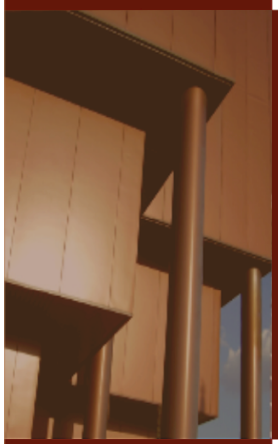
Trabajo Final de Grado
Ingeniería de las Tecnologías Industriales

Desarrollo y resolución de modelos de investigación operativa para la divulgación de la ingeniería de organización

Autora: Julia Caballero Galiano
Tutora: Alicia Robles Velasco



Dpto. de Organización Industrial y Gestión de Empresas II
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2023

Trabajo Final de Grado
Ingeniería de las Tecnologías Industriales

Desarrollo y resolución de modelos de investigación operativa para la divulgación de la ingeniería de organización

Autora:
Julia Caballero Galiano

Tutora:
Alicia Robles Velasco
Profesor Sustituto Interino

Dpto. de Organización Industrial y Gestión de Empresas II
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2023

Trabajo Final de Grado: Desarrollo y resolución de modelos de investigación operativa para la divulgación de la ingeniería de organización

Autora: Julia Caballero Galiano

Tutora: Alicia Robles Velasco

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2023

El Secretario del Tribunal

A mi familia

A mis profesores

Agradecimientos

Este trabajo marca el punto final de cuatro años de carrera, cuatro años de aprendizaje tanto académico como personal. Y no puedo olvidarme de todos aquéllos que han estado conmigo durante esta etapa tan importante en mi vida.

Primero agradecer a mi familia el apoyo constante y empujarme siempre a seguir y dar la mejor versión de mi misma cuando ni yo creía que fuera posible. Suena a cliché, pero de verdad que no podría haber hecho esto sin ellos.

Por supuesto, gracias también a mis amigos. Estos años hemos compartido mucho y hemos ido aprendiendo los unos de los otros, apoyándonos en todo momento y haciendo los momentos duros un poco más amenos. Aunque acabe esta etapa, sé que las amistades que he hecho en esta carrera me acompañarán siempre.

Finalmente, quisiera acordarme también de todos mis profesores, pero especialmente de Alicia Robles. He aprendido muchísimo con el trabajo que me has propuesto, pero sobre todo me ha servido para apreciar lo importante y bonita que puede ser esta carrera. Gracias por tu implicación constante y las facilidades que me has puesto en todo momento.

Julia Caballero Galiano
Sevilla, 2023

Resumen

Este trabajo de fin de grado se centra la resolución de problemas abordados por la investigación operativa para la divulgación de la ingeniería de organización. Para introducir la temática del trabajo es fundamental empezar por la historia tanto de la ingeniería de organización, como de la ingeniería industrial y de la investigación de operaciones: cómo surgieron las distintas disciplinas y la relevancia que han ido teniendo a lo largo de los años.

Con objeto de ver algunas aplicaciones prácticas de estas disciplinas se han resuelto tres tipos de problemas relacionados con la investigación de operaciones: el *problema de la dieta*, el *problema del viajante* y el *problema de la mochila*. Primero, se ha abordado cada tipo de problema desde una perspectiva más sencilla y fácil de comprender y después se ha planteado una versión más realista y compleja de cada problema. Para la resolución de los problemas se han usado distintas metodologías. Uno de los problemas se ha resuelto con el *Solver* de Excel. También se ha utilizado la librería *Gurobi* para resolver el escenario en el que, por el número de variables y restricciones, el *Solver* de Excel es menos eficiente. Por último, se ha utilizado un algoritmo genético programado en el lenguaje de programación de *Python* para resolver problemas no lineales.

El objetivo de este trabajo es destacar la importancia de la ingeniería de organización e investigación de operaciones, desconocidas por muchos pese a estar presente en el día a día de todos y haber sido fundamental para el desarrollo de la sociedad en muchos de sus aspectos.

Abstract

This undergraduate dissertation focuses on the resolution of problems addressed by operational research for the dissemination of industrial engineering. To introduce the subject of this work it is crucial to begin with the history of industrial engineering and operations research: how these disciplines emerged and the relevance they have gained over the years.

In order to see some practical applications of these disciplines, three types of problems related to operations research have been solved: the diet problem, the traveling salesman problem, and the knapsack problema. Each type of problem has initially been approached from a simpler and easier-to-understand perspective, and then a more realistic and challeging version of each problem has been proposed. Different methodologies have been used to solve the problems. One of the problems has been solved with the Excel Solver. The Gurobi library has also been used to solve the scenarios in which, due to the number of variables and restrictions, the Excel Solver is less efficient. Finally, a genetic algorithm programmed in the Python programming language has been used to solve nonlinear problems.

The aim of this work is to highlight the importance of organizational engineering and operations research, unknown by many despite being present in everyone's day-to-day life and having been essential for the development of society in many of its aspects.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE ILUSTRACIONES

1 INTRODUCCIÓN

La relevancia de la ingeniería industrial y de la ingeniería de organización es indudable; sin embargo, hoy en día sigue existiendo confusión acerca de lo que realmente tratan y para qué sirven. Ambas son disciplinas muy versátiles que se complementan entre sí y se centran en mejorar la eficiencia y productividad de procesos de todo tipo.

Relacionada con ambas ramas está la investigación de operaciones, que consiste en la aplicación de técnicas matemáticas, estadísticas y de programación para la optimización de procesos productivos y ayudar en la toma de decisiones. Con ese fin, se plantean modelos matemáticos en los que la función objetivo suele ser maximizar beneficios o minimizar costes o distancias y se busca la solución óptima, cumpliendo siempre una serie de restricciones. Algunas de las muchas aplicaciones de la investigación operativa son el *problema de la dieta*, el *problema del viajante* y el *problema de la mochila*. Son problemas aplicables a todo tipo de ámbitos en los que hay que tomar decisiones, desde empresas hasta situaciones cotidianas.

En conclusión, estas disciplinas tienen en común la búsqueda de la optimización y la resolución de problemas complejos, fomentando la innovación y la calidad. Para continuar su desarrollo es clave concienciar a la sociedad de su importancia e incentivar su estudio.

El objetivo de este trabajo es ayudar a la difusión de la investigación operativa. Para ello, se desarrollan y resuelven los tres problemas mencionados. En concreto, se propone una versión sencilla de cada problema y una versión más realista y, por tanto, compleja. Asimismo, en el trabajo se explican las ténicas de resolución más utilizadas en esta rama de la ingeniería que se han utilizado para resolver los problemas.

El trabajo esta dividido en seis secciones. En esta primera sección se ha realizado una breve introducción al mismo. A continuación, en la sección 2 se realiza un análisis de la historia e hitos más importantes de la ingeniería e investigación operativa. En la siguiente sección se explicarán los tres tipos de problemas que se van a resolver y sus diferentes escenarios. En la sección 4 se verán las metodologías usadas para la resolución de los problemas y en la 5 se mostrarán los resultados obtenidos al aplicar estas metodologías a los escenarios propuestos. Finalmente, en la sección 6, se resumen los aspectos más relevantes del trabajo y se comentarán áreas de investigación y desarrollo futuras.

2 HISTORIA

2.1. La ingeniería industrial

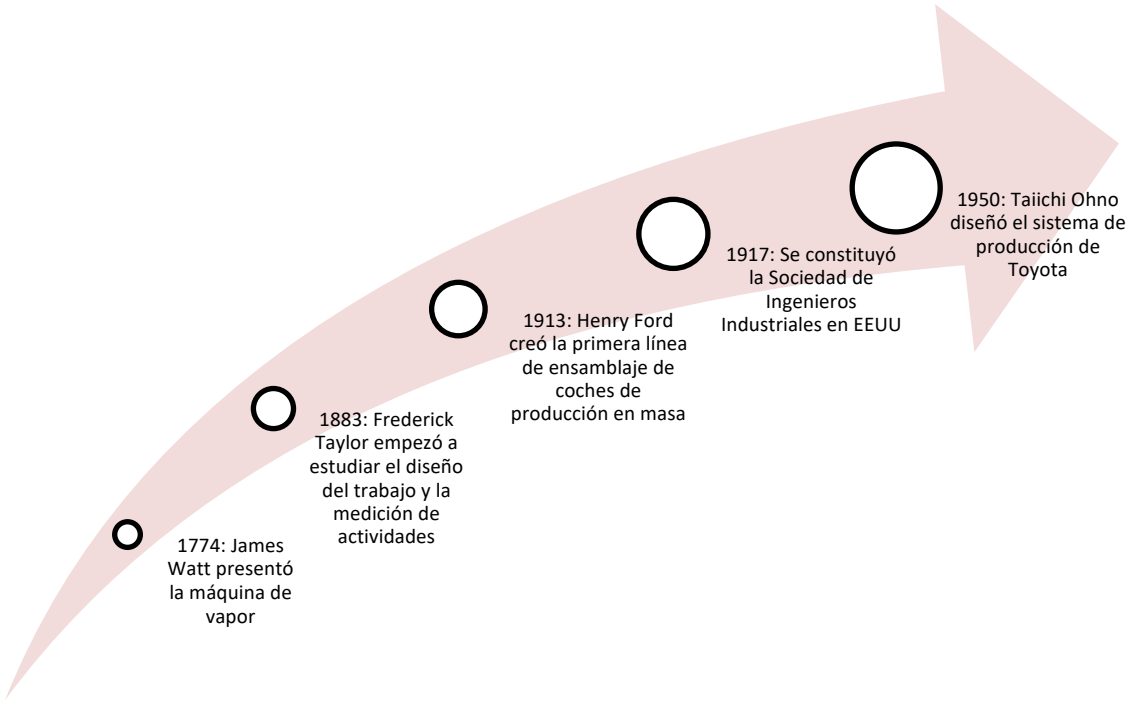
La RAE define la ingeniería como "conjunto de conocimientos orientados a la invención y utilización de técnicas para el aprovechamiento de los recursos naturales o para la actividad industrial" [1]. La ingeniería forma parte de nuestras vidas desde la prehistoria, aunque no se estableció el conocimiento de esta disciplina hasta la Revolución Industrial (1760 - 1840). La primera escuela de ingeniería fue fundada en Francia, en 1795, durante el gobierno de Napoleón y desde entonces han surgido múltiples ramas enfocadas a las distintas necesidades del ser humano. Quizás una de las más generalistas sea la ingeniería industrial.

Según el Instituto de Ingeniería Industrial (IIE), la ingeniería industrial es "lo concerniente con el diseño, mejoramiento e instalación de los sistemas integrados de personas, materiales, información, equipo y energía, soportado por el conocimiento especializado y la habilidad en las matemáticas, la física y las ciencias sociales que, junto con los principios y métodos de análisis de la ingeniería y el diseño, especifican, predicen y evalúan los resultados que serán obtenidos de cada uno de los sistemas de la industria" [2].

La práctica de la ingeniería industrial se ha caracterizado desde el principio por su adaptación, diversificación y formación multidisciplinar. Abarca numerosos campos de aplicación y tiene un papel importante en varios ámbitos. Las principales áreas de conocimiento de un ingeniero industrial son las de manufactura y producción, administrativa y financiera, de logística y distribución, de automatización y robótica, eléctrica, energética y análisis, desarrollo y gestión de proyectos.

La historia de la ingeniería industrial está marcada por una serie de acontecimientos que impulsaron el avance y mejora de la sociedad. En la Ilustración 1 y en el resto de la sección se tratan algunos de los hitos más relevantes. Estos hitos han influido directamente en el desarrollo de la sociedad, impulsando el crecimiento económico y el avance tecnológico en numerosos ámbitos.

Ilustración 1. Hitos de la ingeniería industria



Fuente: Elaboración propia

Aunque esta ingeniería ´oficialmente´ surgiera a raíz de la Primera Revolución Industrial, el verdadero origen de muchas de sus técnicas se remonta a la Revolución Agrícola, que tuvo lugar en el mismo período de tiempo que la Industrial. Ante la necesidad de optimizar la productividad de las actividades económicas rurales, se perfeccionaron los procedimientos de abonado y se reorganizó la explotación. Estas mejoras permitieron que se produjera al auge de la ingeniería en la Revolución Industrial, momento en el que se implementó el uso de máquinas, aumentando así la producción mecanizada y, por lo tanto, la productividad. Esto supuso un antes y un después en la sociedad y la economía. Después las aportaciones de la Ingeniería industrial no hicieron más que crecer, con inventos como el ferrocarril o la máquina de vapor [3].

No se puede hablar de la historia de la ingeniería industrial sin mencionar a sus principales precursores:

Uno de los primeros fue Richard Arkwright, que inventó la versión mecánica del torno de hilar y fue el primero en establecer el método de control administrativo.

En 1760, el arquitecto francés Jean Perronet estudió los tiempos para la fabricación de elementos para la construcción, que resultó clave para el desarrollo conceptual de lo que ahora conocemos como ingeniería industrial.

La máquina de vapor (1774), uno de los inventos más significativos, se atribuye a James Watt y Matthew Boulton, y originó el gran cambio mencionado anteriormente.

Más tarde, a finales del siglo XIX, llegó Frederick W. Taylor, considerado por muchos como el padre de la Dirección Científica y de la Ingeniería Industrial. Entre las contribuciones de Taylor a la ingeniería destacan el desarrollo del análisis del trabajo, el estudio de tiempos y movimientos y la estandarización de herramientas.

En la misma época que Taylor, realizó también avances Henry Lawrence Gantt, que creó una gráfica para el control de actividades a través del tiempo. El diagrama de Gantt sigue siendo hoy en día una herramienta muy usada para la organización y gestión de proyectos.

También cabe destacar al matrimonio formado por Frank y Lillian Gilbreth, que a principios del siglo XX contribuyeron al desarrollo del *Estudio de Movimientos*, logrando la optimización de numerosos procesos en ámbitos tanto industriales como del hogar [4].

A Henry Ford, fundador de la *Ford Motor Company*, le debemos la implementación del sistema de producción en línea, también conocido como *fordismo*. La idea era disminuir el tiempo de producción y el volumen de materia prima en transformación y aumentar la capacidad de producción a través de la especialización y la línea de montaje. Consiguió así la producción en masa y a precio asequible de automóviles y se convirtió en pocos años en líder de la industria automovilística. Gracias a estos avances la gente empezó a interesarse en los métodos de producción y se constituyó en 1917 la Sociedad de Ingenieros Industriales. [5]

Una vez introducida la historia de la ingeniería industrial a nivel mundial, es interesante poner el foco en cómo avanzó esta disciplina en España.

El 4 de septiembre de 1850 surge en España por un Real Decreto la carrera de ingeniero industrial. A finales de ese año se creó en Madrid el Real Instituto Industrial y Escuelas de Ampliación en Barcelona, Sevilla y Vergara.

Sin embargo, debido a la situación socioeconómica de España durante esos años, hasta el siglo XX no empezó la expansión de esta ingeniería en nuestro país. Tras la contribución de los ingenieros industriales en el proceso de modernización de aquella época, se publicó el Decreto de 1935 de Alcalá Zamora que reconocía y regulaba las competencias profesionales del ingeniero industrial [6].

Desde la creación de la carrera han ido surgiendo nuevas especialidades y titulaciones, como, por ejemplo, la Titulación de Ingeniería de Organización. Ésta se origina en la especialidad en Organización Industrial de la carrera de Ingeniería Industrial, en el plan de estudios de 1964. Inicialmente la titulación era de segundo ciclo, lo que implicaba que solo se pudiera acceder a ella si se realizaba desde los primeros ciclos de las Ingenierías Superiores, Técnicas o Arquitectura. Sin embargo, debido al éxito que tuvo la especialidad, se decidió otorgarle una identidad,

consiguiendo una mayor especialización. Hoy en día se accede mediante el grado especialista en Ingeniería en Organización Industrial [7].

Una de las escuelas en la que se imparte esta carrera es la Escuela Técnica Superior de Ingeniería (ETSi), creada en 1963 e inagurada oficialmente en abril de 1967. La primera promoción de ingenieros industriales contaba solo con las especialidades de Eléctrica, Mecánica y Química, pero a lo largo de los años éstas se fueron ampliando. El grado de Ingeniería en Organización Industrial se empezó a imparitr en 1994. [8]

2.2. La ingeniería de organización

La ingeniería de organización industrial (IOI) se define como una rama de la ingeniería que se dedica a estudiar la conexión entre la tecnología, las personas y la organización en búsqueda de una mayor productividad [9].

Para hablar bien de la IOI hay que remontarse al siglo XVIII, cuando surgen los primeros pensadores que resaltaron la importancia de la división del trabajo y la especialización. Destacaron Adam Smith, Charles Babbage, David Ricardo o Matthew Boulton, entre otros.

Adam Smith, famoso economista y filosofo británico, indicaba en su obra *"La Riqueza de las Naciones"* (1776) la importancia de una adecuada división del trabajo y especialización. Lo explicaba mediante el ejemplo de la fabricación de alfileres, mostrando cómo la división del trabajo era crucial para incrementar la productividad.

En el siglo XIX David Ricardo, economista inglés, siguió estudiando las teorías de Smith y trató el tema de la especialización productiva en el comercio internacional entre países. También desarrolló el concepto de ventaja competitiva en el comercio, aunque, al igual que las observaciones de Adam Smith, estaban limitadas por falta de medios. [10]

Pero el nacimiento de la disciplina se produjo a raíz de la Segunda Revolución Industrial (1870-1914). Uno de los momentos clave de la ingeniería de organización lo marcó la publicación en 1903 del artículo *Shop Management*, de Frederick W. Taylor, ya mencionado entre los pioneros de la Ingeniería Industrial y considerado padre de la IOI. En esta publicación Taylor se centró en el estudio de tiempos y movimientos para medir el rendimiento y la productividad de los obreros. Se trataban también temas como el estudio de incentivos, estandarización de herramientas, métodos de estimación de costes, estudios de planificación, etc.

En 1911 publica *Principios de Administración Científica*, donde se tratan aspectos tan importantes como la racionalización del trabajo, la formación rápida de operarios, el desgaste de éstos o la fabricación en serie.

En 1915, año en el que falleció Taylor, se creó la primera agrupación profesional en el campo de la Organización Industrial se empezaron a conocer sus ideas y las de sus seguidores como "*Scientific Management"*.

A las aportaciones de Taylor se le sumó el sistema de producción en línea de Ford comentado anteriormente, que consistía en fabricar en series enormes el mismo modelo de coche (*Ford T*). Tuvo mucho éxito, pero después de un tiempo la demanda de este modelo cayó y *General Motors*, presidida por Alfred Sloan, pasó a ser el primer grupo empresarial del mundo en el sector del automóvil. Alfred Sloan también tiene un importante papel en la historia de la ingeniería de organización. Introdujo técnicas innovadoras de gestión (modelo de gestión descentralizada) y contabilidad (sistema de contabilidad). Por esto y otras importantes aportaciones, lleva su nombre la facultad de gestión empresarial del Instituto de Tecnología de Massachusetts (MIT), MIT Sloan School of Management de Boston [11].

Otros nombres relevantes son Charles Bedaux, Henry Gantt. Frank y Lillian Gilbreth, T.C. Fry y Walter Shewhart. A los dos últimos les debemos la introducción de la estadística como herramienta en el control de la calidad.

La Segunda Guerra Mundial y la Posguerra también influyeron en el desarrollo de la IOI y la Investigación Operativa, de la que se hablará más adelante. Se empezaron a usar y ver la importancia de técnicas de logística, coordinación, planificación y optimización. Además, después de la Guerra, se dio un crecimiento económico y el mercado se volvió más competitivo, lo que propulsó el desarrollo de nuevas técnicas y metodologías para mejorar la logística y estrategias. [12]

Más tarde, en 1950, nace en Japón gracias al ingeniero Taiichi Ohno el sistema de producción de Toyota, la filosofía del *Lean Manufacturing*, y aunque no tuviera demasiado éxito en aquel momento, acabó resultando clave para la mejora de sistemas de producción y otras industrias. Hoy en día sigue estando presente en numerosas e importantes empresas como Toyota, Nike o Intel [13].

A raíz de la popularidad de la rama, surgieron en todo el mundo titulaciones relacionadas con la ingeniería de organización. Como se mencionó anteriormente, en España existe la titulación de Ingeniería de Organización Industrial desde 1964 [12], pero en otros países, como Estados Unidos, se creó antes.

Recibe un nombre distinto en cada país (*Industrial Engineering* en Estados Unidos, *Managment Engenieering* en Reino Unido, *Ingenieur de Génie Industriel* en Francia, etc… [14]) pero el objetivo es siempre el mismo: combinar la gestión y la organización con una formación técnica.

Otros objetivos de la titulación son: el diseño y optimización de procesos productivos y sistemas empresariales, principios de mejora continua en todo tipo de procesos, toma de decisiones, etc.… Por todo esto los graduados en Ingeniería de Organización e Ingeniería Industrial son muy polivalentes y están capacitados para trabajar en una gran variedad de sectores, desde logística hasta consultoría o fabricación.

Ilustración 2. Representación estadística del número de egresados en IOI de 2º ciclo, Grado y Máster

Fuente: Ministerio de Universidades

La importancia de esta profesión ha hecho que se imparta la titulación en múltiples universidades de España y que se convierta en titulación de referencia en las escuelas universitarias militares.

En 1999 se fundó en la Escuela de Ingenieros Industriales de la Universidad Politécnica de Cataluña la Asociación para el Desarrollo de la Ingeniería de Organización (ADINGOR, https://adingor.net/) con el objetivo de promover el desarrollo y difusión de todo lo relacionado con la ingeniería de organización. Además, se encarga de organizar Congresos de Ingeniería de la Organización (CIOs) y sirve de enlace de comunicación entre sus miembros. [15]

Otra asociación que cabe destacar es la Asociación Profesional de Ingenieros de Organización Industrial de España (https://www.aingoi.com/). Fue constituida en 2003 por estudiantes y titulados en IOI y hace de interlocutora de los Ingenieros en Organización ante los distintos organismos y administraciones [16].

2.3. La investigación operativa

La ORS define la investigación operativa como "un enfoque científico para la solución de problemas en la gestión de sistemas complejos que permite a los responsables de la toma de decisiones tomar mejores decisiones" [17]. Es un campo interdisciplinario que tiene aplicaciones en una gran variedad de áreas como la planificación financiera, la logística o la construcción.

En resumen, es la disciplina que se encarga de la formulación de métodos analíticos para la toma de decisiones

La metodología general de la investigación de operaciones es la siguiente:

1) Definir el problema y recolectar datos
2) Formular el modelo matemático
3) Plantear y analizar soluciones a partir del modelo
4) Obtener y elegir la solución
5) Control y evaluación de la solución elegida

Aunque de una manera u otra la investigación operativa haya estado presente en la sociedad desde hace siglos y Charles Babbage (1791-1871) sea considerado por muchos el padre de esta disciplina gracias al desarrollo de la investigación de costos de transporte, el nacimiento oficial de la investigación de operaciones fue durante la Segunda Guerra Mundial [18]. Al igual que la ingeniería industrial y la ingeniería de organización, el avance de la investigación operativa se debe principalmente al desarrollo de tácticas militares, sobretodo de los científicos ingleses y norteamericanos. Buscaban encontrar la forma óptima de utilizar los escaso recursos de los que disponían.

El *problema de la dieta*, que se planteará más adelante en este trabajo, fue uno de los primeros problemas de optimización estudiados, con el objetivo de buscar una dieta saludable que suministrar a los soldados de las fuerzas armadas al menor coste posible.

El origen de la investigación operativa también está relacionado con las investigaciones asociadas al radar en Reino Unido, que se usó para tácticas de defensa, dirección y control de flotas. De ello se encargó el posteriormente ganador de un premio nobel, P.M.S. Blackett.

Además, durante la guerra se crearon varios comités y grupos que fueron clave para el desarrollo de la disciplina, como el NDRC o el OSRD. Se abordaron diversos problemas como la precisión de bombardeos, maniobras de barcos, análisis de sistemas, etc. [19]

Cuando acabó la guerra, en Gran Bretaña se redujo la investigación operativa en el sector militar; sin embargo, en 1948 se fundó el 'Operational Research Club of Britain', que en 1953 pasó a llamarse como lo conocemos actualmente, Operational Research Society. En América y otros países se crearon en 1952 y más tarde se unieron en la IFORS.

Muchos de los ciéntificos que trabajaron durante la guerra en tácticas militares, para seguir con los avances realizados, perfeccionaron técnicas ya existentes, como el método *simplex*. Pero no se quedaron ahí: en Inglaterra se empezó a usar para la planificación social y económica, para lo que sigue siendo útil hoy en día. Otros campos de aplicación en los que desarrolla un papel importante la IO son, como ya se mencionó antes, la gestión, servicios financieros, investigación y desarrollo de la industria, etc.

2.4. Asociaciones

A lo largo de los años son varias las entidades y asociaciones que se han dedicado al estudio y desarrollo de la Investigación Operativa. La más importante es la Federación Internacional de Sociedades de Investigación Operativa (IFORS, *International Federation of Operational Research Societies*), fundada en 1959 por las sociedades más relevantes de aquel momento: ORSA (*Operations Research Society of America*), ORS (*Operational Research Society*) y SOFRO (*French Société Française de Recherche Opérationnelle*).

Todo empezó en 1955, cuando el vicepresidente de la ORSA envió a la ORS una propuesta para una conferencia internacional. Más tarde se unió la sociedad francesa (SOFRO) como sociedad patrocinadora de la primera serie de conferencias trienales, la Conferencia de Oxford de 1957, momento histórico para la IO. Finalmente, en enero de 1959 nació oficialmente la IFORS, cuyo objetivo era "el desarrollo de la investigación operativa como ciencia unificada y su avance en todas las naciones del mundo". [20]

Las agrupaciones regionales miembros de la IFORS son ALIO (*Asociación Latino-Ibero-Americana de Investigación Operativa*), APORS (*Association of Asia Pacific Operational Research Societies*), EURO (*Association of European Operational Research Societies*) y NORAM (*Association of North American Operations Research Societies*) [21].

Además, actualmente forman parte de ella más de 50 sociedades nacionales de todo el mundo, entre las que destacan INFORMS y ORS entre otras.

La ORS (*Operational Research Society*), con sede principal en Birmingham, es la sociedad científica más antigua que se dedica al estudio y desarrollo de la investigación operativa. Fue creada originalmente en 1948 como '*the Operational Research Club'*, después de que les surgiera la idea en una cena a Charles Goodeve, Patrick Blackett, Dr C Gordon y Charles Tizard. Las actividades principales del club eran celebrar reuniones científicas y, a partir de 1950, también elaborar una revista trimestral.

Más tarde, en 1953 pasó a llamarse como la conocemos hoy en día, OR Society. Actualmente la sociedad tiene más de 3000 miembros en 53 países distintos, que se benefician de los distintos servicios y se dedican al desarrollo del conocimiento y educación de la IO. Además, cuenta con cursos de formación que ayudan a profesionales a adquirir y perfeccionar las habilidades apropiadas. [22]

INFORMS (*Institute for Operations Research and the Management Sciences*) surge en 1995 a partir de unión de la ORSA y el TIMS (*The Institute of Management Sciences*) [23]. INFORMS fomenta el estudio de la investigación operativa mediante varios programas, servicios y publicaciones. Además, se encarga de crear redes de profesionales interesados en este campo y facilita la colaboración entre unos y otros. [24]

Los socios de esta asociación, provenientes de casi 90 países de todo el mundo, son investigadores de operaciones y profesionales analíticos que trabajan para universidades, corporaciones, el gobierno, etc. Los distintos niveles de membresía son miembro regular, miembro estudiante y miembro jubilado.

En España la sociedad encargada de desarrollar y promover la Investigación Operativa es la SEIO (*Sociedad de Estadística e Investigación Operativa*), fundada en 1962 por profesionales de distintos ámbitos interesados estudio de la IO bajo el nombre de Sociedad Española de Investigación Operativa. En 1976 se amplió el campo de actividades a la Estadística y a la Informática y se cambió el nombre a el actual, Sociedad de Estadística e Investigación Operativa. En la actualidad la sociedad cuenta con casi 700 socios que intervienen en las distintas actividades: congresos, reuniones, publicación de revistas y boletines… [25]

3 CASOS DE ESTUDIO

En este apartado se presentarán tres problemas diferentes relacionados con la investigación operativa. Para cada problema se realizará una breve introducción y, a continuación, se desarrollará el modelo matemático que lo respalda. Posteriormente se particularizará cada problema para dos casos o escenarios. El primero de ellos persigue ser más simple y, con ello, más fácil de entender para el público general, siendo el segundo de mayor complejidad.

3.1 Problema de la dieta

El *problema de la dieta*, planteado por George Stigler durante la Segunda Guerra Mundial, fue uno de los primeros sobre optimización y programación lineal. Surgió a raíz de la necesidad de alimentar correctamente al ejército estadounidense al menor coste posible. Partiendo de una serie de alimentos, se pretendía encontrar la ración óptima por soldado que cumpliese unos requerimientos nutricionales de la forma más económica. El problema original tenía 77 variables y 9 restricciones.

Stigler lo resolvió en 1939 usando un método heurístico y obtuvo un coste anual de 39,93 $ por soldado. Más tarde, en 1947, Jack Landerman solucionó el problema usando el método *simplex*, mejorando el resultado de Stigler, y consiguió reducir el coste anual a 39,69 $, lo cual supuso un gran avance, siendo el ahorro total de 0,24 $ multiplicado por el número de soldados del ejército estadounidense [26]. En el supuesto de existir 1.000.000 soldados, el ahorro sería de 240.000 $ (equivalente a multiplicar 0,24 por 1.000.000).

Hoy en día son varias las aplicaciones de este problema. Entre las más destacadas están la optimización de dietas en función de intolerancias y alergias alimentarias y el fomento de la alimentación sostenible. En el caso de las intolerancias y alergias, el *problema de la dieta* sirve para encontrar una combinación de alimentos que cumpla con unos requisitos nutricionales específicos, evitando los alimentos que causan problemas de salud. En cuanto a la alimentación sostenible, debido a la concienciación medioambiental, cada vez se buscan más dietas que sean saludables para las personas sin perjudicar al planeta. Para ello, es crucial considerar factores como la huella ambiental o la disponibilidad de alimentos de temporada.

Otros ámbitos en los que este problema es de gran utilidad son la nutrición deportiva, dietas terapéuticas, dietas escolares, balanceo de la dieta animal, etc.

3.1.1 Modelo matemático

A continuación, se presenta el modelo matemático asociado al *problema de la dieta*.

$$Min. \sum_{i=1}^{n} c_i x_i$$
$$s.a.\ b_{jmin} \leq \sum_{i=1}^{n} a_{ij} x_i \leq b_{jmax}\ ;\ j = 1, \dots, m$$
$$x_i \geq 0$$

Donde:

- x_i: cantidad del alimento i
- c_i: coste del alimento i
- n: número de alimentos
- m: número de nutrientes
- a_{ij}: cantidad del nutriente j en el alimento i
- b_{jmin}: cantidad mínima del nutriente j
- b_{jmax}: cantidad máxima del nutriente j

La función objetivo del modelo busca minimizar el coste total de la dieta, siendo el sumatorio de la multiplicación del coste unitario de cada alimento (c_i) por la variable x_i que representa la cantidad de dicho alimento que debe contener la dieta óptima. Este sumatorio tendrá *n* términos, uno correspondiente a cada alimento.

A continuación, se establecen restricciones nutricionales para garantizar que la dieta cumpla con los requisitos establecidos. El modelo suele tener $2 \cdot m$ restricciones, es decir, *m* restricciones para asegurar que la dieta tiene al menos b_{jmin} nutrientes, y otro conjunto de m restricciones para asegurar que no se supera la máxima cantidad permitida para cada nutriente. Estas restricciones se establecen mediante ecuaciones lineales en las que se multiplica cada variable x_i por la cantidad del nutriente j en dicho alimento (a_{ij}) y se fuerza esta multiplicación a ser mayor o menor, dependiendo del caso, que un determinado valor nutricional.

Aunque no sea el caso en este trabajo, también se pueden añadir restricciones adicionales, como preferencias dietéticas o limitaciones económicas.

Por último, las variables del modelo son continuas y mayores que 0, representando la cantidad de cada alimento que debe contener la dieta.

Es importante tener en cuenta que la formulación del modelo puede depender de los requisitos y datos de cada problema en particular.

3.1.2 Particularización del modelo: escenario 1

En este escenario, se presenta un problema nutricional de bebés. El objetivo es demostrar la utilidad de la modelización matemática y de las técnicas propuestas para su resolución. Cabe destacar que los datos incluidos en el enunciado son datos reales, obtenidos a partir de la web de Hero Baby [27], marca de alimentos para bebés y niños pequeños. Es decir, tanto los costes como los nutrientes de cada alimento son valores reales. Para la obtención de estos datos se ha llevado a cabo una búsqueda minuciosa y comparación de información en varias fuentes.

Enunciado:

Unos padres están preparando la compra de potitos de bebés para su hijo de ocho meses. Después de un pequeño estudio de mercado han seleccionado tres tipos diferentes: de pollo y arroz (potito A), de frutas variadas (potito B) y de verdura y pescado (potito C). Los expertos recomiendan los siguientes valores nutricionales para bebés de 6-12 meses: 60-100 g/día de hidratos de carbono, 10-30 g/día de proteína (30 máximo), 5 g/día de fibra y 40-50 mg/día de vitamina C.

El potito A cuesta 8,49 €/kg y contiene por cada 100 gramos 9,5 g de hidratos de carbono, 2,7 g de proteínas y 1 g de fibra.

El potito B cuesta 6,36 €/kg y contiene por cada 100 gramos 15,5 g de hidratos de carbono, 1,1 g de proteínas, 1,4 g de fibra y 25 mg de vitamina C.

El potito C cuesta 18,37 €/kg y contiene por cada 100 gramos 8,5 g de hidratos de carbono, 3 g de proteínas y 1 g de fibra.

Determinar en kilos qué cantidad de cada potito al día (x_A, x_B y x_c) deben darle al bebé para cumplir con los requisitos nutricionales al menor coste posible.

En la tabla siguiente se encuentran los tres tipos de potitos con sus respectivos precios y valores nutricionales.

Tabla 1. Resumen de los datos del problema de la dieta escenario 1

	Precio (€/kg)	Hidratos de carbono (g)	Proteínas (g)	Fibra (g)	Vitamina C (mg)
Potito A	8,49	9,5	2,7	1	0
Potito B	6,36	15,5	1,1	1,4	25
Potito C	18,37	8,5	3	1	0

Formulación matemática del problema:

$$Min.\ 8{,}49x_A + 6{,}36x_B + 18{,}37x_C$$

$$s.a.\ 0{,}06 \leq 0{,}095x_A + 0{,}155x_B + 0{,}085x_C \leq 0{,}1$$

$$0{,}01 \leq 0{,}027x_A + 0{,}011x_B + 0{,}03x_C \leq 0{,}03$$

$$0{,}05 \leq 0{,}01x_A + 0{,}014x_B + 0{,}01x_C \leq 0{,}006$$

$$0{,}00004 \leq 0{,}00025x_B \leq 0{,}00005$$

$$x_A, x_B, x_C \geq 0$$

3.1.3 Particularización del modelo: escenario 2

Debido a la importancia e impacto que tuvo el *problema original de la dieta de Stigler* explicado al comienzo de esta sección, en este escenario se ha decidido emplear los mismos datos usados inicialmente por Stigler y poder comparar resultados. El *problema de la dieta Stigler* consiste en encontrar la combinación más económica de alimentos que cumpla los requerimientos nutricionales de una persona.

Los datos utilizados son los usados en 1939 e incluyen la lista con los 77 alimentos, las cantidades de los distintos nutrientes de cada alimento, así como su precio y los requisitos nutricionales mínimos diarios. Al igual que para el escenario anterior, los datos utilizados en este escenario son datos reales tomados del artículo científico original donde se presentaba el problema.

Restricciones del modelo:

- Calorías (kcal) ≥ 3
- Proteína (g) ≥ 70
- Calcio (g) ≥ 0.8
- Hierro (mg) ≥ 12
- Vitamina A (KIU) ≥ 5
- Vitamina B1 (Tiamina) (mg) ≥ 1.8
- Vitamina B2 (Riboflavina) (mg) ≥ 2.7
- Vitamina B3 (Niacina) (mg) ≥ 18
- Vitamina C (Ácido ascórbico) (mg) ≥ 75

Cada alimento viene expresado en su respectiva unidad y el precio en centavos. Sin embargo, las cantidades de cada nutriente (calcio, proteínas, hierro…) vienen expresadas en gramos por dólar, por lo que habrá que realizar una conversión para que estén en la misma unidad que cada alimento. Es decir, si la unidad del 'alimento 1' son 10 libras, habrá que dividir el precio entre 10 para saber cuánto cuesta una libra y posteriormente habrá que relacionar este precio con los gramos por 1 $ para obtener las cantidades de cada nutriente por libra de alimento. A continuación, se muestra en la Tabla 2 parte de los datos [28]. No obstante, los datos completos se encuentran en la Tabla 17 del Anexo 1.

Tabla 2. Datos del problema de la dieta escenario 2

TABLE A. NUTRITIVE VALUES OF COMMON FOODS PER DOLLAR OF EXPENDITURE, AUGUST 15, 1939

Commodity	Unit	Price Aug. 15, 1939 (cents)	Edible Weight per $1.00 (grams)	Calories (1,000)	Protein (grams)	Calcium (grams)	Iron (mg.)	Vitamin A (1,000 I.U.)	Thiamine (mg.)	Riboflavin (mg.)	Niacin (mg.)	Ascorbic Acid (mg.)
**1. Wheat Flour (Enriched)	10 lb.	36.0	12,600	44.7	1,411	2.0	365		55.4	33.3	441	
2. Macaroni	1 lb.	14.1	3,217	11.6	418	.7	54		3.2	1.9	68	
3. Wheat Cereal (Enriched)	28 oz.	24.2	3,280	11.8	377	14.4	175		14.4	8.8	114	
4. Corn Flakes	8 oz.	7.1	3,194	11.4	252	.1	56		13.5	2.3	68	
5. Corn Meal	1 lb.	4.6	9,861	36.0	897	1.7	99	30.9	17.4	7.9	106	
6. Hominy Grits	24 oz.	8.5	8,005	28.6	680	.8	80		10.6	1.6	110	
7. Rice	1 lb.	7.5	6,048	21.2	460	.6	41		2.0	4.8	60	
8. Rolled Oats	1 lb.	7.1	6,389	25.3	907	5.1	341		37.1	8.9	64	
9. White Bread (Enriched)	1 lb.	7.9	5,742	15.0	488	2.5	115		13.8	8.5	126	
10. Whole Wheat Bread	1 lb.	9.1	4,985	12.2	484	2.7	125		13.9	6.4	160	
11. Rye Bread	1 lb.	9.2	4,930	12.4	439	1.1	82		9.9	3.0	66	
12. Pound Cake	1 lb.	24.8	1,829	8.0	130	.4	31	18.9	2.8	3.0	17	
13. Soda Crackers	1 lb.	15.1	3,004	12.5	288	.5	50					
14. Milk	1 qt.	11.0	8,867	6.1	310	10.5	18	16.8	4.0	16.0	7	177
**15. Evaporated Milk (can)	14½ oz.	6.7	6,055	8.4	422	15.1	9	26.0	3.0	23.5	11	60

Formulación matemática del problema:

$$
\begin{aligned}
Min.\ & 3{,}6x_1 + 14{,}1x_2 + 0{,}86x_3 + \cdots + 0{,}57x_{75} + 0{,}756x_{76} + 20{,}5x_{77} \\
s.a.\ & 1{,}6x_1 + 1{,}636x_2 + 0{,}1x_3 + \cdots + 0{,}08x_{75} + 0{,}07x_{76} + 1{,}31x_{77} \geq 3 \\
& 50{,}8x_1 + 58{,}94x_2 + 3{,}26x_3 + \cdots + 0x_{75} + 0x_{76} + 2{,}25x_{77} \geq 70 \\
& 0{,}072x_1 + 0{,}097x_2 + 0{,}124x_3 + \cdots + 0{,}003x_{75} + 0{,}078x_{76} + 0{,}082x_{77} \geq 0{,}8 \\
& 13{,}14x_1 + 7{,}61x_2 + 1{,}51x_3 + \cdots + 0{,}42x_{75} + 1{,}84x_{76} + 1{,}43x_{77} \geq 12 \\
& 0x_1 + 0x_2 + 0x_3 + \cdots + 0x_{75} + 0x_{76} + 0{,}041x_{77} \geq 5 \\
& 1{,}99x_1 + 0{,}45x_2 + 0{,}12x_3 + \cdots + 0x_{75} + 0{,}014x_{76} + 0{,}041x_{77} \geq 1{,}8 \\
& 1{,}2x_1 + 0{,}27x_2 + 0{,}076x_3 + \cdots + 0x_{75} + 0{,}057x_{76} + 0{,}082x_{77} \geq 2{,}7 \\
& 15{,}88x_1 + 9{,}59x_2 + 0{,}99x_3 + \cdots + 0{,}029x_{75} + 1{,}1x_{76} + 0{,}61x_{77} \geq 18 \\
& 0x_1 + 0x_2 + 0x_3 + \cdots + 0x_{75} + 0x_{76} + 0x_{77} \geq 75 \\
& x_1, x_2, x_3, \ldots, x_{75}, x_{76}, x_{77} \geq 0
\end{aligned}
$$

3.2 Problema del viajante

El *problema del viajante* – TSP (*Travelling Salesman Problem*) está presente en nuestra sociedad desde el siglo XIX gracias al matemático irlandés William Rowan Hamilton y al británico Thomas Kirkman. Sin embargo, no fue hasta los años 1930s que Karl Menger y, más tarde, Hassler Whitney y Merrill Flood lo formularon matemáticamente y comenzaron a estudiarlo en profundidad. A partir de entonces, y debido a la complejidad del problema, el interés por él no hizo más que crecer. [29]

Es considerado como NP-complejo y existen varias heurísticas y algoritmos que buscan encontrar la solución óptima de este problema. Puede ser considerado como un problema de grafos ponderados no dirigido, donde las ciudades son los vértices y los caminos con sus respectivos pesos son las aristas y la distancia o coste entre vértices.

El problema consiste en, dado un número de nodos y los respectivos costes para ir de uno a otro, encontrar el camino más barato o corto para visitar todos los nodos y volver al de inicio. [30]

Las principales aplicaciones del problema actualmente son planificación y logística, aunque también se ha utilizado en otro tipo de ámbitos como la cristalografía de rayos-X, la secuenciación de ADN o el taladro de placas de circuitos. [31]

3.2.1 Modelo matemático

A continuación, se presenta el modelo matemático asociado al *problema del viajante* [32].

$$Min. \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij}$$

$$s.a. \sum_{i=1\ i\neq j}^{n} x_{ij} = 1\ ;\ \ j = 1,\dots,n$$

$$\sum_{j=1\ i\neq j}^{n} x_{ij} = 1\ ;\ \ i = 1,\dots,n$$

$$\sum_{i\in S}\sum_{j\in S, j\neq i} x_{ij} \leq\ |S| - 1, \qquad \forall S = 1,2,\dots,n$$

$$x_{ij}\ \in \{0,1\}$$

Donde:

- x_{ij}: nodo que va del arco i al j

  Variable binaria, toma valor 1 si la arista está en la solución y 0 si no
- c_{ij}: coste o distancia al ir del nodo i al j
- n: número de nodos
- S: subconjunto de {1, 2, …, n}

Mediante la función objetivo del modelo matemático del TSP se quiere minimizar el coste o la distancia de ir de un nodo a otro. Se calcula sumando la multiplicación de las distancias o costes entre los nodos visitados (c_{ij}) por las variables que indican si se va del nodo i al j (x_{ij}). Los sumatorios tendrán tantos términos como nodos.

Las dos primeras restricciones indican que hay que visitar y partir de cada nodo exactamente una vez.

Además, es necesario añadir la tercera restricción para que no se formen ciclos más pequeños dentro del recorrido y garantizar que la solución es un ciclo hamiltoniano, es decir, una ruta cerrada que visita cada nodo una vez y regresa al inicial.

Por último, las variables del modelo son binarias: solo pueden tomar el valor 0 o el 1. Este tipo de varibles se utiliza para problemas de toma de decisiones en los que las dos posibles opciones son mutuamnete excluyentes. En este problema x_{ij} toma valor 1 si la ruta incluye el camino del nodo i al j, y 0 en caso contrario.

3.2.2 Particularización del modelo: escenario 1

En el primer escenario se pretende desarrollar un ejemplo sencillo de la aplicación de este problema. En concreto se presenta el problema de diseño de una gira para un cantante que tiene que pasar por 10 ciudades. Para ello se implementará un algoritmo genético en *Python*, cuyo funcionamiento se explicará más adelante. El objetivo es comprender el *problema del viajante* y analizar la eficiencia de los modelos de investigación operativa y el algoritmo genético para resolver este tipo de problemas.

Enunciado:

Un cantante está planeando una pequeña gira por la Península Ibérica. Su idea es dar conciertos en 10 de las principales ciudades del país recorriendo la menor distancia posible. Sabiendo que tiene que empezar y acabar en la misma ciudad, ¿cuál sería la mejor ruta?

En la Tabla 3 se muestran las distancias medidas en kilómetros (km) entre las 10 ciudades en cuestión. Estas distancias corresponden a los costes del modelo. Por simplicidad se han medido en línea recta, aunque en la realidad debería haberse tenido en cuenta la distancia real por carretera que existe entre las mismas. Los datos se han obtenido mediante la herramienta de *Google Maps* que permite medir la distancia entre dos puntos. Para rellenar la matriz de distancias se ha medido la distancia de cada ciudad respecto a todas las demás.

Tabla 3. Matriz de distancias (km) 10 ciduades

Cij	Barcelona	Bilbao	La Coruña	Lisboa	Madrid	Murcia	Salamanca	Sevilla	Valencia	Zaragoza
Barcelona	0	470	895	1005	506	470	655	822	298	256
Bilbao	470	0	425	720	321	606	334	702	470	248
La Coruña	895	425	0	517	506	856	347	694	793	645
Lisboa	1005	720	517	0	499	702	373	308	760	770
Madrid	506	321	506	499	0	352	175	392	305	274
Murcia	470	606	856	702	352	0	509	436	180	405
Salamanca	655	334	347	373	175	509	0	399	478	405
Sevilla	822	702	694	308	392	436	399	0	545	647
Valencia	298	470	793	760	305	180	478	545	0	243
Zaragoza	256	248	645	770	274	405	405	647	243	0

3.2.3 Particularización del modelo: escenario 2

En este escenario se amplia la lista de ciudades a visitar con respecto al escenario anterior, el cantante pasará de recorrer 10 ciudades a visitar 30. El objetivo es, una vez comprendido el escenario 1, aumentar la complejidad del modelo y hacerlo más realista. La metodología empleada

para resolver este escenario es la misma que en el escenario anterior, el algoritmo genético. Al ser el problema más complejo, se valorará más la utilidad del algoritmo y las ventajas que proporciona, como la precisión y velocidad de resolución.

Enunciado:

Viendo el éxito de su álbum, el cantante ha decidido ampliar la gira y pasar por varias ciudades más de la Península y Francia, sumando un total de 30 ciudades. Su objetivo sigue siendo empezar y acabar en la misma ciudad minimizando la distancia recorrida. ¿Qué ruta debe seguir en este caso?

En la Tabla 4 se muestran las distancias entre algunas de las 30 ciudades. Por falta de espacio, la tabla completa se encuentra en las distintas tablas del Anexo 2 . Al igual que en el escenario anterior, las distancias entre ciudades son en línea recta, para que el problema fuese aún más realista deberían obtenerse las distancias realeas por carretera. En este escenario las distancias también se han obtenido mediante la herramienta de *Google Maps* que permite medir la distancia entre dos puntos.

Tabla 4. Parte de la matriz de distancias (km) 30 ciudades

Cij	Barcelona	Bilbao	Cáceres	Ciudad Real	Coímbra	Córdoba	Évora	Faro	La Coruña …
Barcelona	0	470	752	580	892	710	914	995	895
Bilbao	470	0	506	485	562	617	668	813	425
Cáceres	752	506	0	222	185	225	166	304	454
Ciudad Real	580	485	222	0	400	702	373	308	760
Coímbra	892	562	185	400	0	400	185	355	352
Córdoba	710	617	225	702	400	0	284	295	681
Évora	914	668	166	373	185	284	0	172	535
Faro	995	813	304	308	355	295	172	0	706
La Coruña	895	425	454	760	352	681	535	706	0

3.3 Problema de la mochila

El *Problema de la Mochila* - KP (*Knapsack Problem*) es un problema de Optimización Combinatoria y pertenece a la lista de los 21 problemas NP-completos que publicó Richard Karp en 1972. Consiste en simular llenar una mochila de una determinada capacidad con objetos de distinto valor y peso. Este problema puede aplicarse a varios procesos de toma de decisiones reales, como, por ejemplo, asignación de espacio (ej. carga del avión), generación de claves, selección de proyectos o inversiones, etc.

Aunque aparentemente pueda parecer un problema sencillo, realmente es bastante complejo a la hora de resolverlo. Uno de los métodos más utilizados para resolver problemas de gran tamaño son las metaheurísticas y, en concreto, el algoritmo genético, el cual se explicará en la sección 4.

3.3.1 Modelo matemático

A continuación, se presenta el modelo matemático asociado al *problema de la mochila* [33].

$$Max. \sum_{i=1}^{n} b_i x_i$$
$$s.a. \sum_{i=1}^{n} w_i x_i \leq c$$
$$x_i \in \{0,1\}$$

Donde:

- x_i: variable de decisión que toma el valor de 1 si se decide meter el elemento i en la mochila, y 0 e.c.c
- b_i: valor del elemento i
- n: número de elementos
- w_i: peso del elemento i
- c: capacidad de la mochila

La función objetivo del modelo matemático del problema de la mochila busca maximizar el valor de los elementos que se meten en la mochila. Se calcula sumando la multiplicación del valor de cada elemento (b_i) por la variable x_i que representa si ese elemento se incluye o no en la mochila. Este sumatorio tendrá *n* términos, uno correspondiente a cada elemento.
La primera restricción indica que el peso total de los elementos incluidos en la mochila no puede superar la capacidad de ésta. Esto se expresa mediante el sumatorio de la multiplicación del peso de cada elemento (w_i) por la variable x_i. Al igual que en la función objetivo, el sumatorio tendrá tantos términos como elementos.
En cuanto a las variables de decisión, el tipo depende del problema que se vaya a tratar. En este trabajo serán binarias, ya que en los casos propuestos solo habrá dos opciones: meter el elemento entero en la mochila, en cuyo caso la variable de decisión x_i valdrá 1; o no meterlo, tomando x_i el valor 0.

3.3.2 Particularización del modelo: escenario 1

En este escenario se presenta un sencillo ejemplo del *problema de la mochila*, en el que un repartidor tiene que decidir qué elementos incluir primero para repartirlos. Es un caso con pocos

elementos que podría intentar resolverse sin técnicas avanzadas; sin embargo, al igual que en el caso del problema del viajante, se implementará un algoritmo genético en *Python* para su resolución. La aplicación del algoritmo servirá para encontrar rápida y eficientemente la solución óptima y así comparar esta solución con la obtenida mediante procedimientos tradicionales.

El objetivo principal de este escenario es entender bien en qué consiste el problema y cuáles son los objetivos para luego poder abordar casos de mayor envergadura. Por ello se ha elegido este ejemplo, ya que es uno de los más típicos del *problema de la mochila* y sirve como introducción al tema.

Enunciado:

Un repartidor de comidas caseras a domicilio dispone de una mochila de 30 litros de capacidad para hacer la entrega de pedidos. Cada vez que sale de la tienda debe decidir, en función del tamaño y la hora de entrega al cliente, qué pedidos meter en la mochila.

Los pedidos que tengan que ser entregados en los próximos 30 minutos tendrán una urgencia de 5, los que tengan que estar en 1 hora tendrán una urgencia de 4, en 2 horas urgencia 3, en 3 horas tendrán urgencia 2 y los que puedan entregarse a lo largo del día, urgencia 1.

Al realizar el primer viaje, ¿qué pedidos debe meter primero en la mochila para asegurarse que cumple con los plazos establecidos?

La lista de pedidos es la siguiente:

Tabla 5. Resumen de los datos del escenario 1 del KP

	Urgencia	Volumen (litros)
Producto A	5	15
Producto B	2	12
Producto C	2	5
Producto D	5	2
Producto E	1	7

Formulación matemática:

$$Max.\ 5x_A + 2x_B + 2x_C + 5x_D + x_E$$
$$s.a. 15x_A + 12x_B + 5x_C + 2x_D + 7x_E \leq 30$$
$$x_A, x_B, x_C, x_D, x_E \in \{0,1\}$$

3.3.3 Particularización del modelo: escenario 2

En este escenario se presenta un caso similar al anterior pero más complejo y realista. Consiste en decidir qué productos de un almacén entregar primero para maximizar beneficios sin superar la capacidad del camión en el que se transportan. Aunque por claridad no se vaya a realizar en este

caso, una forma de hacer este problema más realista y útil en el ámbito de la logísitca sería penalizar retrasos en la entrega de los productos mediante nuevas restricciones y añadiendo costes en la función objetivo.

Nuevamente se resolverá implementando el algoritmo genético en *Python.* Al haber muchos más productos que en el escenario anterior, los cálculos manuales se complican significativamente, por lo que se verá aún más la importancia de usar técnicas avanzadas que faciliten la búsqueda de la solución óptima.

Los datos han sido obtenidos en función de los precios y pesos medios de los distintos productos.

Enunciado:

El gerente de una pequeña empresa de logística, que cuenta con un camión con una capacidad máxima de 1500 kg, debe decir qué productos del almacén transportar para maximizar el valor total sin superar la capacidad. El almacén está lleno de electrodomésticos y muebles, cada uno con su respectivo peso y precio. El precio indica el beneficio que se obtiene por cada producto que se entrega.

Aunque eventualmente se vayan a transportar todos los productos del almacén, dado que la empresa está empezando y necesita generar ingresos para cubrir gastos operativos y poder invertir en más recursos, ¿qué productos debe meter en el camión para maximizar las ganancias y asegurar que el transporte sea eficiente y rentable?

Los distintos productos con sus respectivos datos se encuentran en la siguiente tabla:

Tabla 6. Resumen de los datos del escenario 2 del KP

	Valor (€)	Peso (kg)	Nombre
Producto A	800	150	Lavadora
Producto B	1500	300	Frigorífico
Producto C	150	30	Microondas
Producto D	1000	200	Secadora
Producto E	100	10	Aspiradora
Producto F	900	180	Aire acondicionado
Producto G	1000	200	Televisor
Producto H	1000	400	Sofá
Producto I	1200	250	Mesa de comedor
Producto J	500	120	Estantería
Producto K	400	180	Escritorio
Producto L	150	40	Sillón
Producto M	800	120	Cortadora de césped
Producto N	700	80	Equipo de sonido

Producto O	250	70	Escalera plegable aluminio
Producto P	1100	8	Thermomix
Producto Q	1300	300	Máquina de hacer ejercicio
Producto R	600	150	Bicicleta
Producto S	1200	5	Portátil
Producto T	180	24	Juego de vajilla completa

Formulación matemática:

$$Max.\ 800x_A + 1500x_B + 150x_C + 1000x_D + 100x_E + 900x_F + 1000x_G + 1000x_H + 1200x_I + 500x_J + 400x_K + 150x_L + 800x_M + 700x_N + 250x_O + 1100x_P + 1300x_Q + 600x_R + 1200x_S + 180x_T$$

$$s.a.\ 150x_A + 300x_B + 30x_C + 200x_D + 10x_E + 180x_F + 200x_G + 400x_H + 250x_I + 120x_J + 180x_K + 40x_L + 120x_M + 80x_N + 70x_O + 8x_P + 300x_Q + 150x_R + 5x_S + 24x_T \leq 1500$$

$$x_A, x_B, x_C, x_D, x_E, x_F, x_G, x_H, x_I, x_J, x_K, x_L, x_M, x_N, x_O, x_P, x_Q, x_R, x_S, x_T \in \{0,1\}$$

4 METODOLOGÍA

En este apartado se explica la metodología usada para resolver los problemas de optimización que se han planteado en la sección anterior. En concreto se van a tratar dos técnicas: el algoritmo *simplex* y las metaheurísticas.

Dentro del algoritmo *simplex*, que es el que se usa para la resolución del problema de la dieta, se habla sobre el *Solver* de Excel, herramienta usada para problemas de menor complejidad, y sobre *Gurobi*, útil para optimizar problemas con más variables.

En cuanto a las metaheurísticas, se va a utilizar la del algoritmo genético para resolver tanto el *problema del viajante* como el *problema de la mochila*.

Tanto el código de *Gurobi* como los algoritmos genéticos se van a desarrollar en el lenguaje de programación *Python,* que se explicará brevemente a continuación.

4.1 Python

Python es un lenguaje de programación potente utilizado para desarrollar aplicaciones en varias áreas. Surgió en 1991 y sus principales características son la legibilidad de su código y la facilidad para entenderlo. Además, es muy dinámico e interpretado, lo que lo ha convertido en uno de los lenguajes más populares. [34]

Para resolver los problemas de este trabajo se ha utilizado *Spyder*, que es un entorno de desarrollo integrado para la programación en *Python*. *Spyder* utiliza por defecto el intérprete *CPython*, que es uno de los más utilizados. En programación, un intérprete es un programa informático que analiza y ejecuta otros programas.

4.2 Método simplex

El método Simplex es uno de los algoritmos más utilizados para la resolución de problemas de programación lineal, que buscan maximizar o minimizar una función lineal sujeta a restricciones también lineales.

Es un algoritmo iterativo que permite ir mejorando la solución de la función objetivo, generando una amplia variedad de soluciones sucesivas hasta que se encuentre el mejor resultado. El método consiste en ir de vértice en vértice de un poliedro (ver iIlustración 3) de manera que aumente o disminuya la función objetivo del modelo. Dado que el número de vértices que presenta un poliedro solución es finito, si se puede satisfacer el conjunto de restricciones, se encontrará por lo menos una solución óptima.

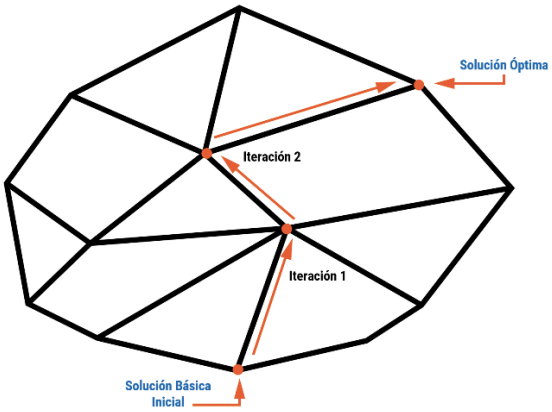


Ilustración 3. Poliedro que representa la solución factible

Fuente: Todas las ilustraciones usadas para explicar el método Simplex pertenecen al artículo *´Método simplex paso a paso: ejemplos de maximizar y minimizar`*

Pasos del método:

1. Definir el problema en forma estándar

   En este primer paso hay que normalizar el signo de los términos independientes multiplicándolos por -1 en caso de ser negativos.

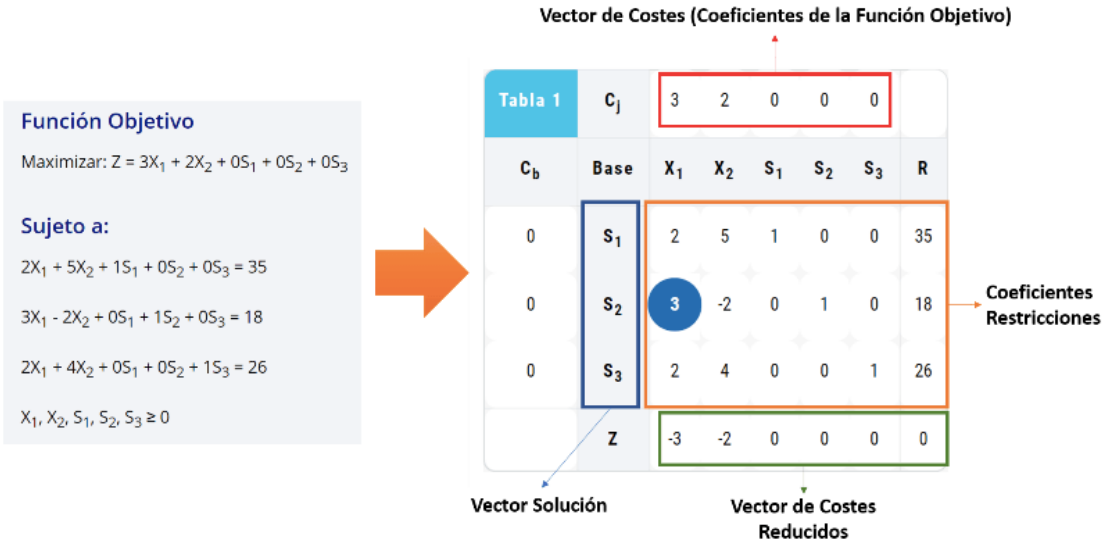
   Además, habrá que convertir las restricciones en igualdades: si la restricción es ‘menor igual’, se introduce una holgura sumando, mientras que si es ‘mayor igual’ se introduce una holgura negativa y una variable artificial. Cuando la restricción es ‘igual’ se añade una variable artificial pero ninguna holgura.

   Finalmente se iguala la función objetivo a cero.

2. Generar matriz

   La matriz inicial está compuesta por el vector de costes (coeficientes de la función objetivo), los coeficientes de las restricciones, el vector costes reducidos, que si no existen variables artificiales será igual al vector de costes multiplicado por -1, y el vector solución, que es donde se coloca la solución básica inicial y se va actualizando al ir mejorarandola. En la Ilustración 4 se ve un ejemplo de la matriz inicial y los distintos vectores de los que está compuesta.

Ilustración 4. Ejemplo generación matriz método simplex



3. Condición de parada y elección de variable entrante y variable saliente

   La condición de parada, también conocida como condición de optimalidad, indica si hemos llegado o no al óptimo fijándonos en la fila de costes reducidos.

   Si el problema es de maximización, se sabe que se ha encontrado la solución óptima cuando todos los costes relativos sean mayores o iguales que cero.

   El caso contrario es el de los problemas de minimización, en los que se busca que los coeficientes del vector de costes reducidos sean menores o iguales que cero.

   Cuando no se ha llegado al óptimo, se utiliza la condición de optimalidad para elegir la variable de entrada (ver Ilustración 5) que entrará en la base para formar parte de la solución. Al igual que antes, la elección de la variable de entrada dependerá de si el problema es maximización o minimización. En el primer caso, se escoge la variable con el coeficiente más negativo en el vector de costes reducidos, y en problemas de minimización ese elige la variable con el coeficiente más positivo. La columna de la variable elegida se llama *columna pivote*. A continuación, se selecciona la variable de salida (ver Ilustración 6). Para ver cuál es, hay que dividir los coeficientes de las restricciones que están en la columna de la variable de entrada entre su respectivo término independiente (coeficientes de la columna R). La división que dé menor valor corresponde a la fila de la variable de salida, que se llamará fila pivote.

Ilustración 5. Variable de entrada método Simplex

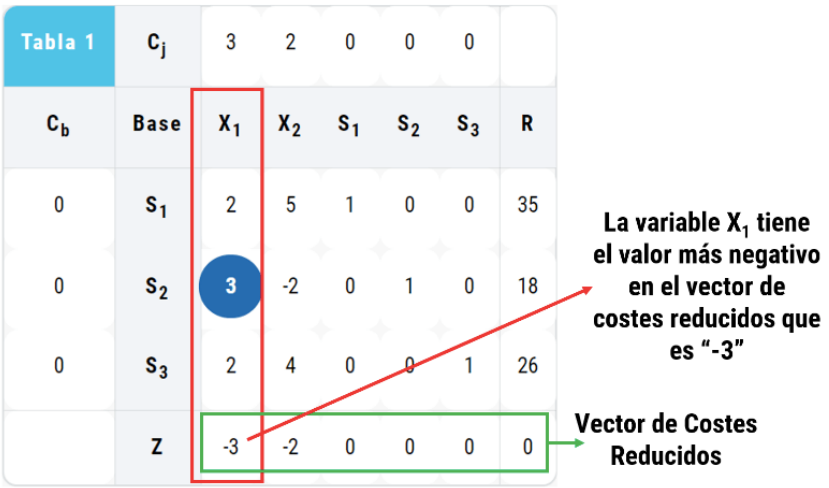
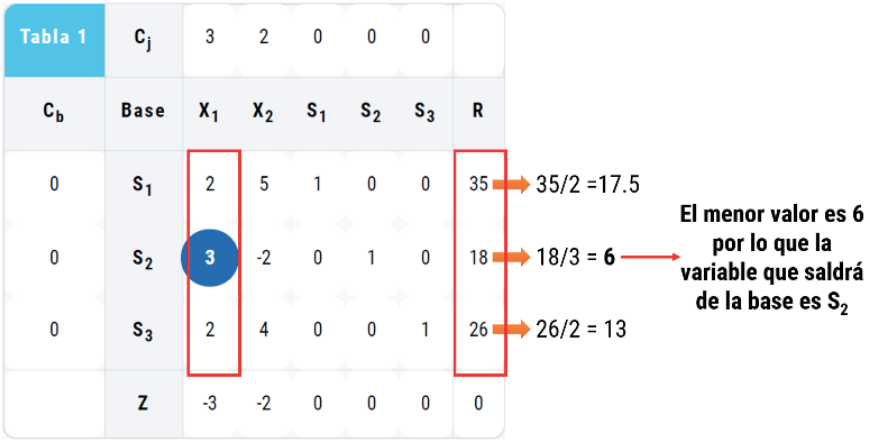


Ilustración 6. Variable de salida método Simplex



4. Actualizar matriz

   Cuando se conoce el elemento pivote, se procede a actualizar la matriz realizando Gauss-Jordan.

   El nuevo valor de cada elemento de la fila pivote se calculará mediante la siguiente fórmula:

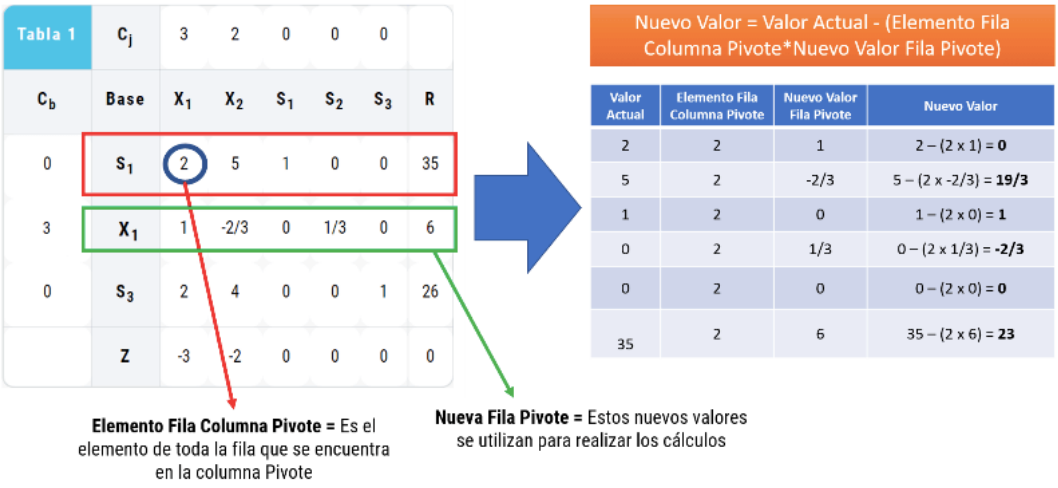
   $$Nuevo\ valor\ fila\ pivote = \frac{Valor\ actual}{Elemento\ pivote}$$

   Para el resto de las filas:

   $$Nuevo\ valor = Valor\ actual - (Anterior\ elemento\ fila\ en\ columna\ pivote * Nuevo\ valor\ fila\ pivote)$$

   Se muestra a continuación cómo se haría esto en la fila 1 (S1) del ejemplo anterior:

Ilustración 7. Actualización matriz método simplex



Se repite esto para todas las filas hasta obtener la matriz actualizada.

5. Comprobar condición de parada

   Si indica que aún no se ha llegado al óptimo volver al paso 3. En caso contrario, se habrá llegado al fin del algoritmo.

La información de esta sección se basa en el contenido de la web *PHPSimplex* [35].

4.2.1 Solver de Excel

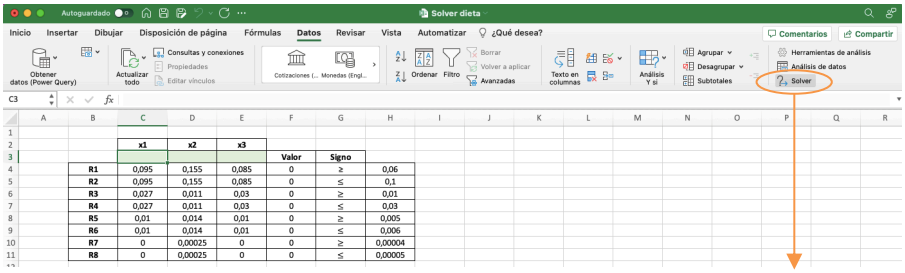
Solver es una herramienta de Excel de gran utilidad a la hora de resolver problemas de programación lineal. Sirve para encontrar el valor óptimo de una celda, la *celda objetivo*. Para ello, *Solver* ajusta los valores de las llamadas *celdas variables de decisión*, cumpliendo siempre con los límites de las *celdas de restricción*.

Esta herramienta va a ser utilizada para resolver el escenario 1 del *problema de la dieta*, ya que es el único de los tres problemas que es lineal y, por tanto, puede resolverse utilizando el método *simplex*.

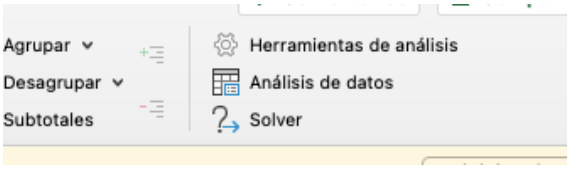
Escribiremos en una hoja de cálculo la fórmula que representa el objetivo que queremos optimizar y los valores de las variables que hay que ajustar para conseguir el objetivo.

Abrimos *Solver*, que está en la pestaña “Datos” tal y como puede verse en la Ilustración 8:

Ilustración 8. Planteamiento problema de la dieta en Excel

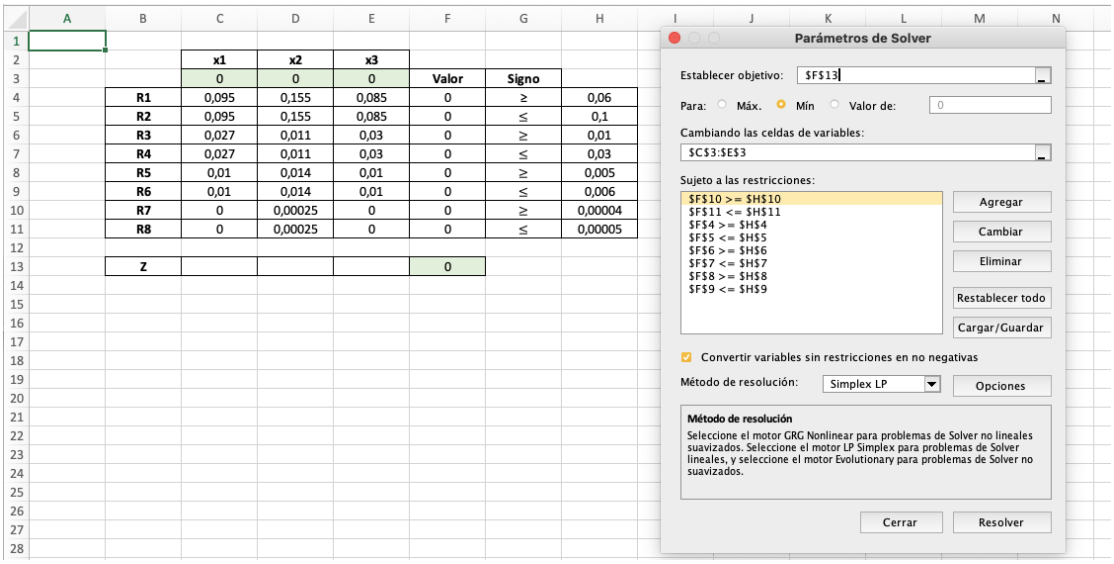


Fuente: Elaboración propia



Programamos el problema: se establece la *celda objetivo* y se señala si se quiere maximizar o minimizar. También se ponen las celdas que contienen las variables que vamos a ajustar y otras restricciones necesarias:

Ilustración 9. Programación del problema en Solver

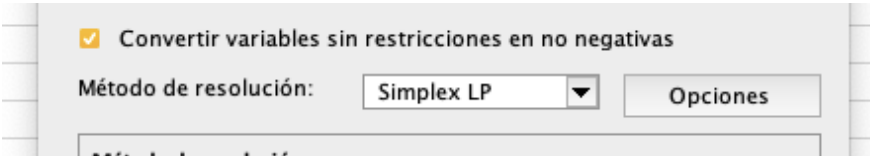


Fuente: Elaboración propia

Se pueden agregar tantas restricciones como hagan falta, pero el número máximo de celdas de variables es 200.

El *Solver* permite usar otros métodos además del *simplex*; por ello, es importante elegir el método de resolución, en este caso *Simplex LP*.

Ilustración 10. Método resolución *Solver*



Fuente: Elaboración propia

Finalmente se pulsa 'Resolver' y, en caso de haberla, *Solver* nos proporcionará la solución óptima con el valor de cada variable y la función objetivo.

La información de esta sección se basa en el contenido de la web del Soporte técnico de Microsoft 365 [36].

4.2.2 Librería Gurobi

La *librería Gurobi* es un software de optimización matemática que, al igual que *Solver*, sirve para resolver diversos problemas de programación. Una de las principales diferencias entre ambas herramientas es que *Gurobi* es mucho rápida y eficiente, pudiendo resolver problemas muy complejos en muy poco tiempo. Por ello, usaremos esta librería para resolver el segundo escenario del *problema de la dieta*, ya que, al tener tantas variables, el *Solver* de Excel es menos eficiente debido a sus limitaciones.

Gurobi incluye implementaciones de alto rendimiento del método simplex y el dual simplex, y un solucionador de barrera paralela [37].

Para el caso de estudio del *problema de la dieta de Stigler* se usará la *librería Gurobi* en el leguaje de programación de *Python*. Para ello, una vez instalado *Gurobi*, habrá que importar la interfaz *Gurobipy*, que nos permitirá resolver complejos problemas de optimización.

Lo siguiente será definir los parámetros y datos del problema, en este caso, lista de alimentos con sus respectivos precios y valores nutricionales, y los requisitos de nutrientes mínimos.

Después se crea el modelo mediante *m=gp.model* (Ilustración 11). Cuando se tenga el modelo creado se añaden variables (*m.addVar*) y, para mayor claridad de los resultados, se les asignan los nombres de los alimentos a los índices correspondientes de las variables de decisión (Ilustración 12).

Ilustración 11. Línea del código para la creación del modelo

```
#Creación del modelo
modelo = gp.Model("Problema de la dieta")
```

Ilustración 12. Líneas del código para la creación de variables

```
#Definición de las variables de decisión
x = modelo.addVars(len(alimentos), name="x")

#Asignación de los nombres de alimentos a los índices correspondientes de las variables de decisión
x = {alimentos[i]: x[i] for i in range(len(alimentos))}
```

La parte clave del código es el establecimiento de la función objetivo y de las restricciones. Como se ve en la Ilustración 13, la función objetivo se establece así: *m.setObjective(* objetivo,

gp.GRB.MINIMIZE). En *m.setObjective* por 'objetivo' se entiende la suma de todos los alimentos multiplicados por sus precios.

Ilustración 13. Línea del código para establecer la función objetivo

```
#Función objetivo
modelo.setObjective(gp.quicksum(precio[i]*x[alimentos[i]] for i in range(len(alimentos))), gp.GRB.MINIMIZE)
```

Para las restricciones se usa *m.addConstr* y se establece que la suma de todos los alimentos multiplicados por los distintos valores nutricionales sea mayor que los mínimos fijados antes (Ilustración 14).

Ilustración 14. Líneas del código para establecer las restricciones

```
#Restricciones
modelo.addConstr(gp.quicksum(calorias[i]*x[alimentos[i]] for i in range(len(alimentos))) >= min_calorias, "calorias_min")
modelo.addConstr(gp.quicksum(proteina[i]*x[alimentos[i]] for i in range(len(alimentos))) >= min_proteina, "proteinas_min")
modelo.addConstr(gp.quicksum(calcio[i]*x[alimentos[i]] for i in range(len(alimentos))) >= min_calcio, "calcio_min")
modelo.addConstr(gp.quicksum(hierro[i]*x[alimentos[i]] for i in range(len(alimentos))) >= min_hierro, "hierro_min")
modelo.addConstr(gp.quicksum(vitaminaA[i]*x[alimentos[i]] for i in range(len(alimentos))) >= min_vitaminaA, "vitaminaA_min")
modelo.addConstr(gp.quicksum(vitaminaB1[i]*x[alimentos[i]] for i in range(len(alimentos))) >= min_vitaminaB1, "vitaminaB1_min")
modelo.addConstr(gp.quicksum(vitaminaB2[i]*x[alimentos[i]] for i in range(len(alimentos))) >= min_vitaminaB2, "vitaminaB2_min")
modelo.addConstr(gp.quicksum(vitaminaB3[i]*x[alimentos[i]] for i in range(len(alimentos))) >= min_vitaminaB3, "vitaminaB3_min")
modelo.addConstr(gp.quicksum(vitaminaC[i]*x[alimentos[i]] for i in range(len(alimentos))) >= min_vitaminaC, "vitaminaC_min")
```

Finalmente se resuelve el modelo mediante *m.optimize* y se imprime la solución.

Ilustración 15. Líneas del código para resolver el modelo

```
#Resolver el modelo
modelo.optimize()
```

La información de esta sección se basa en el contenido de la *web GUROBI OPTIMIZATION* [38].

4.3 Metaheurísticas

El término 'metaheurística' fue introducido por primera vez en el artículo semanal sobre búsqueda tabú de Fred Glover en 1986 y se obtiene de combinar el prefijo 'meta', que significa 'a un nivel superior', y la palabra griega 'heurística', encontrar.

En investigación operativa, un método heurístico es aquél que, aunque no garantice la optimalidad, proporciona una solución de alta calidad con un coste computacional razonable. Por lo tanto, las metaheurísticas se pueden definir *como métodos aproximados diseñados para mejorar problemas de optimización combinatoria en los que los métodos heurísticos no son efectivos*. El objetivo es encontrar el óptimo de un problema complejo en el menor tiempo posible, lo que supone un gran avance. [39]

Es una combinación de varias áreas distintas: las matemáticas aplicadas, la ingeniería informática, ingeniería industrial y las disciplinas empresariales, lo que supone una gran ayuda para la toma de decisiones. Por ello, es una disciplina que se ha convertido en vital para las grandes empresas y otros tipos de sectores como la medicina. [40]

A lo largo de los años, se han desarrollado muchas metaheurísticas según el tipo de problema a resolver. En este caso, para el *problema del viajante* y el *de la mochila*, se ha decidido utilizar algoritmos genéticos. Como hemos dicho, no se garantiza siempre la solución óptima, algunos de los motivos son los siguientes:

- Convergencia hacia un óptimo local
- Configuración inicial inadecuada: la efectividad y rendimiento del algoritmo dependen de la configuración inicial de parámetros y población
- Limitaciones computacionales

4.3.1 Algoritmo Genético

Los principios básicos de los algoritmos genéticos fueron desarrollados por John H. Holland en 1975 y están fundamentados en las Leyes de la Vida Natural descritas por Charles Darwin [41].

El algoritmo genético es un método para resolver problemas de búsqueda y optimización basados en la mecánica de la selección natural, la evolución y la genética. El método elige las mejores soluciones a medida que se desarrolla la población y las combina para producir nuevas soluciones que se evalúan y compiten con la población inicial.

Aunque el algoritmo cambia según la aplicación, hay una estructura común para todos:

1. Crear una población inicial de x individuos.

   En el algoritmo genético, *individuo* hace referencia a una posible solución y la *población* es un conjunto de *individuos*, es decir, un conjunto de posibles soluciones. Esta población irá evolucionando mediante distintos procesos de selección para intentar encontrar la solución óptima. Se establece una población inicial y a partir de ésta ir realizando cambios y mejoras. Esta población inicial suele generarse aleatoriamente; sin embargo, en algunas ocasiones puede ser conveniente generarla siguiendo algún criterio específico.

2. Calcular la aptitud (*fitness*) de cada individuo.

   La función aptitud es la que evalúa la idoneidad de una solución y es la encargada de guiar el proceso de selección en función de los resultados que se vayan obteniendo. Está estrechamente

relacionada con la función objetivo por lo que la aptitud será muy diferente en función del tipo de problema y lo que se quiera optimizar en cada caso.

3. Se seleccionan los mejores individuos

   Para empezar a buscar la solución óptima se seleccionan unos primeros individuos (padres) siguiendo distintos métodos de selección, como, por ejemplo, la selección por ruleta, por torneo, por ranking o elitista. Todas estas técnicas evalúan y comparan individuos de la población inicial para identificar a los mejores y ordenar según la aptitud para empezar el proceso de cruce, mutación, etc.

4. Se cruzan los individuos seleccionados (fase de reproducción)

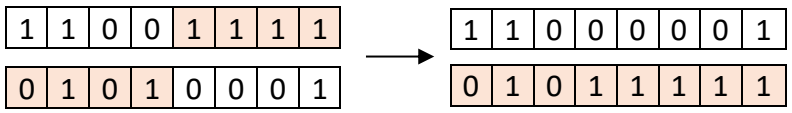
   El cruce es una función cuya finalidad es combinar genes de los padres para que los descendientes hereden las características beneficiosas de los dos padres, es decir, combinar dos soluciones distintas y crear una nueva a partir de éstas.

   Hay distintos tipos de cruce: cruce en un punto, cruce en dos puntos, aleatorio, etc. Una vez más, la selección de un tipo u otro depende del caso en concreto.

   Cruce en un punto:

   En la Ilustración 16 se presenta un ejemplo de cruce en un punto. Este tipo de cruce consiste en seleccionar un punto, normalmente al azar, y a partir de ahí hacer el intercambio de genes. Como puede observarse, a partir de dos soluciones iniciales llamadas de padres (parte izquierda de la ilustración), se generan dos soluciones nuevas como combinación de las anteriores, también llamadas de hijos (parte derecha de la ilustración).

Ilustración 16. Cruce en un punto

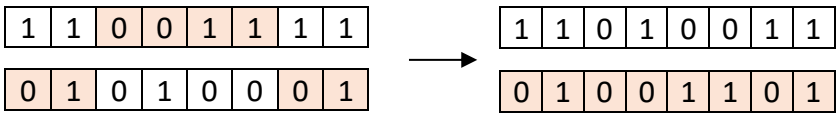


Fuente: Elaboración propia

   Cruce en dos puntos:

   Este caso es similar al anterior, pero en lugar de realizarse el cruce en un punto se realiza en dos, normalmente aleatorios. Como se puede observar en la Ilustración 17, se seleccionan dos puntos, y se realiza el intercambio de genes en los dos, obteniendo nuevamente dos soluciones nuevas (hijos) como combinación de las anteriores (padres).

Ilustración 17. Cruce en dos puntos

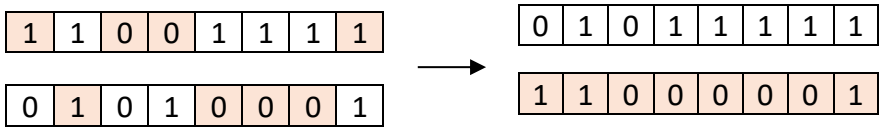


Fuente: Elaboración propia

Cruce aleatorio:

El proceso de intercamabio de genes vuelve a ser muy parecido a los casos anteriores. En este caso la diferencia es que no hay un número determinado de puntos de cruces, sino que es aleatorio. Sin embargo, como se ve en la Ilustración 18, el resultado es el mismo: dos hijos (parte derecha de la ilustración) a partir de unos padres (parte izquierda de la ilustración) que han intercambiado sus genes.

Ilustración 18. Cruce aleatorio

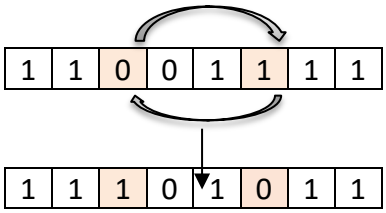


Fuente: Elaboración propia

5. Se introducen pequeñas mutaciones en algunos individuos aleatoriamente.

   Mediante la mutación, que se realiza en función a una determinada tasa, se introducen cambios aleatorios en los individuos para evitar que el algoritmo quede atrapado en óptimos locales y poder explorar un rango más amplio y variado de soluciones. Consiste en seleccionar aleatoriamente puntos de un individuo, generalmente dos puntos, e intercambiar los genes de posición. En el caso de la Ilustración 19 se observa cómo el gen de la posición 3, que vale 0, pasa a la posición 6, y viceversa. Por lo que en la solución nueva (parte inferior de la ilustración) el gen de la posición 3 valdrá 1 y el de la posición 6 será 0.

Ilustración 19. Mutación



Fuente: Elaboración propia

La información de esta sección se basa en el contenido de la *web cienciadedatos.net* [42].

En este algoritmo también son muy importantes los parámetros y el criterio de selección de éstos. Los más comunes y los que se han utilizado en este trabajo son el tamaño de la población, el número de generaciones y tasa de mutación.

El tamaño de población es el tamaño de individuos en cada generación de la población y es de los parámetros más importantes, ya que tiene un gran impacto en la diversidad y tiempo computacional. Un tamaño de la pobación grande implica una exploración más exhaustiva del espacio de posibles soluciones, pero a costa de un mayor tiempo de ejecución. Mientras que una población pequeña proporciona soluciones menos diversas pero mayor eficiencia computacional.

El número de generaciones es el número de veces que se ejecuta el algoritmo hasta parar. Al igual que ocurre con el tamaño de población, con un número grande de generaciones se consigue mayor convergencia hacia soluciones óptimas, pero también mayor tiempo de ejecución, por lo que hay que encontrar el equilibrio, en función de la complejidad del problema, entre calidad de soluciones y coste computacional.

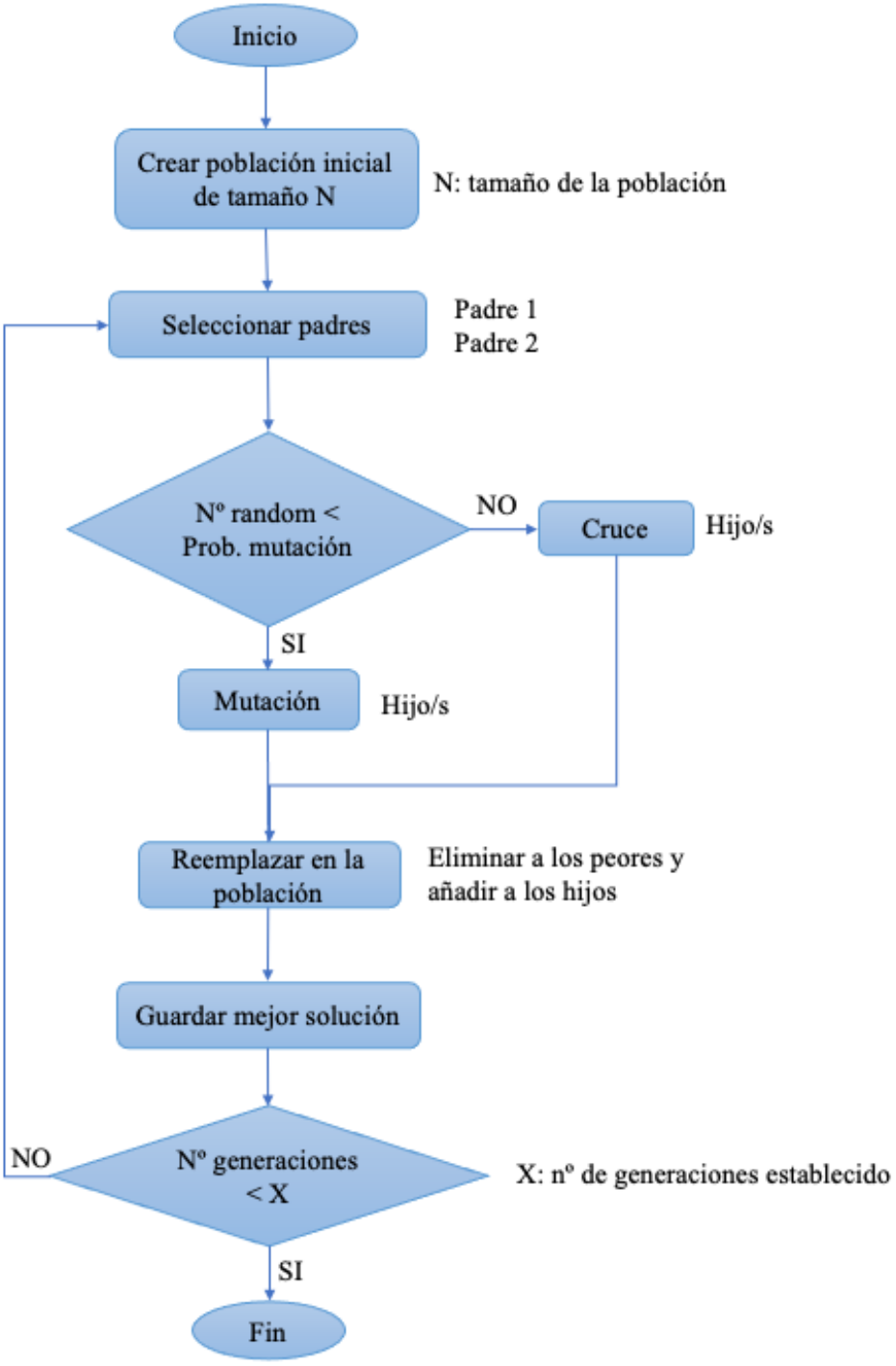
Respecto a la tasa de mutación, determina la probabilidad de que se mute algún gen de un individuo. Suele ser bastante pequeña, ya que esto suele ser suficiente para garantizar la diversidad genética adecuada, aunque depende del tipo de problema que se esté tratando. La tasa de mutación es muy útil para evitar que el algritmo se quede atrapado en un óptimo local, que es una solución mejor que sus vecinas, pero no necesariamente la mejor.

Cabe destacar que la elección de estos parámetros puede llegar a ser algo subjetiva y depende de muchos factores, por lo que es conveniente dar distintos valores que ayuden a evaluar y comparar distintas regiones del espacio de búsqueda para determinar cuáles funcionan mejor en cada situación. En este trabajo se realizarán varias ejecuciones con distintas combinaciones de los parámetros para encontrar los valores que ayuden a encontrar la solución óptima.

En resumen, es una técnica de programación que imita a la evolución biológica para resolver problemas de optimización. Se parte de una población inicial consituida por posibles soluciones. Éstas se evalúan por la función de aptitud. Las mejores soluciones se reproducen y se las hace evolucionar por procesos similares a los que han ido experimentando los seres vivos a lo largo de la historia: selección, cruce y mutación. Tras realizar un determinado número de iteraciones, se selecciona la mejor solución encontrada a lo largo de la ejecución del algoritmo.

En la Ilustración 20 se presenta un esquema general con los distintos pasos del algoritmo comentados en esta sección.

Ilustración 20. Esquema con los pasos del algoritmo genético



Fuente: Elaboración propia

5 RESULTADOS

En este apartado se expondrán los resultados obtenidos de los problemas planteados en el punto 3 y se verá si se han cumplido los objetivos deseados.

Como se explicó en la metodología, cada problema ha sido resuelto de una forma distinta y utilizando diferentes softwares.

5.1 Problema de la dieta

5.1.1 Escenario 1

Como se ha explicado en el apartado de la metodología, el primer escenario del *problema de la dieta* se ha resuelto con *Solver* de Excel ya que, al tener pocas variables y restricciones, *Solver* garantiza que se encontrará la solución óptima.

Cada restricción se ha dividido en dos, una para establecer el mínimo y otra para establecer el máximo de cada nutriente.

A continuación, se muestra la tabla de Excel con las restricciones y los valores de las variables y función objetivo generada por *Solver*:

Tabla 7. Resultados del escenario 1 del problema de la dieta

	x1	**x2**	**x3**			
	0,305	0,2	0	**Valor**	**Signo**	
R1	0,095	0,155	0,085	0,06	≥	0,06
R2	0,095	0,155	0,085	0,06	≤	0,1
R3	0,027	0,011	0,03	0,01044	≥	0,01
R4	0,027	0,011	0,03	0,01044	≤	0,03
R5	0,01	0,014	0,01	0,005852	≥	0,005
R6	0,01	0,014	0,01	0,00585	≤	0,006
R7	0	0,00025	0	0,00005	≥	0,00004
R8	0	0,00025	0	0,00005	≤	0,00005

Z	8,49	6,36	18,37	3,86368

La solución óptima proporcionada nos dice que se le deberá dar al bebé diariamente 0,31kg del potito A y 0,2kg del potito B. No será necesario darle nada del potito C, ya que con los dos primeros se cumplen los requisitos nutricionales a menor precio.

Si interpretáramos el resultado en un horizonte temporal mayor, por ejemplo, de una semana, la solución óptima se corresponde a darle al bebé algo más de 2 potitos A y algo más de 1 potito B.

En la última fila de la tabla se ve la suma de estas variables multiplicada cada una por su respectivo precio, lo que da como resultado el valor de la función objetivo, que es lo que habrá que gastarse al día para comprar la cantidad necesaria de potitos. Este coste mínimo es de 3,86 € al día. Lo que supondría un coste de 27 € semanales.

También se puede comprobar en la solución que se cumplen las restricciones. Las únicas restricciones que se cumplen con signo de igualdad son la primera (correspondiente al mínimo de hidratos de carbono) y la última (correspondiente al máximo de vitamina C), mientras que en el resto de las restricciones aparecen holguras. Volviendo a las ocho restricciones originales tenemos:

Hidratos de carbono: $0{,}06 \leq 0{,}06 \leq 0{,}1$

Proteínas: $0{,}01 \leq 0{,}01044211 \leq 0{,}03$

Fibra: $0{,}005 \leq 0{,}00585263 \leq 0{,}006$

Vitamina C: $0{,}00004 \leq 0{,}00005 \leq 0{,}00005$

5.1.2 Escenario 2

Mediante la aplicación del método *simplex*, esta vez con la *librería Gurobi*, se han identificado las cantidades necesarias de cada alimento que cumplan con los requisitos y a un coste mínimo.

La selección de alimentos que proporciona la dieta óptima a menor precio es la siguiente: harina de trigo (x1), hígado (x30), col (x46), espinacas (x52) y alubias blancas (x69).

En la siguiente tabla se muestran los resultados obtenidos, donde se ven las cantidades diarias requeridas de cada alimento seleccionado y el precio total diario en centavos. Además, a la derecha de la tabla se puede comprobar el cumplimiento de los distintos requerimientos nutricionales. Para mejor visualización se han omitido de la tabla los alimentos no elegidos.

Tabla 8. Resultados del escenario 2 del problema de la dieta, valores diarios

	x1	**x30**	**x46**	**x52**	**x69**			
	0,81997	0,00706	0,30309	0,06182	1,03438	**Valor**	**Signo**	
Calorías	1,6092	0,5896	0,0962	0,0891	1,5871	3	≥	3
Proteína	50,796	89,244	4,625	8,586	99,769	147,4135	≥	70
Calcio	0,072	0,0536	0,148	0	0,6726	0,8	≥	0,8
Hierro	13,14	37,252	1,332	11,178	46,728	60,4669	≥	12
Vitamina A	0	45,3456	0,2664	74,3904	0	5	≥	5
Vitamina B1	1,9944	1,7152	0,333	0,4617	2,2656	4,1204	≥	1,8
Vitamina B2	1,1988	13,6144	0,1665	1,1178	1,4514	2,7	≥	2,7
Vitamina B3	15,876	84,688	0,962	2,673	12,803	27,316	≥	18
Vitamina C	0	140,7	198,653	223,155	0	75	≥	75

Z	3,6	26,8	3,7	8,1	0,62667	10,866

Para ver lo que supone esto al año y compararlo con los resultados obtenidos por Stigler en 1939, se multiplican los valores por 365. En las tablas mostradas a continuación se ven las diferencias entre ambos resultados:

Tabla 9. Resultados de Stigler 1939

	Agosto 1939	
Alimento	**Cantidad**	**Coste**
Harina de trigo	370 lb.	13,33 $
Leche evaporada	57 cans	3,84 $
Col	111 lb.	4,11 $
Espinacas	23 lb.	1,85 $
Alubias blancas secas	285 lb.	16,80 $
	Coste total	39,93 $

Tabla 10. Resultados de este trabajo 2023

	Mayo 2023	
Alimento	**Cantidad**	**Coste**
Harina de trigo	299,29 lb.	10,77 $
Hígado	2,58 lb	0,69 $
Col	110,63 lb.	4,09 $
Espinacas	22,57 lb.	1,83 $
Alubias blancas secas	377,55 lb.	22,28 $
	Coste total	39,66 $

Comparando los resultados obtenidos ahora con los obtenidos por Stigler inicialmente se observan algunas diferencias: la leche evaporada es sustituida por hígado y las cantidades anuales de los alimentos seleccionados varían un poco. Todo esto produce una pequeña disminución en el precio final, que pasa de 39,93 $ a 39,66 $. Aunque la mejora parezca pequeña, al multiplicarlo por el número total de soldados, el impacto que tiene en el ahorro total es bastante significativo.
El motivo de la mejora de la solución probablemente se deba a la precisión de los ordenadores actuales y a haber arrastrado más decimales al resolverlo, ya que los datos usados han sido los mismos que usó Stigler en 1939, basados en los precios y valores nutricionales de la época.

Hay que tener en cuenta las limitaciones del problema, ya que los requisitos nutricionales y precios están basados en los datos de Estados Unidos en 1939 y no se consideran las necesidades específicas de algunos grupos de la población. A pesar de esto, con la resolución de este problema se demuestra que es factible encontrar rápida y eficientemente una combinación óptima de alimentos que cumpla con los requisitos nutricionales y al menor precio.
Actualmente hay mucha más información y conocimiento sobre el tema, por lo que, con los datos y consideraciones necesarias en cada caso, la aplicación de este problema es una herramienta para tratar y resolver cuestiones relacionadas con la nutrición, salud y sostenibilidad.

Como ya se ha comentado, el algoritmo genético es una técnica efectiva para resolver problemas complejos de este tipo, sin embargo, se sabe que no siempre garantiza la solución óptima. Por ello, es conveniente ejecutar el código varias veces para explorar distintas soluciones y acercarse lo máximo posible al óptimo. Esto se hará en ambos escenarios del *problema del viajante* y el *de la mochila*, y después de varias ejecuciones se seleccionará la que proporcione el mejor resultado.

5.2 Problema del viajante

Antes de exponer los resultados obtenidos se explicará más detalladamente el funcionamiento del algoritmo genético aplicado al *problema del viajante*.

5.2.1 Algoritmo genético aplicado al *problema del viajante* (TSP)

Particularizando lo visto en el apartado 3 sobre el algoritmo genético para el *problema del viajante*. Se tiene como dato un vector con las distintas ciudades y una matriz con las distancias de unas a otras, por lo que un gen es una ciudad, un individuo representa una posible ruta que pase por todas las ciudades del problema y vuelva a la primera, y una población será un conjunto de rutas, de entre las cuáles se buscará la mejor mediante exploraciones y modificaciones.
El TSP busca encontrar la ruta más corta que permita visitar todas las ciudades y volver a la inicial, de manera que la aptitud o *fitness* en este caso es la distancia total recorrida y el objetivo es minimizarla. Para ello se creará un bucle que recorra la ruta entera y vaya sumando la distancia entre una ciudad y su sucesora. También es importante tener en cuenta fuera del bucle la distancia entre la última ciudad de la ruta y la primera. Finalmente se suman todas estas distancias para obtener la total. Todo esto se llevará a cabo en la función '*calcular_fitness'*.

Parámetros
Para la selección de parámetros se ha experimentado combinando los distintos parámetros (tamaño de población, número de generaciones y tasa de mutación), teniendo en cuenta el número de ciudades, por lo que los valores del escenario 1 y 2 serán distintos. En estos experimentos se han realizado 10 ejecuciones para cada combinación y se ha estudiado la media, desviación típica y mejor y peor resultado en cada caso. El número de generaciones se estableció desde el principio, sin necesidad de realizar combinaciones, de manera que fuera un número suficientemente grande para poder explorar suficientes posibles soluciones sin un coste computacional excesivo. Aún así es importante prestar atención en las distintas ejecuciones para ver a partir de que generación se suele estabilizar la solución.

A la hora de escoger parámetros tendrán especial importancia el mejor resultado y la media. Esta última indica que, cuánto menor sea, menores serán las distancias que se obtienen con esa combinación.

A la vista de los resultados, para el escenario 1 se han escogido los siguientes parámetros:

Tabla 11. Pruebas combinación de parámetros TSP escenario 1

		Media	Desviación	Mejor resultado	Peor resultado
TP 300	**TM 0,3**	3161,6	73,92	3086	3300
	TM 0,5	3148,7	46,68	3086	3218
	TM 0,7	3366,7	119,03	3203	3560
TP 500	**TM 0,3**	3178,1	39,04	3086	3218
	TM 0,5	3156,8	55,34	3086	3232
	TM 0,7	3330,8	139,46	3086	3565
TP 700	**TM 0,3**	3199,8	104,12	3086	3477
	TM 0,5	3210,5	109,15	3086	3498
	TM 0,7	3383,2	138,35	3158	3587

Tamaño de población = 300

Número de generaciones = 500

Tasa de mutación = 0,5

Y para el segundo escenario:

Tabla 12. Pruebas combinación de parámetros TSP escenario 2

		Media	Desviación	Mejor resultado	Peor resultado
TP 500	**TM 0,3**	5896	76,40	5814	6050
	TM 0,5	5901,8	102,34	5814	6050
	TM 0,7	6027,3	44,77	5911	6050
TP 1000	**TM 0,3**	5926,8	94,50	5814	6050
	TM 0,5	5880,2	57,49	5814	5931
	TM 0,7	5902	91,81	5814	6050
TP 1500	**TM 0,3**	5888,7	75,46	5814	6050
	TM 0,5	5937,1	68,99	5814	6050
	TM 0,7	5953,2	93,98	5814	6050

Tamaño de población = 1000

Número de generaciones = 500

Tasa de mutación = 0,5

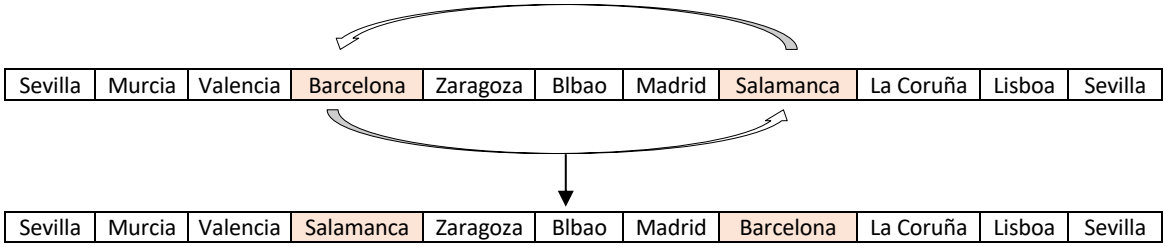
Procedimiento

Como se comentó previamente, aunque normalmente la población inicial se suele generar aleatoriamente, en este trabajo se ha decido que para el algoritmo genético aplicado al *problema del viajante* con 30 ciudades (escenario 2) puede ser más eficiente hacerlo siguiendo un criterio específico para arrancar desde un buen punto de partida. En este caso, para crear a un individuo se seleccionará una ciudad al azar y a partir de ésta se irán seleccionando las siguientes según estén

más cerca de la última ciudad generada. Para crear la población inicial completa se repetirá la función '*crear_individuo'* tantas veces como el tamaño de la población. Con este método se consigue una población inicial más informada y se podrá llegar antes a las mejores soluciones.
Sin embargo, para el problema con 10 ciudades (escenario 1), al ser más simple y de menor tamaño, la aleatoriedad puede ser beneficiosa. Por lo que en este escenario la función '*crear_individuo'* crea individuos completamente aleatorios

Después de haber creado una población inicial y haber definido la función '*calculo_fitness'* (función para el cálculo de la aptitud), se crean unas variables en las que guardar las mejores soluciones , '*mejores_distancias'*, *'mejor_individuo'*, que se inicializa a 0, y *'mejor_fitness'*, que inicialmente valdrá infinito. Entonces comienza el bucle principal, que ejecuta el algoritmo tantas veces como número de generaciones haya.
Primero se calcula el *fitness* de todos los individuos de la población y se ordena la población de menor a mayor distancia, es decir, de mejor a peor *fitness*.
A partir de la población ordenada se seleccionan los padres. Esto se hará llamando a la función *padres*, donde se cogen los dos primeros individuos de '*poblacion_ordenada'* y estos serán los dos padres a partir de los cuales se irán generando nuevos individuos.
A continuación, se genera un número aleatorio entre 0 y 1. Si es mayor que la tasa de mutación, se llevará a cabo el cruce y en caso contrario, mutación. Para el cruce se define un punto de cruce aleatorio y se generan dos hijos intercambiando los genes de los padres. El *hijo1* está formado por los genes del *padre1* antes del punto de cruce y luego se añaden los genes del *padre2* que no estén en el segmento cogido del *padre1* en el orden que aparezcan en el *padre2*. En el *hijo2* se copian primero los genes del *padre2* hasta el punto de cruce y luego los que falten del 'padre1'. Para el caso de la mutación se seleccionan aleatoriamente dos índices entre 0 y la longitud del individuo y se cambian sus posiciones en el individuo, introduciendo cambios en la secuencia de los genes. A continuación, se muestra un ejemplo de este tipo de mutación:

Ilustración 21. Ejemplo de mutación aplicado al TSP



Fuente: Elaboración propia

Después de realizar estas operaciones, se eliminarán de la población ordenada los dos últimos individuos, que corresponden a las rutas con mayor distancia, y se añadirán en su lugar los dos

hijos. Esta será la nueva población, para la cuál se volverá a calcular el *fitness* de todos sus individuos y se ordenarán en una nueva población ordenada.

El último paso del bucle es asegurar que la mejor solución de cada generación se está guardando, lo que en los algoritmos genéticos se conoce como elitismo. En este caso, se creará una variable llamada *'mejor_actual'* y se igualará al primer elemento de la población ordenada, se calculará su aptitud y se le asignará este valor a otra nueva variable llamada '*fitness_mejor_actual'*. Una vez se tienen estos valores se compararán con los mejores valores de la generación anterior (*'mejor_individuo'* y *'mejor_fitness'*). Si *'fitness_mejor_actual'* es mejor que '*mejor_fitness'* se actualizará el valor de *'mejor_fitness'* y '*mejor_individuo'*.

Otro aspecto clave que hay que tener en cuenta a la hora de codificar este algoritmo es la restricción del problema del viajante que indica que hay que pasar por todas las ciudades una única vez. En este caso no se ha puesto una condición explícita en la función *fitness* que evite pasar por la misma ciudad dos veces, pero hay una serie de funciones que garantizan que esto no pase. Para empezar, los primeros individuos se crean añadiendo en cada paso una de las ciudades del vector 'ciudades' y eliminándola para que no se vuelva a incluir en ese individuo, de modo que en cada solución (individuo) aparezcan todas las ciudades y solo una vez. Además, también se tiene en cuenta que los cruces y mutaciones no provoquen repeticiones de ciudades. Por un lado, la función cruce genera *hijos* asegurando que no hay duplicados en éstos ya que se intercambian segmentos de las rutas de los *padres* sin repetir ciudades. En cuanto a la mutación, lo que se hace en este caso es intercambiar al azar las posiciones de dos genes de un individuo. Por lo que, si el individuo inicial no tenía ciudades repetidas, el mutado tampoco las tendrá, ya que no se ha añadido ninguna nueva, sino que se han cambiado de lugar.

A continuación, se muestran los datos obtenidos en ambos escenarios tras implementar en *Python* el algoritmo explicado. El código escrito en *Python* para la resolución del TSP da como resultado un vector con las ciudades en el orden que se realiza la ruta y la distancia total de ésta. Además, se imprime una gráfica con las mejores soluciones en cada generación para ver cómo ha ido evolucionando la solución.

5.2.2 Escenario 1

La ventaja de aplicar el algoritmo genético en este tipo de problemas radica en la rapidez con la que proporciona soluciones óptimas y cálculo de las distancias totales.

Se presentan a continuación algunos de los resultados que se han ido obteniendo las distintas veces que se ha ejecutado el código del algoritmo genético aplicado al TSP con 10 ciudades. Como se vió en la Tabla 13 al explicar el algoritmo genético aplicado al problema del viajante, los parámetros utilizados son tamaño de población = 300, número de generaciones = 500 y tasa de mutación = 0,5.

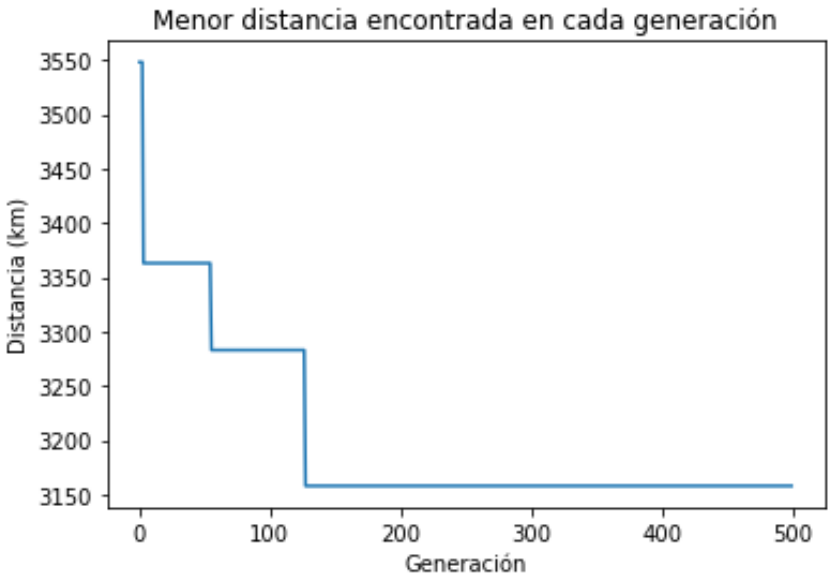
Primero se ve una ruta aleatoria y luego se comparará con las rutas proporcionadas por el agortimo:

La ruta aleatoria es: ['Valencia', 'Murcia', 'Barcelona', 'Bilbao', 'Lisboa', 'Madrid', 'Sevilla', 'Zaragoza', 'Salamanca', 'La Coruña', 'Valencia']
Con una distancia de: 4923 km

Ilustración 22. Mapa ruta aleatoria

Comparando este resultado con los siguientes se puede observar cómo, gracias al algoritmo genético, se consigue reducir significativamente la distancia total recorrida, obteniendo rutas mucho más eficientes:

Ilustración 23. Evolución de la mejor solución encontrada por el algoritmo. Ruta de 3158 km

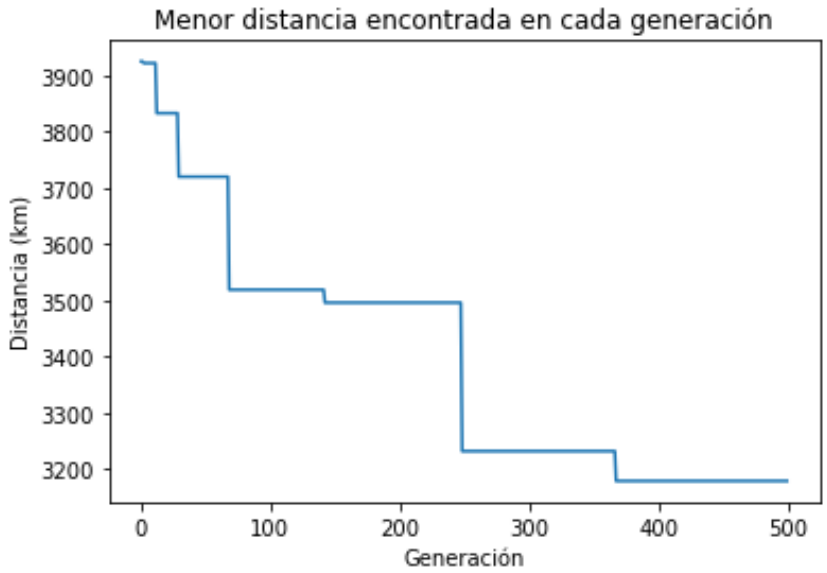


La mejor ruta encontrada es: ['Madrid', 'Murcia', 'Valencia', 'Barcelona', 'Zaragoza', 'Bilbao', 'La Coruña', 'Lisboa', 'Sevilla', 'Salamanca', 'Madrid']
Con una distancia de: 3158 km

Esta solución se obtiene después de 500 iteraciones del algoritmo. En la Ilustración 23 se observa cómo la función objetivo va mejorando a medida que avanzan las iteraciones hasta estabilizarse alrededor de la iteración 120.

Para asegurarnos del correcto funcionamiento del algoritmo, este se ha implementado varias veces. Por ejemplo, en la Ilustración 24 se muestra la evolución de la mejor solución encontrada tras 500 iteraciones. En este caso la función objetivo se estabiliza aproximadamente en la iteración 370. Aquí se ve la importancia de establecer un número de generaciones alto, ya que, aunque la mayoría de las veces la solución se estabiliza pronto, habrá ejecuciones, como esta, en las que tarde más la función objetivo en estabilizarse.
Además, puede comprobarse que en este caso se obtiene una solución final peor que la anterior. Esto es un problema común cuando utilizamos metaheurísticas, ya que, como se comentó antes, estos métodos no aseguran alcanzar el óptimo global del problema y por ello es importante realizar varias ejecuciones.

Ilustración 24. Evolución de la mejor solución encontrada por el algoritmo. Ruta de 3179 km



La mejor ruta encontrada es: ['Murcia', 'Valencia', 'Barcelona', 'Zaragoza', 'Bilbao', 'La Coruña', 'Salamanca', 'Lisboa', 'Sevilla', 'Madrid', 'Murcia']
Con una distancia de: 3179 km

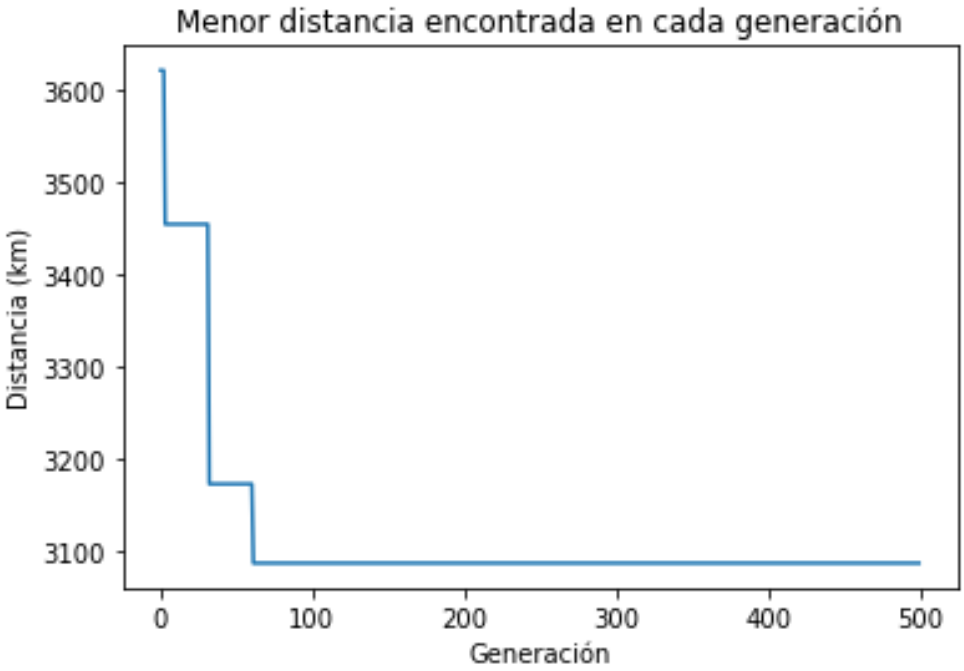
Tras implementar el algoritmo diez veces, la mejor solución encontrada es la que corresponde a una ruta de 3086 km. En la Tabla 13 se observa la media de la mejor solución encontrada en estas diez ejecuciones, así como la desviación y el mejor y peor resultado. Los valores mostrados en esta tabla demuestran que el algoritmo proporciona siempre soluciones relativamente cercanas a la óptima.

En la Ilustración 25 se ve la evolución de la solución hasta llegar a la mejor solución encontrada en las 500 generaciones.

Tabla 13. Resultado de las 10 ejecuciones del escenario 1 del TSP

Media	Desviación	Mejor resultado	Peor resultado
3148,7	46,68	3086	3218

Ilustración 25. Evolución de la mejor solución encontrada por el algoritmo. Ruta de 3086 km



La mejor ruta encontrada es: ['Sevilla', 'Lisboa', 'La Coruña', 'Salamanca', 'Madrid', 'Bilbao', 'Zaragoza', 'Barcelona', 'Valencia', 'Murcia', 'Sevilla']
Con una distancia de: 3086 km

A la vista de los resultados proporcionados por el código en *Python*, el cantante recorrerá un total de 3086 km durante su gira. Empezará y acabará en Sevilla y la ruta que debe seguir es la siguiente: Sevilla – Lisboa – La Coruña – Salamanca – Madrid – Bilbao – Zaragoza – Barcelona – Valencia – Murcia – Sevilla.

Vista en el mapa, la ruta queda así:

Ilustración 26. Mapa con la mejor ruta. TSP 10 ciudades

Se puede observar que no importa en qué ciudad se empiece mientras se siga el orden del resto de la ruta establecida, por ejemplo: Madrid – Bilbao – Zaragoza – Barcelona – Valencia – Murcia – Sevilla – Lisboa – La Coruña – Salamanca – Madrid.

La distancia total es la suma de ir de una ciudad a otra siguiendo la ruta establecida en la solución. En este caso la mejor solución da como resultado una distancia de 3086 km, que son casi 2000 km menos que la ruta aleatoria mostrada al principio. Esta mejora se consigue en menos de 5 segundos, tiempo que tarda en proporcionar la solución el código.

5.2.3 Escenario 2

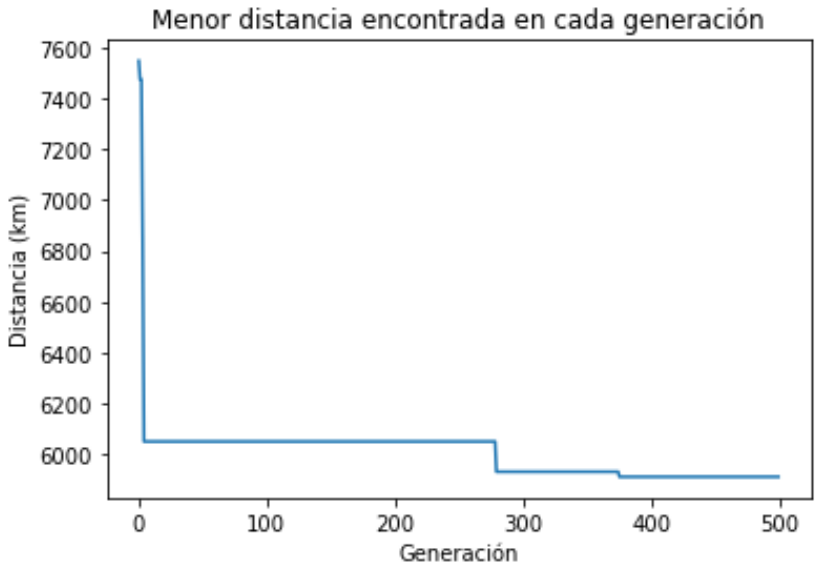
Para 30 ciudades, el número de posibles soluciones es aproximadamente $2{,}6525286\ x\ 10^{32}$. Esto hace que sea extremadamente difícil llegar a la mejor solución. Es aquí donde cobra especial importancia el uso de alguna técnica, como el algoritmo genético, que proporcione soluciones óptimas en un tiempo razonable.

Aunque se tarde algo más en ejecutar el código en este escenario que en el anterior, en menos de un minuto se obtiene la solución. Los parámetros en este escenario son tamaño de población = 1000, número de generaciones = 500 y tasa de mutación = 0,5. Se observa que al haber más ciudades sea necesario explorar un mayor número de soluciones, lo que se puede conseguir aumentando el tamaño de la población.

Nuevamente, será necesario ejecutar varias veces para analizar el mayor rango de soluciones posibles.

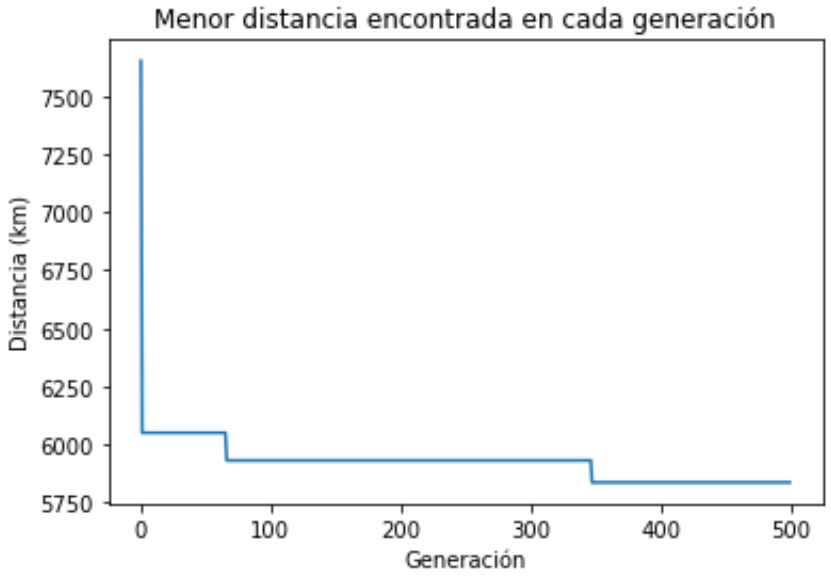
Algunas de las soluciones obtenidas son las siguientes:

Ilustración 27. Evolución de la mejor solución encontrada por el algoritmo. Ruta de 5911 km



La mejor ruta encontrada es: ['Paris', 'Nantes', 'San Sebastián', 'Bilbao', 'Vitoria', 'León', 'Oviedo', 'La Coruña', 'Santiago', 'Oporto', 'Coimbra', 'Lisboa', 'Évora', 'Mérida', 'Cáceres', 'Salamanca', 'Segovia', 'Madrid', 'Ciudad Real', 'Faro', 'Sevilla', 'Málaga', 'Córdoba', 'Murcia', 'Valencia', 'Zaragoza', 'Barcelona', 'Toulouse', 'Marsella', 'Lyon', 'Paris']
Con una distancia de: 5911 km

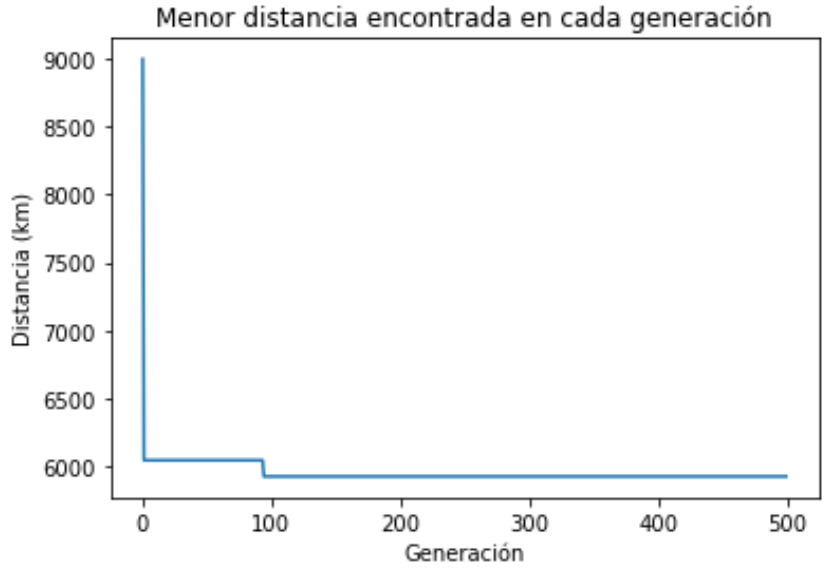
Ilustración 28. Evolución de la mejor solución encontrada por el algoritmo. Ruta de 5836 km



La mejor ruta encontrada es: ['Paris', 'Nantes', 'San Sebastián', 'Bilbao', 'Vitoria', 'León', 'Oviedo', 'La Coruña', 'Santiago', 'Oporto', 'Coimbra', 'Lisboa', 'Évora', 'Mérida', 'Cáceres', 'Salamanca', 'Segovia', 'Madrid', 'Ciudad Real', 'Córdoba', 'Faro', 'Sevilla', 'Málaga', 'Murcia', 'Valencia', 'Zaragoza', 'Barcelona', 'Toulouse', 'Marsella', 'Lyon', 'Paris']
Con una distancia de: 5836 km

Como se vió en el escenario 1 y se observa en la Ilustración 29, es posible que la solución empeore en las distintas veces que se implemente el código.

Ilustración 29. Evolución de la mejor solución encontrada por el algoritmo. Ruta de 5931 km



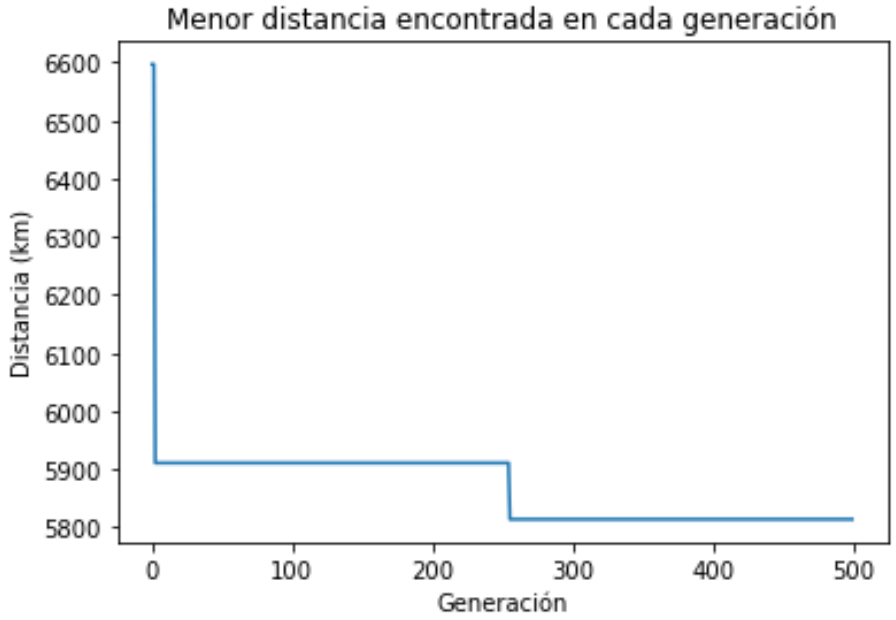
La mejor ruta encontrada es: ['Paris', 'Nantes', 'San Sebastián', 'Bilbao', 'Vitoria', 'León', 'Oviedo', 'La Coruña', 'Santiago', 'Oporto', 'Coimbra', 'Lisboa', 'Évora', 'Mérida', 'Cáceres', 'Salamanca', 'Segovia', 'Madrid', 'Ciudad Real', 'Córdoba', 'Málaga', 'Sevilla', 'Faro', 'Murcia', 'Valencia', 'Zaragoza', 'Barcelona', 'Toulouse', 'Marsella', 'Lyon', 'Paris']
Con una distancia de: 5931 km

Tras realizar 10 ejecuciones, la mejor solución encontrada es la que corresponde a una ruta de 5814 km. Como se comprueba en la Tabla 14, las soluciones obtenidas en las distintas ejecuciones no varían mucho, lo que muestra la estabilidad del código.

Tabla 14. Resultado de las 10 ejecuciones del escenario 2 del TSP

Media	Desviación	Mejor resultado	Peor resultado
5880,2	57,49	5814	5931

Ilustración 30. Evolución de la mejor solución encontrada por el algoritmo. Ruta de 5814



La mejor ruta encontrada es: ['Málaga', 'Faro', 'Sevilla', 'Córdoba', 'Ciudad Real', 'Madrid', 'Segovia', 'Salamanca', 'Cáceres', 'Mérida', 'Évora', 'Lisboa', 'Coimbra', 'Oporto', 'Santiago', 'La Coruña', 'Oviedo', 'León', 'Vitoria', 'Bilbao', 'San Sebastián', 'Nantes', 'Paris', 'Lyon', 'Marsella', 'Toulouse', 'Barcelona', 'Zaragoza', 'Valencia', 'Murcia', 'Málaga']
Con una distancia de: 5814 km

En la Ilustración 30 se observa como evolucionan las soluciones en las 500 iteraciones. La solución se estabiliza alrededor de la generación 250 y la menor distancia encontrada es 5814 km.

En conclusión, al ampliar la gira, la distancia mínima que tendrá que recorrer el cantante es 5814 km, un poco menos que el doble que antes. La ruta vista en el mapa quedaría así:

Ilustración 31. Mapa con la mejor ruta. TSP 30 ciudades

Al haber creado la población inicial según la cercanía de unas ciudades a otras, se consigue reducir bastante el tiempo de ejecución y garantizar una buena solución. Al igual que en el escenario 1, los resultados de las distintas ejecuciones no varían mucho entre sí, lo que indica que el algoritmo es estable.

En este caso, para los parámetros elegidos, el peor resultado obtenido es 5931 km y el mejor es 5814 km, diferencia relativamente pequeña para la cantidad de ciudades.

5.3 Problema de la mochila

Al igual que para el problema del viajante, se comenzará esta sección explicando el funcionamiento del algoritmo genético aplicado a este problema y luego se mostrarán los resultados.

5.3.1 Algoritmo genético aplicado al *problema de la mochila* (KP)

En los escenarios del *problema de la mochila* propuestos en este trabajo, los datos son la capacidad máxima de la mochila y un vector *pedidos* donde en cada componente se encuentra el nombre del pedido, su valor y su peso o volumen. El valor no representa lo mismo en todos los problemas,

pero lo que se busca siempre es maximizar la suma del valor de los pedidos. Es decir, el objetivo de este tipo de problemas es buscar la combinación de objetos/pedidos que meter en la mochila sin superar la capacidad máxima y maximizando el valor total.

En este trabajo se resolverán dos escenarios. En uno el valor será la urgencia o prioridad de entrega de un pedido, es decir, se querrá maximizar la urgencia, y en el otro se tiene como dato el precio al que se vende cada producto, por lo tanto, el objetivo será que el precio total de los productos seleccionados sea lo mayor posible para maximizar el beneficio. De ahora en adelante se utilizará el término *valor* para hablar de los dos escenarios, en el primero representará la urgencia y en el segundo el beneficio. Por tanto, la función *'calculo_fitness'* es sumar el valor de cada pedido a una variable que guarde el valor total, a la vez que se suma su respectivo peso a la variable *peso_total* y se va comprobando que éste no supere la capacidad. Si se supera la capacidad se utiliza un sistema de penalización en el que se le otorga a la función objetivo un valor de 0. La función *'calculo_fitness'* devuleve el valor total.

Cada *gen* es un pedido, el *individuo* o solución será un vector cuyas variables tomarán valor 1 si el pedido se incluye en la *mochila* y 0 en caso contrario, y la aptitud será la suma de los valores de las variables que están incluidas.

Parámetros

Al igual que en el problema del viajante, los parámetros se determinarán a través del mismo proceso empírico.

Finalmente, los parámetros elegidos para el primer escenario son:

Tabla 15. Pruebas combinación de parámetros KP escenario 1

		Media	Desviación	Mejor resultado	Peor resultado
TP 25	**TM 0,1**	12,8	0,42163702	13	12
	TM 0,3	12,6	0,51639778	13	12
	TM 0,5	12,2	1,22927259	13	10
TP 50	**TM 0,1**	12,7	0,48304589	13	12
	TM 0,3	12,8	0,42163702	13	12
	TM 0,5	12,9	0,31622777	13	12
TP 100	**TM 0,1**	13	0	13	13
	TM 0,3	12,8	0,42163702	13	12
	TM 0,5	12,9	0,31622777	13	12

Tamaño de población = 100

Número de generaciones = 25

Tasa de mutación = 0,1

En este caso se observa cómo, para estos parámetros, se garantiza siempre la misma solución.

Mientras que los valores de los parámetros del segundo escenario, al haber más pedidos, serán mayores:

Tabla 16. Pruebas combinación de parámetros KP escenario 2

		Media	Desviación	Mejor resultado	Peor resultado
TP 100	**TM 0,1**	9419	224,373201	9780	8980
	TM 0,3	9594	335,797558	10080	9030
	TM 0,5	9397	507,39093	10080	8500
TP 500	**TM 0,1**	9694	305,657761	10080	9150
	TM 0,3	9718	273,284548	10180	9150
	TM 0,5	9924	135,170674	10180	9780
TP 1000	**TM 0,1**	9820	182,878223	10180	9550
	TM 0,3	9802	137,663035	10030	9530
	TM 0,5	10009	126,002645	10180	9800

Tamaño de población = 1000

Número de generaciones = 500

Tasa de mutación = 0,5

Procedimiento

Para un mismo número de datos, por ejemplo, 20 ciudades y 20 productos, el KP tiende a tener menos posibles soluciones; por lo tanto, en este caso sí que se partirá de una población completamente aleatoria. La función '*crear_individuo'* generará un vector con tantos componentes como productos y asignará valores binarios aleatorios. El 0 representará que el producto no se incluye en la solución y el 1 lo contrario.

Igual que para el TSP, cuando se tiene la población inicial y la función *'calculo_fitness',* se crean unas variables para almacenar los mejores resultados. La única diferencia es que, como en este caso estamos maximizando, el valor inicial de la variable '*mejor_fitness'* será 0. Después de esto se ejecuta el bucle principal hasta igualar el número de generaciones. Los pasos dentro del bucle son los mismos que el *problema del viajante*, la diferencia radica en cómo funcionan algunas de las funciones a las que se llama. La función '*padres'* es igual, sin embargo, las funciones '*cruce'* y '*mutación'* varían. En el cruce se selecciona un punto aleatorio y se crea un vector *hijo* combinando genes del *padre1* hasta el punto de corte y los del *padre2* después. En la mutación se selecciona un índice al azar del *padre1* y se cambia el gen correspondiente a dicho índice, es decir, si antes valía 1 después valdrá 0, y viceversa.

En este caso, como solo se genera un hijo en cada generación, se eliminará únicamente el último elemento del vector '*población_ordenada'* y se sustituirá por el *hijo*. Se vuelve a calcular la aptitud de cada individuo y se ordena la población. Con base en esta nueva población se obtiene el mejor individuo de la generación y su aptitud y se comparara con el mejor individuo hasta el momento. En caso de ser mejor, se actualizará el mejor individuo y su *fitness*.

Tras la explicación del algoritmo genético aplicado a este problema, se muestran los resultados de ambos escenarios. Para mayor claridad, el código implementado en *Python* imprimirá por pantalla dos ilustraciones junto con el resultado. En la ilustración de la izquierda se puede observar la evolución de la aptitud durante toda la ejecución del algoritmo, es decir, el valor de la mejor solución en cada generación. Mientras que en la gráfica de la derecha se muestra la solución óptima, donde las barras que aparecen representan los pedidos incluidos: la barra azul indica el valor o urgencia y la naranja el peso o volumen de cada producto seleccionado.

5.3.2 Escenario 1

Al tener el escenario 1 solo 5 pedidos, haciendo algunas operaciones y cálculos a mano puede comprobarse cuál es la solución óptima: (1, 0, 1, 1, 1) con aptitud 13 y un volumen total de 29 litros. Este mismo resultado se obtiene en segundos gracias al algoritmo genético.

Se ejecuta el código 10 veces para comprobarlo y, efectivamente, con los parámetros seleccionados en la Tabla 15 (tamaño de población = 100, número de generaciones = 25 y tasa de mutación = 0,1) el resultado es siempre el esperado.

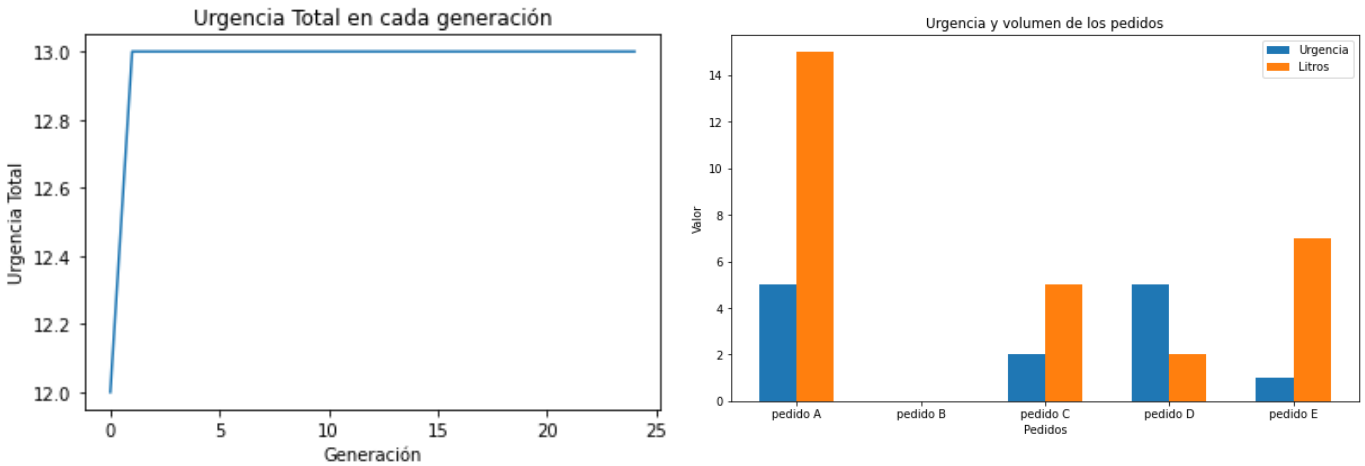
En la Tabla 17 se observan la media y desviación de las soluciones encontradas en las 10 ejecuciones realizadas y el mejor y peor resultado, que como se ha comentado, en este escenario es el mismo.

Tabla 17. Resultado de las 10 ejecuciones del escenario 1 del KP

Media	Desviación	Mejor resultado	Peor resultado
13	0	13	13

La mejor solución es la siguiente:

Ilustración 32. Productos incluidos en la mochila. Aptitud 13



Mejor solución encontrada: [1, 0, 1, 1, 1] Aptitud: 13
Pedidos incluidos en la mochila: ['pedido A', 'pedido C', 'pedido D', 'pedido E']
Volumen total: 29 litros

En la gráfica de la izquierda se observa que, en este caso, la función objetivo se estabiliza muy pronto, antes de la generación 5. Sin embargo, se ha optado por poner un número de generaciones lo suficientemente alto para garantizar que siempre se llegue a la solución óptima.

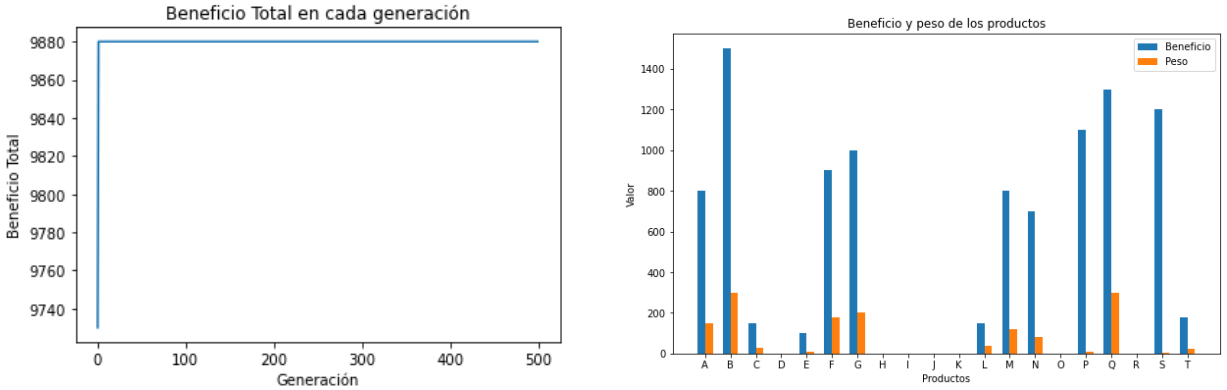
En la ilustración de la gráfica de la derecha se pueden observar los pedidos que se deben meter en la mochila, que son los mismos que valen 1 en el vector de la mejor solución. Además, sale cada pedido con su respectiva urgencia y volumen. Por tanto, los pedidos que deberá entregar antes el repartidor son el A, C, D y E. El volumen total que ocupan estos pedidos es 29 litros, lo que garantiza que no se sobrepase la capacidad de la mochila. En este caso se ve cómo los pedidos con mayor urgencia (A y D) entran en la mochila. No sorprende el hecho de que el pedido B no se incluya, ya que la urgencia de éste es baja y su volumen bastante alto, por lo que lo óptimo es llenar la mochila con el resto de los pedidos que, aunque sus urgencias tampoco sean muy altas, ocupan menos.

5.3.3 Escenario 2

En este caso, al haber más productos entre los que elegir, la solución no es directa y no se obtiene lo mismo en las distintas ejecuciones como ocurría en el escenario 1.
Además, como se vió en la explicación de algoritmo, al haber más productos aumentan los valores de los parámetros: tamaño de población = 1000, número de generaciones = 500 y tasa de mutación = 0,5.

Algunos de los resultados obtenidos con esta combinación de parámetros son los siguientes:

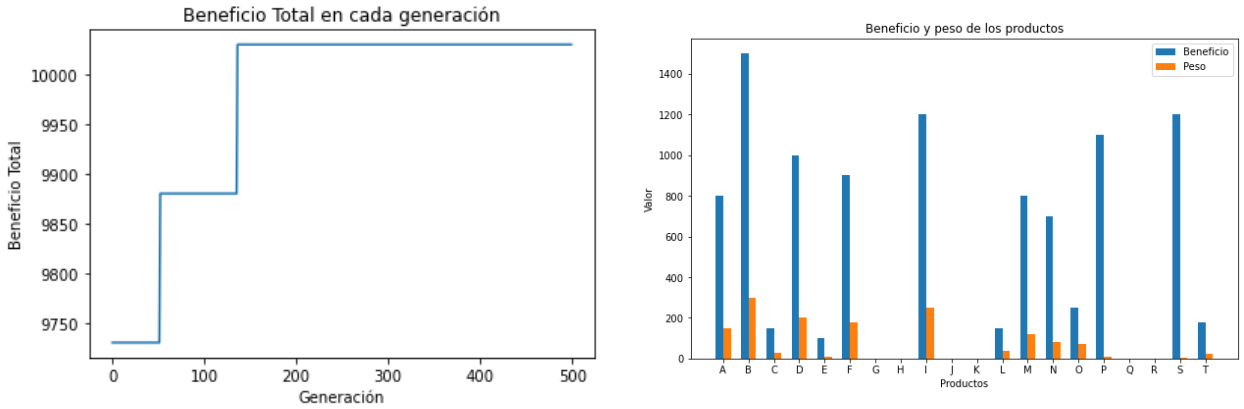
Ilustración 33. Productos incluidos en el camión. Beneficio 9880 €



Mejor solución encontrada: [1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1] Aptitud: 9880

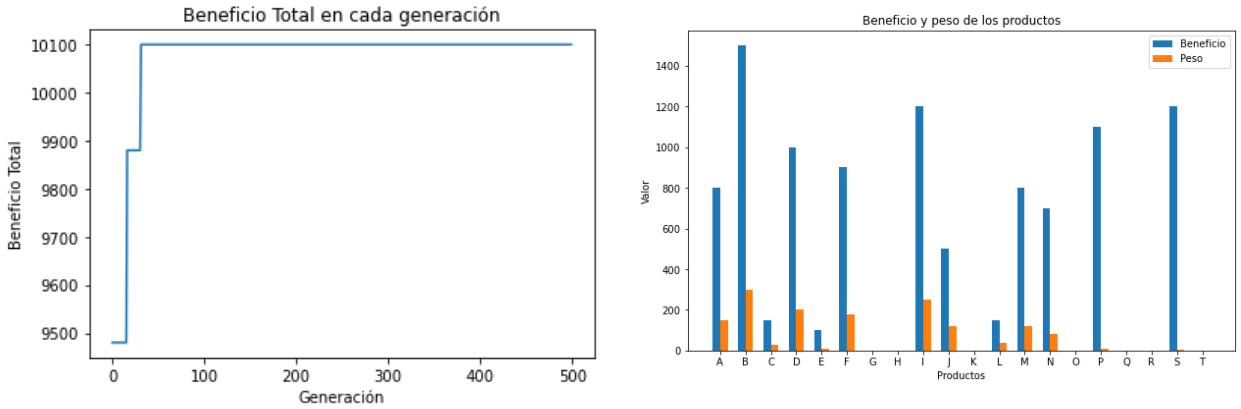
Productos incluidos en el camión: ['Lavadora', 'Frigorífico', 'Microondas', 'Aspiradora', 'Aire acondicionado', 'Televisor', 'Sillón', 'Cortadora de césped', 'Equipo de sonido', 'Thermomix', 'Máquina de hacer ejercicio', 'Portátil', 'Vajilla completa']
Peso total: 1447 kg

Ilustración 34. Productos incluidos en el camión. Beneficio 10030 €



Mejor solución encontrada: [1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1] Aptitud: 10030
Productos incluidos en el camión: ['Lavadora', 'Frigorífico', 'Microondas', 'Secadora', 'Aspiradora', 'Aire acondicionado', 'Mesa de comedor', 'Sillón', 'Cortadora de césped', 'Equipo de sonido', 'Escalera plegable', 'Thermomix', 'Portátil', 'Vajilla completa']
Peso total: 1467 kg

Ilustración 35. Productos incluidos en el camión. Beneficio 10100 €



Mejor solución encontrada: [1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0] Aptitud: 10100
Productos incluidos en el camión: ['Lavadora', 'Frigorífico', 'Microondas', 'Secadora', 'Aspiradora', 'Aire acondicionado', 'Mesa de comedor', 'Estantería', 'Sillón', 'Cortadora de césped', 'Equipo de sonido', 'Thermomix', 'Portátil']
Peso total: 1493 kg

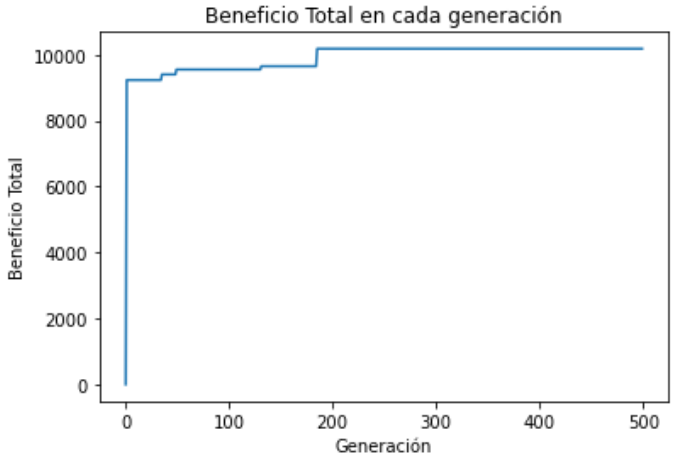
Aunque el beneficio sea mejor en unos casos que en otros, siempre debe cumplirse la restricción de la capacidad. Como se puede comprobar en las distintas soluciones, nunca se superan los 1500 kg. También se observa en las ilustraciones de las distintas ejecuciones cómo evoluciona la solución en las 500 iteraciones y cuándo se estabiliza, en la mayoría de los ejemplos ocurre antes de la generación 200.

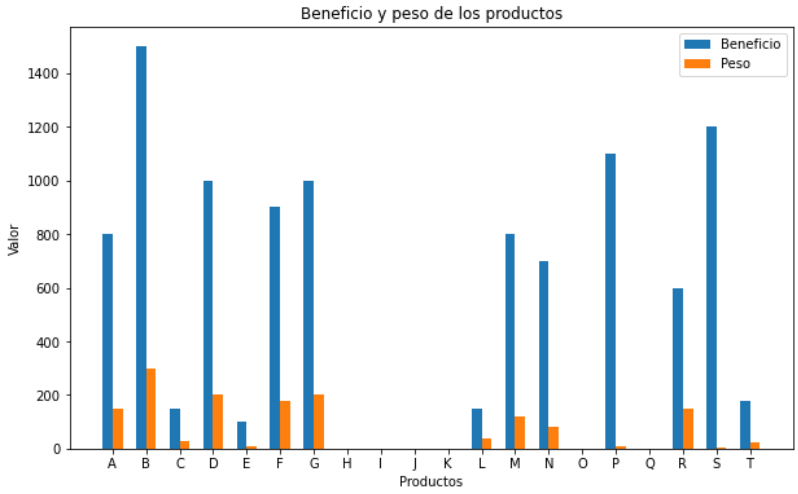
Finalmente, tras 10 ejecuciones, la mejor solución encontrada es la que corresponde a un beneficio de 10180 €. Este resultado también se puede observar en la Tabla 18, así como el peor y la media y desviación de las soluciones obtenidas en las distintas ejecuciones.

Tabla 18. Resultado de las 10 ejecuciones del escenario 2 del KP

Media	Desviación	Mejor resultado	Peor resultado
10009	126,002645	10180	9800

Ilustración 36. Productos incluidos en el camión. Beneficio 10180





Mejor solución encontrada: [1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1] Aptitud: 10180
Productos incluidos en el camión: ['Lavadora', 'Frigorífico', 'Microondas', 'Secadora', 'Aspiradora', 'Aire acondicionado', 'Televisor', 'Sillón', 'Cortadora de césped', 'Equipo de sonido', 'Thermomix', 'Bicicleta', 'Portátil', 'Vajilla completa']
Peso total: 1497 kg

Como ya se ha visto en el anterior escenario, la solución es un vector que tiene tantos elementos como productos. El elemento valdrá 1 si se incluye en el camión y 0 en caso contrario. Asociada a este vector está la *aptitud*, que es la suma del valor de cada producto seleccionado, es decir, la

suma de los beneficios de los elementos que valen 1 en el vector *solución*. El código genera también un vector con los productos incluidos en el camión, pero esta vez con sus respectivos nombres en vez de 0 o 1. Los que en el vector de la mejor solución valían 0 se omiten en este segundo vector, por lo que solo aparecerán los nombres de los productos seleccionados.

Finalmente se obtiene también el peso total de los productos elegidos, 1497 kg, que como ya se ha comentado, tiene que ser menor de 1500 kg.

En resumen, esta selección de productos permitirá que el gerente maximice sus beneficios lo antes posible sin sobrepasar la capacidad del camión. Las ganancias serán de 10180 € y el peso total de la carga será 1497 kg.

6 CONCLUSIONES Y FUTURAS LÍNEAS DE INVESTIGACIÓN

El objetivo de este trabajo era mostrar la importancia de la ingeniería industrial y de la ingeniería de organización, así como algunas de las aplicaciones de la investigación operativa.

La historia demuestra que la ingeniería y la investigación operativa están presentes en la mayoría de los ámbitos de nuestra vida y contribuyen al desarrollo de la sociedad. La ingeniería de organización y la investigación operativa se complentan proporcionando enfoques y herramientas para el diseño, análisis y optimización de situaciones y problemas complejos. Esto se ha visto mediante la resolución de distintos escenarios de los tres tipos de problemas propuestos. Gracias a los escenarios sencillos se ha conseguido llevar el entendimiento de una disciplina compleja a la población general y con los casos complejos se han mostrado versiones más realistas y completas de estos problemas y sus soluciones.
Todos se han resuelto y desarrollado con éxito y de forma eficiente, demostrando la gran utilidad de estas ramas y lo importante que son para la optimización y toma de decisiones, no solo en el ámbito empresarial, sino también en situaciones cotidianas como ir a comprar comida para un bebé o planear la ruta de un viaje.

Aquí se han estudiado tres tipos de problemas y algunas de sus aplicaciones, pero la investigación operativa es mucho más. Aborda una amplia variedad de problemas en diferentes áreas, a pequeña y gran escala. Otros ejemplos de problemas de investigación operativa son el problema de ruta de vehículos (VRP), la programación de taller, problema de localización de instalaciones o el problema de programación de trabajos. Todos estos problemas, junto con los estudiados en este trabajo, se centran en encontrar soluciones óptimas dentro de restricciones y objetivos específicos. En definitiva, la investigación operativa hace la complicada tarea de la toma de decisiones más fácil y rápida.

Como se comentó al principio, la divulgación de la ingeniería y la investigación operativa es fundamental para promover su desarrollo. El futuro de estas disciplinas siempre ha sido prometedor, pero ahora, que avanzamos hacia una sociedad cada vez más tecnológica y digitalizada con una gran cantidad de datos, la importancia que cobran es mayor. Algunas de las

áreas en las que más avance puede haber son la optimización de la cadena de suministro, optimización en tiempo real, sostenibilidad y energías renovables, e inteligencia artificial y aprendizaje automático.

Debido al enorme crecimiento del comercio *online* en los últimos años y la previsión de que éste siga aumentando, las empresas se enfrentarán a problemas cada vez más complejos de logística, gestión de pedidos, planificación de cadena de suministros, etc. Aquí entrará en juego la investigación de operaciones, que será crucial para ayudar a desarrollar modelos para optimizar la gestión de inventarios, rutas y logística, entre otras áreas.

Otro ámbito en auge es la inteligencia artificial, el aprendizaje automático y el análisis de *big data.* La combinación de estas disciplinas con la investigación de operaciones permitirá mejorar los algoritmos de optimización que ayuden a la toma de decisiones a medida que se recopila nueva información en un entorno en cambio constante. El desarrollo de estos enfoques adaptativos permitirá a las empresas adecuarse a nuevas situaciones y aprovechar oportunidades en tiempo real.

La investigación operativa también puede influir enormemente en la protección del medioambiente, algo que cada vez cobra más importancia en la sociedad para combatir el cambio climático e impulsar una economía sostenible. En este caso la IO puede contribuir a optimizar la generación y distribución de energías renovables, consiguiendo una mayor eficiencia y rentabilidad en su uso.

REFERENCIAS

[1] R. A. Española, «Ingeniería».

Fecha de acceso: marzo 2023: https://dle.rae.es/ingenier%C3%ADa

[2] A. E. Varela, «La ingeniería industrial como herramienta para la internacionalización» 2010.

Fecha de acceso: febrero 2023. https://www.laccei.org/LACCEI2010-Peru/Papers/Papers_pdf/IGE089_Espinal.pdf

[3] «Historia de la Ingeniería Industrial».

Fecha de acceso: abril 2023. https://www.ingenieriaindustrialonline.com/conceptos-generales/historia-de-la-ingenieria-industrial/

[4] «Precursores de la ingeniería industrial».

Fecha de acceso: febrero 2023. https://predictiva21.com/precursores-ingenieria-industrial-2/

[5] «Historia de la ingeniería Industrial».

Fecha de acceso: marzo 2023. https://ingenierosconingenio.com/blog/historia-de-la-ingenieria-industrial/

[6] «¿Qué es la ingeniería industrial?».

Fecha de acceso: abril 2023. https://portal.coiim.es/colegio/que-es-la-ingenieria-industrial/

[7] AingOI, «La ingeniería de OI».

Fecha de acceso: abril 2023. https://www.aingoi.com/la-ingenieria-de-oi/

[8] E. Sevilla, «Historia de la ETSi».

Fecha de acceso: marzo 2023. https://www.etsi.us.es/etsi/nuestra-escuela/historia-de-la-etsi

[9] G. Carman, «Ingeniería de organización industrial,» 2017.

Fecha de acceso: abril 2023. https://grupocarman.com/blog/2017/03/18/ingenieria-de-organizacion-industrial/

[10] U. d. Carlemany, «Los orígenes e historia de la ingeniería en organización industrial,» 2021.

Fecha de acceso: marzo 2023. https://www.universitatcarlemany.com/actualidad/blog/los-origenes-e-historia-de-la-ingenieria-en-organizacion-industrial/#:~:text=La%20aparici%C3%B3n%20de%20la%20Ingenier%C3%ADa,de%20Frederick%20Taylor%20en%201903.

[11] L. Castaño, «La historia de la Organización Industrial,» 2011.

Fecha de acceso: marzo 2023. https://tuotrodiario.es/consejos/historia-organizacion-industrial

[12] AingOI, «Historia de la IOI».

Fecha de acceso: abril 2023. https://www.aingoi.com/la-ingenieria-de-oi/historia/

[13] «Lean Manufacturing: qué es, principios, herramientas y ejemplos».

Fecha de acceso: abril 2023. https://www.cursosaula21.com/que-es-lean-manufacturing/

[14] «Ingeniería de organización industrial».

Fecha de acceso: abril 2023. https://es.wikipedia.org/wiki/Ingenier%C3%ADa_de_organizaci%C3%B3n_industrial

[15] ADINGOR, «Asociación para el Desarrollo de la Ingeniería de Organización».

Fecha de acceso. marzo 2023. https://adingor.net/quienes-somos/#

[16] «Ingeniería de organización industrial».

Fecha de acceso: abril 2023. https://es.wikipedia.org/wiki/Ingenier%C3%ADa_de_organizaci%C3%B3n_industrial#Asociaci%C3%B3n_Profesional_de_Ingenieros_de_Organizaci%C3%B3n_Industrial_de_Espa%C3%B1a_(AingOI)

[17] T. O. Society, «About OR-The Operational Research Society».

Fecha de acceso: abril 2023. https://www.theorsociety.com/about-or/

[18] «Historia de la Investigación de Operaciones,» 2015.

Fecha de acceso: https://www.scielo.br/j/bolema/a/Vbm7ZdpKvyyshsWHwHNqksS/?lang=es&format=pdf#:~:text=La%20investigación%20operacional%20tuvo%20sus,determinar%20cuáles%20tácticas%20militares%20funcionaban.

[19] «Perspectivas históricas de la Investigación Operacional,» 2018.

Fecha de acceso: abril 2023. https://www.gestiondeoperaciones.net/programacion_lineal/historia-de-la-investigacion-de-operaciones/

[20] IFORS, «IFORS History».

Fecha de acceso: marzo 2023. https://www.ifors.org/history/

[21] IFORS, «Regional Groupings».

Fecha de acceso: marzo 2023. https://www.ifors.org/regional-groupings/

[22] T. O. Society, «History of The OR Society».

Fecha de acceso: febrero 2023. https://www.theorsociety.com/about-us/history-of-the-or-society/

[23] «Institute for Operations Research and the Management Sciences».

Fecha de acceso: marzo 2023. https://en.wikipedia.org/wiki/Institute_for_Operations_Research_and_the_Management_Sciences

[24] «informs».

Fecha de acceso: abril 2023. https://www.informs.org/

[25] Seio, «Breve historia».

Fecha de acceso: marzo 2023. https://www.seio.es/breve-historia/

[26] «¿Qué son los Problemas de Dieta?» 2019.

Fecha de acceso: marzo 2023. https://knuth.uca.es/moodle/mod/page/view.php?id=4498

[27] «La Tienda Hero».

Fecha de acceso: mayo 2023. https://www.latiendahero.es

[28] G. J. Stigler, «The Cost of Subsistence,» *Journal of Farm Economics, Vol. 27, No. 2,* May 1945.

[29] «Problema del viajante».

Fecha de acceso: marzo 2023. https://es.wikipedia.org/wiki/Problema_del_viajante#Historia

[30] N. Cummings, «A brief History of the Travelling Salesman Problem».

Fecha de acceso: marzo 2023. https://www.theorsociety.com/about-or/or-methods/heuristics/a-brief-history-of-the-travelling-salesman-problem/

[31] «Problema del Viajante de Comercio» 2017.

Fecha de acceso: abril 2023. https://knuth.uca.es/moodle/mod/page/view.php?id=4131&lang=en

[32] « Tres métodos diferentes para resolver el problema del viajante» 2020.

Fecha de acceso: mayo 2023. https://baobabsoluciones.es/blog/2020/10/01/problema-del-viajante/

[33] «Problema de la mochila ».

Fecha de aaceso: abril 2023. https://www.uaeh.edu.mx/scige/boletin/tlahuelilpan/n6/e2.html

[34] « El tutorial de Python ».

Fecha de acceso: junio 2023. https://docs.python.org/es/3/tutorial/

[35] «Teoría del método Simplex».

Fecha de acceso: mayo 2023. http://www.phpsimplex.com/teoria_metodo_simplex.htm

[36] M. 365, «Definir y resolver un problema con Solver».

Fecha de acceso: abril 2023. https://support.microsoft.com/es-es/office/definir-y-resolver-un-problema-con-solver-5d1a388f-079d-43ac-a7eb-f63e45925040

[37] «Gurobi Optimization» 2011.

Fecha de acceso: abril 2023. http://investigaciondeoperacion2es.blogspot.com/2011/10/gurobi-optimization.html

[38] M. Karbstein, «Introduction to Modeling with Python,» 2022.

Fecha de acceso: mayo 2023. https://www.gurobi.com/wp-content/uploads/IntroModeling.pdf?x58432

[39] «Metaheurísticas» 2021.

Fecha de acceso: abril 2023. http://www.cs.us.es/~fsancho/?e=207

[40] «Metaheurística: la disciplina desconocida que se ha vuelto imprescindible en las grandes empresas,» 2017.

Fecha de acceso: abril 2023. https://www.expansion.com/empresas/2017/07/08/59609d2b268e3ecb7a8b45aa.html

[41] J. C. López, «Introducción a los algoritmos genéticos,» 2010.

Fecha de acceso: mayo 2023.
https://www.adictosaltrabajo.com/2010/10/07/jgap/#:~:text=Los%20principios%20b%C3%A1sicos%20de%20los,existen%20en%20una%20determinada%20poblaci%C3%B3n.

[42] J. A. Rodrigo, «Optimización con algoritmo genético y Nelder-Mead,» 2019.

Fecha de acceso: mayo 2023.
https://www.cienciadedatos.net/documentos/48_optimizacion_con_algoritmo_genetico

ANEXO

Anexo 1

En este anexo se muestran la tabla completa con los datos del escenario 2 del *problema de la dieta*. Éstos han sido obtenidos del artículo 'The Cost of Subsistence' escrito por George Stigler en 1945.

Tabla 19. Datos del escenario 2 del *problema de la dieta*

TABLE A. NUTRITIVE VALUES OF COMMON FOODS PER DOLLAR OF EXPENDITURE, AUGUST 15, 1939

Commodity	Unit	Price Aug. 15, 1939 (cents)	Edible Weight per $1.00 (grams)	Calories (1,000)	Protein (grams)	Calcium (grams)	Iron (mg.)	Vitamin A (1,000 I.U.)	Thiamine (mg.)	Ribo-flavin (mg.)	Niacin (mg.)	Ascorbic Acid (mg.)
**1. Wheat Flour (Enriched)	10 lb.	36.0	12,600	44.7	1,411	2.0	365		55.4	33.3	441	
2. Macaroni	1 lb.	14.1	3,217	11.6	418	.7	54		3.2	1.9	68	
3. Wheat Cereal (Enriched)	28 oz.	24.2	3,280	11.8	377	14.4	175		14.4	8.8	114	
4. Corn Flakes	8 oz.	7.1	3,194	11.4	252	.1	56		13.5	2.3	68	
5. Corn Meal	1 lb.	4.6	9,861	36.0	897	1.7	99	30.9	17.4	7.9	106	
6. Hominy Grits	24 oz.	8.5	8,005	28.6	680	.8	80		10.6	1.6	110	
7. Rice	1 lb.	7.5	6,048	21.2	460	.6	41		2.0	4.8	60	
8. Rolled Oats	1 lb.	7.1	6,389	25.3	907	5.1	341		37.1	8.9	64	
9. White Bread (Enriched)	1 lb.	7.9	5,742	15.6	488	2.5	115		13.8	8.5	126	
10. Whole Wheat Bread	1 lb.	9.1	4,985	12.2	484	2.7	125		13.9	6.4	160	
11. Rye Bread	1 lb.	9.2	4,930	12.4	439	1.1	82		9.9	3.0	66	
12. Pound Cake	1 lb.	24.8	1,829	8.0	130	.4	31	18.9	2.8	3.0	17	
13. Soda Crackers	1 lb.	15.1	3,004	12.5	288	.5	50					
14. Milk	1 qt.	11.0	8,867	6.1	310	10.5	18	16.8	4.0	16.0	7	177
**15. Evaporated Milk (can)	14½ oz.	6.7	6,035	8.4	422	15.1	9	26.0	3.0	23.5	11	60
16. Butter	1 lb.	30.8	1,473	10.8	9	.2	3	44.2		.2	2	
*17. Oleomargarine	1 lb.	16.1	2,817	20.6	17	.6	6	55.8	.2			
18. Eggs	1 doz.	32.6	1,857	2.9	238	1.0	52	18.6	2.8	6.5	1	
**19. Cheese (Cheddar)	1 lb.	24.2	1,874	7.4	448	16.4	19	28.1	.8	10.3	4	
20. Cream	½ pt.	14.1	1,689	3.5	49	1.7	3	16.9	.6	2.5		17
21. Peanut Butter	1 lb.	17.9	2,534	15.7	661	1.0	48		9.6	8.1	471	
22. Mayonnaise	½ pt.	16.7	1,198	8.6	18	.2	8	2.7	.4	.5		
23. Crisco	1 lb.	20.3	2,234	20.1								
24. Lard	1 lb.	9.8	4,628	41.7				.2		.5	5	
25. Sirloin Steak	1 lb.	39.6	1,145*	2.9	166	.1	34	.2	2.1	2.9	69	
26. Round Steak	1 lb.	36.4	1,246*	2.2	214	.1	32	.4	2.5	2.4	87	
27. Rib Roast	1 lb.	29.2	1,553*	3.4	213	.1	33			2.0		
28. Chuck Roast	1 lb.	22.6	2,007*	3.6	309	.2	46	.4	1.0	4.0	120	
29. Plate	1 lb.	14.6	3,107*	8.5	404	.2	62		.9			
**30. Liver (Beef)	1 lb.	26.8	1,692*	2.2	333	.2	139	169.2	6.4	50.8	316	525
31. Leg of Lamb	1 lb.	27.6	1,643*	3.1	245	.1	20		2.8	3.9	86	
32. Lamb Chops (Rib)	1 lb.	36.6	1,239*	3.3	140	.1	15		1.7	2.7	54	
33. Pork Chops	1 lb.	30.7	1,477*	3.5	196	.2	30		17.4	2.7	60	
34. Pork Loin Roast	1 lb.	24.2	1,874*	4.4	249	.3	37		18.2	3.6	79	
35. Bacon	1 lb.	25.6	1,772*	10.4	152	.2	23		1.8	1.8	71	
36. Ham—smoked	1 lb.	27.4	1,655*	6.7	212	.2	31		9.9	3.3	50	
37. Salt Pork	1 lb.	16.0	2,835*	18.8	164	.1	26		1.4	1.8		
38. Roasting Chicken	1 lb.	30.3	1,497*	1.8	184	.1	30	.1	.9	1.8	68	46
39. Veal Cutlets	1 lb.	42.3	1,072*	1.7	156	.1	24		1.4	2.4	57	
40. Salmon, Pink (can)	16 oz.	13.0	3,489	5.8	705	6.8	45	3.5	1.0	4.9	209	
41. Apples	1 lb.	4.4	9,072	5.8	27	.5	36	7.3	3.6	2.7	5	544
42. Bananas	1 lb.	6.1	4,982	4.9	60	.4	30	17.4	2.5	3.5	28	498
43. Lemons	1 doz.	26.0	2,380	1.0	21	.5	14		.5		4	952
44. Oranges	1 doz.	30.9	4,439	2.2	40	1.1	18	11.1	3.6	1.3	10	1,998
*45. Green Beans	1 lb.	7.1	5,750	2.4	138	3.7	80	69.0	4.3	5.8	37	862
**46. Cabbage	1 lb.	3.7	8,949	2.6	125	4.0	36	7.2	9.0	4.5	26	5,369
47. Carrots	1 bunch	4.7	6,080	2.7	73	2.8	43	188.5	6.1	4.3	89	608
48. Celery	1 stalk	7.3	3,915	.9	51	3.0	23	.9	1.4	1.4	9	313
49. Lettuce	1 head	8.2	2,247	.4	27	1.1	22	112.4	1.8	3.4	11	449
*50. Onions	1 lb.	3.6	11,844	5.8	166	3.8	59	16.6	4.7	5.9	21	1,184
*51. Potatoes	15 lb.	34.0	16,810	14.3	336	1.8	118	6.7	29.4	7.1	198	2,522
**52. Spinach	1 lb.	8.1	4,592	1.1	106	—	138	918.4	5.7	13.8	33	2,755
**53. Sweet Potatoes	1 lb.	5.1	7,649	9.6	138	2.7	54	290.7	8.4	5.4	83	1,912
54. Peaches (can)	No. 2½	16.8	4,894	3.7	20	.4	10	21.5	.5	1.0	31	196
55. Pears (can)	No. 2½	20.4	4,080	3.0	8	.3	8	.8	.8	.8	5	81
56. Pineapple (can)	No. 2½	21.3	3,993	2.4	16	.4	8	2.0	2.8	.8	7	399
57. Asparagus (can)	No. 2	27.7	1,945	.4	33	.3	12	16.3	1.4	2.1	17	272
58. Green Beans (can)	No. 2	10.0	5,386	1.0	54	2.0	65	53.9	1.6	4.3	32	431
59. Pork and Beans (can)	16 oz.	7.1	6,389	7.5	364	4.0	134	3.5	8.3	7.7	56	
60. Corn (can)	No. 2	10.4	5,452	5.2	136	.2	16	12.0	1.6	2.7	42	218
61. Peas (can)	No. 2	13.8	4,109	2.3	136	.6	45	84.9	4.9	2.5	37	370
62. Tomatoes (can)	No. 2	8.6	6,263	1.3	63	.7	38	53.2	3.4	2.5	36	1,253
63. Tomato Soup (can)	10½ oz.	7.6	3,917	1.6	71	.6	43	57.9	3.5	2.4	67	862
*64. Peaches, Dried	1 lb.	15.7	2,889	8.5	87	1.7	173	86.8	1.2	4.3	55	57
*65. Prunes, Dried	1 lb.	9.0	4,284	12.8	99	2.5	154	85.7	3.9	4.3	65	257
66. Raisins, Dried	15 oz.	9.4	4,524	13.5	104	2.5	136	4.5	6.3	1.4	24	136
67. Peas, Dried	1 lb.	7.9	5,742	20.0	1,367	4.2	345	2.9	28.7	18.4	162	
**68. Lima Beans, Dried	1 lb.	8.9	5,097	17.4	1,055	3.7	459	5.1	26.9	38.2	93	
**69. Navy Beans, Dried	1 lb.	5.9	7,688	26.9	1,691	11.4	792		38.4	24.6	217	
70. Coffee	1 lb.	22.4	2,025	—	—	—	—		4.0	5.1	50	
71. Tea	¼ lb.	17.4	652	—	—	—	—			2.3	42	
72. Cocoa	8 oz.	8.6	2,637	8.7	237	3.0	72		2.0	11.9	40	
73. Chocolate	8 oz.	16.2	1,400	8.0	77	1.3	39		.9	3.4	14	
74. Sugar	10 lb.	51.7	8,773	34.9	—	—	—					
75. Corn Sirup	24 oz.	13.7	4,966	14.7	—	.5	74				5	
76. Molasses	18 oz.	13.6	3,752	9.0	—	10.3	244		1.9	7.5	146	
77. Strawberry Preserves	1 lb.	20.5	2,213	6.4	11	.4	7	.2	.2	.4	3	

* Quantities including inedible portions.

Anexo 2

En el anexo 2 se presentan los datos del escenario 2 del problema del viajante. En la Tabla 20 se observa la matriz completa con las 30 ciudades y las respectivas distancias.

Tabla 20. Matriz distancias (km) 30 ciudades

Cij	Barcelona	Bilbao	Cáceres	Ciudad Real	Coimbra	Córoba	Évora	Faro	La Coruña	León	Lyon	Lisboa	Madrid	Málaga	Marsella	Mérida	Murcia	Nantes	Oporto	Oviedo	Paris	Salamanca	Santiago	San Sebastián	Segovia	Sevilla	Toulouse	Valencia	Vitoria	Zaragoza
Barcelona	0	470	752	580	892	710	914	995	895	685	532	1005	506	770	338	772	470	712	900	694	831	655	897	403	528	822	261	298	430	256
Bilbao	470	0	506	485	562	617	668	813	425	256	676	720	321	738	672	560	606	455	522	236	743	334	457	77	275	702	360	470	50	248
Cáceres	752	506	0	222	185	225	166	304	454	353	1150	261	245	350	1070	61	454	945	264	435	1253	167	421	564	253	240	795	514	488	522
Ciudad Real	580	485	222	0	400	143	350	413	610	445	1041	455	163	255	914	208	263	936	470	513	1206	270	581	509	218	253	681	307	443	393
Coimbra	892	562	185	400	0	400	185	355	352	352	1233	176	396	513	1188	222	640	952	110	408	1279	241	300	630	365	373	892	685	555	645
Córdoba	710	617	225	143	400	0	284	295	681	530	1180	392	300	134	1050	178	289	1071	490	615	1347	351	640	648	344	120	823	422	580	534
Évora	914	668	166	350	185	284	0	172	535	490	1316	108	419	367	1234	143	565	1093	295	561	1408	329	483	726	420	215	963	660	650	688
Faro	995	813	304	413	355	295	172	0	706	652	1436	220	530	310	1328	253	579	1250	464	727	1556	480	654	863	545	177	1079	714	788	794
La Coruña	895	425	454	610	352	681	535	706	0	220	1082	517	506	813	1113	523	856	687	235	207	1029	347	54	520	445	694	801	793	469	645
León	685	256	353	445	352	530	490	652	220	0	900	510	310	662	896	414	659	605	274	88	928	187	244	304	221	584	611	586	239	434
Lyon	532	676	1150	1041	1233	1180	1316	1436	1082	900	0	1388	913	1267	278	1188	995	517	1199	887	392	1001	1109	605	899	1293	360	816	678	649
Lisboa	1005	720	261	455	176	392	108	220	517	510	1388	0	499	471	1317	244	702	1126	279	585	1453	373	463	788	495	308	1036	760	710	770
Madrid	506	321	245	163	396	300	419	530	506	310	913	499	0	416	816	283	352	774	423	373	1053	175	488	353	68	392	561	305	282	274
Málaga	770	738	350	255	513	134	367	310	813	662	1267	471	416	0	1108	296	323	737	613	748	1457	484	770	762	470	157	913	472	700	627
Marsella	338	672	1070	914	1188	1050	1234	1328	1113	896	278	1317	816	1108	0	1095	805	1158	1175	907	660	945	1132	595	824	1163	319	642	656	545
Mérida	772	560	61	208	222	178	143	253	523	414	1188	244	283	296	1095	0	434	1001	316	495	1304	235	478	610	294	174	832	518	535	552
Murcia	470	606	454	263	640	289	565	579	856	659	995	702	352	323	805	434	0	1026	736	717	1240	509	806	597	416	436	673	180	549	405
Nantes	712	455	945	936	952	1071	1093	1250	687	605	517	1126	774	737	1158	1001	1026	0	878	545	342	769	731	436	728	1153	466	869	494	623
Oporto	900	522	264	470	110	490	295	464	235	274	1199	279	423	613	1175	316	736	878	0	335	1213	253	192	597	380	478	869	726	525	647
Oviedo	694	236	435	513	408	615	561	727	207	88	887	585	373	748	907	495	717	545	335	0	878	266	225	312	305	664	589	630	264	449
Paris	831	743	1253	1206	1279	1347	1408	1556	1029	928	392	1453	1053	1457	660	1304	1240	342	1213	878	0	1079	1071	700	1015	1441	588	1066	772	840
Salamanca	655	334	167	270	241	351	329	480	347	187	1001	373	175	484	945	235	509	769	253	266	1079	0	319	400	131	399	653	478	324	405
Santiago	897	457	421	581	300	640	483	654	54	244	1109	463	488	770	1132	478	806	731	192	225	1071	319	0	535	425	649	812	782	478	644
San Sebastián	403	77	564	509	630	648	726	863	520	304	605	788	353	762	595	610	597	436	597	312	700	400	535	0	317	741	278	449	77	206
Segovia	528	275	253	218	365	344	420	545	445	221	899	495	68	470	824	294	416	728	380	305	1015	131	425	317	0	427	544	358	244	280
Sevilla	822	702	240	253	373	120	215	177	694	584	1293	308	392	157	1163	174	436	1153	478	664	1441	399	649	741	427	0	1260	545	670	647
Toulouse	261	360	795	681	892	823	963	1079	801	611	360	1036	561	913	319	832	673	466	869	589	588	653	812	278	544	1260	0	485	344	288
Valencia	298	470	514	307	685	422	660	714	793	586	816	760	305	472	642	518	180	869	726	630	1066	478	782	449	358	545	485	0	424	243
Vitoria	430	50	488	443	555	580	650	788	469	239	678	710	282	700	656	535	549	494	525	264	772	324	478	77	244	670	344	424	0	294
Zaragoza	256	248	522	393	645	534	688	794	645	434	649	770	274	627	545	552	405	623	647	449	840	405	644	206	280	647	288	243	294	0

Para mayor claridad en las siguientes tablas se muestra esta matriz con las 30 ciudades dividida por grupos:

Tabla 21. Matriz distancias (km) Barcelona - Córdoba

Cij	Barcelona	Bilbao	Cáceres	Ciudad Real	Coimbra	Córdoba
Barcelona	0	470	752	580	892	710
Bilbao	470	0	506	485	562	617
Cáceres	752	506	0	222	185	225
Ciudad Real	580	485	222	0	400	143
Coimbra	892	562	185	400	0	400
Córdoba	710	617	225	143	400	0
Évora	914	668	166	350	185	284
Faro	995	813	304	413	355	295
La Coruña	895	425	454	610	352	681
León	685	256	353	445	352	530
Lyon	532	676	1150	1041	1233	1180
Lisboa	1005	720	261	455	176	392
Madrid	506	321	245	163	396	300
Málaga	770	738	350	255	513	134
Marsella	338	672	1070	914	1188	1050
Mérida	772	560	61	208	222	178
Murcia	470	606	454	263	640	289
Nantes	712	455	945	936	952	1071
Oporto	900	522	264	470	110	490
Oviedo	694	236	435	513	408	615
Paris	831	743	1253	1206	1279	1347
Salamanca	655	334	167	270	241	351
Santiago	897	457	421	581	300	640
San Sebastián	403	77	564	509	630	648
Segovia	528	275	253	218	365	344
Sevilla	822	702	240	253	373	120
Toulouse	261	360	795	681	892	823
Valencia	298	470	514	307	685	422
Vitoria	430	50	488	443	555	580
Zaragoza	256	248	522	393	645	534

Tabla 22. Matriz distancias (km) Évora - Lisboa

Cij	Évora	Faro	La Coruña	León	Lyon	Lisboa
Barcelona	914	995	895	685	532	1005
Bilbao	668	813	425	256	676	720
Cáceres	166	304	454	353	1150	261
Ciudad Real	350	413	610	445	1041	455
Coimbra	185	355	352	352	1233	176
Córdoba	284	295	681	530	1180	392
Évora	0	172	535	490	1316	108
Faro	172	0	706	652	1436	220
La Coruña	535	706	0	220	1082	517
León	490	652	220	0	900	510
Lyon	1316	1436	1082	900	0	1388
Lisboa	108	220	517	510	1388	0
Madrid	419	530	506	310	913	499
Málaga	367	310	813	662	1267	471
Marsella	1234	1328	1113	896	278	1317
Mérida	143	253	523	414	1188	244
Murcia	565	579	856	659	995	702
Nantes	1093	1250	687	605	517	1126
Oporto	295	464	235	274	1199	279
Oviedo	561	727	207	88	887	585
Paris	1408	1556	1029	928	392	1453
Salamanca	329	480	347	187	1001	373
Santiago	483	654	54	244	1109	463
San Sebastián	726	863	520	304	605	788
Segovia	420	545	445	221	899	495
Sevilla	215	177	694	584	1293	308
Toulouse	963	1079	801	611	360	1036
Valencia	660	714	793	586	816	760
Vitoria	650	788	469	239	678	710
Zaragoza	688	794	645	434	649	770

Tabla 23. Matriz distancias (km) Madrid - Nantes

Cij	Madrid	Málaga	Marsella	Mérida	Murcia	Nantes
Barcelona	506	770	338	772	470	712
Bilbao	321	738	672	560	606	455
Cáceres	245	350	1070	61	454	945
Ciudad Real	163	255	914	208	263	936
Coimbra	396	513	1188	222	640	952
Córdoba	300	134	1050	178	289	1071
Évora	419	367	1234	143	565	1093
Faro	530	310	1328	253	579	1250
La Coruña	506	813	1113	523	856	687
León	310	662	896	414	659	605
Lyon	913	1267	278	1188	995	517
Lisboa	499	471	1317	244	702	1126
Madrid	0	416	816	283	352	774
Málaga	416	0	1108	296	323	737
Marsella	816	1108	0	1095	805	1158
Mérida	283	296	1095	0	434	1001
Murcia	352	323	805	434	0	1026
Nantes	774	737	1158	1001	1026	0
Oporto	423	613	1175	316	736	878
Oviedo	373	748	907	495	717	545
Paris	1053	1457	660	1304	1240	342
Salamanca	175	484	945	235	509	769
Santiago	488	770	1132	478	806	731
San Sebastián	353	762	595	610	597	436
Segovia	68	470	824	294	416	728
Sevilla	392	157	1163	174	436	1153
Toulouse	561	913	319	832	673	466
Valencia	305	472	642	518	180	869
Vitoria	282	700	656	535	549	494
Zaragoza	274	627	545	552	405	623

Tabla 24. Matriz distancias (km) Oporto - San Sebastián

Cij	Oporto	Oviedo	Paris	Salamanca	Santiago	San Sebastián
Barcelona	900	694	831	655	897	403
Bilbao	522	236	743	334	457	77
Cáceres	264	435	1253	167	421	564
Ciudad Real	470	513	1206	270	581	509
Coimbra	110	408	1279	241	300	630
Córdoba	490	615	1347	351	640	648
Évora	295	561	1408	329	483	726
Faro	464	727	1556	480	654	863
La Coruña	235	207	1029	347	54	520
León	274	88	928	187	244	304
Lyon	1199	887	392	1001	1109	605
Lisboa	279	585	1453	373	463	788
Madrid	423	373	1053	175	488	353
Málaga	613	748	1457	484	770	762
Marsella	1175	907	660	945	1132	595
Mérida	316	495	1304	235	478	610
Murcia	736	717	1240	509	806	597
Nantes	878	545	342	769	731	436
Oporto	0	335	1213	253	192	597
Oviedo	335	0	878	266	225	312
Paris	1213	878	0	1079	1071	700
Salamanca	253	266	1079	0	319	400
Santiago	192	225	1071	319	0	535
San Sebastián	597	312	700	400	535	0
Segovia	380	305	1015	131	425	317
Sevilla	478	664	1441	399	649	741
Toulouse	869	589	588	653	812	278
Valencia	726	630	1066	478	782	449
Vitoria	525	264	772	324	478	77
Zaragoza	647	449	840	405	644	206

Tabla 25. Matriz distancias (km) Segovia - Zaragoza

Cij	Segovia	Sevilla	Toulouse	Valencia	Vitoria	Zaragoza
Barcelona	528	822	261	298	430	256
Bilbao	275	702	360	470	50	248
Cáceres	253	240	795	514	488	522
Ciudad Real	218	253	681	307	443	393
Coimbra	365	373	892	685	555	645
Córdoba	344	120	823	422	580	534
Évora	420	215	963	660	650	688
Faro	545	177	1079	714	788	794
La Coruña	445	694	801	793	469	645
León	221	584	611	586	239	434
Lyon	899	1293	360	816	678	649
Lisboa	495	308	1036	760	710	770
Madrid	68	392	561	305	282	274
Málaga	470	157	913	472	700	627
Marsella	824	1163	319	642	656	545
Mérida	294	174	832	518	535	552
Murcia	416	436	673	180	549	405
Nantes	728	1153	466	869	494	623
Oporto	380	478	869	726	525	647
Oviedo	305	664	589	630	264	449
Paris	1015	1441	588	1066	772	840
Salamanca	131	399	653	478	324	405
Santiago	425	649	812	782	478	644
San Sebastián	317	741	278	449	77	206
Segovia	0	427	544	358	244	280
Sevilla	427	0	1260	545	670	647
Toulouse	544	1260	0	485	344	288
Valencia	358	545	485	0	424	243
Vitoria	244	670	344	424	0	294
Zaragoza	280	647	288	243	294	0

En los siguientes anexos se muestran los códigos utilizados en este trabajo para resolver los distintos casos prácticos:

Anexo 3

Gurobi

```
import gurobipy as gp

#Definir los parámetros
alimentos = ["harina de trigo", "macarrones", "cereal de trigo", "copos de
maíz", "harina de maíz", "sémola", "arroz", "avena en copos", "pan blanco",
"pan de trigo integral", "pan de centeno", "pastel", "galletas de soda",
"leche", "leche evaporada", "mantequilla","oleomargarín", "huevo", "queso",
"crema", "mantequilla de maní", "mayonesa", "crisco", "manteca de cerdo",
"solomillo", "filete redondo", "asado de costilla", "chuck toast", "plato",
"hígado", "pata de cordero", "chuletas de cordero", "chuletas de cerdo",
"lomo de cerdo", "panceta", "ahumado", "cerdo salado", "pollo asado",
chuletas de ternera", "salmón", "manzanas", "bananas", "limones", "narnajas",
"frijoles verdes", "col", "zanahorias", "apio", "lechuga", "cebollas",
"patatas", "espinacas", "batata", "melocotón", "peras", "piña", "espárragos",
"judias verdes", "carne de cerdo y frijoles", "maíz", "guisantes", "tomates",
"sopa de tomate", "melocotones secos", "ciruelas secas", "pasas secas",
"guisantes secos", "habas secas", "alubias blancas secas", "café", "té",
"cacao", "chocolate", "azúcar", "jarabe de maíz", "melaza", "conservas de
fresa"]

#Mínimo de nutrientes
min_calorias = 3
min_proteina = 70
min_calcio = 0.8
min_hierro = 12
min_vitaminaA = 5
min_vitaminaB1 = 1.8
min_vitaminaB2 = 2.7
min_vitaminaB3 = 18
min_vitaminaC = 75

#Precios y nutrientes por alimentos
precio = [3.6, 14.1, 0.864285714, 0.8875, 4.6, 0.354166667, 7.5, 7.1, 7.9,
9.1, 9.2, 24.8, 15.1, 11, 6.7, 30.8, 16.1, 32.6, 24.2, 28.2, 17.9, 33.4,
20.3, 9.8, 39.6, 36.4, 29.2, 22.6, 14.6, 26.8, 27.6, 36.6, 30.7, 24.2, 25.6,
27.4, 16, 30.3, 42.3, 13, 4.4, 6.1, 26, 30.9, 7.1, 3.7, 4.7, 7.3, 8.2, 3.6,
2.266666667, 8.1, 5.1, 16.8, 20.4, 21.3, 27.7, 10, 7.1, 10.4, 13.8, 8.6, 7.6,
15.7, 9, 0.626666667, 7.9, 8.9, 5.9, 22.4, 34.8, 1.075, 2.025, 5.17,
0.570833333, 0.755555556, 20.5]
```

```
calorias = [1.6092, 1.635574759, 0.101989055, 0.101184815, 1.655941588,
0.101286159, 1.589964947, 1.796185162, 1.184926855, 1.11008987, 1.140875456,
1.983991252, 1.887441744, 0.0062541, 0.5628, 3.32578411, 3.3317, 0.9448,
1.79116, 0.987024867, 2.81, 2.872411486, 4.081091764, 4.087014477,
1.148830568, 0.80088122, 0.993049581, 0.81361435, 1.240912456, 0.5896,
0.855830189, 1.208108959, 1.07485782, 1.064992529, 2.662153499, 1.836285801,
3.007933686, 0.545398798, 0.7193125, 0.754036572, 0.2552, 0.298898,
0.2600104, 0.6798, 0.1704, 0.0962, 0.126903109, 0.0657, 0.0328, 0.2088,
0.32413, 0.0891, 0.4896, 0.6216, 0.612, 0.5112, 0.1108, 0.1, 0.532468305,
0.5408, 0.3174, 0.1118, 0.1216, 1.334555902, 1.152, 0.084597314, 1.579909439,
1.548460035, 1.5871, 0, 0, 0.093530774, 0.161997143, 1.804440989,
0.083918174, 0.068002532, 1.311788884]

proteina = [50.796, 58.93709046, 3.258463872, 2.236716969, 41.26054457,
2.408202374, 34.49923942, 64.39288308, 38.54962034, 44.03963089, 40.3906714,
32.23985785, 43.48665779, 0.31783, 28.274, 2.77148, 2.73738, 77.539,
108.437994, 13.81834813, 118.32265, 6.01202404, 0, 0, 65.76064629,
77.90390048, 62.21163554, 69.83523169, 58.97983907, 89.244, 67.63819233,
51.25310734, 60.19203791, 60.26889541, 38.90839729, 58.1033716, 26.23942152,
55.75187709, 66.0075, 91.6544454, 1.188, 3.65997591, 5.460218409, 12.36,
9.798, 4.625, 3.431084046, 3.723, 2.214, 5.976, 7.616, 8.586, 7.038, 3.36,
1.632, 3.408, 9.141, 5.4, 25.84246173, 14.144, 18.768, 5.418, 5.396,
13.65957217, 8.91, 0.651712644, 107.9868102, 93.88651364, 99.769, 0, 0,
2.547907281, 1.5592225, 0, 0, 0, 2.254637144]

calcio = [0.072, 0.098698477, 0.12446122, 0.000887586, 0.078197242,
0.002833179, 0.044999008, 0.362076851, 0.197487809, 0.245675627, 0.101206694,
0.099199563, 0.07549767, 0.010765, 1.0117, 0.06158859, 0.0966, 0.3257942,
3.9696, 0.479412078, 0.179, 0.066800267, 0, 0, 0.039614847, 0.036403692,
0.029207341, 0.045200797, 0.02919794, 0.0536, 0.027607425, 0.036609362,
0.061420447, 0.072613127, 0.05119526, 0.054814502, 0.015999647, 0.030299933,
0.0423125, 0.884042878, 0.022, 0.02429984, 0.1300052, 0.3399, 0.2627, 0.148,
0.131603224, 0.219, 0.0902, 0.1368, 0.0408, 0, 0.1377, 0.0672, 0.0612,
0.0852, 0.0831, 0.2, 0.283983096, 0.0208, 0.0828, 0.0602, 0.0456, 0.26691118,
0.225, 0.015666169, 0.331780982, 0.329270237, 0.6726, 0, 0, 0.032251991,
0.026324536, 0, 0.00285436, 0.07782512, 0.081986805]

hierro = [13.14, 7.613882499, 1.512549543, 0.497048215, 4.553839367,
0.283317926, 3.074932209, 24.20945218, 9.08443922, 11.37387161, 7.544498986,
7.687966102, 7.549766977, 0.01845, 0.603, 0.9238, 0.966, 16.9413, 4.5989,
0.846021314, 8.592265, 2.672010684, 0, 0, 13.46904803, 11.64918138,
9.638422408, 10.39618336, 9.051361442, 37.252, 5.521, 5.49140, 9.213067028,
8.955618997, 5.887454853, 8.496247734, 4.159908289, 9.08997996, 10.155,
5.850283749, 1.584, 1.82998796, 3.640145606, 5.562, 5.68, 1.332, 2.021049507,
1.679, 1.804, 2.124, 2.675, 11.178, 2.754, 1.68, 1.632, 1.704, 3.324, 6.5,
9.513433714, 1.664, 6.21, 3.268, 3.268, 27.16213776, 13.86, 0.852239611,
27.25343783, 40.84730783, 46.728, 0, 0, 0.774047782, 0.789736071, 0,
0.422445228, 1.8436242, 1.434769092]
```

```
vitaminaA = [0, 0, 0, 0, 1.421349863, 0, 0, 0, 0, 0, 0, 4.687179333, 0,
0.0172, 1.742, 13.61, 8.985, 6.059773, 6.8, 4.765920071, 0, 0.901803606, 0,
0.019601988, 0.079229694, 0.145614767, 0, 0.090401594, 0, 45.3456, 0, 0, 0,
0, 0, 0, 0, 0.030299933, 0, 0.455022069, 0.3212, 1.061, 0, 3.4299, 4.899,
0.2664, 8.859717023, 0.0657, 9.2168, 0.5976, 0.15186668, 74.3904, 14.8257,
3.612, 0.1632, 0.426, 4.5151, 5.39, 0.248485209, 1.248, 4.8162, 4.5752,
4.4004, 13.62817085, 7.713, 0.028199105, 0.229086869, 0.45385897, 0, 0, 0, 0,
0, 0, 0, 0, 0.040993403]

vitaminaB1 = [1.9944, 0.451193037, 0.12446122, 0.119824123, 0.800371768,
0.037539625, 0.149996693, 2.633931601, 1.090132706, 1.264774524, 0.910860243,
0.694396938, 0, 0.0041, 0.201, 0, 0.032, 0.912, 0.1936, 0.169204263, 1.71845,
0.133600534, 0, 0, 0.83191179, 0.910092295, 0, 0.226003986, 0.131390731,
1.7152, 0.7728, 0.622359161, 5.343578876, 4.405196371, 0.460757336,
2.713317825, 0.223995062, 0.272699399, 0.592375, 0.130006306, 0.1584,
0.152499, 0.1300052, 1.1124, 0.3053, 0.333, 0.286707023, 0.1022, 0.1476,
0.1692, 0.6664, 0.4617, 0.4284, 0.084, 0.1632, 0.5964, 0.3878, 0.16,
0.589264924, 0.1664, 0.6762, 0.2924, 0.266, 0.188407892, 0.351, 0.039478747,
2.267170045, 2.393883618, 2.2656, 0.895984198, 0, 0.021501327, 0.018224679,
0, 0, 0.01435609, 0.040993403]

vitaminaB2 = [1.1988, 0.267895866, 0.076059634, 0.02041448, 0.363387182,
0.005666359, 0.359992063, 0.631859602, 0.671458551   , 0.582342227,
0.276018256, 0.74399672, 0, 0.00164, 1.5745, 0.06158859, 0, 2.11766, 2.4931,
0.705017762, 1.44994, 0.167000668, 0, 0.04900497, 1.148830568, 0.873688604,
0.584146813, 0.904015944, 0, 13.6144, 1.076689592, 0.988452785, 0.829176032,
0.871357524, 0.460757336, 0.904439275, 0.287993651, 0.545398798, 1.0155,
0.637030897, 0.1188, 0.21349859, 0, 0.4017, 0.4118, 0.1665, 0.202104951,
0.1022, 0.2788, 0.2124, 0.16093, 1.1178, 0.2754, 0.168, 0.1632, 0.1704,
0.5817, 0.43, 0.54666746, 0.2808, 0.345, 0.215, 0.1824, 0.67512828, 0.387,
0.008773055, 1.453516684, 3.399492721, 1.4514, 1.142379852, 1.600094479,
0.127932897, 0.068848786, 0, 0, 0.056668777, 0.081986805]

vitaminaB3 = [15.876, 9.587852036, 0.985317988, 0.603558547, 4.875828009,
0.389562149, 4.499900794, 4.543709501, 9.95338558, 14.55855567, 6.072401623,
4.215981411, 0, 0.0071, 0.737, 0.61588595, 0, 0.325794, 0.9682, 0, 84.3116,
0, 0, 0.490049697, 27.33424454, 31.67121188, 0, 27.12047833, 0, 84.688,
23.74238588, 19.76905569, 18.42613406, 19.12145678, 18.17431716, 13.70362538,
0, 20.60395458, 24.118125, 27.17131786, 0.22, 1.708, 1.040041602, 3.09,
2.627, 0.962, 4.183102467, 0.657, 0.902, 0.756, 4.488, 2.673, 4.233, 5.208,
1.02, 1.491, 4.709, 3.2, 3.975763343, 4.368, 5.106, 3.096, 5.092,
8.635361717, 5.85, 0.150395225, 12.79726646, 8.276251913, 12.803,
11.19980247, 29.21911656, 0.430026545, 0.283495, 0, 0.028543596, 1.103152186,
0.614901039]

vitaminaC = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.1815, 4.02, 0, 0, 0, 0,
4.794120782, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 140.7, 0, 0, 0, 0, 0, 0, 0,
```

```
13.93796927, 0, 0, 23.936, 30.3778, 247.5299012, 617.382, 61.202, 198.653,
28.5767, 22.849, 36.818, 42.624, 57.1653, 223.155, 97.512, 32.928, 16.524,
84.987, 75.344, 43.1, 0, 22.672, 51.06, 107.758, 65.512, 8.94937487, 23.13,
0.852239611, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


#Creación del modelo
modelo = gp.Model("Problema de la dieta")

#Definición de las variables de decisión
x = modelo.addVars(len(alimentos), name="x")

#Asignción de los nombres de alimentos a los índices correspondientes de las
variables de decisión
x = {alimentos[i]: x[i] for i in range(len(alimentos))}

#Función objetivo
modelo.setObjective(gp.quicksum(precio[i]*x[alimentos[i]] for i in
range(len(alimentos))), gp.GRB.MINIMIZE)

#Restricciones
modelo.addConstr(gp.quicksum(calorias[i]*x[alimentos[i]] for i in
range(len(alimentos))) >= min_calorias, "calorias_min")
modelo.addConstr(gp.quicksum(proteina[i]*x[alimentos[i]] for i in
range(len(alimentos))) >= min_proteina, "proteinas_min")
modelo.addConstr(gp.quicksum(calcio[i]*x[alimentos[i]] for i in
range(len(alimentos))) >= min_calcio, "calcio_min")
modelo.addConstr(gp.quicksum(hierro[i]*x[alimentos[i]] for i in
range(len(alimentos))) >= min_hierro, "hierro_min")
modelo.addConstr(gp.quicksum(vitaminaA[i]*x[alimentos[i]] for i in
range(len(alimentos))) >= min_vitaminaA, "vitaminaA_min")
modelo.addConstr(gp.quicksum(vitaminaB1[i]*x[alimentos[i]] for i in
range(len(alimentos))) >= min_vitaminaB1, "vitaminaB1_min")
modelo.addConstr(gp.quicksum(vitaminaB2[i]*x[alimentos[i]] for i in
range(len(alimentos))) >= min_vitaminaB2, "vitaminaB2_min")
modelo.addConstr(gp.quicksum(vitaminaB3[i]*x[alimentos[i]] for i in
range(len(alimentos))) >= min_vitaminaB3, "vitaminaB3_min")
modelo.addConstr(gp.quicksum(vitaminaC[i]*x[alimentos[i]] for i in
range(len(alimentos))) >= min_vitaminaC, "vitaminaC_min")

#Resolver el modelo
modelo.optimize()

#Imprimir la solución
if modelo.status == gp.GRB.OPTIMAL:
    print("La dieta óptima es:")
    for i in range(len(alimentos)):
        if x[alimentos[i]].x >0:
```

```
            print(f"{alimentos[i]}: {round(x[alimentos[i]].x,5)} diarios y
{365*round(x[alimentos[i]].x,5)} anualmente ")
    print(f"Coste total anual: {365*modelo.objVal/100}$")
else:
    print("No se encontró una solución óptima")
```

Anexo 4

Algoritmo genético

El código es el mismo para ambos escenarios, lo único que cambian son los datos. Por ello, se presentará el código del escenario 1.

- **TSP**
  - **Escenario 1**

```
import random
import matplotlib.pyplot as plt


#Crear individuo aleatorio
def crear_individuo():
    return random.sample(ciudades, len(ciudades))


#Cálculo de la aptitud
def calculo_fitness(route):
    distancia = 0
    distancia_total=
distancias[ciudades.index(route[0])][ciudades.index(route[-1])]
    for i in range(len(route)-1):
        distancia+=
distancias[ciudades.index(route[i])][ciudades.index(route[i+1])]
    distancia_total += distancia
    return distancia_total


#Selección de los padres
def padres(poblacion_ordenada):
    padre1 = poblacion_ordenada[0]
    padre2 = poblacion_ordenada[1]
    return padre1, padre2


#Cruce de los padres
def cruce(padre1, padre2):
    punto_cruce = random.randint(1, len(padre1)-1)
    hijo1 = padre1[:punto_cruce] + [x for x in padre2 if x not in
padre1[:punto_cruce]]
    hijo2 = padre2[:punto_cruce] + [x for x in padre1 if x not in
```

```
padre2[:punto_cruce]]
    return hijo1, hijo2



#Mutación
def mutacion(hijo):
    index1 = random.randint(0, len(hijo)-1)
    index2 = random.randint(0, len(hijo)-1)
    hijo[index1], hijo[index2] = hijo[index2], hijo[index1]
    return hijo


#Datos del problema
ciudades = ["Barcelona", "Bilbao", "La Coruña", "Lisboa", "Madrid",
"Murcia", "Salamanca", "Sevilla", "Valencia", "Zaragoza"]
distancias = [
    [0,470,895,1005,506,470,655,822,298,256],
    [470,0,425,720,321,606,334,702,470,248],
    [895,425,0,517,506,856,347,694,793,645],
    [1005,720,517,0,499,702,373,308,760,770],
    [506,321,506,499,0,352,175,392,305,274],
    [470,606,856,702,352,0,509,436,180,405],
    [655,334,347,373,175,509,0,399,478,405],
    [822,702,694,308,392,436,399,0,545,647],
    [298,470,793,760,305,180,478,545,0,243],
    [256, 248,645,770,274,405,405,647,243,0]
]


#Parámetros del algoritmo
tam_poblacion = 300
num_generaciones = 500
tasa_mutacion = 0.5


mejores_distancias = []
mejor_individuo = None
mejor_fitness = float('inf')
```

```
#Se crea población inicial aleatoria
poblacion = [crear_individuo() for i in range(tam_poblacion)]

#Bucle principal del algoritmo
for generacion in range(num_generaciones):
    fitness = [calculo_fitness(individuo) for individuo in poblacion]
    poblacion_ordenada = [x for _, x in sorted(zip(fitness, poblacion), reverse=False)]

    padre1, padre2 = padres(poblacion_ordenada)

    if random.random() > tasa_mutacion:
        hijo1, hijo2 = cruce(padre1, padre2)
    else:
        hijo1 = mutacion(padre1)
        hijo2 = mutacion(padre2)

    #Se actualiza la población
    poblacion_ordenada.pop()
    poblacion_ordenada.pop()
    poblacion_ordenada.append(hijo1)
    poblacion_ordenada.append(hijo2)

    poblacion = poblacion_ordenada

    #Se actualiza el fitness y ordenar nuevamente la población
    fitness = [calculo_fitness(individuo) for individuo in poblacion]
    poblacion_ordenada = [x for _, x in sorted(zip(fitness, poblacion), reverse=False)]
    poblacion = poblacion_ordenada.copy()

    #Obtener el mejor individuo de esta generación
    mejor_actual = poblacion[0]
    fitness_mejor_actual = fitness[0]
```

```
    #Comprobar si el mejor individuo de esta generación es mejor que el mejor encontrado hasta ahora
    if fitness_mejor_actual < mejor_fitness:
        mejor_individuo = mejor_actual.copy()
        mejor_fitness = fitness_mejor_actual

    #Almacenar la distancia del mejor individuo de esta generación
    mejores_distancias.append(mejor_fitness)

#Gráfica de la evolución de la distancia total
plt.plot(mejores_distancias)
plt.xlabel("Generación")
plt.ylabel("Distancia (km)")
plt.title("Menor distancia encontrada en cada generación")
plt.show()

mejor_individuo.append(mejor_individuo[0])

print("La mejor ruta encontrada es:", mejor_individuo)
print("Con una distancia de:", mejor_fitness, "km")
```

- **Escenario 2**

```
import random
import matplotlib.pyplot as plt

#Se crean individuos para la poblaciñon inicial. No es completamente aleatoria
def crear_individuo():

    ciudad_inicial = random.choice(ciudades)
    individuo = [ciudad_inicial]

    for _ in range(len(ciudades)-1):
```

```
        ciudades_restantes = [ciudad for ciudad in ciudades if ciudad
not in individuo]
        distancia_minima = float('inf')
        ciudad_mas_cercana = None

        for ciudad in ciudades_restantes:
            distancia    =    distancias[ciudades.index(individuo[-
1])][ciudades.index(ciudad)]
            if distancia < distancia_minima:
                distancia_minima = distancia
                ciudad_mas_cercana = ciudad

        individuo.append(ciudad_mas_cercana)

    return individuo

#Cálculo de la aptitud, distancia total
def calculo_fitness(route):
    distancia = 0
    distancia_total=
distancias[ciudades.index(route[0])][ciudades.index(route[-1])]
    for i in range(len(route)-1):
        distancia+=
distancias[ciudades.index(route[i])][ciudades.index(route[i+1])]
    distancia_total += distancia
    return distancia_total

#Selección de los padres
def padres(poblacion_ordenada):
    padre1 = poblacion_ordenada[0]
    padre2 = poblacion_ordenada[1]
    return padre1, padre2

#Cruce
def cruce(padre1, padre2):
    punto_cruce = random.randint(1, len(padre1)-1)
```

```
    hijo1 = padre1[:punto_cruce] + [x for x in padre2 if x not in
padre1[:punto_cruce]]
    hijo2 = padre2[:punto_cruce] + [x for x in padre1 if x not in
padre2[:punto_cruce]]
    return hijo1, hijo2


#Mutación
def mutacion(hijo):
    index1 = random.randint(0, len(hijo)-1)
    index2 = random.randint(0, len(hijo)-1)
    hijo[index1], hijo[index2] = hijo[index2], hijo[index1]
    return hijo


#Datos
ciudades = ["Barcelona", "Bilbao", "Cáceres", "Ciudad Real",
"Coimbra", "Córdoba", "Évora", "Faro", "La Coruña", "León", "Lyon",
"Lisboa", "Madrid", "Málaga", "Marsella", "Mérida", "Murcia",
"Nantes", "Oporto", "Oviedo", "Paris", "Salamanca", "Santiago", "San
Sebastián", "Segovia", "Sevilla", "Toulouse", "Valencia", "Vitoria",
"Zaragoza"]
distancias = [
    [0, 470, 752, 580, 892, 710, 914, 995, 895, 685, 532, 1005, 506,
770, 338, 772, 470, 712, 900, 694, 831, 655, 897, 403, 528, 822, 261,
298, 430, 256],
    [470, 0, 506, 485, 562, 617, 668, 813, 425, 256, 676, 720, 321,
738, 672, 560, 606, 455, 522, 236, 743, 334, 457, 77, 275, 702, 360,
470, 50, 248],
    [752, 506, 0, 222, 185, 225, 166, 304, 454, 353, 1150, 261, 245,
350, 1070, 61, 454, 945, 264, 435, 1253, 167, 421, 564, 253, 240,
795, 514, 488, 522],
    [580, 485, 222, 0, 400, 143, 350, 413, 610, 445, 1041, 455, 163,
255, 914, 208, 263, 936, 470, 513, 1206, 270, 581, 509, 218, 253,
681, 307, 443, 393],
    [892, 562, 185, 400, 0, 400, 185, 355, 352, 352, 1233, 176, 396,
513, 1188, 222, 640, 952, 110, 408, 1279, 241, 300, 630, 365, 373,
892, 685, 555, 645],
    [710, 617, 225, 143, 400, 0, 284, 295, 681, 530, 1180, 392, 300,
134, 1050, 178, 289, 1071, 490, 615, 1347, 351, 640, 648, 344, 120,
823, 422, 580, 534],
    [914, 668, 166, 350, 185, 284, 0, 172, 535, 490, 1316, 108, 419,
```

367, 1234, 143, 565, 1093, 295, 561, 1408, 329, 483, 726, 420, 215, 963, 660, 650, 688],

[995, 813, 304, 413, 355, 295, 172, 0, 706, 652, 1436, 220, 530, 310, 1328, 253, 579, 1250, 464, 727, 1556, 480, 654, 863, 545, 177, 1079, 714, 788, 794],

[895, 425, 454, 610, 352, 681, 535, 706, 0, 220, 1082, 517, 506, 813, 1113, 523, 856, 687, 235, 207, 1029, 347, 54, 520, 445, 694, 801, 793, 469, 645],

[685, 256, 353, 445, 352, 530, 490, 652, 220, 0, 900, 510, 310, 662, 896, 414, 659, 605, 274, 88, 928, 187, 244, 304, 221, 584, 611, 586, 239, 434],

[532, 676, 1150, 1041, 1233, 1180, 1316, 1436, 1082, 900, 0, 1388, 913, 1267, 278, 1188, 995, 517, 1199, 887, 392, 1001, 1109, 605, 899, 1293, 360, 816, 678, 649],

[1005, 720, 261, 455, 176, 392, 108, 220, 517, 510, 1388, 0, 499, 471, 1317, 244, 702, 1126, 279, 585, 1453, 373, 463, 788, 495, 308,1036, 760, 710, 770],

[506, 321, 245, 163, 396, 300, 419, 530, 506, 310, 913, 499, 0, 416, 816, 283, 352, 774, 423, 373, 1053, 175, 488, 353, 68, 392, 561, 305, 282, 274],

[770, 738, 350, 255, 513, 134, 367, 310, 813, 662, 1267, 471, 416, 0, 1108, 296, 323, 737, 613, 748, 1457, 484, 770, 762, 470, 157, 913, 472, 700, 627],

[338, 672, 1070, 914, 1188, 1050, 1234, 1328, 1113, 896, 278, 1317, 816, 1108, 0, 1095, 805, 1158, 1175, 907, 660, 945, 1132, 595, 824, 1163, 319, 642, 656, 545],

[772, 560, 61, 208, 222, 178, 143, 253, 523, 414, 1188, 244, 283, 296, 1095, 0, 434, 1001, 316, 495, 1304, 235, 478, 610, 294, 174, 832, 518, 535, 552],

[470, 606, 454, 263, 640, 289, 565, 579, 856, 659, 995, 702, 352, 323, 805, 434, 0, 1026, 736, 717, 1240, 509, 806, 597, 416, 436, 673, 180, 549, 405],

[712, 455, 945, 936, 952, 1071, 1093, 1250, 687, 605, 517, 1126, 774, 737, 1158, 1001, 1026, 0, 878, 545, 342, 769, 731, 436, 728, 1153, 466, 869, 494, 623],

[900, 522, 264, 470, 110, 490, 295, 464, 235, 274, 1199, 279, 423, 613, 1175, 316, 736, 878, 0, 335, 1213, 253, 192, 597, 380, 478, 869, 726, 525, 647],

[694, 236, 435, 513, 408, 615, 561, 727, 207, 88, 887, 585, 373, 748, 907, 495, 717, 545, 335, 0, 878, 266, 225, 312, 305, 664, 589, 630, 264, 449],

[831, 743, 1253, 1206, 1279, 1347, 1408, 1556, 1029, 928, 392, 1453, 1053, 1457, 660, 1304, 1240, 342, 1213, 878, 0, 1079, 1071,

```
700, 1015, 1441, 588, 1066, 772, 840],

    [655, 334, 167, 270, 241, 351, 329, 480, 347, 187, 1001, 373,
175, 484, 945, 235, 509, 769, 253, 266, 1079, 0, 319, 400, 131, 399,
653, 478, 324, 405],

    [897, 457, 421, 581, 300, 640, 483, 654, 54, 244, 1109, 463, 488,
770, 1132, 478, 806, 731, 192, 225, 1071, 319, 0, 535, 425, 649, 812,
782, 478, 644],

    [403, 77, 564, 509, 630, 648, 726, 863, 520, 304, 605, 788, 353,
762, 595, 610, 597, 436, 597, 312, 700, 400, 535, 0, 317, 741, 278,
449, 77, 206],

    [528, 275, 253, 218, 365, 344, 420, 545, 445, 221, 899, 495, 68,
470, 824, 294, 416, 728, 380, 305, 1015, 131, 425, 317, 0, 427, 544,
358, 244, 280],

    [822, 702, 240, 253, 373, 120, 215, 177, 694, 584, 1293, 308,
392, 157, 1163, 174, 436, 1153, 478, 664, 1441, 399, 649, 741, 427,
0, 1260, 545, 670, 647],

    [261, 360, 795, 681, 892, 823, 963, 1079, 801, 611, 360, 1036,
561, 913, 319, 832, 673, 466, 869, 589, 588, 653, 812, 278, 544,
1260, 0, 485, 344, 288],

    [298, 470, 514, 307, 685, 422, 660, 714, 793, 586, 816, 760, 305,
472, 642, 518, 180, 869, 726, 630, 1066, 478, 782, 449, 358, 545,
485, 0, 424, 243],

    [430, 50, 488, 443, 555, 580, 650, 788, 469, 239, 678, 710, 282,
700, 656, 535, 549, 494, 525, 264, 772, 324, 478, 77, 244, 670, 344,
424, 0, 294],

    [256, 248, 522, 393, 645, 534, 688, 794, 645, 434, 649, 770, 274,
627, 545, 552, 405, 623, 647, 449, 840, 405, 644, 206, 280, 647, 288,
243, 294, 0]

]


#Parámetros

tam_poblacion = 1000

num_generaciones = 500

tasa_mutacion = 0.3


mejores_distancias = []

mejor_individuo = None

mejor_fitness = float('inf')


poblacion = [crear_individuo() for i in range(tam_poblacion)]
```

```
#Bucle principal del algoritmo
for generacion in range(num_generaciones):
    fitness = [calculo_fitness(individuo) for individuo in poblacion]
    poblacion_ordenada = [x for _, x in sorted(zip(fitness, poblacion), reverse=False)]

    padre1, padre2 = padres(poblacion_ordenada)

    if random.random() > tasa_mutacion:
        hijo1, hijo2 = cruce(padre1, padre2)
    else:
        hijo1 = mutacion(padre1)
        hijo2 = mutacion(padre2)

    #Se actualiza la población
    poblacion_ordenada.pop()
    poblacion_ordenada.pop()
    poblacion_ordenada.append(hijo1)
    poblacion_ordenada.append(hijo2)

    poblacion = poblacion_ordenada

    # Se actualiza el fitness y ordenar nuevamente la población
    fitness = [calculo_fitness(individuo) for individuo in poblacion]
    poblacion_ordenada = [x for _, x in sorted(zip(fitness, poblacion), reverse=False)]
    poblacion = poblacion_ordenada.copy()

    # Obtener el mejor individuo de esta generación
    mejor_actual = poblacion[0]
    fitness_mejor_actual = fitness[0]

    # Comprobar si el mejor individuo de esta generación es mejor que el mejor encontrado hasta ahora
    if fitness_mejor_actual < mejor_fitness:
```

```
                mejor_individuo = mejor_actual.copy()
                mejor_fitness = fitness_mejor_actual

            #Almacenar la distancia del mejor individuo de esta generación
              mejores_distancias.append(mejor_fitness)

        #Gráfica de la evolución de la distancia total
        plt.plot(mejores_distancias)
        plt.xlabel("Generación")
        plt.ylabel("Distancia (km)")
        plt.title("Menor distancia encontrada en cada generación")
        plt.show()

        mejor_individuo.append(mejor_individuo[0])

        print("La mejor ruta encontrada es:", mejor_individuo)
        print("Con una distancia de:", mejor_fitness, "km")
```

En el problema de la mochila el código es el mismo para ambos escenarios, lo único que cambian son los datos y valores de los parámetros. Por ello, se presentará el código de un único escenario, el 1.

- **KP**

```
import random
import matplotlib.pyplot as plt

#Datos
pedidos = [('pedido A', 5, 15), ('pedido B', 2, 12), ('pedido C', 2, 5),
('pedido D', 5, 2), ('pedido E', 1, 7)]
capacidad = 30

#Parámetros del algoritmo genético
tam_poblacion = 100
num_generaciones = 25
tasa_mutacion = 0.1
```

```
#Crear individuo aleatorio
def crear_individuo():
    individuo = [random.randint(0, 1) for _ in pedidos]
    return individuo

#Función para el cálculo de la aptitud
def calculo_fitness(individuo):
    urgencia_total = 0
    peso_total = 0
    for i, pedido in enumerate(pedidos):
        if individuo[i] == 1:
            urgencia_total += pedido[1]
            peso_total += pedido[2]
    if peso_total > capacidad:
        urgencia_total = 0
    return urgencia_total



#Selección de dos padres
def padres(poblacion_ordenada):
    padre1 = poblacion_ordenada[0]
    padre2 = poblacion_ordenada[1]
    return padre1, padre2

#Cruce de los padres
def cruce(padre1, padre2):
    punto_corte = random.randint(0, len(pedidos) - 1)
    hijo = padre1[:punto_corte] + padre2[punto_corte:]
    return hijo

#Mutación
def mutacion(hijo):
    indice_mutacion = random.randint(0, len(hijo) - 1)  # Seleccionar
un índice aleatorio para la mutación
```

```
    hijo[indice_mutacion] = 1 - hijo[indice_mutacion]   # Mutar el
elemento en el índice seleccionado
    return hijo



poblacion = [crear_individuo() for _ in range(tam_poblacion)]
urgencia = []
mejor_solucion = None
mejor_fitness = None



#Bucle principal del algoritmo genético
for _ in range(num_generaciones):
    fitness = [(calculo_fitness(individuo), individuo) for individuo in
poblacion]
    poblacion_ordenada = [x for _, x in sorted(zip(fitness, poblacion),
reverse=True)]

    padre1, padre2 = padres(poblacion_ordenada)

    if random.random() > tasa_mutacion:
        hijo = cruce(padre1, padre2)
    else:
        hijo = mutacion(padre1)

    poblacion_ordenada.pop()
    poblacion_ordenada.append(hijo)

    poblacion = poblacion_ordenada

    #Actualizar el fitness y ordenar la población
    fitness = [calculo_fitness(individuo) for individuo in poblacion]
    poblacion_ordenada = [x for _, x in sorted(zip(fitness, poblacion),
reverse=True)]
    poblacion = poblacion_ordenada
```

```
    #Obtener el mejor individuo de esta generación
    mejor_actual = poblacion[0]
    fitness_mejor_actual = fitness[0]

    #Elitismo
    if mejor_fitness is None or fitness_mejor_actual > mejor_fitness:
        mejor_solucion = mejor_actual.copy()
        mejor_fitness = fitness_mejor_actual


    urgencia.append(mejor_fitness)


#Gráfica de la evolución de la aptitud
plt.plot(urgencia)
plt.xlabel("Generación")
plt.ylabel("Urgencia Total")
plt.title("Urgencia Total en cada generación")
plt.show()

nombres_pedidos = [pedido[0] for i, pedido in enumerate(pedidos) if
mejor_solucion[i] == 1]
litros_pedidos = [pedido[2] for i, pedido in enumerate(pedidos) if
mejor_solucion[i] == 1]
litros_totales = sum(litros_pedidos)

print("Mejor solución encontrada:", mejor_solucion, "Aptitud:",
mejor_fitness)
print("Pedidos incluidos en la mochila:", nombres_pedidos)
print("Volumen total:", litros_totales, "litros")

#Gráfica de los pedidos incluidos en la mochila
nombres = [pedido[0] for pedido in pedidos]
urgencias = [pedido[1] if mejor_solucion[i] == 1 else 0 for i, pedido
in enumerate(pedidos)]
litros = [pedido[2] if mejor_solucion[i] == 1 else 0 for i, pedido in
```

```
enumerate(pedidos)]


posicion = range(len(nombres))
ancho_barra = 0.3


plt.figure(figsize=(10, 6))
plt.bar(posicion, urgencias, width=ancho_barra, label='Urgencia')
plt.bar([p + ancho_barra for p in posicion], litros, width=ancho_barra,
label='Litros')
plt.xlabel('Pedidos')
plt.ylabel('Valor')
plt.title('Urgencia y volumen de los pedidos')
plt.xticks([p + ancho_barra / 2 for p in posicion], nombres)
plt.legend()
plt.show()
```