran carga de trabajo.

Mapbox GL llega a manejar con soltura, gracias al uso de las tarjetas gráficas, 50.000 teselas simultáneas, lo que constituye más de la mitad de todas las teselas de 1 kilómetro para todo el área de estudio. Esto quiere decir que todo el área de estudio puede visualizarse sin problemas a 5 kilómetros de resolución, entrando rápidamente a niveles de más resolución con áreas de visulización más acotadas. El rendimiento y la *API* del componente son realmente sobresalientes, y además es un producto con una muy buena política de precios.

Mostramos a continuación unas figuras en las que se muestra el visor *Mapbox*. Remitimos al lector, no obstante, a consultar los geovisores en `https://cell.37north.io`.

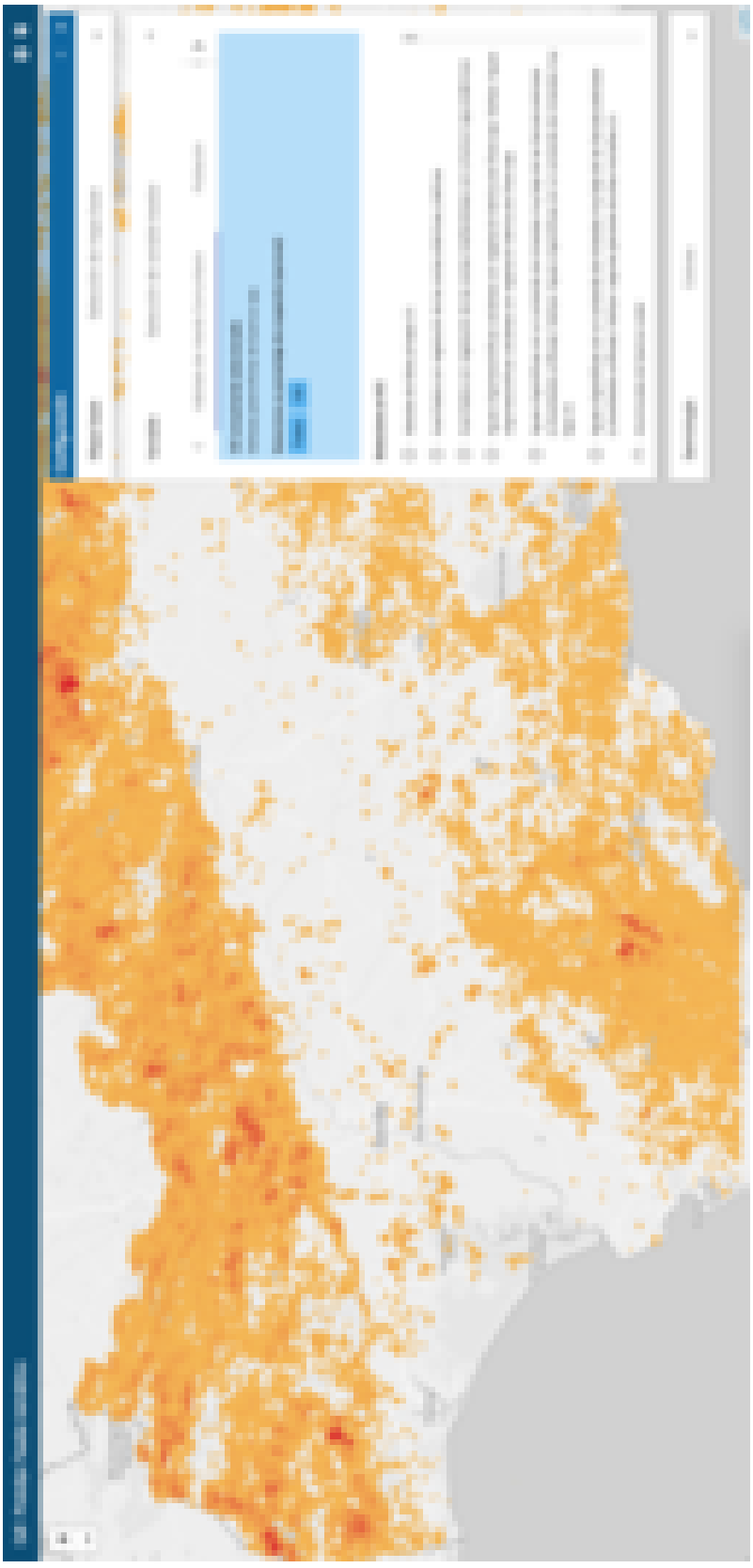

Figura III.9.1: Visor *Mapbox* mostrando una selección de HICS a 1 km de resolución. Fuente: elaboración propia.

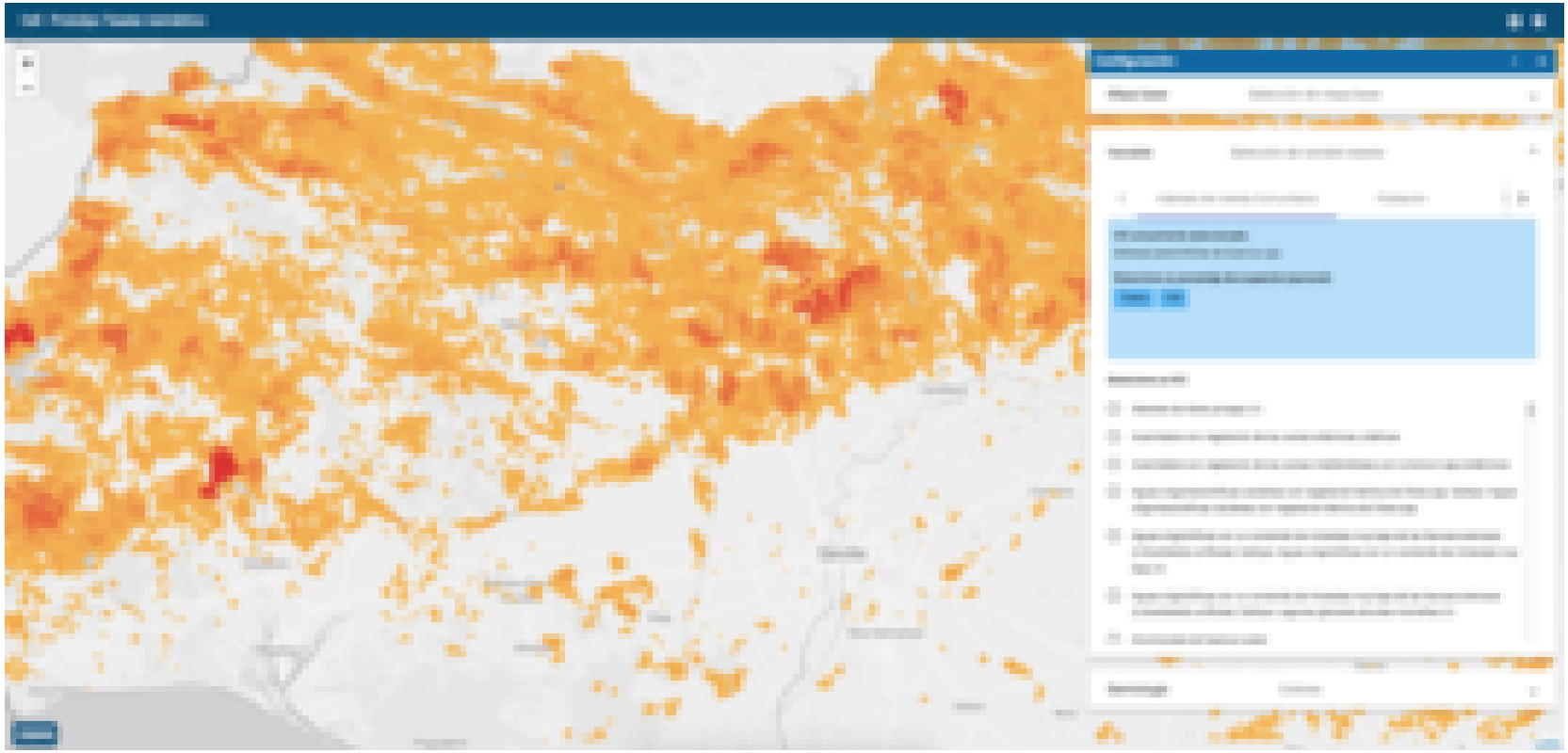

Figura III.9.2: Visor *Mapbox* mostrando una selección de HICS a 1 km de resolución. Fuente: elaboración propia.

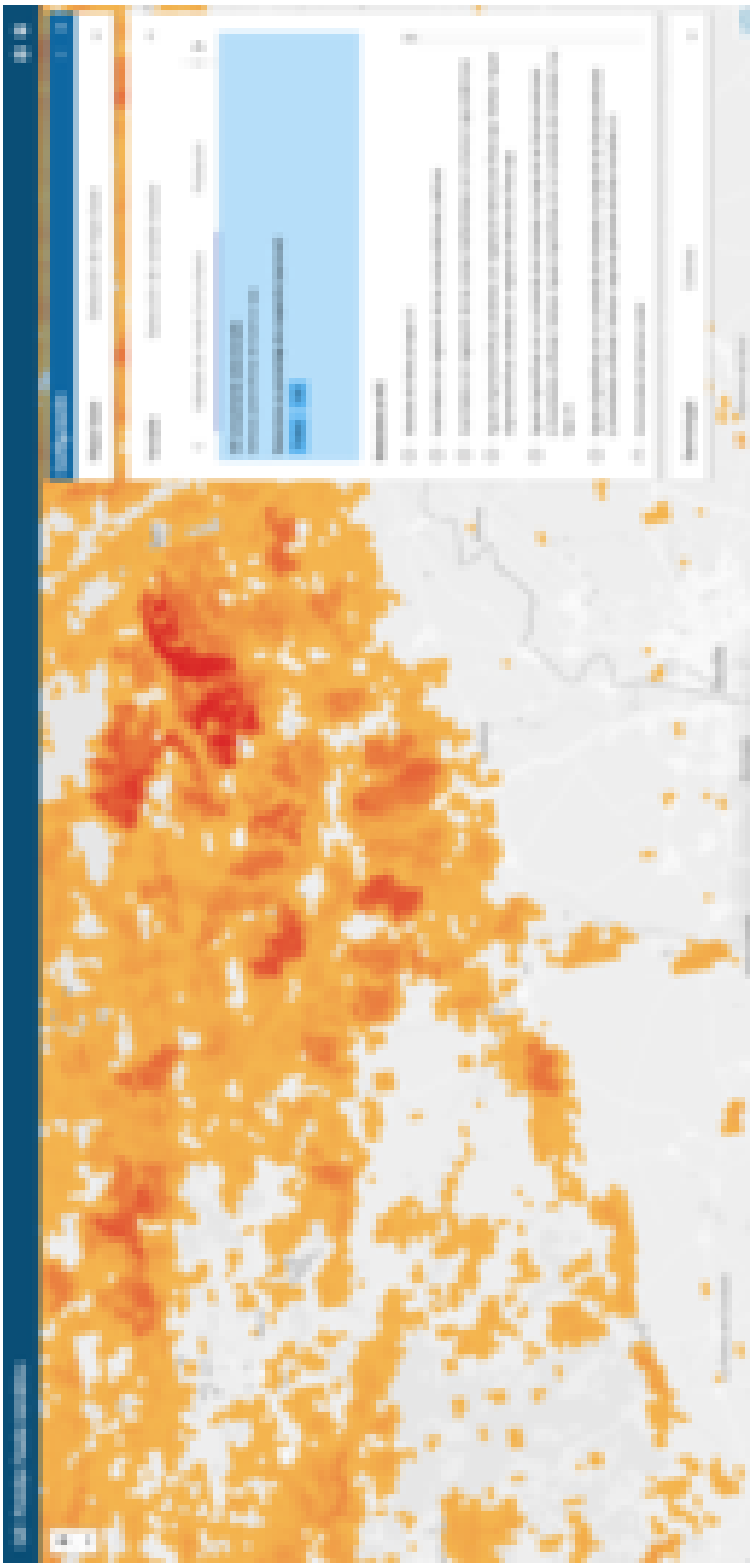

Figura III.9.3: Visor *Mapbox* mostrando una selección de HICS a 1 km de resolución, vista de detalle. Fuente: elaboración propia.

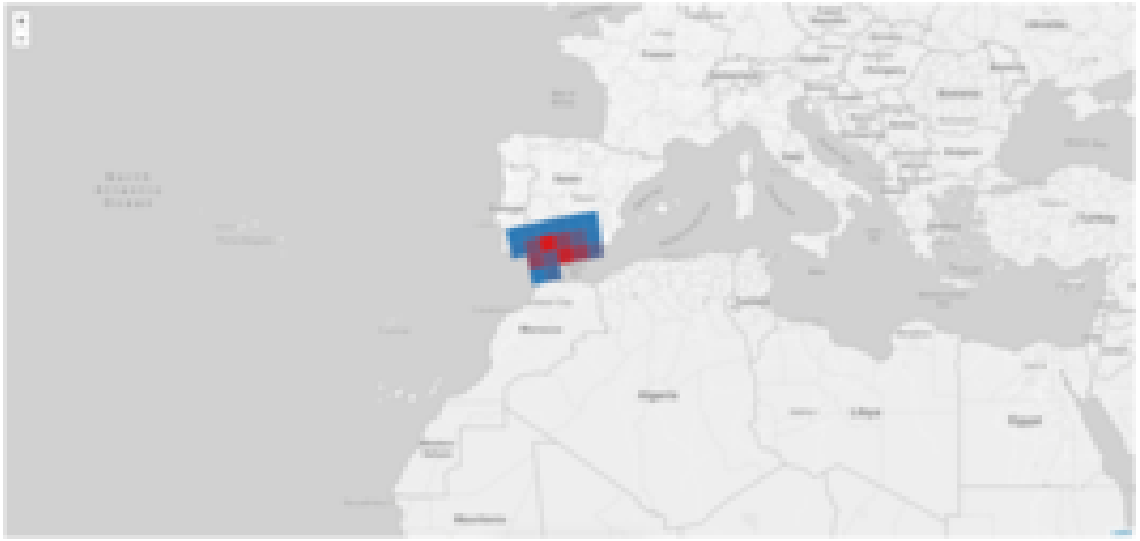

Figura III.9.4: Una de las características de la combinación *Mapbox + D3* es su filosofía de orientación a datos. *D3* recibe los datos de los vectores de adscripción de las teselas y, mientras *Mapbox* se encarga de dibujarlas, *D3* está, con cada petición al servidor, reajustando la escala de colores a los nuevos datos recibidos, permitiendo una aproximación contextual, reactiva y adaptativa de la semiología cartográfica. Los datos mostrados son de población. Fuente: elaboración propia.

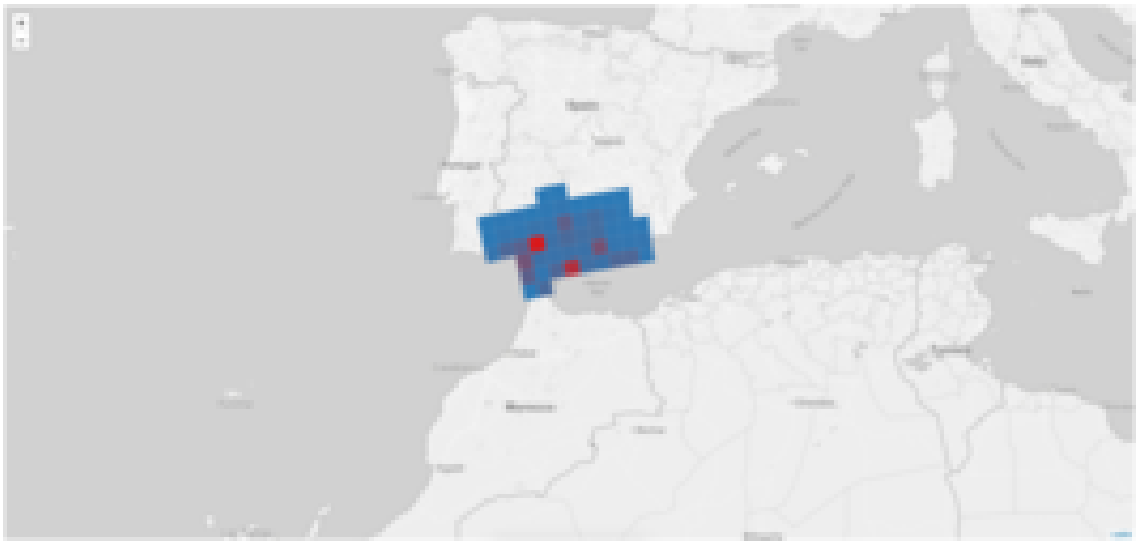

Figura III.9.5: Semiología adaptativa al rango de datos de población con *Mapbox + D3*. Fuente: elaboración propia.

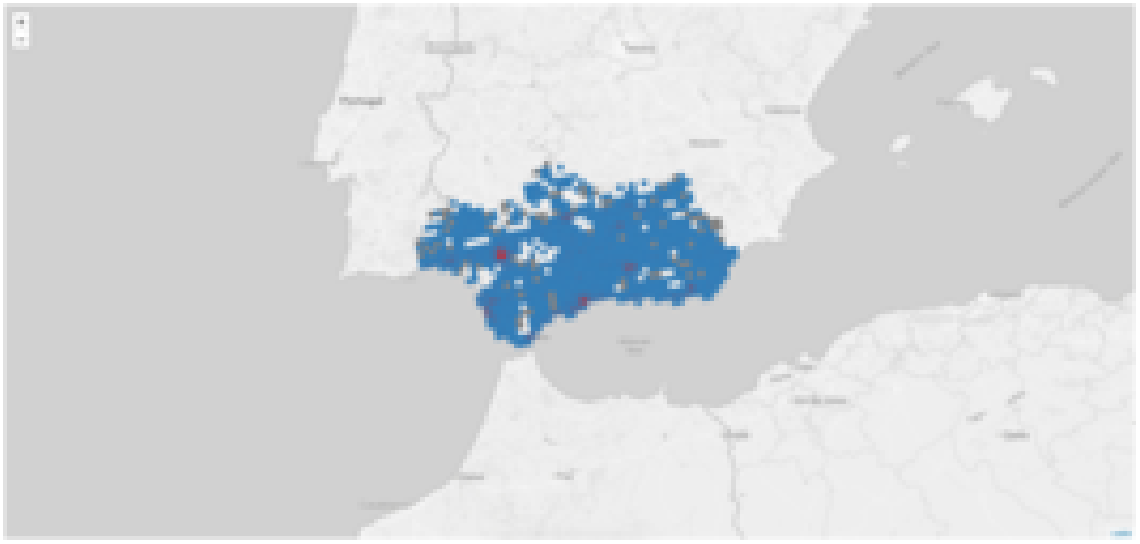

Figura III.9.6: Semiología adaptativa al rango de datos de población con *Mapbox + D3*. Fuente: elaboración propia.

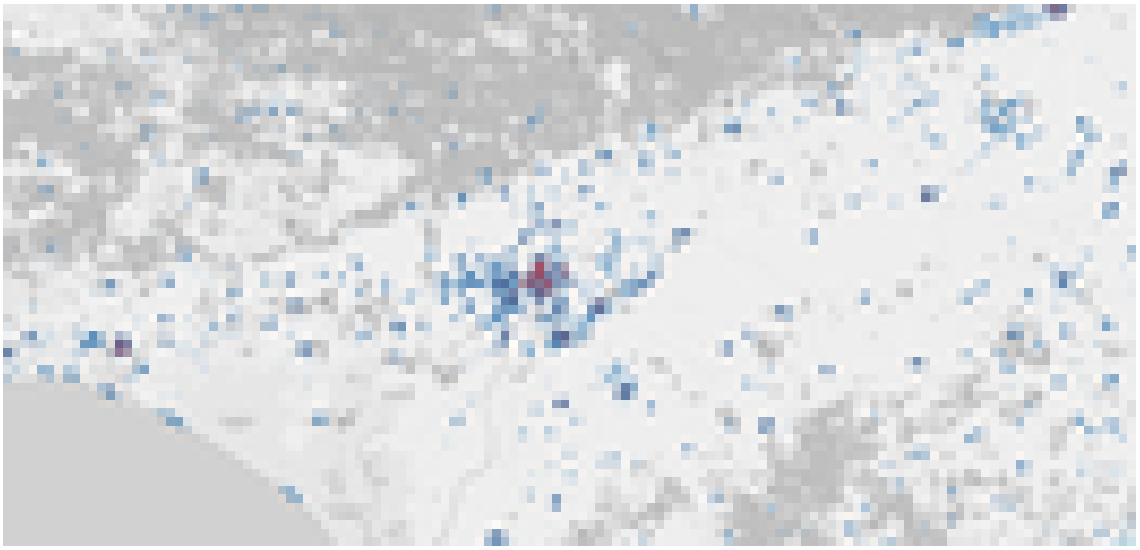

Figura III.9.7: Semiología adaptativa al rango de datos de población con *Mapbox + D3*. Fuente: elaboración propia.

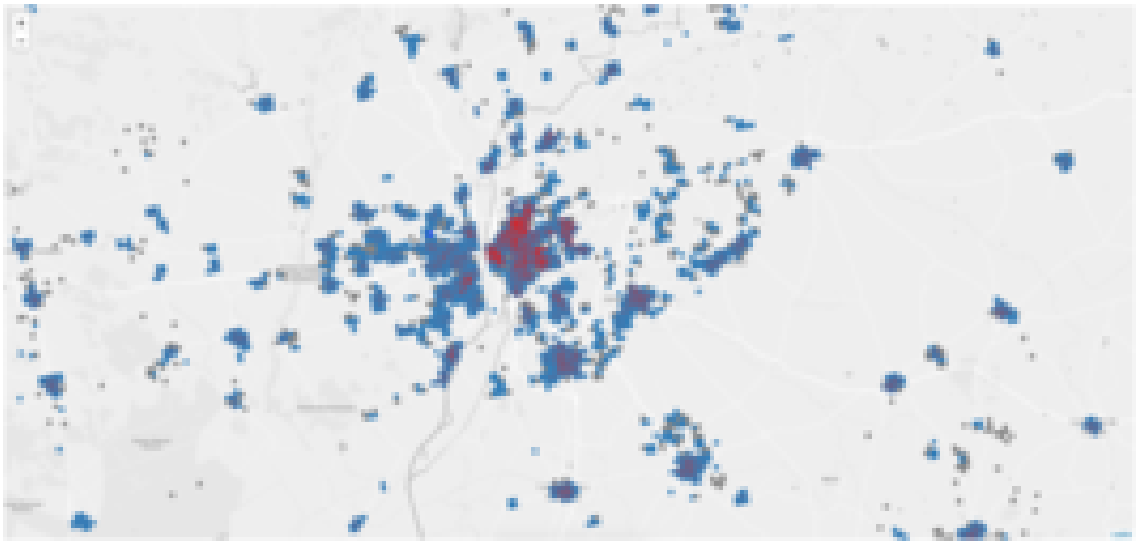

Figura III.9.8: Semiología adaptativa al rango de datos de población con *Mapbox + D3*. Fuente: elaboración propia.

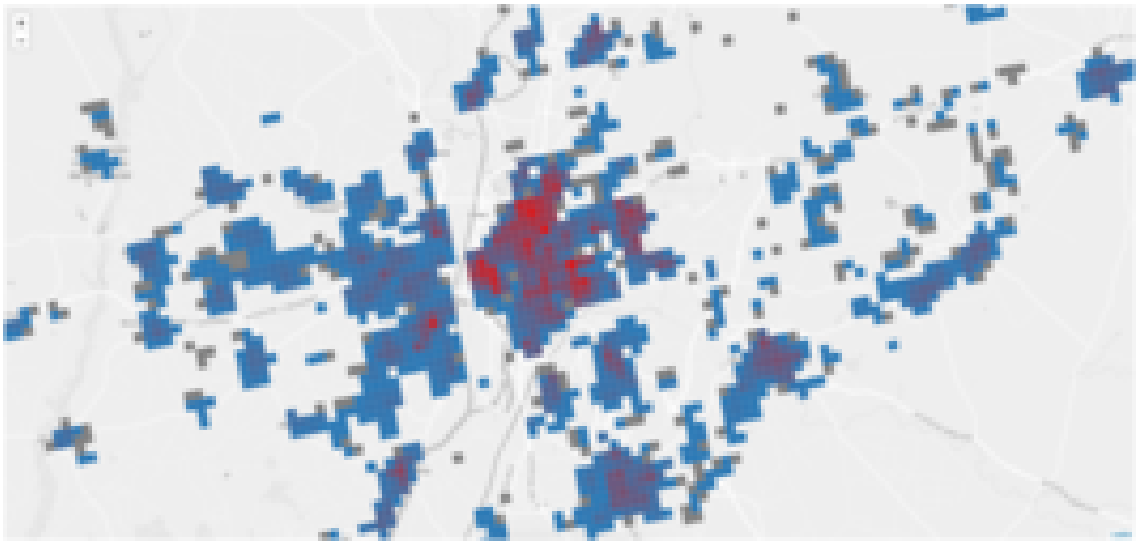

Figura III.9.9: Semiología adaptativa al rango de datos de población con *Mapbox + D3*. Fuente: elaboración propia.

Figura III.9.10: Semiología adaptativa al rango de datos de población con *Mapbox + D3*. Fuente: elaboración propia.

III.9.5. Resultados con *CARTO*

CARTO es una plataforma de *Location Intelligence* especializada en la visualización y exploración de datos mediante técnicas mixtas de cartografía y *widgets*. En una visualización de *CARTO* se sincroniza la visualización de los datos cartográficos con dispositivos de visualización estadística (*widgets*), como pueden ser histogramas, en los que todos los sistemas implicados sirven para fijar el contexto de datos a explorar. Así, por ejemplo, un zoom de detalle en el mapa provoca una visualización de información más limitada, hecho que recogen los *widgets*, adaptándose a las distribuciones de las variables visibles en ese momento en la extensión de mapa. Y al contrario: una manipulación en un *widget*, por ejemplo en un histograma, seleccionando un rango dentro del mismo, filtra dichos valores de la distribución en la representación cartográfica actual. Este juego de sincronías se están generalizando en el sector del *Web GIS*[4], y *CARTO* lo implementa de una forma especialmente fácil de usar, haciendo esta funcionalidad muy accesible a un amplio espectro de científicos que no tienen que estar directamente relacionados con el mundo *IT*, como sí han de estarlo los usuarios de *Mapbox*. Así como este último es un producto para desarrolladores, *CARTO* es, además, un producto para científicos especialistas temáticos.

A pesar de ello, *CARTO* ofrece, para los usuarios más vinculados a las *IT*, una profundidad técnica realmente interesante. El hecho de que esto sea así radica en que el núcleo de la operativa de *CARTO* es un despliegue masivo de *PostgreSQL / PostGIS*, es decir, una base de datos relacional geográfica. Aparte del cómodo constructor de visualizaciones interactivas sobre datos, llamado comercialmente *Builder*, *CARTO* habilita al usuario con un corte más especialista el acceso directo a un instancia *PostgreSQL* en la nube, extraordinariamente bien dimensionada, con la que se puede operar directamente en *SQL* espacial. Esto lo hace una opción muy interesante para muchos casos de uso distintos, a la par que satisface las necesidades de muchos perfiles distintos de usuarios.

Dado que el núcleo operativo de esta plataforma es *PostgreSQL*, que es precisamente también el de la plataforma *Cell*, se podría esperar que pudiera tenderse un vínculo de información entre ambas, y así es. *CARTO* permite conectar a una cuenta de usuario una base de datos externa de la que leer datos para ser usados en el *Builder* anteriormente comentado. Esta capacidad ha sido utilizada en este trabajo para darle una dimensión extra a la visualización y publicación de los datos integrados en rejilla, muy difícil de conseguir por medios de desarrollo propios en el marco de este trabajo. A continuación mostramos algunas figuras de los visores creados con esta herramienta, remitiendo, una vez más, al lector a consultar los geovisores en `https://cell.37north.io`.

[4]Por ejemplo, *ArcGIS Online* también tiene notables capacidades en este aspecto.

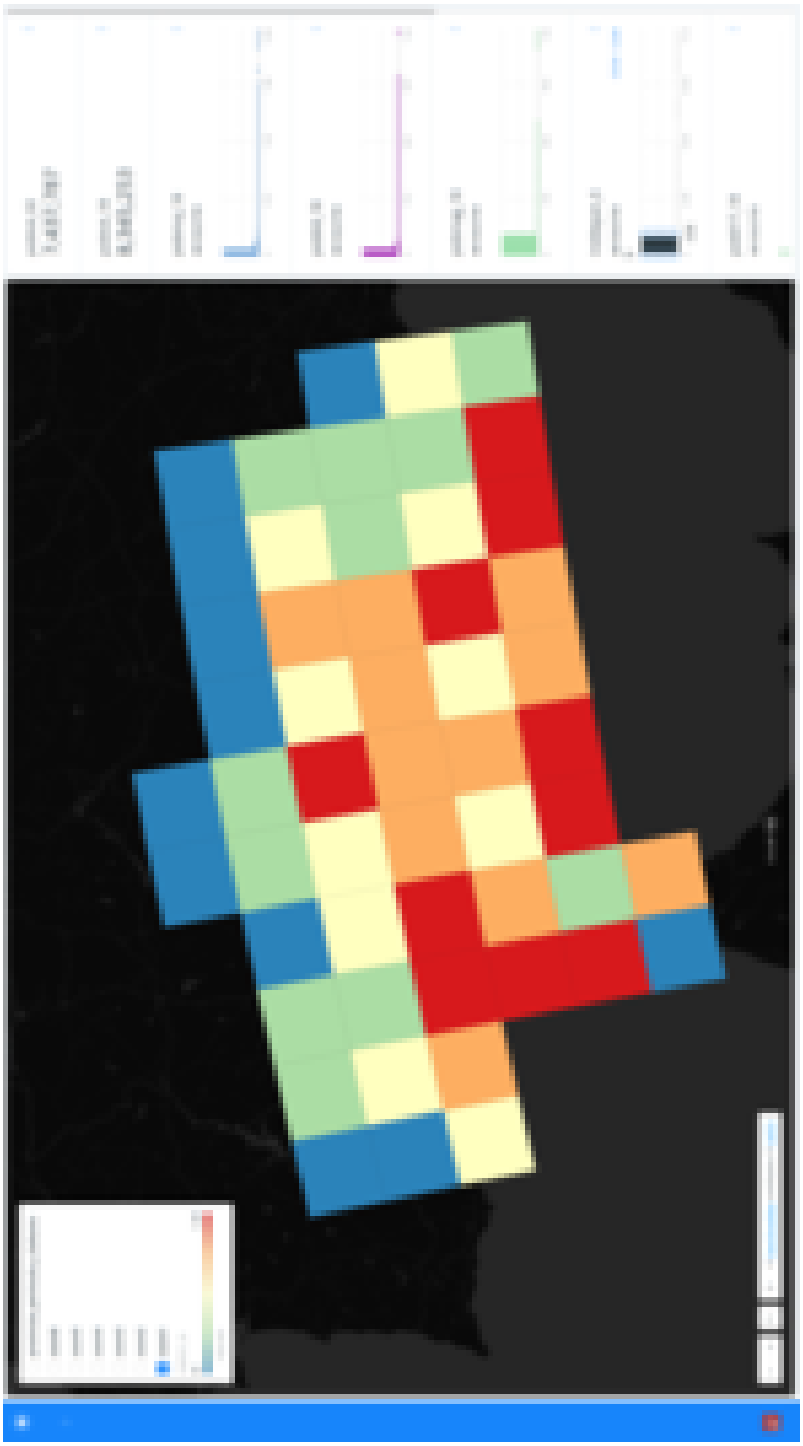

Figura III.9.11: *Dashboard* interactivo con datos de población sobre la resolución de 50 kilómetros. Fuente: elaboración propia.

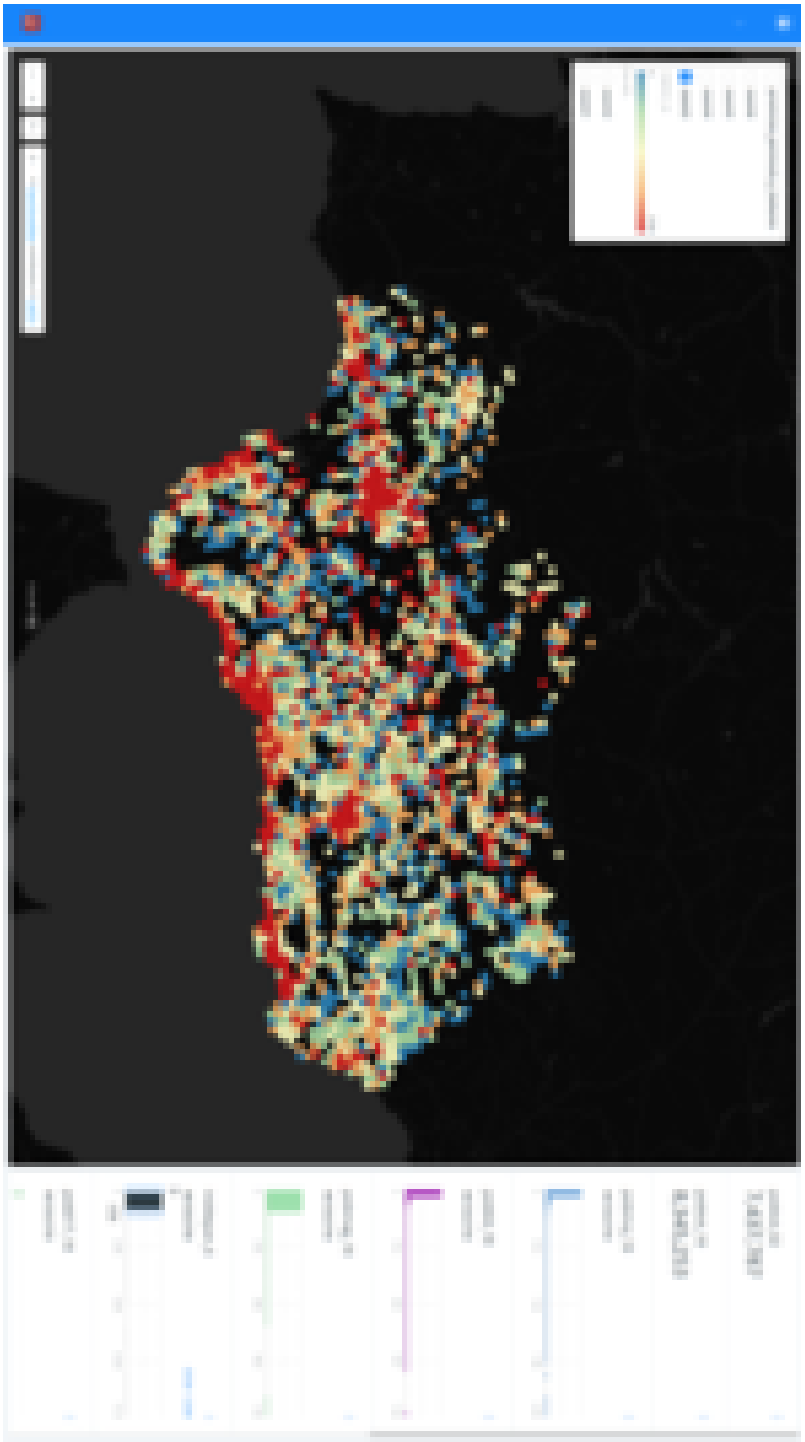

Figura III.9.12: *Dashboard* interactivo con datos de población sobre la resolución de 5 kilómetros. Fuente: elaboración propia.

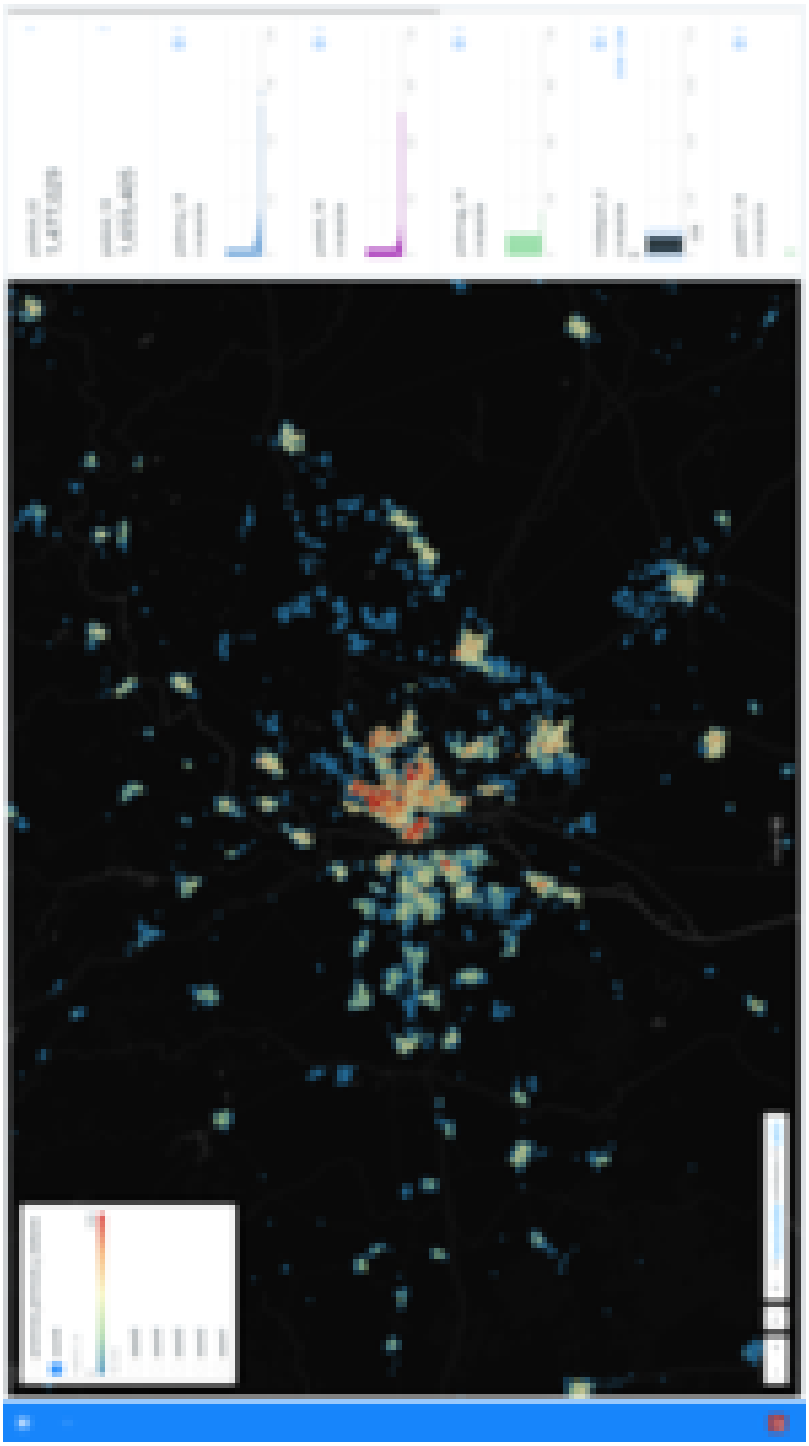

Figura III.9.13: *Dashboard* interactivo con datos de población sobre la resolución de 250 metros. Fuente: elaboración propia.

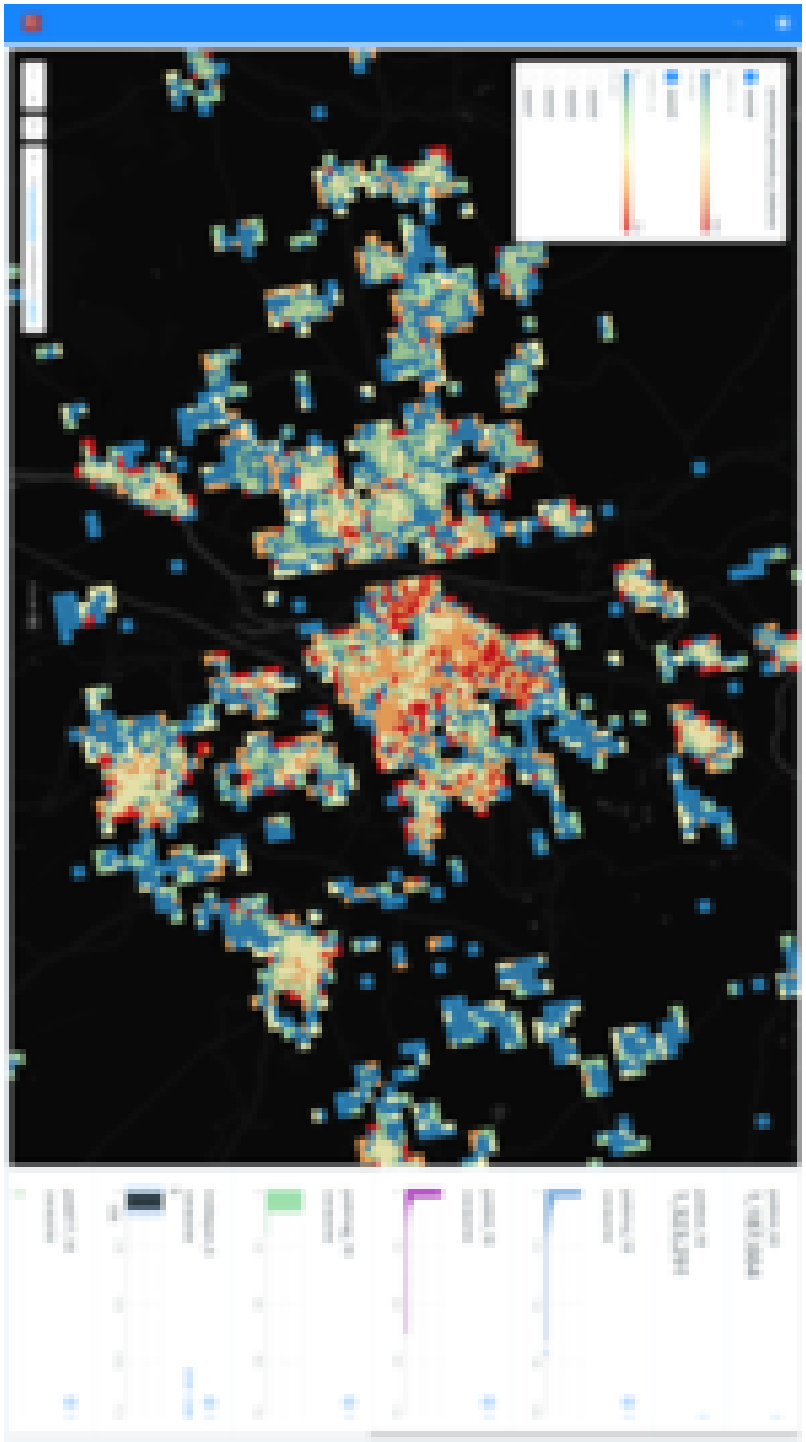

Figura III.9.14: *Dashboard* interactivo con datos de población sobre las resoluciones de 500 y 250 metros simultáneamente. Fuente: elaboración propia.

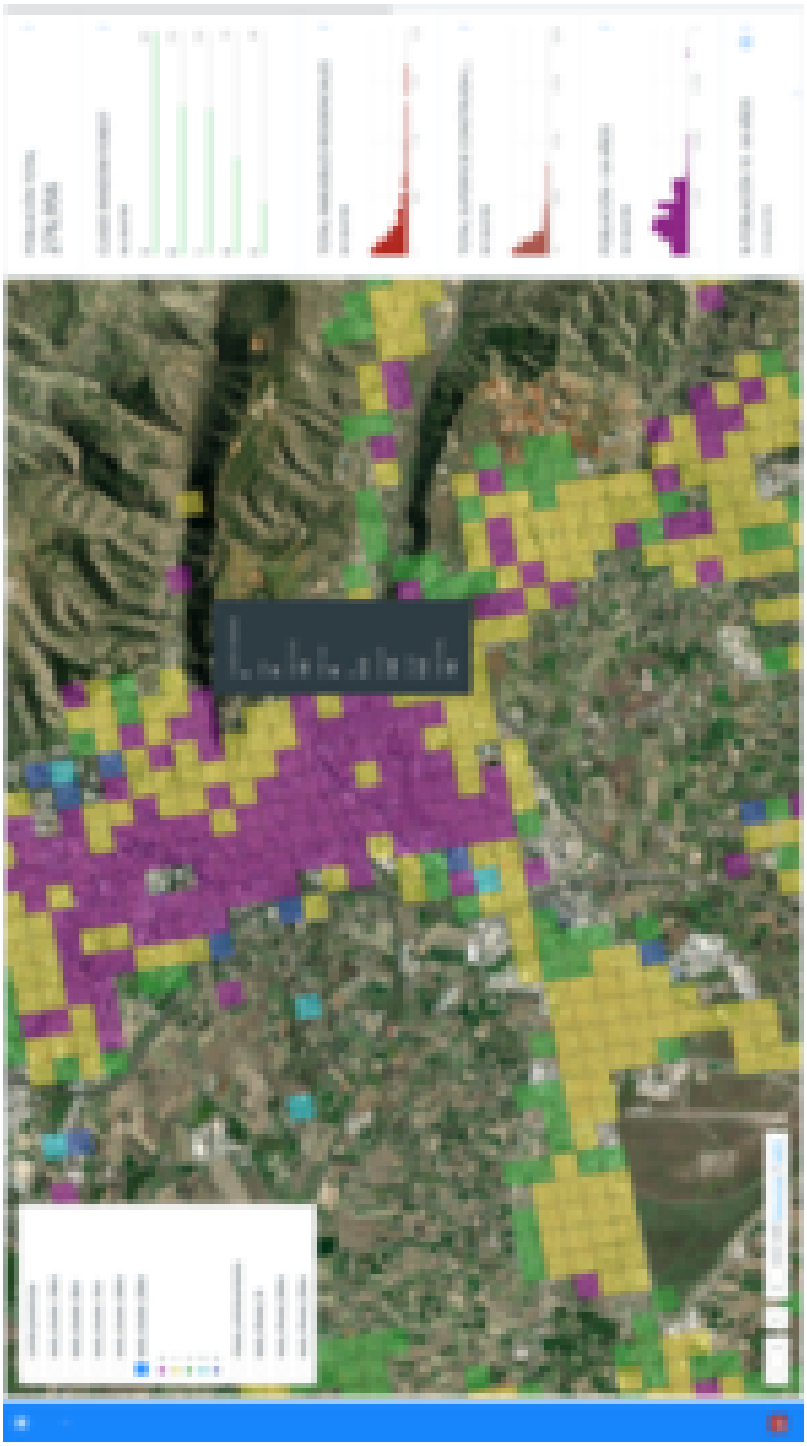

Figura III.9.15: *Dashboard* con los datos de clasificación *K-Means* para Granada. Fuente: elaboración propia.

Figura III.9.16: *Dashboard* con los datos de clasificación *Random forest* para Granada. Fuente: elaboración propia.

Parte IV

DISCUSIÓN, CONCLUSIONES Y LÍNEAS DE FUTURO

IV.1 Discusión

En relación al objetivo general de esta tesis (pag. 41), es decir, el desarrollo de un prototipo de plataforma *Cloud* que, de forma integral, posibilite la realización de las tareas de modelado y almacenamiento en bases de datos, de los procesos de integración espacial de la geoinformación ambiental y socio-económica en estructuras de teselas multiescalares y multidimensionales, de su análisis espacial (*Machine Learning*), así como su difusión *web* y el acceso a datos y resultados a través de clientes ligeros, creemos que los resultados confirman las hipótesis de partida.

En este sentido, pensamos que la tesis incorpora importantes avances tanto tecnológicos como metodológicos en todos los procesos de integración espacial de grandes conjuntos de datos geográficos, en origen con diferentes estructuras y formatos. Es obvio que gran parte de los procesos comentados están implícitos en las funcionalidades de las grandes plataformas *Cloud* empresariales ligadas a la información geográfica y el *Web GIS* (*Argis Online*, *CARTO*, *Mapbox*, etc.), aunque requerirían de un trabajo adicional para explicitarlas y estructurarlas para conseguir los objetivos generales y específicos de esta tesis. Generalmente, estas plataformas construyen teselaciones hexagonales y cuadrangulares en tiempo real por contención de puntos (un proceso conocido como *binning*) y realizando, sobre los aspectos temáticos numéricos de los mismos, sencillas operaciones de agregación estadística como recuentos, medias y/o sumas. La plataforma propuesta va mucho más allá, permitiendo procesos de adscripción a la rejilla mucho más sofisticados, como la adscripción de categorías poligonales que hacen los *Gridder Task* de tipo *DISCRETEPOLYAREASUMMARY*. Es más, la plataforma propuesta pone a disposición del programador un *framework* flexible para la implementación de cualquier nuevo algoritmo de adscripción de información a las teselas que sea necesario, excediendo, por tanto, la oferta típica de los proveedores *Cloud*.

Por otra parte, en esta tesis se ha desarrollado una plataforma *Cloud* con este objetivo específico que integra todos los procesos citados, habiéndose realizado para ello un enorme esfuerzo de programación e integración de las mas importantes librerías y desarrollos de *software* de código abierto, cuyo núcleo principal es la base de datos *PosgreSQL / PostGIS*, utilizadas en el ámbito de las tecnologías de información geográfica. El desarrollo completo de la librería para la parte servidora (*libcellbackend*) y su correspondiente en la parte cliente (*libcellfrontend*), ha significado un enorme esfuerzo de programación y constituyen la principal aportación metodológica de la tesis en forma de *software* generado bajos los principios *Open Source* en un nuevo contexto innovador (entornos *Cloud*) para el almacenamiento, gestión, integración espacial, análisis y difusión *web* de información geográfica.

La plataforma desarrollada no solo recoge todos estos procesos de forma integral sino que se ha diseñado para que sea escalable, gracias a su arquitectura de microservicios, y extensible, es decir, que sobre la misma estructura se irán incorporando nuevos microservicios, especialmente los de adscripción de los datos a las estructuras de teselas multiescalares asimétricas que serán muy dependientes de los tipos, estructura y temática de los datos que se vayan incorporando a la plataforma, en caso de que algunas fuentes de datos o algoritmos de adscripción tengan necesidades muy especiales. La generalización de estos procesos de adscripción, con ejemplos que cubren gran parte de las necesidades y casos de uso más comunes, (*POINTAGGREGATIONS*, *DISCRETEPOLYAREASUMMARY*) entran en contraste con ejemplos muy especializados en las particularidades de adscripción necesarias para ciertas fuentes de datos, como es el caso del muy especializado *Gridder Task* de tipo *MDTPROCESSING*, que adscribe datos de Modelos Digitales de Terreno. La plataforma, en este sentido, como ya se ha comentado, presenta un sólido *framework* que permite su extensibilidad. Lo mismo se puede decir de los microservicios de *Machine Learning* con Python, demostrando la flexibilidad y el acierto de la arquitectura elegida. No hemos encontrado, en las referencias bibliográficas consultadas, ningún trabajo parecido en el ámbito de la geografía española, como un producto que se derive de una tesis doctoral.

En relación a los resultados de objetivo específico relacionado con los datos, en la tesis se aportan nuevas ideas y métodos para la incorporación de datos, tanto estructurados como no estructurados, aspecto crucial en el contexto actual del área del *Spatial Big Data*. Creemos que en esta tesis se realiza una profunda reflexión sobre las estructuras de datos clásicos en la geoinformación ambiental y socio-económica, presentadas como vector y ráster, especialmente sobre su integración espacial en las estructuras de teselas haciendo uso para su almacenamiento de las posibilidades de la BBDD espacial *PostgreSQL / PostGIS*, la referencia para bases de datos geográficas de código abierto. El uso que se hace de este sistema también aporta la novedad de hacer un uso intensivo de sus cada vez más sofisticadas funcionalidades *NoSQL* con el uso del tipo de dato de estructura laxa *JSONB* como repositorio final del vector de adscripción de las teselas, demostrando una gran flexibilidad y capacidad, aún en *hardware* relativamente modesto, de servir las teselas definidas por la rejilla con un rendimiento más que aceptable.

Los resultados operacionales de la plataforma también han arrojado una evaluación muy positiva del concepto de asimetria de la estructura de datos propuesta. En ciertos escenarios, como la adscripción del juego de datos de catastro, la ganancia en espacio es más que notable, aunque depende mucho, como es evidente, del análisis de adscripción elegido. La ganancia, en un ráster de extensión completa del área de estudio, no es positiva, puesto que el ráster es capaz de comprimir su matriz numérica de forma extremadamente eficiente, mucho más de lo que PostgreSQL es capaz de hacer con su tipo *JSONB*. Si bien la ganancia de espacio no es un objetivo alcanzado en todos los datos procesados, la relación prestaciones del vector de adscripción / tamaño del juego de datos es más que positiva. Y teniendo en cuenta las técnicas de ahorro en espacio implementadas sobre las claves de las variables de adscripción y catálogos en el vector de adscripción, a través del uso intensivo de *hash*, aún se está muy lejos de alcanzar el límite operativo que la PostgreSQL impone al tamaño de su tipo *JSONB*, establecido en la actualidad en 255 *megabytes*.

Dado que muchos análisis de adscripción, como el *DISCRETEPOLYAREASUMMARY*, generan una gran cantidad de variables sobre el vector de adscripción, pero dada también la naturaleza asimétrica de los mismos, donde una tesela sólo recibe las variables que realmente se encuentran en su extensión, es bastante difícil llegar a estimar cuál puede ser el límite operativo de una base de datos *PostgreSQL* que gestione esta estructura de datos. Habría que adscribir muchísima más información de la que ha dado lugar los tests del prototipo presentado en este trabajo para tener una noción de este límite operativo del sistema.

En el comportamiento de la rejilla también destaca su capacidad autoindexativa y su aprovechamiento de las relaciones topológicas inter-resolución. Aunque la teselación con hexágonos ofrece ventajas distintas a la cuadrangular, el hecho de perder la capacidad de la adscripción por herencia de información de teselas madres a hijas tendría un notable impacto, para muchos de los análisis de adscripción, en el rendimiento de la plataforma. En rejillas cuadrangulares bien conformadas según las reglas de definición de rejilla expuestas en el capítulo de metodología II.3 "Definición de la rejilla" (pag. 79), las ventajas en la aceleración tanto del proceso de adscripción como de la recuperación de teselas para servirlas por la *API* son más que notables y positivas. Habría que estudiar en profundidad la topología de las teselaciones hexagonales para intentar sacarles el mismo provecho que a las cuadrangulares.

Continuando con la evaluación de los procesos de adscripción, queda demostrado el acierto de plantear la plataforma, desde el principio, como una arquitectura de microservicios con escalabilidad horizontal. El *hardware* avanzado y potente es caro y escaso, pero sin embargo generalmente existe una gran disponibilidad de *commodity hardware*, es decir, máquinas antiguas, desfasadas, que con una instalación de una distribución *Linux* tienen una segunda juventud. Los requerimientos individuales de los microservicios de trabajadores de adscripción son tan modestos que se pueden aprovechar máquinas antiguas para crear un *cluster* de teselación a bajo coste, incrementando la velocidad de adscripción: a más trabajadores, menos tiempo de proceso. Aunque la plataforma, en esta fase de prototipo, carece de ciertas capacidades de gestión de los trabajos concurrentes que poseen plataformas de otros entornos mucho más funcionales, como es el caso de *Apache Spark*, la plataforma propuesta cumple con creces con sus objetivos iniciales.

En lo que se refiere a la acomodación de la información adscrita en el vector de adscripción de la tesela a las necesidades de estructura de la información de los procesos de *Machine Learning* probados, el no supervisado *K-Means* y el supervisado *Random forest*, el cumplimiento del objetivo planteado es total. El diseño del vector de adscripción responde exactamente a esas necesidades, y las capacidades de búsqueda y extracción de un determinado sub-vector, gracias a las funcionalidades de indexación de *JSON* de la *PostgreSQL*, y gracias al haber incorporado, cuando el sistema comenzaba a evidenciar una patente falta de rendimiento en este aspecto[1], el concepto de *variable de indexación*, son

[1]Recordemos siempre que esta no es la especialidad de un sistema como *PostgreSQL*, que es una funcionalidad muy reciente en el sistema y que aún le queda un largo recorrido para mejorarla en el caso, claro está, de que la cúpula del *PostgreSQL Global Development Group*, que toma las últimas decisiones de prioridades y gestión del proyecto, lo considere una línea de desarrollo que merezca la pena potenciar. Personalmente, este autor espera que así sea ya que le da al sistema una versatilidad extraordinaria como

más que adecuadas para la adquisición de estos sub-vectores que ya están prácticamente en el formato nativo de las librerías de *Machine Learning Python* utilizadas. Dados los resultados de integración de estructura de datos y de la comunicación efectiva *Node.js* y *Python*, la implementación de nuevos algoritmos de *Machine Learning* a la plataforma es factible y hasta cierto punto sencilla.

Cabe destacar, en los resultados de las pruebas de *Machine Learning* con el algoritmo *K-Means*, como con una simple exploración de los datos ya emerge un patrón claramente vinculado a un problema clásico en Geografía, el *Modifiable Areal Unit Problem* (*MAUP*) (Openshaw et al., 1979, Openshaw et al., 1981, Chasco Yrigoyen, 2003). La estructura multiescalar de la rejilla propuesta se muestra, de esta forma, ventajosa, ya que permite la detección de estos patrones muy dependientes de la escala que se producen en las distribuciones estadísticas de las variables a medida que el área de agrupación de las mismas se va reduciendo. El prototipo, por lo tanto, valida su utilidad en el campo no sólo de los estudios multivariantes, sino de los multivariantes multiescalares, pudiendo el investigador estudiar el comportamiento de sus variables a medida que la escala de significación aumenta en un entorno controlado y eficiente.

En cuanto a los visores, la tendencia de los clientes *web* ligeros de orientar su funcionamiento al consumo de datos, y no de estructuras de información o visualización prefabricadas por el servidor, como hacían los antiguos visores con los mapas predibujados, aporta unas capacidades semiológicas, interactivas y de exploración de datos más que notables. El uso combinado en el visor *Mapbox* propuesto de cartografía generada en tiempo real en el cliente más exploración del mismo set de datos por medio de un método tan sencillo como un histograma hace del conjunto una herramienta muy versátil y con grandes posibilidadesd de extensión y adaptación. La creación de *dashboards* de navegación e interpretación de datos, aunque muy costosa en tiempo de desarrollo, puesto que la programación gráfica de los *frontends* es una árdua tarea, está más que probada. La estructura de datos asimétrica creada en los vectores de adscripción, más la propiedad añadida de la definición de las rejillas que permite obviar el tránsito desde los servidores al cliente de la vertiente geométrica de la información geográfica realmente cumple el objetivo de ser capaz de enviar una buena cantidad de información a una resolución más que adecuada para una zona de visualización amplia del conjunto teselar. La capacidad que adquiere el visor de recrear, localmente y en tiempo real, la geometría de soporte de la información temática constituye un más que interesante mecanismo para desarrollar visualizaciones cartográficas de la información teselada. El relativamente nuevo conjunto de tecnologías *web* formadas por *HTML5*, *CSS3* y *WebGL* permite crear visores de increibles prestaciones. Si bien es cierto que en la demostración de estas capacidades, en este trabajo de tesis, se ha utilizado un visor de tecnología no abierta, aunque basado en ella, como son el *framework* de *Mapbox* y la plataforma *CARTO*, también es verdad que el desarrollo de un visor con estas características excede con mucho la capacidad de dedicación de este trabajo. El foco del objetivo estaba más centrado en la parte *backend* que en la parte *frontend*. De todas formas, aún con esta peculiaridad, queda demostrado la capacitación de la plataforma para cumplir este objetivo.

---

sistema mixto *SQL / NoSQL*.

IV.2 Conclusiones

Con el trabajo realizado en esta tesis se han confirmados las hipótesis de trabajo con todas las tareas realizadas para cada uno de los objetivos planteados en el capitulo I.2 (pag. 41).

En el desarrollo de esta tesis doctoral se ha intentado proporcionar una solución tecnológica integral a todo el proceso de integración espacial de diferentes tipos de datos ambientales o socio-demográficos en una nueva estructura para los datos integrados: las estructuras de teselas multiescalares y multidimensionales asimétricas. La solución tecnológica, y el principal resultado de esta tesis es, sin duda, una plataforma *cloud* (*Cell*) desarrollada y programada íntegramente por el autor, cuyo código se hace público a través de la plataforma de desarrollo *GitHub* y en el que se integra el uso de un elevado conjunto de librerías *Open Source* muy relacionadas con el acceso, almacenamiento, transformación y análisis de datos geográficos.

En este sentido, se trata de una tesis con un alto contenido innovador en lo tecnológico que, además, incorpora también soluciones metodológicas novedosas en los procesos de integración espacial de geodatos de diferentes formatos (vector y ráster) y de variables de diferente naturaleza (categóricas, numéricas, discretas, continuas, etc.) que se almacenan en estructuras "laxas" en BBDD espaciales (combinando *SQL* y *NoSQL*), así como en su acceso y difusión en *Internet* (*API*, servicios *web* geográficos, etc.), proporcionando finalmente estructuras muy adecuadas a los procesos analíticos ligados al campo de la Inteligencia Artificial y el *Machine Learning* con vectores n-dimensionales.

Como complemento a esta solución integral del proceso de integración espacial, almacenamiento y difusión, la plataforma se complementa con la incorporación de clientes *web*, también desarrollados en parte por el autor sobre una base de productos comerciales, que hacen un uso innovador de la rejilla multiescalar y asimétrica desarrollada al acelerar los procesos de geovisualización, ya que la rejilla, al estar definida matemáticamente e integrada en el cliente, acelera los procesos del tránsito de los datos desde el servidor al cliente (solo se envían los datos multidimensionles temáticos, ahorrándose el envío de la pesada información geométrica). Por otra parte, los clientes *web* hacen también un uso muy eficiente de las nuevas potencialidades de *HMTL5* y *WebGL*, posibilitando la incorporación, también innovadora, de los procesos del tratamiento semiológico en el cliente, ya que los clientes *web* más habituales utilizan servicios *web* (generalmente servicios interoperables *OGC* del tipo *WMS* o servicios de mapas parecidos) donde el trabajo de semiología y renderización se realiza en la parte servidora de la aplicación *web*. El aspecto más innovador de estos clientes, sin duda, es el consumo directo de los datos temáticos nativos almace-

nados en la rejilla, abriendo un amplio campo para la incorporación de herramientas más interactivas en el cliente, en forma de *widgets*, que permiten filtrar o agregar los datos recibidos a gusto del usuario, facilitando el desarrollo de cuadros de mando (*dashboards*) para la exploración interna de los datos, representando, mediante gráficos estadísticos y/o científicos, la información representada en el mapa por otros medios de expresión que complementan a la geovisualización en el geovisor *web*, cuyas representaciones semiológicas interactúan sincrónicamente con todos los componentes antes comentados en tiempo real.

El alto contenido tecnológico de la tesis, junto a las aportaciones metodológicas, especialmente las que se incorporan en los métodos de adscripción de las variables temáticas multidimensionales a la nueva estructura propuesta de rejillas multiescalares y multidimensionales, entre otras, la situarían en el corazón de un debate clásico en nuestra disciplina geográfica: ¿son las tecnologías de la información geográfica (TIG) meras herramientas tecnológicas o aportan nuevas concepciones metodológicas y avances significativos a diferentes temáticas de la ciencia geográfica? Este debate no solo es un clásico en la geografía española (Chuvieco et al., 2005), sino también un debate de más larga tradición aún en el mundo anglosajón (Wright et al., 1997) y está más vigente en la actualidad con la aparición de nuevas propuestas disciplinares altamente relacionadas con los contenidos de esta tesis (*Spatial Data Science*, *Spatial Big Data*, *Web GIS*, etc.).

Sin duda, estamos de acuerdo con Wright, Goodchild y Proctor al defender los importantes avances metodológicos que estas tecnologías han incorporado en nuestra disciplina científica, y creemos que esta tesis se enmarca en este contexto innovador para la ciencia geográfica.

IV.3 Líneas de futuro

Una vez terminada la tesis Doctoral se espera poder seguir desarrollando la plataforma *Cell*, contando para ello con financiación en el Grupo de Investigación procedente de tres proyectos que comparten este entorno tecnológico pero cuyos objetivos temáticos son diferentes. Estos proyectos son los siguientes:

1. Plataforma *Cloud* para la Integración Espacial de Geoinformación ambiental y socioeconómica (*Spatial Big Data*) y Clientes *Web* Interactivos (Geovisores, *DashBoard*) para su difusión, acceso, explotación y análisis (*Maching Learning*). Proyecto singular de actuaciones de transferencia CEI RIS3. Junta de Andalucía. Andalucía Tech CEI-9.

2. Prototipo *Cloud* de Evaluación de Riesgos a Través de la Integración Espacial de Geodatos (*Grid* Multiescalares) y Clientes *Web* (*Dashboard*) para su Gestión y Medidas de Adaptación. Proyecto Plan Estatal I+D+i Retos. Ministerio de Ciencia e Innovación. PID2019-106834RB-I00.

3. E-Infraestructura (*Cloud Computing*) para la Integración Espacial de Variables Antrópicas en el Cálculo de la Vulnerabilidad ante Riesgos Costeros y Herramientas *Web* (*Dashboard*) para su Gestión en Andalucía (Clodrisk_A). Proyecto I+D+i Frontera. Junta de Andalucía. P18-FR-2574.

De todo lo expuesto en los capítulos anteriores IV.1 “Discusión” y IV.2 “Conclusiones”, y siguiendo el conocido dicho de “las tesis no se terminan, sólo se abandonan”, es evidente que la plataforma *Cell* tiene un buen número de aspectos de mejora y extensibilidad, que pasamos a detallar a continuación.

IV.3.1. Mejoras en los microservicios de datos

Sería una buena idea experimentar, como repositorio final de las teselas, un sistema más especializado en el almacenamiento y recuperación de información de estructuras de datos laxas como el *JSON*. Existen sistemas *NoSQL* que podrían ser buenos candidatos para ello, entre otros *MongoDB* (Pirtle, 2010), *Cassandra* (Lakshman et al., 2010) o *ClickHouse* (ClickHouse Contributors, 2021). *PostgreSQL* ofrece una ayuda inestimable en tareas de desarrollo por su altísimo grado de integración con el ecosistema de las TIG de *Software Libre*, especialmente con el SIG de escritorio *QGIS*, y es muy difícil renunciar a esta combinación como herramienta de desarrollo, prueba y depuración. Sin embargo, hay que tener en cuenta que, actualmente, este sistema cuenta con un límite duro al tamaño de los

documentos *JSON* que puede albergar, situado en 255 *megabytes*, que son bastantes para dar cabida a muchísimos datos adscritos, especialmente con el cuidado que se ha puesto, con el uso intensivo de *hashes*, en que estos sean lo más pequeños posible. Y además, dada la orientación al microservicio de la plataforma, sería natural para ella implementar la posibilidad de utilizar un sistema u otro como repositorio final de los datos adscritos, sin perjuicio de seguir contando con *PostgreSQL / PostGIS* en la fase de desarrollo y depuración.

En línea con lo anterior, actualmente el prototipo tiene una dependencia fuerte de la propia *PostGIS* como motor de adscripción, ya que realiza una parte del trabajo analítico de algunos de las clases *Gridder Task*. El éxito en la adscripción del *Gridder Task DISCRETEPOLYTOPAREA* utilizando como motor de adscripción exclusivamente la librería *Turf.js*, en un modo de ejecución completamente independiente, salvo en la fase de adquisición de datos, de *PostGIS*, aunque hace el desarrollo un tanto más largo, convierte en deseable que todas las clases de adscripción lo utilicen internamente. La base de datos ha demostrado ser, como era de esperar, un cuello de botella en la faceta no de almacén de datos, sino de motor de geoprocesamiento para algunos análisis de teselado.

También sería un buen añadido para la plataforma la posibilidad de crear llamadas en la *API* de servicio que permitieran la subida y lectura de información original de adscripción en otros formatos que no sean una segunda instancia *PostGIS*. Juegos de datos sencillos podrían ser proporcionados de esta manera en formatos populares como *KML*, *GeoJSON*, *GeoPackage* o incluso *SHAPEFILE*. La presencia en el sistema de la librería *GDAL / OGR* posibilitaría esta funcionalidad.

Se echa en falta disponer, aparte de la API de servicios, con la que se puede interaccionar directamente con herramientas como *Postman* (Postman Contributors, 2021) o *curl* (Stenberg, 2021), de un *frontend* de administración para la consulta y control del sistema, permitiendo a un usuario con menos conocimientos técnicos inspeccionar el catálogo de variables de adscripción, controlar y consultar procesos de adscripción en curso, solicitar nuevas adscripciones de datos, etc.;

IV.3.2. Nuevos análisis de teselado

La propia naturaleza extensible de la plataforma para la implementación de nuevos análisis de adscripción en la forma de clases de la familia *GridderTask* deja unas posibilidades muy abiertas para incluir nuevos análisis de adscripción para tipos de datos con características muy distintas a las tratadas en este trabajo. Entre los candidatos más interesantes se encuentran la adscripción directa de imágenes de satélite multiespectrales, indicadores de redes sociales, indicadores de vulnerabilidad frente a riesgos naturales, fuentes socio-económicas normalizadas a nivel europeo, análisis sobre fuentes de datos lineales vectoriales, etc.

Sería muy interesante explorar el desarrollo e implementación de análisis de adscripción no atómicos, es decir, no exclusivamente ceñidos al universo de la tesela que es objeto del análisis. Estos análisis tendrían en cuenta relaciones topológicas con otras teselas en su nivel de resolución, por lo que se podrían realizar análisis propios del álgebra ráster.

Ahondando aún más en esta línea, también sería muy interesante incorporar análisis exclusivos que trabajaran sobre redes geométricas. La aplicación del análisis de redes en todos los estudios que tengan que ver con accesibilidad, por ejemplo, aportaría a la plataforma, con su base teselar, un interesante cuerpo adicional de funcionalidad analítica avanzada.

En el ámbito de la adscripción, la plataforma es lo suficientemente flexible, en su estado actual, y con un esfuerzo de implementación no trivial pero tampoco excesivo, de utilizar una rejilla de tipo triangular en lugar de cuadrangular. Los triángulos también tienen una capacidad autoindexativa, por lo que la clase *Grid* sólo necesitaría ser refactorizada[1] en dos clases de rejilla distintas, una para las cuadrangulares (la actual) y otra para las nuevas triangulares, implementando sus particularidades topológicas. En cuanto a la implementación de rejillas hexagonales también sería posible con algo más de esfuerzo, aunque las actuales clases de la familia *GridderTask* necesitarían detectar este tipo de rejillas para no intentar hacer ningún procedimiento de adscripción por herencia, algo no posible, como ya se comentó en II.4.1, "Análisis de la herencia inter-resolución", (pag. 93), por las propiedades geometrico-topológicas de este tipo de rejilla.

IV.3.3. Machine Learning

En cuanto a las capacidades del *Machine Learning* del sistema, una vez probada su capacidad de servir vectores multivariantes, es muy extensible. El sistema es tremendamente adaptable a la implementación en la librería *PyCell* de nuevos métodos de *Machine Learning*. Los dos métodos implementados, procedentes de la librería *Scikit-learn*, demuestran sus capacidades de integración, capacidades directamente utilizables para la implementación de cualquier otro algoritmo presente en la misma. De igual forma, procedimientos de otras áreas del *Machine Learning*, como el *Deep Learning*, es decir, aprendizaje con redes neuronales, un campo bastante más avanzado pero con unas prestaciones muy potentes, precisan de suministrarles la información exactamente en el mismo formato del que se sirven ahora mismo los procedimientos implementados. Sería cuestión de estudiar la integración de la librería *Python PyTorch* (Paszke et al., 2019), que trabaja con estos algoritmos, y establecer clases de la familia *MlTask* que las controlaran.

IV.3.4. Visualización

En cuanto a los *frontend*, aquí el margen de actuación se amplía muchísimo. Como ya se ha descrito en IV.2 "Conclusiones", el prototipo implementado en esta tesis utiliza plataformas comerciales no disponibles bajo licencias de *Software Libre* como apoyo a la implementación de este componente. Es una tecnología compleja, que se escapa del ámbito profesional usual de este autor, pero estaría bien que algún colaborador con más experiencia en éste ámbito intentara crear un visor dedicado en el dominio del *Software Libre* a partir del excelente proyecto *Leaflet* (Agafonkin et al., 2021a).

[1] En ingeniería de *software*, la refactorización es el proceso de reestructurar la arquitectura interna de un programa.

La flexibilidad que aportan librerías como *D3* posibilitan la implementación de innumerables dispositivos (*widgets*) de representación de información estadística. En la vertiente cartográfica, el margen de aprovechamiento de la definición puramente matemática de la tesela abre muchas posibilidades de aplicaciones semiológicas novedosas y originales (Belmonte, 2015). Y dado que la *API* de la plataforma está orientada al dato, el tratamiento de visualización de los mismos no se restringe necesariamente a la cartografía, sino que queda abierto a las técnicas más modernas del campo del *DataViz* (Cairo, 2012).

Parte V

BIBLIOGRAFÍA Y ANEXOS

V.1 Bibliografía

Adair, Michael et al. (2021). *proj4js*. URL: `http://proj4js.org/` (visitado 05-03-2021).

Agafonkin, Vladimir et al. (2021a). *Leaflet*. URL: `https://leafletjs.com/` (visitado 05-03-2021).

— (2021b). *Turf.js - Advanced geospatial analysis for browsers and Node.js*. URL: `https://turfjs.org/` (visitado 05-03-2021).

Álvarez Francoso, J. I. (2016). «Geovisualización de grandes volúmenes de datos ambientales. Diseño e implementación de un sistema para el acceso y la difusión de datos globales». Tesis doct. Universidad de Sevilla.

Álvarez Francoso, J. I., Camarillo Naranjo, J. M., Limones Rodríguez, N. y Pita López, M. F. (2014). «Globalclimatemonitor.org: una herramienta de acceso a datos climáticos globales». En: *GeoFocus* 14, págs. 1-6.

Álvarez Francoso, J. I., Ojeda Zújar, J., Díaz Cuevas, P., Guisado Pintado, E., Camarillo Naranjo, J. M., Prieto Campos, A. y Fraile Jurado, P. (2020). «A Specialized Geoviewer and Dashboard for Beach Erosion Rates Visualization and Exploration». En: *Journal of Coastal Research* 95.SI, págs. 1006-1010.

Anthes, Gary (2012). «HTML5 leads a web revolution». En: *Communications of the ACM* 55.7, págs. 16-17.

Armstrong, Margaret (1998). *Basic Linear Geostatistics*. Springer. ISBN: 3-540-61845-7.

Atlassian (2021). *BitBucket*. URL: `https://bitbucket.org` (visitado 05-03-2021).

Baer, Kim y Vacarra, Jill (2008). *Information Design Workbook*. Rockport Publishers. ISBN: 978-1-59253-627-6.

Barbolla, Rosa y Sanz, Paloma (1998). *Álgebra lineal y teoría de matrices*. Prentice Hall. ISBN: 84-8322-008-3.

Barros, A. P. y Dumas, M. (2006). «The Rise of Web Service Ecosystems». En: *IT Professional* 8.5, págs. 31-37. DOI: `10.1109/MITP.2006.123`.

Batista e Silva, F., Gallego, J. y Lavalle, C. (2013). «A high-resolution population grid map for Europe». En: *Journal of Maps* 9.1, págs. 16-28.

Battersby, Sarah E, Finn, Michael P, Usery, E Lynn y Yamamoto, Kristina H (2014). «Implications of web Mercator and its use in online mapping». En: *Cartographica: The International Journal for Geographic Information and Geovisualization* 49.2, págs. 85-101.

Belmonte, Nicolas (2015). «Data Visualization Techniques with WebGL». En: *WebGL Insights*, pág. 297.

Berners-Lee, Tim, Cailliau, Robert, Luotonen, Ari, Nielsen, Henrik Frystyk y Secret, Arthur (ago. de 1994). «The World-Wide Web». En: *Commun. ACM* 37.8, págs. 76-82.

ISSN: 0001-0782. DOI: 10.1145/179606.179671. URL: https://doi.org/10.1145/179606.179671.

Biehl, Matthias (2016). *RESTful API Design*. Vol. 3. API-University Press.

Birch, Colin PD, Oom, Sander P y Beecham, Jonathan A (2007). «Rectangular and hexagonal grids used for observation, experiment and simulation in ecology». En: *Ecological modelling* 206.3-4, págs. 347-359.

BOE (2006a). «Ley 12/1989, de 9 de mayo, de la Función Estadística Pública». En: *BOE num. 112 de 11/05/1989*.

— (2006b). «Ley 2/2018, de 23 de mayo, por la que se modifica la Ley 14/2010, de 5 de julio, sobre las infraestructuras y los servicios de información geográfica en España». En: *BOE num. 126 de 24/5/2018*.

— (2020). «Ley 3/2018, de 5 de diciembre, de Protección de Datos Personales y garantía de los derechos digitales.» En: *BOE num. 294 de 06/12/2018*.

— (2021). *Directiva 92/43/CEE del Consejo, de 21 de mayo de 1992, relativa a la conservación de los hábitats naturales y de la fauna y flora silvestres*. URL: https://www.boe.es/buscar/doc.php?id=DOUE-L-1992-81200 (visitado 05-03-2021).

Boettiger, Carl (2015). «An introduction to Docker for reproducible research». En: *ACM SIGOPS Operating Systems Review* 49.1, págs. 71-79.

Bondaruk, B., Roberts, S. A. y Robertson, C. (2019). «Discrete Global Grid Systems: Operational Capability of the Current State of the Art». En: *Spatial Knowledge and Information - Canada - Proceedings*.

Brassard, G. y Bratley, P. (1997). *Fundamentos de algorítmia*. Prentice Hall. ISBN: 84-89660-00-X.

Breiman, L. (2001). «Random Forests». En: *Machine Learning* 45, págs. 5-32. DOI: 10.1023/A:1010933404324.

Bressert, Eli (2012). *SciPy and NumPy*. O'Reilly. ISBN: 978-1-449-30546-8.

Breunig, Martin, Bradley, Patrick Erik, Jahn, Markus, Kuper, Paul, Mazroob, Nima, Rösch, Norbert, Al-Doori, Mulhim, Stefanakis, Emmanuel y Jadidi, Mojgan (2020). «Geospatial data management research: Progress and future directions». En: *ISPRS International Journal of Geo-Information* 9.2, pág. 95.

Cady, Field (2017). *The Data Science Handbook*. Wiley. ISBN: 978-1-119-09294-0.

Cairo, Alberto (2012). *The Functional Art: An introduction to information graphics and visualization*. New Riders.

Campbell-Kelly, Martin, Aspray, William, Snowman, Daniel P, McKay, Susan R, Christian, Wolfgang et al. (1997). «Computer: A history of the information machine». En: *Computers in Physics* 11.3, págs. 256-257.

Caparros Santiago, J. A. y Rodríguez-Galiano, V. F. (2020). «Estimación de la fenología de la vegetación a partir de imágenes de satélite: el caso de la península ibérica e islas Baleares (2001-2017)». En: *Revista de Teledetección* 57, págs. 25-36.

CARTO Contributors (2021a). *CARTO*. URL: https://carto.com/ (visitado 05-03-2021).

— (2021b). *CARTO API1*. URL: https://docs.carto.com/ (visitado 05-03-2021).

Chamberlin, Donald D. (2012). «Early history of SQL». En: *IEEE Annals of the History of Computing* 34.4, págs. 78-82.

Chasco Yrigoyen, C. (2003). *Econometría espacial aplicada a la predicción-extrapolación de datos microterritoriales*. Dirección General de Economía y Planificación.

Chen, Jian, Yang, Shengtian, Li, Hongwei, Zhang, Bin y Lv, Junrong (2013). «Research on geographical environment unit division based on the method of natural breaks (Jenks)». En: *Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci* 3, págs. 47-50.

Chen, Peter Pin Shan (mar. de 1976). «The Entity-Relationship Model—toward a Unified View of Data». En: *ACM Trans. Database Syst.* 1.1, págs. 9-36. ISSN: 0362-5915. DOI: 10.1145/320434.320440. URL: https://doi.org/10.1145/320434.320440.

Chen, S., Tang, X., Wang, H., Zhao, H. y Guo, M. (2016). «Towards Scalable and Reliable In-Memory Storage System: A Case Study with Redis». En: *2016 IEEE Trustcom/-BigDataSE/ISPA*, págs. 1660-1667. DOI: 10.1109/TrustCom.2016.0255.

Church, Richard L. y Murray, Alan T. (2009). *Business Site Selection, Location Analysis, and GIS*. Wiley. ISBN: 978-0-470-19106-4.

Chuvieco, E. (1996). *Fundamentos de teledetección espacial*. Rialp. ISBN: 84-321-3127-X.

Chuvieco, E., Bosque Sendra, J., Pons Fernández, X., Conesa García, C., Santos Preciado, J. M., Gutiérrez Puebla, J., Salado García, M. J., Martín, M. P., Riva, J. de la, Ojeda Zújar, J. y Prados Velasco, M. J. (2005). «¿Son las tecnologías de la información geográfica (TIG) parte del núcleo de la geografía?» En: *Boletín De La Asociación De Geógrafos Españoles* 40, págs. 35-55.

ClickHouse Contributors (2021). *ClickHouse*. URL: https://clickhouse.tech/ (visitado 05-03-2021).

Codd, Edgar Frank (1983). «A relational model of data for large shared data banks». En: *Communications of the ACM* 26.1, págs. 64-69.

Coetzee, S., Ivánová, I., Mitasova, H. y Brovelli, M. A. (2020). «Open Geospatial Software and Data: A Review of the Current State and A Perspective into the Future». En: *ISPRS International Journal of Geo-Information* 9.2. ISSN: 2220-9964. DOI: 10.3390/ijgi9020090. URL: https://www.mdpi.com/2220-9964/9/2/90.

Dalton, John-David et al. (2021). *lodash*. URL: https://lodash.com/ (visitado 05-03-2021).

Dark, Shawna J y Bram, Danielle (2007). «The modifiable areal unit problem (MAUP) in physical geography». En: *Progress in Physical Geography* 31.5, págs. 471-479.

Das, A. K. (2020). «UNESCO Recommendation on Open Science: An Upcoming Milestone in Global Science». En: *Science Diplomacy Review*.

Date, Christopher J. (2003). «Edgar F. Codd: a tribute and personal memoir». En: *ACM SIGMOD Record* 32.4, págs. 4-13.

de Berg, Mark, van Kreveld, Marc, Overmars, Mark y Schwarzkopf, Otfried (2000). *Computational Geometry: Algorithms and Applications*. Springer. ISBN: 3-540-65620-0.

Diener, Michael (2015). *Python Geospatial Analysis Cookbook*. Packt Publishing Ltd.

Dirección General del Catastro (2021). *Sede Electrónica del Catastro - Página oficial*. URL: https://www.sedecatastro.gob.es/ (visitado 05-03-2021).

Dragićević, S. (2004). «The potential of Web-based GIS». En: *Journal of Geographical Systems* 6, págs. 79-81. DOI: 10.1007/s10109-004-0133-4.

Dutton, Geoffrey (1996). «Encoding and handling geospatial data with hierarchical triangular meshes». En: *Proceeding of 7th International symposium on spatial data handling*. Vol. 43. Citeseer.

DVC Contributors (2021). *DVC*. URL: https://dvc.org/ (visitado 05-03-2021).

Dyevre, Arthur (2020). «Text-mining for Lawyers: How Machine Learning Techniques Can Advance our Understanding of Legal Discourse». En: *Available at SSRN 3734430*.

Enrique, I., Molina, J. E., Ojeda, S., Escudero, M. y Pérez, G. (2013). «Distribución espacial de la población en Andalucía». En: *Cuadernos Geográficos* 52(2), págs. 153-157.

Etherington, Thomas R. (2016). «Teaching introductory GIS programming to geographers using an open source Python approach». En: *Journal of Geography in Higher Education* 40.1, págs. 117-130. DOI: 10.1080/03098265.2015.1086981. eprint: https://doi.org/10.1080/03098265.2015.1086981. URL: https://doi.org/10.1080/03098265.2015.1086981.

EuroStat (2012). «ESSnet project GEOSTAT 1A-Representing Census data in a European population grid-Final Report». En: *GEOSTAT1A*.

Fain, Yakov y Moiseev, Anton (2017). *Angular 2 Development with TypeScript*. Manning Publications. ISBN: 978-1-61729-312-2.

Fernandez Núñez, M., Burningham, H. y Ojeda Zújar, J. (2017). «Improving accuracy of LiDAR-derived digital terrain models for saltmarsh management». En: *Journal of Coastal Conservation* 21.1, págs. 209-222.

Fernández-Palacios Carmona, A., Moreira Madueño, J. M., Sánchez Rodríguez, E. y Ojeda Zújar, J. (1995). «Evaluation of different methodological approaches for monitoring water quality parameters in the coastal waters of Andalusia (Spain) using Landsat-TM data». En: *EARSeL Advances in Remote Sensing, 4 (1), 67-75.*

Ferreira, N., Poco, J., Vo, H. T., Freire, J. y Silva, C. T. (2013). «Visual Exploration of Big Spatio-Temporal Urban Data: A Study of New York City Taxi Trips». En: *IEEE Transactions on Visualization and Computer Graphics* 19.12, págs. 2149-2158. DOI: 10.1109/TVCG.2013.226.

Flanagan, David (2011). *JavaScript: The Definitive Guide*. O'Reilly. ISBN: 978-0-596-80552-4.

Fraile Jurado, P., Álvarez Francoso, J. I., Guisado Pintado, E., Sánchez Carnero, N., Ojeda Zújar, J. y Leatherman, S. P. (2017). «Mapping inundation probability due to increasing sea level rise along El Puerto de Santa María (SW Spain)». En: *Natural Hazards* 87.2, págs. 581-598.

Gallego, Francisco Javier (2010). «A population density grid of the European Union». En: *Population and Environment* 31.6, págs. 460-473.

GDAL/OGR Contributors (2021). *GDAL/OGR geospatial data abstraction software library*. URL: https://gdal.org/ (visitado 05-03-2021).

Geographic Information Science, University Consortium for (2020). *Tomlinson, Roger*. URL: https://www.ucgis.org/roger-tomlinson (visitado 10-01-2020).

GEOS Contributors (2021). *GEOS - Geometry Engine, Open Source*. URL: https://trac.osgeo.org/geos (visitado 05-03-2021).

Gibb, RG (2016). «The rHEALPix discrete global grid system». En: *IOP Conference Series: Earth and Environmental Science*. Vol. 34. 1. IOP Publishing, pág. 012012.

GitHub Inc. (2021). *The 2020 State of the OctoVerse*. URL: https://octoverse.github.com/ (visitado 04-03-2021).

GitLab Inc. (2021). *GitLab*. URL: https://about.gitlab.com/ (visitado 05-03-2021).

Goerlich, F. J. y Cantarino, I. (2012). «Una grid de densidad de población para España». En: *Fundación BBVA*.

— (2013). «Geodemografía: coberturas del suelo, sistemas de información geográfica y distribución de la población». En: *Investigaciones Regionales* 25, págs. 165-191.

— (jun. de 2017). «Grid poblacional 2011 para España: Evaluación metodológica de diversas posibilidades de elaboración». En: *Estudios Geográficos, Instituto de Economía, Geografía y Demografía (CSIC)* LXXVIII (282), págs. 135-163.

Goerlich, F.J., Reig, E. y Cantarino, I. (jun. de 2016). «Construcción de una tipología rural/urbana para los municipios españoles». En: *Investigaciones Regionales* 35, págs. 151-173.

Gold, Christopher (2016). «Tessellations in GIS: Part II–making changes». En: *Geo-spatial Information Science* 19.2, págs. 157-167.

González Jiménez, A., Rubio Iglesias, J. M., Velasco Tirado, A. et al. (2012). «CartoCiudad: An inspired national thoroughfares network map». En: INSPIRE Conference 2012.

Goodchild, M. F. (2000). «Discrete global grids for digital earth». En: *International Conference on Discrete Global Grids*. Citeseer, págs. 26-28.

— (2007). «Citizens as sensors: the world of volunteered geography». En: *GeoJournal* 69.4, págs. 211-221.

— (2013). «The quality of big (geo) data». En: *Dialogues in Human Geography* 3.3, págs. 280-284.

Goodchild, M. F. y Shiren, Y. (1992). «A hierarchical spatial data structure for global geographic information systems». En: *CVGIP: Graphical Models and Image Processing* 54.1, págs. 31-44. ISSN: 1049-9652. DOI: https://doi.org/10.1016/1049-9652(92)90032-S. URL: https://www.sciencedirect.com/science/article/pii/104996529290032S.

Graser, Anita (2013). *Learning QGIS 2.0*. Packt Publishing Ltd.

Grommé, F. y Ruppert, E. (2020). «Population Geometries of Europe: The Topologies of Data Cubes and Grids». En: *Science, Technology, & Human Values* 45.2, págs. 235-261. DOI: 10.1177/0162243919835302. eprint: https://doi.org/10.1177/0162243919835302. URL: https://doi.org/10.1177/0162243919835302.

Gutiérrez Puebla, J. (2018). «Big Data y nuevas geografías: la huella digital de las actividades humanas». En: *Documentsd'AnàlisiGeogràfica* 64/2, págs. 195-217.

Gutiérrez Puebla, J., García Palomares, J. C., Romanillos, Gustavo y Salas Olmedo, M. H. (2017). «The eruption of Airbnb in tourist cities: Comparing spatial patterns of hotels and peer-to-peer accommodation in Barcelona». En: *Tourism Management* 62, págs. 278-291.

Gutiérrez Puebla, J., García Palomares, J. C. y Salas Olmedo, M. H. (2016). «Big (geo) data en ciencias sociales: retos y oportunidades». En: *Revista de Estudios Andaluces* 33 (1), págs. 1-23.

Hastie, T. (2001). *The elements of statistical learning : data mining, inference, and prediction : with 200 full-color illustrations*. New York: Springer. ISBN: 0387952845.

Herron, D. (2020). *Node.js Web Development - 5th Edition*. ISBN: 9781838987572.

Hidalgo, Adolfo Jorge Sánchez (2019). «Neuro-evolucionismo y deep machine learning: nuevos desafíos para el derecho». En: *Journal of Ethics and Legal Technologies* 1.1.

Hsu, Leo S. y Obe, Regina O. (2015). *PostGIS in action*. Manning Publications.

IBM Corp. (2021). *Watson Analytics*. URL: https://www.ibm.com/analytics (visitado 05-03-2021).

Iglesias Campos, A., Malvarez García, G., Ojeda Zújar, J. y Moreira Madueño, J.M. (2011). *Coastal Informatics: Web Atlas Design and Implementation: Web Atlas Design*

and Implementation. Ed. por D. Wright, N. Dwyer y V. Cummins. IGI Global. Cap. 14 Spain, págs. 214-228.

INE (2021a). *INE - Población a nivel de sección censal*. URL: https://www.ine.es/censos2011_datos/cen11_datos_resultados_seccen.htm (visitado 05-03-2021).

— (2021b). *Rejilla censo 1 km*. URL: https://www.ine.es/censos2011_datos/cen11_datos_resultados_rejillas.htm (visitado 05-03-2021).

INSPIRE Directive (2007). «Directive 2007/2/EC of the European Parliament and of the Council of 14 March 2007 establishing an Infrastructure for Spatial Information in the European Community (INSPIRE)». En: *Published in the official Journal on the 25th April* 43.

INSPIRE Thematic Working Group Coordinate Reference Systems & Geographical Grid Systems (2014). «D2.8.I.2Data Specification on Geographical Grid Systems - Technical Guidelines». En: *INSPIRE*.

Instituto de Cartografía y Estadística de Andalucía (IECA) (2021a). *Datos Espaciales de Referencia de Andalucía (DERA)*. URL: https://www.juntadeandalucia.es/institutodeestadisticaycartografia/DERA/ (visitado 15-03-2021).

— (2021b). *Datos Espaciales de Referencia de Andalucía (DERA): MDT*. URL: https://www.juntadeandalucia.es/institutodeestadisticaycartografia/DERA/g01.htm (visitado 15-03-2021).

— (2021c). *Página oficial*. URL: http://www.juntadeandalucia.es/institutodeestadisticaycartograf (visitado 15-03-2021).

— (2021d). *Portal Callejero Digital Unificado de Andalucía*. URL: https://www.callejerodeandalucia.es/ (visitado 05-03-2021).

IV Jornadas Ibéricas de Infraestructuras de Datos Espaciales (2013).

Jaramillo, D., Nguyen, D. V. y Smart, R. (2016). «Leveraging microservices architecture by using Docker technology». En: *SoutheastCon 2016*, págs. 1-5. DOI: 10.1109/SECON.2016.7506647.

Jobs, S. (2010). *Thoughts on Flash*. URL: https://web.archive.org/web/20170615060422/https://www.apple.com/hotnews/thoughts-on-flash/ (visitado 05-03-2021).

Jong, Jos de et al. (2021). *mathjs*. URL: https://mathjs.org/ (visitado 05-03-2021).

Jung, M., Youn, S., Bae, J. y Choi, Y. (2015). «A Study on Data Input and Output performance Comparison of MongoDB and PostgreSQL in the Big Data Environment». En: *2015 8th International Conference on Database Theory and Application (DTA)*, págs. 14-17. DOI: 10.1109/DTA.2015.14.

Kanevski, M, Pozdnukhov, A y Timonin, V (2008). «Machine learning algorithms for geospatial data. Applications and software tools». En: *EMSs 2008: International Congress on Environmental Modelling and Software*.

Karau, Holden, Konwinski, Andy, Wendell, Patrick y Zaharia, Matei (2015). *Learning Spark*. O'Reilly. ISBN: 978-1-449-35862-4.

Kimerling, Jon A, Sahr, Kevin, White, Denis y Song, Lian (1999). «Comparing geometrical properties of global grids». En: *Cartography and Geographic Information Science* 26.4, págs. 271-288.

Kirk, Andy (2016). *Data Visualisation: A Handbook for Data Driven Design*. SAGE. ISBN: 978-1-4739-1214-4.

Knapp, Ellen M. (1980). «SPATIAL DATA INTEGRATION». En: *Map Data Processing*. Ed. por HERBERT FREEMAN y GOFFREDO G. PIERONI. Academic Press,

págs. 47-61. ISBN: 978-0-12-267180-7. DOI: https://doi.org/10.1016/B978-0-12-267180-7.50008-6. URL: https://www.sciencedirect.com/science/article/pii/B9780122671807500086.

Lakhani, Karim y Wolf, Robert (sep. de 2003). «Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects». En: *Perspectives on Free and Open Source Software*. DOI: 10.2139/ssrn.443040.

Lakshman, Avinash y Malik, Prashant (abr. de 2010). «Cassandra: A Decentralized Structured Storage System». En: *SIGOPS Oper. Syst. Rev.* 44.2, págs. 35-40. ISSN: 0163-5980. DOI: 10.1145/1773912.1773922. URL: https://doi.org/10.1145/1773912.1773922.

LAS Working Group (LWG) Contributors (2021). *ASPRS LAS Specification*. URL: https://github.com/ASPRSorg/LAS (visitado 05-03-2021).

Law, Michael y Collins, Amy (2019). *Getting to know ArcGIS PRO*. Esri press.

Leiner, Barry M., Cerf, Vinton G., Clark, David D., Kahn, Robert E., Kleinrock, Leonard, Lynch, Daniel C., Postel, Jon, Roberts, Larry G. y Wolff, Stephen (oct. de 2009). «A Brief History of the Internet». En: *SIGCOMM Comput. Commun. Rev.* 39.5, págs. 22-31. ISSN: 0146-4833. DOI: 10.1145/1629607.1629613. URL: https://doi.org/10.1145/1629607.1629613.

Leszczynski, Agnieszka y Crampton, Jeremy (2016). «Introduction: Spatial big data and everyday life». En: *Big Data & Society* 3.2, pág. 2053951716661366.

Lewis, Grace (2010). «Basics about cloud computing». En: *Software engineering institute carniege mellon university, Pittsburgh*.

«Ley 14/2010, de 5 de julio, sobre las infraestructuras y los servicios de información geográfica en España» (2010). En: *BOE núm. 163, de 06/07/2010*.

Likas, Aristidis, Vlassis, Nikos y Verbeek, Jakob J. (2003). «The global k-means clustering algorithm». En: *Pattern Recognition* 36.2. Biometrics, págs. 451-461. ISSN: 0031-3203. DOI: https://doi.org/10.1016/S0031-3203(02)00060-2. URL: https://www.sciencedirect.com/science/article/pii/S0031320302000602.

Liu, Zhen Hua, Hammerschmidt, Beda, McMahon, Doug, Liu, Ying y Chang, Hui Joe (2016). «Closing the Functional and Performance Gap between SQL and NoSQL». En: *Proceedings of the 2016 International Conference on Management of Data*. SIGMOD '16. San Francisco, California, USA: Association for Computing Machinery, págs. 227-238. ISBN: 9781450335317. DOI: 10.1145/2882903.2903731. URL: https://doi.org/10.1145/2882903.2903731.

Lutz, Mark (2010). *Programming Python, 4th Edition*. O'Reilly Media Inc. ISBN: 9780596158101.

Ma, Ting, Zhou, Chenghu, Xie, Yichun, Qin, Biao y Ou, Yang (2009). «A discrete square global grid system based on the parallels plane projection». En: *International Journal of Geographical Information Science* 23.10, págs. 1297-1313.

Maclean, Malcolm (2014). *Leaflet Tips and Tricks: Interactive Maps Made Easy*. Lean Publishing.

MacWright, Tom et al. (2021). *simple statistics*. URL: https://simplestatistics.org/ (visitado 05-03-2021).

Mantyla, Dan (2015). *Functional Programming in JavaScript*. Packt Publishing. ISBN: 978-1-78439-822-4.

Mapbox (2021a). *Mapbox*. URL: https://www.mapbox.com/ (visitado 05-03-2021).

— (2021b). *Mapbox API*. URL: https://www.mapbox.com/api (visitado 05-03-2021).

Martínez Murillo, J. F., Hueso González, P. y Ruíz Sinoga, J. D. (2017). «Topsoil moisture mapping using geostatistical techniques under different Mediterranean climatic conditions». En: *Science of The Total Environment* 595, págs. 400-412.

Mateos, Pablo (2013). «Geovisualización de la población: Nuevas tendencias en la web social». En: *Investigaciones Geográficas (Esp)* 60, págs. 87-100.

Meng, Michael, Steinhardt, Stephanie y Schubert, Andreas (2018). «Application programming interface documentation: what do software developers want?» En: *Journal of Technical Writing and Communication* 48.3, págs. 295-330.

Merchán, J. I., Moreno, J. A. y Villar, A. (nov. de 2014). «Callejero digital de Andalucía Unificado». En: V Jornadas Ibéricas de Infra-estruturas de Dados Espaciais. V Jornadas Ibéricas de Infra-estruturas de Dados Espaciais.

Metternicht, Graciela (2006). «Consideraciones acerca del impacto de Google Earth en la valoración y difusión de los productos de georrepresentación». En: *GeoFocus. Revista Internacional de Ciencia y Tecnología de la Información Geográfica* 6, págs. 1-10.

Microsoft Corp. (2021a). *GitHub*. URL: `https://github.com/` (visitado 05-03-2021).

— (2021b). *Power BI*. URL: `https://powerbi.microsoft.com/` (visitado 05-03-2021).

Miller, Christopher C (2006). «A beast in the field: The Google Maps mashup as GIS/2». En: *Cartographica: The International Journal for Geographic Information and Geovisualization* 41.3, págs. 187-199.

Mora-García, RT y Marti-Ciriquian, P (2015). «Desagregación poblacional a partir de datos catastrales». En: *Análisis espacial y representación geográfica: innovación y aplicación*, págs. 305-314.

Moreira Madueño, J. M. y Ojeda Zújar, J. (1992). «Andalucía, una visión inédita desde el Espacio». En: *Agencia de Medio Ambiente y Agua de Andalucía. Junta de Andalucía.*, pág. 214.

Netek, Rostislav, Masopust, Jan, Pavlicek, Frantisek y Pechanec, Vilem (2020). «Performance Testing on Vector vs. Raster Map Tiles—Comparative Study on Load Metrics». En: *ISPRS International Journal of Geo-Information* 9.2, pág. 101.

Neteler, M., Bowman, M.H., Landa, M. y Metz, M. (2012). «GRASS GIS: a multi-purpose Open Source GIS». En: *Environmental Modelling & Software* 31, págs. 124-130. DOI: `10.1016/j.envsoft.2011.11.014`.

Newham, Cameron y Rosenblatt, Bill (2005). *Learning the bash shell: Unix shell programming*. .ºReilly Media, Inc."

Newton, Thomas, Villarreal, Oscar y Verspohl, Lars (2017). *Learning D3.js 4 Mapping*. Packt Publishing. ISBN: 978-1-78728-017-5.

Nguyen, TT (2009). «Indexing PostGIS databases and spatial query performance evaluations». En: *International Journal of Geoinformatics* 5.3, pág. 1.

Nicolai, R y Simensen, G (2008). «The New EPSG Geodetic Parameter Registry». En: *70th EAGE Conference and Exhibition incorporating SPE EUROPEC 2008*. European Association of Geoscientists & Engineers, cp-40.

Noguero Hernández, M.D., Vallejo Villalta, L. I., Ramírez Moreno, E. y Ramírez Torres, A. (2016). «Identificación del espacio residencial en Andalucía a partir de datos catastrales». En: *Aplicaciones de las Tecnologías de la Información Geográfica para el desarrollo económico sostenible. XVII Congreso Nacional de Tecnologías de la Información Geográfica. Málaga*, págs. 421-431.

Obe, Regina O y Hsu, Leo S (2017). *PostgreSQL: Up and Running: a Practical Guide to the Advanced Open Source Database*. ."O'Reilly Media, Inc."

Ofoeda, Joshua, Boateng, Richard y Effah, John (2019). «Application programming interface (API) research: A review of the past to inform the future». En: *International Journal of Enterprise Information Systems (IJEIS)* 15.3, págs. 76-95.

OGC (2021a). *Simple Feature Access - Part 1: Common Architecture*. URL: https://www.ogc.org/standards/sfa (visitado 05-03-2021).

— (2021b). *Web Feature Service (WFS) OGC standard*. URL: https://www.ogc.org/standards/wfs (visitado 05-03-2021).

Ojeda Zújar, J. (2010). «Geovisualización: espacio, tiempo y territorio». En: *Ciudad y Territorio* XLII, págs. 165-166.

— (jun. de 2015). *Georreferenciación, caracterización estadística y estrategias de difusión residencial del espacio residencial en Andalucía*. Proyecto. Proyecto convocatoria FEDER de la Consejería de Vivienda y Fomento de la Junta de Andalucía.

Ojeda Zújar, J., Álvarez Francoso, J. I., Díaz Cuevas, P., Prieto Campos, A. y Cabrera, A. (2013). «Instrumentos para el conocimiento, la difusión y gobernanza de las zonas litorales: visores 3D (desktop y web). Costa de Andalucía». En: *Geotemas, 14, 31-34*.

Ojeda Zújar, J., Álvarez Francoso, J. I., Martín Cajaraville, D. y Fraile Jurado, P. (2009). «El uso de las TIG para el cálculo del indice de vulnerabilidad costera (CVI) ante una potencial subida del nivel del mar en la costa andaluza (España)». En: *GeoFocus. Revista Internacional de Ciencia y Tecnología de la Información Geográfica* 9, págs. 83-100.

Ojeda Zújar, J., Díaz Cuevas, María del Pilar, Álvarez Francoso, J. I., Pérez Alcántara, Juan Pedro y Prieto Campos, Antonio (2015). «Geoportales y geovisores web: Un nuevo entorno colaborativo para la producción, acceso y difusión de la información geográfica». En: *Análisis espacial y representación geográfica: innovación y aplicación. XXIV Congreso de la Asociación de Geógrafos Españoles (2015), p 777-786*. Departamento de Geografía y Ordenación del Territorio, Universidad de Zaragoza.

Ojeda Zújar, J., Fraile Jurado, P. y Álvarez Francoso, J. I. (2021). «Sea level rise inundation risk assessment in residential cadastral parcels along the Mediterranean Andalusian coast». En: *Cuadernos de Investigación Geográfica* 47. DOI: http://doi.org/10.18172/cig.4744.

Ojeda Zújar, J., Sanchez, E., Fernandez-Palacios, A. y Moreira, J.M. (1995). «Study of the dynamics of estuarine and coastal waters using remote sensing: the Tinto-Odiel estuary, SW Spain». En: *Journal of Coastal Conservation* 1.2, págs. 109-118. DOI: https://doi.org/10.1007/BF02905119.

Ojeda Zújar, J. y Vallejo Villalta, L. I. (1995). «La Flecha de El Rompido: análisis morfométrico y modelos de evolución durante el periodo 1943-1991». En: *Revista de la Sociedad Geológica de España* 8 (3), págs. 229-237.

Ojeda Zújar, J., Vallejo Villalta, L. I., Hernández Calvento, L. y Álvarez Francoso, J. I. (2007). «Fotogrametría digital y LIDAR como fuentes de información en geomorfología litoral (marismas mareales y sistemas dunares): el potencial de su análisis espacial a través de SIG». En: *BAGE: Boletín de la Asociación de Geógrafos Españoles* 44, págs. 215-233. URL: http://documents.mx/documents/fotogrametria-digital-y-lidar-como-fuentes-de-informacion-en-geomorfologia.html.

Olaya, V. (2016). *Sistemas de Información Geográfica*. Autoedición. ISBN: 9781530295944.

Oliphant, Travis E. (2006). *Guide to NumPy*. NumPy Collaborators.

Oluwatosin, Haroon Shakirat (2014). «Client-server model». En: *IOSRJ Comput. Eng* 16.1, págs. 2278-8727.

Openshaw, S. y Taylor, P. (1979). «A million or so correlation coefficients, three experiments on the modifiable areal unit problem». En: *Statistical applications in the spatial science*, págs. 127-144.

— (1981). «The modifiable areal unit problem». En: *Quantitative geography: A British view*, págs. 60-69.

Paneque, Pilar, Lafuente, Regina y Vargas, Jesús (2018). «Public attitudes toward water management measures and droughts: A study in Southern Spain». En: *Water* 10.4, pág. 369.

Paszke, Adam, Gross, Sam, Massa, Francisco, Lerer, Adam, Bradbury, James, Chanan, Gregory, Killeen, Trevor, Lin, Zeming, Gimelshein, Natalia, Antiga, Luca, Desmaison, Alban, Kopf, Andreas, Yang, Edward, DeVito, Zachary, Raison, Martin, Tejani, Alykhan, Chilamkurthy, Sasank, Steiner, Benoit, Fang, Lu, Bai, Junjie y Chintala, Soumith (2019). «PyTorch: An Imperative Style, High-Performance Deep Learning Library». En: *Advances in Neural Information Processing Systems 32*. Ed. por H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox y R. Garnett. Curran Associates, Inc., págs. 8024-8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Pedregosa, Fabian, Varoquaux, Gaël, Gramfort, Alexandre, Michel, Vincent, Thirion, Bertrand, Grisel, Olivier, Blondel, Mathieu, Prettenhofer, Peter, Weiss, Ron, Dubourg, Vincent et al. (2011). «Scikit-learn: Machine learning in Python». En: *The Journal of machine Learning research* 12, págs. 2825-2830.

Peng, Dunlu, Cao, Lidong y Xu, Wenjie (2011). «Using JSON for data exchanging in web service applications». En: *Journal of Computational Information Systems* 7.16, págs. 5883-5890.

Pereira, Francisco Câmara y Borysov, Stanislav S. (2019). «Chapter 2 - Machine Learning Fundamentals». En: *Mobility Patterns, Big Data and Transport Analytics*. Ed. por Constantinos Antoniou, Loukas Dimitriou y Francisco Pereira. Elsevier, págs. 9-29. ISBN: 978-0-12-812970-8. DOI: https://doi.org/10.1016/B978-0-12-812970-8.00002-6. URL: https://www.sciencedirect.com/science/article/pii/B9780128129708000026.

Pérez Alcántara, J. P., Díaz Cuevas, P., Álvarez Francoso, J. I. y Ojeda Zújar, J. (2016). «Métodos de adscripción y tratamiento espacial para la generación y visualización de indicadores de vivienda (GRID) a través de Catastro». En: *Aplicaciones de las Tecnologías de la Información Geográfica para el desarrollo económico sostenible. XVII Congreso Nacional de Tecnologías de la Información Geográfica.*, págs. 224-235.

Pérez Alcántara, J. P., Ojeda Zújar, J., Cuevas, P. y Álvarez Francoso, J. I. (2017). «Integración de datos poblacionales y catastrales en estructuras GRID: Primeros resultados para el espacio residencial en Andalucía». En: *Naturaleza, territorio y ciudad en un mundo global*, págs. 1619-1628. DOI: https://doi.org/10.15366/ntc.2017.

Peterson, Perry R (2016). «Discrete global grid systems». En: *International Encyclopedia of Geography: People, the Earth, Environment and Technology: People, the Earth, Environment and Technology*, págs. 1-10.

Petković, Dušan (2017). «JSON integration in relational database systems». En: *Int J Comput Appl* 168.5, págs. 14-19.

Pilicita Garrido, A., Borja López, Y. y Gutiérrez Constante, G. (2020). «Rendimiento de MariaDB y PostgreSQL». En: *Revista Científica Y Tecnológica UPSE* 7.2, págs. 09-16. DOI: https://doi.org/10.26423/rctu.v7i2.538.

Pirtle, Mitch (2010). *MongoDB for web development*. Boston, Mass. London: Addison-Wesley. ISBN: 9780321705334.

Portal CartoCiudad (2021). URL: https://www.cartociudad.es/ (visitado 05-03-2021).

Postman Contributors (2021). *Postman - A Platform for Collaborative API Development*. URL: https://www.postman.com/ (visitado 05-03-2021).

Prieto Campos, A., Díaz Cuevas, P., Fernández Núñez, M. y Ojeda Zújar, J. (2018). «Methodology for Improving the Analysis, Interpretation, and Geo-Visualisation of Erosion Rates in Coastal Beaches—Andalusia, Southern Spain». En: *Geosciences* 8.9, pág. 335.

PROJ Contributors (2021). *PROJ coordinate transformation software library*. Open Source Geospatial Foundation. URL: https://proj.org/.

Project Jupyter Contributors (2021). *A gallery of interesting Jupyter Notebooks*. URL: https://github.com/jupyter/jupyter/wiki/A-gallery-of-interesting-Jupyter-Notebooks#earth-science-and-geo-spatial-data (visitado 05-03-2021).

Purss, M., Gibb, R., Samavati, F., Peterson, P. y Ben, J. (2016). «The OGC Discrete Global Grid System core standard: A framework for rapid geospatial integration». En: *IEEE International Geoscience And Remote Sensing Symposium (IGARSS)*. DOI: 10.1109/igarss.2016.7729935. URL: https://www.sciencedirect.com/science/article/pii/1049965292900325.

pyproj Contributors (2021). *pyproj*. URL: https://pyproj4.github.io/pyproj/ (visitado 05-03-2021).

Raschka, S. y Mirjalili, V. (2019). *Python Machine Learning: Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow 2*. Packt Publishing. ISBN: 978-1-78995-575-0.

REDIAM (2021). *Red de Información Ambiental de Andalucía*. URL: http://www.juntadeandalucia.es/medioambiente/site/rediam/ (visitado 05-03-2021).

Reig, E., Goerlich, F.J. y Cantarino, I. (2016). «Delimitación de áreas rurales y urbanas a nivel local: Demografía, coberturas del suelo y accesibilidad». En: *Fundación BBVA* 25, pág. 133.

Reinhardt, Robert (2007). *Macromedia Flash 8 Bible*. John Wiley & Sons.

Rey, Sergio, Anselin, Luc, Li, Xun, Pahle, Robert, Laura, Jason, Li, Wenwen y Koschinsky, Julia (jun. de 2015). «Open geospatial analytics with PySAL». En: *ISPRS International Journal of Geo-Information* 4, págs. 815-836. DOI: 10.3390/ijgi4020815.

Rigaux, Philippe, Scholl, Michel y Voisard, Agnès (2002). *Spatial Databases with Application to GIS*. Morgan Kaufmann Publishers. ISBN: 978-1-55860-588-6.

Robinson, A. H., Sale, R. D., Morrison, J. L. y Muehrcke, P. C. (1987). *Elementos de cartografía*. Ediciones Omega. ISBN: 84-282-0768-2.

Rodriguez, Alex (2008). «Restful web services: The basics». En: *IBM developerWorks* 33, pág. 18.

Rodríguez Galiano, V. F., Sánchez Castillo, M., Dash, J., Atkinson, P. M. y Ojeda Zújar, J. (2016). «Modelling interannual variation in the spring and autumn land surface phenology of the European forest». En: *Biogeosciences* 13.11, págs. 3305-3317.

Ruíz Sinoga, J. D., Romero Díaz, A., Martínez Murillo, J. F., Gabarrón Galeote, M. A. et al. (2015). «Incidencia de la dinámica pluviométrica en la degradación del suelo. Sur de España». En: *Boletín de la Asociación de Geógrafos Españoles* 68, págs. 177-204. DOI: https://doi.org/10.21138/bage.1858.

Ryza, Sandy, Laserson, Uri, Owen, Sean y Wills, Josh (2015). *Advanced Analytics with Spark: Patterns for Learning from Data at Scale*. O'Reilly. ISBN: 978-1-491-91276-8.

Sahr, Kevin (2011). «Hexagonal discrete global grid systems for geospatial computing». En: *Archiwum Fotogrametrii, Kartografii i Teledetekcji* 22, págs. 363-376.

Sahr, Kevin y White, Denis (1998). «Discrete global grid systems». En: *Computing Science and Statistics*, págs. 269-278.

Sahr, Kevin, White, Denis y Kimerling, A Jon (2003). «Geodesic discrete global grid systems». En: *Cartography and Geographic Information Science* 30.2, págs. 121-134.

Sánchez Carnero, N., Ojeda Zújar, J., Rodríguez Pérez, D. y Márquez Pérez, J. (2014). «Assessment of different models for bathymetry calculation using SPOT multispectral images in a high-turbidity area: the mouth of the Guadiana Estuary». En: *International Journal of Remote Sensing* 35.2, págs. 493-514.

Sayar, Ahmet, Pierce, Marlon y Fox, Geoffrey (2005). «OGC compatible geographical information systems web services». En: *Indiana Computer Science Report TR610*.

Shapiro, F.R., ed. (2006). *The Yale Book of Quotations*. Yale University Press. ISBN: 9780300107982.

Sherman, Gary (2014). *The PyQGIS Programmer's Guide: Extending QGIS 2.x with Python*. Locate Press. ISBN: 978-0-9894217-2-0.

Smith, Gregory (2010). *PostgreSQL 9.0: High Performance*. Packt Publishing Ltd.

Stack Exchange Inc. (2021). *Stack Overflow's 2020 Developer Survey*. URL: https://insights.stackoverflow.com/survey/2020 (visitado 04-03-2021).

Stancu-Mara, Sorin y Baumann, Peter (2008). «A Comparative Benchmark of Large Objects in Relational Databases». En: *Proceedings of the 2008 International Symposium on Database Engineering & Applications*. IDEAS '08. Coimbra, Portugal: Association for Computing Machinery, págs. 277-284. ISBN: 9781605581880. DOI: 10.1145/1451940.1451980. URL: https://doi.org/10.1145/1451940.1451980.

Stenberg, Daniel (2021). *curl*. URL: https://curl.se/ (visitado 05-03-2021).

Stonebraker, Michael (2010). «SQL databases v. NoSQL databases». En: *Communications of the ACM* 53.4, págs. 10-11.

Summerfield, Mark (2010). *Programación en Python 3*. Anaya Multimedia. ISBN: 978-84-415-2613-6.

Szalay, Alexander S., Gray, Jim, Fekete, George, Kunszt, Peter Z., Kukol, Peter y Thakar, Ani (2007). «Indexing the sphere with the hierarchical triangular mesh». En: *arXiv preprint cs/0701164*.

Teorey, Toby J., Yang, Dongqing y Fry, James P. (jun. de 1986). «A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model». En: *ACM Comput. Surv.* 18.2, págs. 197-222. ISSN: 0360-0300. DOI: 10.1145/7474.7475. URL: https://doi.org/10.1145/7474.7475.

Tiobe Software BV (2021). *Tiobe Programming Language Index*. URL: https://www.tiobe.com/tiobe-index/ (visitado 04-03-2021).

Tobler, Waldo y Chen, Zi-tan (1986). «A quadtree for global information storage». En: *Geographical Analysis* 18.4, págs. 360-371.

Togootogtokh, Enkhtogtokh y Amartuvshin, Amarzaya (2018). *Deep Learning Approach for Very Similar Objects Recognition Application on Chihuahua and Muffin Problem*. arXiv: 1801.09573 [cs.AI].

Toro Bonilla, J.M. (2014). «La informática, una ciencia joven muy importante». En: Lección Inaugural leída en la Solemne Apertura del Curso Académico 2014-2015. Secretariado de Publicaciones de la Universidad de Sevilla, pág. 64. ISBN: 978-84-472-1576-8.

UNESCO (2020). «Preliminary study of the technical, financial and legal aspects of the desirability of a UNESCO recommendation on Open Science». En: ed. por UNESCO. UNESCO 40th General Conference, 2019. UNESCO.

Van Vliet, Hans, Van Vliet, Hans y Van Vliet, JC (2008). *Software engineering: principles and practice*. Vol. 13. John Wiley & Sons Hoboken, NJ.

Vargas, Jesús y Paneque, Pilar (2019). «Challenges for the integration of water resource and drought-risk management in Spain». En: *Sustainability* 11.2, pág. 308.

Vassiliadis, Panos, Simitsis, Alkis y Skiadopoulos, Spiros (2002). «Conceptual modeling for ETL processes». En: *Proceedings of the 5th ACM international workshop on Data Warehousing and OLAP*, págs. 14-21.

Vitolo, Claudia, Elkhatib, Yehia, Reusser, Dominik, Macleod, Christopher JA y Buytaert, Wouter (2015). «Web technologies for environmental Big Data». En: *Environmental Modelling & Software* 63, págs. 185-198.

Wilson, Matt, Masetti, Giuseppe y Calder, Brian (mayo de 2016). «NOAA QC Tools: Origin, Development, and Future». En:

Wright, D., Goodchild, M. F. y Proctor, J. (1997). «GIS: Tool or Science? Demystifying the Persistent Ambiguity of GIS as "Tool" Versus "Science"». En: *Annals of the Association of American Geographers* 87 (2), págs. 346-362.

Xing, Jin, Sieber, Renee y Roche, Stéphane (2020). «Rethinking spatial tessellation in an era of the smart city». En: *Annals of the American Association of Geographers* 110.2, págs. 399-407.

Yang, Yu (2013). *A study of pattern recognition of Iris flower based on Machine Learning*.

Yau, Nathan (2011). *Visualize This: The FlowingData Guide to Design, Visualization, and Statistics*. Wiley. ISBN: 978-0-470-94488-2.

Zhang, Hao, Tudor, Bogdan Marius, Chen, Gang y Ooi, Beng Chin (jun. de 2014). «Efficient In-Memory Data Management: An Analysis». En: *Proc. VLDB Endow.* 7.10, págs. 833-836. ISSN: 2150-8097. DOI: 10.14778/2732951.2732956. URL: https://doi.org/10.14778/2732951.2732956.

V.2 Repositorios GitHub

En este anexo se listan los distintos repositorios *Git* en los que se guarda el código y los datos generados en esta tesis.

Dado que la ubicación de los repositorios puede cambiar en el tiempo, remitimos al lector a `https://cell.37north.io`, donde se mantiene actualizado el catálogo de recursos descrito en este anexo con sus vínculos actualizados.

Git es un sistema de control de versiones de código. Es una herramienta fundamental para el desarrollo de *software*, ya que permite a un grupo de desarrolladores trabajar a la vez sobre una misma base de código. El programa hace un seguimiento exhaustivo de los cambios, a nivel de línea en cada fichero. Permite implementar flujos de trabajo muy sofisticados, imprescindibles para progresar simultáneamente, evitando conflictos, en varias funcionalidades nuevas y/o corrección de errores, que pueden estar al cargo de desarrolladores distintos. A medida que esas líneas de trabajo van concluyendo, *Git* proporciona un entorno seguro para unir el trabajo realizado en cada una de ellas en un todo cohesionado.

Todos los proyectos de *software* utilizan un sistema de estas características para su desarrollo. *Git* es *Software Libre*, y alrededor de él se han creado una serie de plataformas comerciales para su explotación en la nube. Estas plataformas implementan alrededor de los flujos de trabajo de la herramienta todo un ecosistema de funcionalidad orientado a la coordinación de grupos de desarrolladores: gestión de tareas y *bugs*, sistemas de discusión técnica, etc. Las plataformas más populares de este tipo son *GitHub*, actualmente propiedad de *Microsoft* (Microsoft Corp., 2021a), y *GitLab* (GitLab Inc., 2021), ocupando quizás *BitBucket* (Atlassian, 2021) el tercer lugar. Alrededor de este ecosistema se ha organizado, además, un sistema de prestigio entre los desarrolladores similar al de las revisiones *peer-to-peer*. Hoy en día, tener un perfil personal de desarrollador en *GitHub* interesante es uno de los factores que se miran en la contratación de desarrolladores en ciertos sectores de la industria del *software*.

V.2.1. Repositorios de datos

Los repositorios de datos presentan un problema para *Git*. El objetivo de diseño de *Git* es el control de repositorios de código. Los repositorios de código están compuestos de sencillos ficheros de texto, que es en lo que se codifican los listados de código de los programas. Como tales, no suelen ser muy grandes, por lo que por lo general las plataformas como *GitHub* no admiten ficheros de más de 50 *megabytes*. Además, *Git* hace

un excelente trabajo controlando cambios en ficheros de texto, pero el control de cambios en fichero binarios, como los que se usan como fuentes de datos (ficheros de tipo *SHAPEFILE*, *GeoTIFF*, etc.), es mucho más limitado, por la propia naturaleza del formato.

Por estas dos razones, hemos de diferenciar, en los repositorios de datos, dos tipos de ficheros bien distintos:

- por un lado, los ficheros que codifican los flujos de datos: *scripts* en distintos lenguajes de programación, consultas *SQL*, etc. Estos ficheros son código y *Git / GitHub* hacen un gran trabajo identificando y controlando su historial de cambios;
- por otro, los ficheros que son fuentes de datos originales. Estos ficheros son a menudo binarios y grandes, con lo que se salen del ámbito de utilidad de *Git*.

Para la gestión de estos últimos se utiliza otra aplicación llamada *DVC* (*Data Version Control*) (DVC Contributors, 2021). Fuertemente inspirada en el flujo de trabajo de *Git*, *DVC* es el "*Git* de los datos". Pero a diferencia de *Git*, no tiene plataformas en la nube asociadas aún[1], por lo que se instala en un servidor local.

Los repositorios que se describen a continuación, por lo tanto, tienen un enfoque mixto:

- para el código de análisis y *ETL* sobre los datos, utiliza *Git*;
- los datos están gestionados por un servidor local *DVC*.

Los repositorios son los siguientes:

- **malkab-phd-data-cell_raw_data:** este repositorio no tiene datos en sí, pero actua como base operativa para los demás al gestionar la creación de la base de datos de origen;
- **malkab-phd-data-proyecto_nacional_data:** la base de datos catastral;
- **malkab-phd-data-context_data:** gestión y tratamiento de los datos de contexto y el MDT;
- **malkab-phd-data-hic:** gestión y tratamiento de los Hábitats de Interés Comunitario;
- **malkab-phd-data-poblacion:** gestión y tratamiento de los datos de población.

V.2.2. Repositorios de la plataforma *Cell*

El código que hace funcionar a la plataforma está muy repartido en diversos repositorios. Hay que tener en cuenta que un repositorio *Git* debe centrarse en el diseño de una pieza de *software* autónomo, como por ejemplo una librería. Por ello, vamos a encontrar en esta sección repositorios de cuatro tipos:

[1] En el momento de escribir esto, hay una en pruebas beta accesible por invitación.

- **repositorios de imágenes *Docker*:** *Docker* es, como *Git*, una herramienta imprescindible en el desarrollo de *software* y de proceso de datos, ya que garantiza la creación de contextos de ejecución reproducibles y homogéneos en todos los niveles del desarrollo[2]. En el desarrollo de la plataforma se usan varias de ellas, de propósito múltiple, tanto en la fase de desarrollo como de producción;

- **repositorios de librerías:** librerías desarrolladas en el contexto de creación de la plataforma pero que tienen una vocación de utilidad más allá de la misma. Estas librerías pueden ser utilizadas en otros proyectos;

- **repositorio *Cell*:** el repositorio de la plataforma propiamente dicha;

- **repositorios accesorios:** repositorios utilizados para otras tareas vinculadas con esta tesis.

Repositorios de imágenes *Docker*

Una imagen *Docker* es una definición compacta de un entorno de ejecución *Linux*. Este entorno de ejecución carga, de forma aislada del resto del sistema operativo anfitrión, las librerías y programas destinados a realizar un trabajo computacional concreto. En cierto sentido, el concepto es similar al de las máquinas virtuales pero ahorrándose el sistema operativo[3]. Estas imágenes son plantillas con las que después se crean *contenedores*, que son instancias de estas imágenes funcionando sobre un sistema operativo y realizando un trabajo concreto. Este concepto ayuda a los *DevOps* a empaquetar el repertorio de programas y librerías base que un nuevo desarrollo necesita para funcionar en un cómodo formato que el resto del equipo puede utilizar, aumentando dramáticamente el nivel de reproducibilidad de una infraestructura de *software*. *Docker* viene acompañado, además, de un ecosistema de repositorios (similar al concepto ya comentado de *Git*) de imágenes en la nube que los miembros de un equipo pueden descargar automáticamente para crear contenedores.

Estas imágenes, además, se pueden extender. A partir de una imagen base, los *DevOps* crean nuevas imágenes derivadas modificándolas puntualmente o ampliando sus capacidades. Gracias al ingenioso sistema de ficheros por capas con las que se guardan las imágenes *Docker*, la imagen madre no se repite en la hija, sino que la hija hereda toda la información de la madre y modifica o extiende puntualmente lo que necesite para cumplir su papel. De hecho, todas las imágenes *Docker* descienden de una única imagen primordial que configura el funcionamiento básico de todas sus decenas de miles de descendientes. De esta manera, no hay que estar constantemente reinventando la rueda, puesto que siempre se

[2] Antes de la llegada de este tipo de técnicas, los despligues de *software* desde el entorno de desarrollo, donde se programa, hasta el entorno de producción, es decir, los servidores donde corre el producto final, eran procesos delicados muy proclives a los fallos. La frase "pues a mí en mi entorno de desarrollo me funciona" es quizás una de las más exasperantes que un *DevOps* (los profesionales al cargo de estos procesos) podía escuchar de un programador. La llegada de las tecnologías de máquinas virtuales y contenerización, como *Docker*, ha facilitado muchísimo este proceso.

[3] Podríamos decir que una imagen *Docker* se comporta como una máquina virtual que coge "prestado" el sistema operativo de su máquina anfitriona, modificándolo a su gusto.

parte de imágenes creadas por especialistas, muy bien diseñadas, como base para nuevos desarrollos.

Las imágenes *Docker* utilizadas en el desarrollo de esta tesis son:

- de producción propia:

  - **docker-grass:** una imagen bastante densa destinada a flujos de trabajo TIG. Incluye instalaciones de *GDAL*, *PostGIS*, *GRASS* y las librerías típicas *Python* para TIG, *Machine Learning* y proceso de datos en general. Esta imagen es muy grande, de varios *gigabytes* de datos, contradiciendo el principio *Docker* de hacer imágenes lo más pequeñas posibles y destinadas a un sólo fin y servicio (esta incluye muchos), pero es una herramienta fantástica para desplegar rápidamente todo el *software* TIG necesario para el proceso de información en el día a día;

  - **docker-nodejs-dev:** la imagen principal para desarrollar en el entorno *Node.js* con *TypeScript* y *JavaScript*. Es la imagen sobre la que se desarrollan las librerías accesorias como *libcellbackend*, la *API*, el controlador, los trabajadores de adscripción y los geovisores;

  - **docker-ngingx-angular:** una imagen de producción (es decir, siguiendo el patrón *Docker* de imagen pequeña y centrada en una sola tarea) para el despliegue de servidores *web NGINX* configurados para la ejecución de aplicaciones *Angular*;

  - **ubuntu-general-purpose:** una imagen pequeña, con algunos paquetes útiles de la distribución *Ubuntu* instalados, para realizar tareas de caracter general y de mantenimiento, como por ejemplo acceder y examinar de forma cómoda volúmenes de datos *Docker*;

  - **docker-public-python:** una imagen de desarrollo y producción *Python*. Es una imagen ligera puesto que se vale del gestor de entornos *virtualenv* de *Python* para instalar, en desarrollo, las librerías *Python* que convengan en cada momento, sin ensuciar la imagen en sí. Con ella se desarrollan los trabajadores de *Machine Learning*;

  - **docker-public-postgis:** una imagen crítica en la plataforma, puesto que gestiona los microservicios de bases de datos *PostGIS*;

  - **docker-public-text_workflows:** una imagen de propósito general para el procesamiento de documentos LaTeX. Esta imagen es la que ha hecho posible la escritura de este documento;

- producidas por terceros:

  - **redis:** la imagen oficial del *software Redis*, el microservicio de memoria compartida.

Repositorios de librerías

Los siguientes repositorios se centran en el desarrollo de librerías desarrolladas para apoyar aspectos concretos de la plataforma, pero cuyo diseño es genérico, pudiendo ser utilizadas en cualquier otro proyecto:

- **ts-utils:** un conjunto de funcionalidades de uso común para el lenguaje *TypeScript*, ejecutables tanto en el entorno del *front* como del *backend*;
- **appian:** una librería escrita sobre el *framework Node.js Express*, destinado a la creación de *API*, para estandarizar las respuestas y los procedimientos de mapeo de objetos desde la base de datos a una *API REST*. Ayuda a crear *APIs* de una más estándar y rápida;
- **rxpg:** una librería *TypeScript* que encapsula el *driver* de la *PostgreSQL* bajo un estilo de programación reactiva según el estándar *ReactiveX*;
- **rxredis:** lo mismo que la anterior, pero con el *driver* de *Redis*;
- **rewhitt:** librería que ayuda a crear microservicios controlador - trabajadores con un microservicio *Redis* actuando como cola de mensajes;
- **node-utils:** una librería con funcionalidad base para el entorno de ejecución *JavaScript / TypeScript Node.js*. Abstrae, por ejemplo, el trabajo con formatos de ficheros de datos comunes, como el *CSV*;
- **node-logger:** una librería para la gestión de *logs* de aplicaciones *Node.js*.

Repositorio *Cell*

El repositorio principal de la plataforma se llama simplemente **cell**, y en él se puede encontrar la base de código de todos los microservicios que la componen, así como flujos de datos de prueba que ayudan a depurar el funcionamiento de la plataforma. También incluye el código de las librerías base ***libcellbackend*** y ***PyCell***.

Repositorios accesorios

Se han utilizado dos repositorios adicionales para los trabajos de esta tesis:

- **cell-home-web:** la *web* de enlaces al proyecto situada en `https://cell.37north.io`;
- **phd-thesis:** el repositorio de este documento, escrito en LaTeXy *Markdown*.