Trabajo Fin de Máster

Máster en Ingeniería Industrial

Desarrollo de algoritmos aproximados para la resolución del problema de gestión del flujo de pacientes en un servicio de urgencia hospitalario

Autor: Jaime Personat Moyano

Tutor: José Manuel Molina Pariente

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

Sevilla, 2023

Trabajo Fin de Máster
Máster en Ingeniería Industrial

Desarrollo de algoritmos aproximados para la resolución del problema de gestión del flujo de pacientes en un servicio de urgencia hospitalario

Autor:

Jaime Personat Moyano

Tutor:

José Manuel Molina Pariente

Dpto. Organización Industrial y Gestión de Empresas I

Escuela Técnica Superior de Ingeniería

Universidad de Sevilla

Sevilla, 2023

Trabajo Fin de Máster: Desarrollo de algoritmos aproximados para la resolución del problema de gestión del flujo de pacientes en un servicio de urgencia hospitalario

Autor: Jaime Personat Moyano

Tutor: José Manuel Molina Pariente

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

Resumen

El presente proyecto se realiza con el objetivo de mejorar la calidad en la atención de los servicios de urgencia hospitalario, en términos de reducir los tiempos de estancia total de los pacientes y la espera para la primera atención por parte del personal médico, dándole mayor importancia a este incumplimiento para aquellos pacientes categorizados con la máxima prioridad.

Para ello, el problema se centrará en la secuenciación de las actividades referentes a los procesos de urgencia de los pacientes, lo que corresponde al nivel operativo en la gestión de un Servicio de Urgencia Hospitalario (SUH). Se proponen distintas metaheurísticas que ayudarán en esta secuenciación, minimizando las respectivas funciones objetivo y cumpliendo con las restricciones planteadas. Las metaheurísticas utilizadas son las propuestas por Ruiz [1], a las que se les aplicarán algunas modificaciones y se comparará su resultado.

Todo esto se llevará a cabo haciendo uso del lenguaje de programación Python y varias de las librerías que ofrece. Se llevará a cabo una experiencia computacional basada en Bedoya [2], considerando como factores la configuración de recursos en el SUH y el nivel máximo de saturación de los mismos. Para cada combinación de configuración de recursos y nivel de saturación, se han generado un conjunto de instancias para obtener unos resultados representativos.

Finalmente, se analizarán los resultados obtenidos por las distintas variantes propuestas de la metaheurística para dos tipos de representación de la solución: por pacientes o por actividades pertenecientes a cada paciente. Para terminar, se concluye en cuál de estos algoritmos y tipo de representación proporciona una mejor solución en la secuenciación para el SUH para un mismo tiempo de computación.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

El Sistema Sanitario Nacional se enfrenta a desafíos y problemas de gran complejidad, ya que tiene que cumplir con un estándar de calidad y de servicio hacia los pacientes minimizando en todo momento los costes que eso conlleva. En la actualidad, muchos investigadores enfocan su trabajo en el desarrollo de herramientas y sistemas que apoyen y faciliten la atención hospitalaria, más concretamente en el SUH [3]. La optimización del SUH es aún más compleja debido a sus características aleatorias y, por tanto, difíciles de predecir, saturándose en las horas punta o en oleadas de enfermedades infecciosas, como ocurrió durante la pandemia de COVID-19. Es por ello por lo que se investigan nuevas técnicas de simulación y optimización con el objetivo de solventar varios de los problemas de toma de decisiones en el ámbito sanitario.

El presente proyecto se centra en el SUH. El principal problema que afecta a este servicio es la congestión, siendo declarado como ineficiente por parte de la población al no cumplirse sus necesidades básicas y con ello incumpliendo el estándar de calidad exigido [2]. Esta congestión se traduce en demoras en la atención del paciente, provocando en ocasiones que éste abandone el SUH sin tratamiento y con la insatisfacción por el servicio prestado. Ante esta congestión, los profesionales son forzados a trabajar a ritmos elevados y con altas cargas de trabajo, lo que puede dar lugar a diagnósticos erróneos o cualquier otro error humano, así como generar la insatisfacción del mismo personal hospitalario al estar sometidos a un mayor estrés. Todo esto, siendo el cuidado de la salud un servicio esencial en la sociedad, hace que la optimización del SUH sea primordial y de constante objeto de estudio [2].

Existen dos puntos de vista para la mejora del SUH. Por un lado, la optimización se puede centrar en la correcta identificación y priorización de aquellos pacientes que requieren una atención más temprana. Además, al haber numerosos pacientes a la vez en el SUH y, como se ha dicho anteriormente, siendo de gran complejidad la previsión de la llegada de éstos, es necesaria una correcta asignación de recursos y secuenciación de actividades para minimizar el tiempo de estancia en el SUH y atender en el menor tiempo posible a los pacientes de mayor gravedad. Además, se incluye la gestión de los turnos para asegurar que los recursos humanos disfruten del descanso adecuado, manteniendo la disponibilidad para la correcta atención de los pacientes y minimizando el impacto por falta de personal. Por otro lado, es necesario que el SUH conste del número de recursos necesarios para atender al volumen de pacientes que acuden al centro, siendo estos recursos tanto médicos como enfermeros, máquinas de diagnóstico, camas de hospital, etc. Esta serie de acciones deben realizarse sin perder de vista uno de los objetivos de la gestión del centro: la minimización de los costes [4]. No obstante, este aspecto adquiere más importancia en centros del ámbito privado. En el ámbito público, donde se enmarca el presente proyecto, se prioriza la calidad del servicio mediante la reducción de los tiempos de espera, atención a pacientes prioritarios, etc. [3].

El proyecto que se presenta se centra, por tanto, en la optimización del SUH, concretamente en la secuenciación y asignación de las actividades de los distintos pacientes a los diferentes recursos que se encuentran disponibles en el respectivo horizonte temporal. Debido a la variabilidad tanto en número de pacientes y su prioridad, como en instantes de llegada y en tiempos de proceso, este problema no tiene fácil solución y su recreación es compleja, por lo que se harán una serie de suposiciones que faciliten la resolución del problema, siempre buscando el mejor ajuste a una

situación real. Para la optimización en la secuenciación de las actividades se utilizarán algoritmos basados en metaheurísticas conocidas de la literatura, como es el algoritmo *Iterated Greedy* [1], siendo la diferenciación del presente proyecto la propia aplicación de estos algoritmos, ya que se estudiará el efecto de dos tipos de representación de la solución. Estos dos tipos se diferencian en cómo se ha llevado a cabo la construcción de la secuencia que marca el orden de asignación de las distintas actividades a los recursos del SUH: por pacientes y por actividades independientes respectivamente, mediante una experimentación computacional basada en la literatura.

1.1 Objetivos

El principal objetivo del presente proyecto es la optimización en la gestión del flujo de pacientes en un SUH a nivel operativo. Esta optimización se llevará a cabo poniendo el foco en la secuenciación de las actividades, a través de la ordenación de los pacientes o de las actividades individuales asociadas a los procesos de urgencias asignados a cada paciente, el cual llega al SUH o ya se encuentra en él en un instante de tiempo determinado. Para ello se hace uso de metaheurísticas siguiendo los siguientes pasos:

- La elaboración del estado del arte de la gestión del flujo de pacientes de un SUH.
- El diseño de metaheurísticas a partir de otras ya existentes, combinándolas y adaptándolas para lograr el mejor resultado en la resolución del problema.
- La validación de las metaheurísticas propuestas para la resolución de la gestión del flujo de pacientes en el SUH, generando los datos de pacientes y recursos a partir de [2].
- La evaluación de las soluciones obtenidas y la eficacia de las metaheurísticas propuestas realizando un análisis computacional estableciendo distintos escenarios.

1.2 Sumario

La memoria se compone de un total de 7 capítulos y un anexo. A continuación, se realiza un resumen de sobre qué trata cada capítulo:

1. Introducción: incluye la presentación del presente proyecto y del problema que intenta resolver, los objetivos planteados y los pasos seguidos para llevarlos a cabo.
2. Estado del arte: se realiza un repaso de la literatura sobre cómo se ha tratado históricamente el problema que se quiere optimizar.
3. Descripción del problema: se describe el problema detalladamente, describiendo las restricciones que lo definen, variables y funciones objetivo a minimizar. Se presenta el entorno sobre el que se basa el problema.
4. Metodología de resolución: se realiza una explicación del método para la construcción de la solución y de su estructura, presentando las dos posibilidades que se plantean en la optimización del problema. Definición de los algoritmos y las metaheurísticas utilizadas en la optimización.

5. Análisis computacional: se definen y generan diferentes escenarios de gestión del SUH. Se describe cómo se generan los datos de entrada y la calibración de los parámetros para el correcto funcionamiento de los algoritmos.

6. Resultados: se presentan los resultados obtenidos en el análisis computacional y se analizan para concluir si se ha resuelto el problema.

7. Conclusiones: se resumen las conclusiones obtenidas en el apartado anterior y se propone una revisión para futuras mejoras.

8. Anexo: se incluye el código generado en lenguaje Python, tanto la generación de los algoritmos como el *decoding* necesario para optimizar la asignación de actividades a los recursos del SUH.

2 ESTADO DEL ARTE

En este apartado se realiza un recorrido de la literatura en la que se ha estudiado la optimización del SUH. Se recogen algunas de las distintas metaheurísticas que se han ido utilizando a lo largo del tiempo y se analizan aquellos artículos que han tratado la programación del flujo de pacientes y qué tipo de secuenciación han utilizado, comparándolo finalmente con lo realizado en el presente proyecto. Para realizar este recorrido se ha revisado y ampliado la investigación realizada por [5].

Existen multitud de artículos y estudios que tratan el problema de la optimización del SUH. Una de las disciplinas que se siguen para llevar cabo la optimización de un SUH es la Investigación Operativa [6]. La Investigación Operativa utiliza métodos analíticos, la mayoría de ellos matemáticos, estadísticos o computacionales, con el objetivo de alcanzar un óptimo o soluciones cercanas al óptimo en problemas complejos de toma de decisiones [6]. La utilización de metodologías enmarcadas en la Investigación Operativa del SUH se ha expandido. Un ejemplo de ello es la aparición y ascensión en el uso del *Big Data* [6]. Esta área de investigación se caracteriza por hacer uso de grandes cantidades de datos y proveerlos de métodos predictivos, que son adoptados en las tomas de decisiones organizacionales [6].

Para la realización de cualquier proyecto de optimización es necesario definir las restricciones del problema que se quiere resolver, para luego llevar a cabo la optimización en todos los escenarios posibles y realizar un análisis de los resultados obtenidos. Uno de los pasos clave en el modelado del problema es la identificación del carácter del modelo de optimización que se quiere desarrollar. Existen 4 tipologías en función de las distintas propiedades que definen un modelo: estático o dinámico, continuo o discreto, determinista o estocástico y cerrado o abierto [7]. En cuanto al problema de gestión del flujo de pacientes de un SUH, se define como un problema dinámico y estocástico, sin embargo, se pueden hacer algunas suposiciones para considerarlo como un problema estático y determinista, tal y como se hace en el presente proyecto.

En primer lugar, los pacientes llegan al SUH por cuatro posibles vías: a pie, traídos por la ambulancia, en helicóptero o procedentes de emergencias policiales [2]. La vía de entrada al SUH puede llevar asociada una determinada prioridad al paciente en cuestión, como podría ser en el caso de que el paciente llegue al hospital por vía aérea, siendo lo común que lleve asociada una urgencia mayor. La predicción de llegada de los pacientes al centro hospitalario, en cantidad e instante de tiempo, se considera que sigue una distribución aleatoria. No obstante, hay investigadores que la aproximan a distribuciones Poisson o Weibull respectivamente [8]. Las mismas distribuciones son usadas para la predicción del modo de llegada, concretamente para la llegada a pie y en ambulancia, respectivamente [2].

La gestión del SUH se puede clasificar según el nivel de decisión en la gestión del SUH, pudiendo ser a nivel táctico u operativo [6]. Si la decisión adoptada afecta al diseño de los recursos disponibles para realizar las distintas actividades pertenecientes al proceso de urgencia (PU) de un paciente, se trata del nivel táctico [6]. Si la mejora se analiza desde el punto de vista de la planificación operativa o la gestión del inventario del SUH, el enfoque es a nivel operativo [6].

El horizonte temporal de los artículos que tratan el nivel táctico es de a medio- largo plazo [5]. La intención en este nivel es dimensionar el SUH y dotarle de los recursos adecuados para satisfacer

la demanda y, además, prestar un servicio de calidad dentro de los niveles establecido por la dirección del hospital [2]. La programación de los turnos del personal pertenece a este nivel de decisión en la gestión del SUH. En [9] se propone un algoritmo genético mediante el cual se busca optimizar un problema multiobjetivo que minimiza el tiempo de espera de los pacientes, categorizados según su prioridad, teniendo en cuenta el tamaño de la plantilla mediante el método del *Sample Average Approximation*. También se minimiza el coste total del SUH, calculado según las necesidades de personal y el diseño de los turnos. Para la estimación de los pacientes que llegan al SUH y su tiempo de espera se hace uso de una simulación Monte Carlo combinada con programación matemática. Otro ejemplo de optimización del SUH a nivel táctico es [10], donde se quiere optimizar la configuración de los recursos que componen el SUH mediante el uso novedoso de metamodelos (red neuronal artificial y función de base radial) combinados con un modelo de simulación de eventos discretos. El objetivo es mejorar el flujo de pacientes y descongestionar el SUH dimensionando los recursos que lo componen. Además, se concluye que esta nueva herramienta es útil para la toma de decisiones tanto a nivel operativo como al táctico y al estratégico. En [11], se aborda el nivel táctico mediante la utilización de la simulación Monte Carlo para medir la efectividad y el coste de un sistema de salud pública que involucra a voluntarios cardíacos. El SUH se revisa con el objetivo de aumentar la supervivencia de pacientes con afecciones cardíacas. Para ello, se intenta reducir el tiempo que transcurre hasta la primera atención para el tratamiento de emergencia, en este caso mediante el uso de desfibriladores.

En cuanto al nivel operativo, la mayor parte de la investigación se basa en la programación de las actividades relativas a los PU de los pacientes que acuden al SUH, el mismo problema que se trata en el presente proyecto. En [6] se realiza una recopilación de distintas técnicas cuantitativas que suelen ser usadas para la toma de decisiones en los SUH, enfocándose en el nivel operativo. La resolución del problema se aborda a través del modelado matemático y técnicas de optimización, análisis de decisión multicriterio, simulación de eventos discretos y modelos de decisión bayesianos. Para llevar a cabo la optimización del SUH se pueden usar distintos indicadores referentes al servicio prestado por el SUH: el más usado en la literatura es el *Length of Stay* (LOS), el cual se refiere al tiempo de estancia total que un paciente permanece en el SUH [12]; otro de los indicadores usados es el denominado *Door to Doctor* [13] el cual mide el tiempo que transcurre desde el instante de llegada de un paciente hasta su primera consulta.

El algoritmo desarrollado en [14] gestiona la optimización del SUH mediante la secuenciación de los pacientes. Se hace uso de una heurística constructiva que aplica dicha secuenciación buscando minimizar el tiempo de espera de los pacientes dándole más importancia a aquellos con prioridades asistenciales más altas. En este artículo también se tiene en cuenta la carga de trabajo de los distintos recursos que componen el SUH en cuestión, intentando equilibrarlas. Además, se hace mención al carácter estocástico de los datos de entrada del algoritmo: las prioridades de los pacientes, su instante de llegada y la duración de las distintas actividades asignadas a cada paciente. Para realizar la predicción de dichos datos se propone la recogida de datos y el uso de inteligencia artificial para su tratamiento.

En [15] se vuelve a realizar la optimización del SUH actuando sobre la secuenciación de los pacientes. Para realizar la optimización se propone un modelo de programación lineal entera que tiene como objetivo la minimización del tiempo de espera total de los pacientes para acceder a cada recurso del SUH (correspondiente a las actividades de registro, triaje, consulta, tratamiento y cama de hospital). La diferenciación que propone este articulo frente a otros de la literatura es la

consideración de todos los niveles del PU y la disponibilidad de los recursos, tanto humanos como materiales. Se concluye que el modelo encuentra soluciones óptimas en problemas de reducidas dimensiones, sin embargo, para problemas de más de 25 pacientes este modelo no es factible debido al elevado tiempo que requiere la resolución.

Una de las principales complejidades que presenta el SUH a la hora de realizar la secuenciación de los pacientes es la necesidad de atender antes a pacientes nuevos, con prioridades asistenciales más altas, que otros que ya se encuentran en el SUH y ya han sido programados. En [16] se intenta resolver este problema mediante el uso de un método de programación dinámica, el cual tiene en cuenta el impacto de la llegada de nuevos pacientes al SUH sobre los pacientes que ya han sido secuenciados, minimizando el tiempo de espera de los pacientes en el SUH. La novedad que aporta este artículo es la asignación de un espacio de tiempo programado a cada paciente en el momento de su llegada al SUH, con el objetivo de reducir el estrés.

Manteniendo el centro de la investigación en la secuenciación de pacientes, en [17] se realiza únicamente para las actividades cuyo recurso utilizado es el laboratorio. El objetivo en este caso es minimizar el tiempo total de finalización ponderado en función de la prioridad asistencial asociada al paciente en cuestión. Con esta función objetivo se pretende dar más importancia a aquellos pacientes más urgentes según el factor asociado en la etapa de triaje, pero sin desatender al resto de pacientes. Para resolver el problema se usa un modelo de programación lineal entera y se aplica un algoritmo genético, cuyos parámetros son optimizados usando *Response Surface Methodology*. Se concluye que los experimentos de optimización realizados muestran una mejoría en la secuenciación de pacientes y en la eficiencia en el uso de los recursos del SUH.

En [18] se hace uso de metaheurísticas para tratar el problema de la secuenciación de pacientes en el SUH, incluyendo las principales actividades que se le realizan al paciente: triaje, consulta, tratamiento y hospitalización. El objetivo vuelve a ser minimizar el tiempo total de espera de los pacientes que acuden al SUH. Se propone un algoritmo híbrido entre una búsqueda local iterada (ILS) y una búsqueda de vecindad variable (VND). La solución inicial de la que parte el algoritmo se obtiene aplicando la heurística *First In First Out,* a la que se le realiza la exploración variando entre 4 generaciones de vecindades para asegurar una mejor búsqueda en el espacio de la secuencia. Los resultados obtenidos muestran que el algoritmo híbrido propuesto encuentra mejores soluciones y de forma eficiente hasta los 24 pacientes. A partir de esta dimensión del problema, el modelo no encuentra soluciones óptimas en un tiempo razonable de resolución.

De nuevo, la metaheurística es utilizada por [19] para minimizar el retraso ponderado asociado al tiempo de espera de los pacientes del SUH. Esta ponderación aumenta a medida que el paciente espera más allá de su tiempo inicialmente asignado. Se usa una heurística *greedy* combinada con una general variable neighborhood search para optimizar la búsqueda. La GVNS usa una variación de vecindades como es la búsqueda local que, a su vez, varía el tipo de búsqueda entre seis estructuras basadas en los distintos movimientos mediante los cuales se puede llevar a cabo la búsqueda local: intercambio, inserción y eliminación. Se resuelve el problema estático, aportando una solución de referencia para el problema dinámico que define realmente la secuenciación de pacientes, en este caso, únicamente para el recurso de consulta médica.

Los autores del artículo mencionado en el párrafo anterior llevan a cabo una nueva investigación en [20], la cual sirve de ampliación a la realizada previamente. En este artículo se vuelve a estudiar

el problema dinámico de la programación de pacientes a las consultas médicas con el objetivo de minimizar el *tardiness* total ponderado. Se propone una heurística simple de reoptimización basada en múltiples listas de pacientes en función de su prioridad asistencial, combinándola de nuevo con una VND. Además, la heurística anterior se compara con una formulación *Arc-Flow* con el objetivo de obtener buenos resultados en el problema estático y evaluar correctamente el problema dinámico estocástico. Por último, se introduce una planificación basada en escenarios que usa muestras de múltiples escenarios para prever futuros eventos con el objetivo de paliar la estocasticidad del problema.

Una investigación reciente propone un nuevo enfoque para la optimización del SUH a partir de la secuenciación en los recursos que componen el SUH. Este innovador enfoque se propone en [21], el cual consiste en integrar *Agent-Based Simulation* y Simulación de Eventos Discretos, siendo por tanto un método de modelado híbrido que introduce una nueva forma de modelar sistemas dinámicos complejos, como es el caso del SUH. Se contemplan cinco estrategias diferentes para llevar a cabo la programación de los pacientes. La primera es FIFO+*Random*, siendo FIFO una heurística que programa los pacientes por orden de llegada y *Random* una mediante la cual los pacientes se asignan a los recursos de forma aleatoria. Otra estrategia es la combinación de FIFO+*Centralised* y *Random*+*Centralised*, siendo *Centralised* un método de asignación basado en la optimización del recurso por parte de un "agente". Por último, se encuentra *Random*+*Random* y *Autonomous*. Estas estrategias se implementan para programar los pacientes a los recursos del SUH. Se concluye que la estrategia que mejores resultados aporta en cuanto al tiempo de estancia de pacientes que no requieren hospitalización es la de FIFO+*Centralised*. Para los pacientes que sí requieren hospitalización, la combinación de estrategias con mejor resultado es *Random*+*Centralised*.

La investigación realizada por [22] se centra de nuevo en la secuenciación de las consultas de un SUH. En este artículo, se considera el SUH como un sistema de red de colas variable en el tiempo con retornos y se facilita una metodología analítica para aproximar el estado del propio sistema y el tiempo de espera de los pacientes. Esta metodología se basa en el método de aproximación puntual del flujo de fluidos estacionario. A continuación, se dividen los pacientes en tres grupos, analizando para cada grupo el tiempo de espera de los pacientes y, combinándolos, el tiempo de espera total. Usando métodos de aproximación y una técnica de linealización puntual, se formula un modelo de programación entera mixta y se diseña un algoritmo tabú que ayude a optimizar dicho modelo. Esta investigación se diferencia de la realizada en [23] en el nivel de gestión desde el que se actúa. En este último artículo, se pretende dimensionar el recurso de los médicos del SUH para unos niveles de servicio determinados mediante la consideración del rendimiento de los sistemas de servicios no estacionarios con retornos. La diferencia entre un artículo y otro radica en que, en [23], se evalúa directamente la carga de los recursos. Sin embargo, en [22] se analiza, como ya se ha dicho, el estado del sistema y el tiempo de espera total de los pacientes.

En cuanto al nivel de gestión del SUH al que se actúa para realizar la optimización, existe la posibilidad de actuar desde varios a la vez. Lo realizado en [24] tiene como principal objetivo el dimensionamiento óptimo de recursos tales como camas de hospital, enfermeros y médicos para diferentes niveles de precisión, lo cual es relativo al nivel táctico de gestión de un SUH. Además, se busca minimizar tanto el tiempo total de espera de los pacientes, como el LOS medio y equilibrar las cargas de trabajo de los distintos recursos, considerando así 3 KPIs de manera simultánea. El trabajo realizado en este articulo combina, por tanto, dos tipos de problema: el

dimensionamiento de los recursos y la relación de los resultados obtenidos para dicha composición con distintas reglas de secuenciación de pacientes: FIFO, con prioridad y vía rápida, lo cual corresponde al nivel operativo de gestión del SUH. Para resolver el problema se consideran todas las fases del proceso de un SUH: admisión y triaje, examinación, exámenes adicionales, unidad de cuidado y cama de hospital, siendo esto también un aspecto novedoso en la literatura. Para optimizar el problema se crea un modelo de programación lineal entera mixta (MILP) estocástico resuelto mediante el método *Sample Average Approximation*. Únicamente se realiza el dimensionamiento del recurso más crítico de un SUH: las camas de hospital. Los resultados obtenidos en cuanto a la estrategia de secuenciación a utilizar mostraron que la de vía rápida y por prioridad resultan mejores en cuanto a la utilización del recurso de camas.

En el capítulo I de [25] se desarrolla un modelo MILP que minimiza el tiempo de espera total de los pacientes y equilibra la carga de trabajo entre los recursos del SUH, en este caso entre los facultativos. Todo esto se lleva a cabo mediante la asignación de los pacientes a los distintos recursos detectando los huecos ociosos y cuellos de botella, los cuales afectan negativamente a los objetivos del problema. En primer lugar, se define un PU, concepto que también es usado en el presente proyecto. Un PU corresponde al conjunto de actividades secuenciadas que se le deben realizar a un paciente desde su entrada al SUH hasta su alta u hospitalización. El modelo creado pretende facilitar la toma de decisiones en la gestión del SUH mediante la asignación de las actividades pertenecientes a los PU de los distintos pacientes al recurso correspondiente. En nivel operativo se distinguen dos etapas: off-line, en la cual se fijan los turnos de trabajo de los recursos humanos del SUH; y on-line, donde se asigna el recurso a la actividad correspondiente [25]. El trabajo desarrollado en [25] y al igual que el llevado a cabo en el proyecto de la presente memoria, se centra en la etapa on-line del nivel operativo. Esto implica que, para un instante de tiempo dado, se considera estático el conjunto de pacientes que se encuentran en el SUH, estando este conjunto compuesto por los pacientes nuevos que llegan al SUH y aquellos que ya se encuentran en él desde instantes previos al estudiado y que pueden haber sido atendidos parcialmente.

El proyecto llevado a cabo por [5] trata el mismo problema de secuenciación de pacientes, donde también se busca minimizar el incumplimiento del tiempo máximo de atención según la prioridad asistencial asignada en la etapa de triaje (TEPCOF) y el tiempo de estancia total de los pacientes en el servicio de urgencias. Se lleva a cabo un análisis computacional mediante el modelado del problema en lenguaje C++, realizando la comparación de dos algoritmos: algoritmo genético y una variante del *Iterated Greedy*.

Una vez realizado este recorrido por la literatura para hacer un resumen de cómo se ha tratado el problema que se intenta resolver en el presente proyecto a lo largo de los años, se destacan las novedades que se aportan con este nuevo trabajo. En primer lugar, en cuanto a la secuenciación de los pacientes para la gestión del flujo de los propios pacientes, en el presente proyecto se realiza de dos formas: secuenciando los pacientes que se encuentran en el SUH mediante la programación consecutiva de las actividades que componen su PU, tal y como se ha realizado en todas las investigaciones explicadas en párrafos anteriores y relacionadas con este aspecto. Se propone un segundo tipo de representación de la solución. En este nuevo modo de construcción de la secuencia, se realiza la programación de las actividades que forman parte de los PU asignados a cada paciente una a una, respetando en todo momento el orden de precedencia y sucesión que en estos se establecen. En este caso, se puede realizar la asignación a los distintos recursos sin que

sea forzosa una programación que realice la asignación consecutiva de las actividades del PU de un paciente antes de pasar al siguiente. Hablando de las heurísticas utilizadas, se sigue lo propuesto por [1] aunque aplicándolo a la gestión del SUH, siendo el trabajo original aplicado a un entorno *Flowshop,* cuando el SUH es entendido como un entorno *Jobshop.* Un entorno *Jobshop* se define como la asignación de una serie de procesos a unos recursos, con una secuencia que impone el orden en esta asignación ya predefinida, sin solapamiento de actividades en cada recurso [26].Tal y como se ha visto en el Estado del Arte, el algoritmo *Iterated Greedy* no es de frecuente utilización para llevar a cabo la optimización de modelos en la gestión del SUH, siendo el único precedente encontrado el trabajo realizado por [5]. En el presente proyecto, además, se incluyen varios de los recursos que componen el SUH y las distintas actividades que pueden asignársele a un paciente en la etapa de triaje y que componen el PU. Esto lo diferencia de muchos de los artículos encontrados, en los que la optimización se centra en un único recurso como pueden ser las camas de hospital, médicos, etc. Esto conlleva que no se incluyan varias de las actividades al no ser realizadas por el recurso que se quiere optimizar. Por último, en el desarrollo del proyecto se ha programado el *decoding* necesario para la correcta asignación de las distintas actividades a sus correspondientes recursos y de las unidades de estos recursos requeridas según una secuencia dada. Con esto, se establece la diferenciación respecto a todo lo realizado con anterioridad y respecto al trabajo realizado por [5], el cual trata el mismo problema que en el presente proyecto.

3 DESCRIPCIÓN DEL PROBLEMA

En este apartado se lleva a cabo la descripción del problema sobre el que se ha basado el proyecto, explicando el funcionamiento del SUH, el procedimiento para la creación de las actividades asignadas a los distintos pacientes y las restricciones aplicadas a la gestión del SUH a nivel operativo. El objetivo es realizar la optimización en base a una jornada de trabajo en el SUH que sea fiel a la realidad, dentro de las limitaciones que establece la complejidad del problema a resolver.

3.1 Flujo de pacientes y recursos del SUH

La llegada de los pacientes a un SUH se produce de manera aleatoria, de manera que no es posible predecir cuándo llegará cada paciente y la prioridad de urgencia que tendrá asociada. Además, el tiempo que necesitará cada actividad asociada a su PU es, de nuevo, difícil de predecir. Es por ello por lo que, para realizar la optimización, se realizarán aproximaciones y suposiciones que faciliten la resolución del problema.

Tras la llegada del paciente al servicio de urgencias y su admisión, donde se le registra en el SUH, tiene lugar el proceso de triaje. En esta etapa se le asigna el nivel de prioridad asistencial o ESI, el cual lleva asociado el PU y, con ello, los recursos de los que precisa el paciente para su correcto tratamiento. En total, el Sistema Español de Triaje establece cinco niveles de prioridad [27], tal y como se muestra en la Tabla 1. Esta prioridad asistencial se asigna en función del estado del paciente y de la urgencia de su tratamiento, siendo los pacientes de ESI 1 los de mayor prioridad asistencial. A este nivel de prioridad pertenecen los pacientes que requieren de procesos llamados tiempo-dependientes, como pueden ser los casos de ictus, infarto o trauma grave [28]. Los ESI llevan asociados los tiempos máximos de espera para la primera asistencia por parte del personal médico (TEPCOF).

Nivel de prioridad	Urgencia	Tiempo de atención (min)
1	Reanimación	1
2	Urgente	10
3	Moderado	60
4	Baja	120
5	No Urgente	240

Tabla 1: Prioridades asistenciales [27]

Tras la primera consulta anteriormente mencionada, se le asignan diversas pruebas de diagnósticos si es necesario, tales como análisis de sangre en el laboratorio, prueba radiológica, etc. [2]. Para el análisis de la gestión del SUH, estas pruebas estarán predefinidas en el PU del paciente en función de su prioridad asistencial. De esta manera, el paciente de prioridad 1 requiere de todas las pruebas diagnósticas, además de una segunda revisión por enfermería (2 unidades de este recurso) y una segunda consulta con el mismo médico que le asistió en la primera, respetando la continuidad asistencial. En el lado opuesto, el paciente de prioridad 5 únicamente requerirá de 1 revisión en enfermería y en consulta por un médico asistente, constando su PU de 3 actividades únicamente, incluyendo el alta [2].

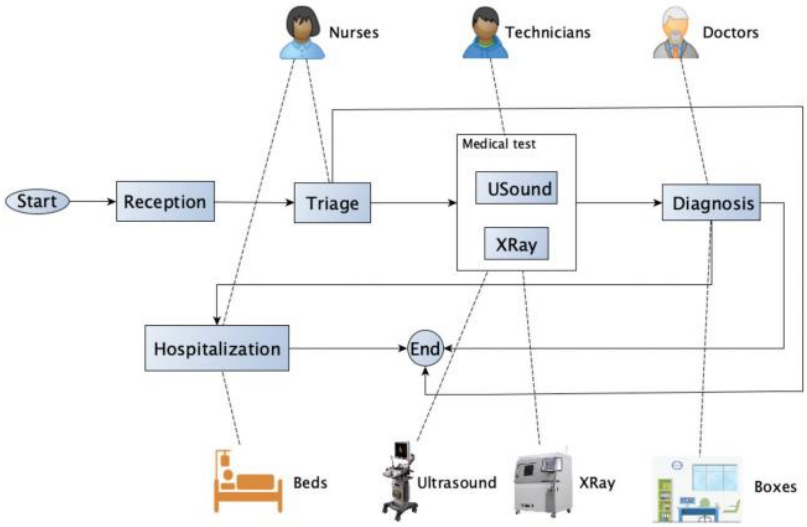

Figura 1: El flujo de pacientes en SUHs [34]

Las duraciones de las distintas actividades que componen un PU son, de nuevo, difíciles de predecir, ya que dependen de múltiples factores como pueden ser el estado del paciente, la experiencia del médico o enfermero, la capacidad del recurso, etc. En el presente proyecto, se sigue lo propuesto por [2], donde se establece la duración de las distintas actividades en función del ESI del paciente. La última actividad asignada a todo paciente es el alta, en el cual el paciente se hospitaliza o se le da el alta hospitalaria, saliendo en este momento del SUH.

La programación de los pacientes, o de las actividades que componen su PU, se realiza asignando las actividades a los recursos libres y que pertenecen al tipo de recurso que puede llevar a cabo la actividad. Esto es a diferencia de lo realizado en [2], donde son los recursos los que se asignan a las distintas actividades. Es decir, la configuración que se resuelve en el presente proyecto se define por el desplazamiento de los pacientes a través de los distintos recursos del SUH y no al contrario.

3.2 Definición del problema

El objetivo de la optimización se centra, principalmente, en dos aspectos. Por un lado, el objetivo es minimizar el tiempo de espera de los pacientes para su primera consulta, dentro del intervalo de tiempo que establece el TEPCOF según el ESI. Por otro lado, se busca que el LOS de cada paciente

en el SUH sea el menor posible. Respecto a la primera función objetivo, se le asigna un peso en función del ESI del paciente correspondiente con el fin de priorizar aquellos que tienen un TEPCOF menor, siendo su atención prioritaria. Esto afectará sobre todo a los pacientes de ESI 1 y 2. La minimización de ambas FO se lleva a cabo mediante la correcta secuenciación de actividades, respetando siempre la precedencia que establece el PU de cada paciente. Esto conlleva, por ejemplo, que la actividad de análisis de sangre nunca se realice con anterioridad a la correspondiente a la primera consulta por parte del médico.

En cuanto a las actividades que pueden componer un PU según [2], se encuentran:

- Primera revisión del paciente por personal de enfermería.
- Primera consulta de evaluación del paciente por médico o médico asistente.
- Analítica en laboratorio.
- Pruebas radiológicas.
- Segunda revisión, si así lo requiere, por parte de enfermería.
- Segunda consulta por personal médico.
- Alta, pudiendo ser ingreso hospitalario o enviado al domicilio.

Patient ESI level	Resources				Required tests		
	Physician (P)	Physician assistant (PA)	Nurse	EMT	Blood	X-Ray	CT-Scan
1	1		2	1	√	√	√ 70%
2	1		2	1	√	√	√ 70%
3	1		1		√	√	√ 30%
4		1	1		X	√	X
5		1	1		X	X	X

Note. Patients rated as ESI levels 1 and 2 are considered high risk. Patients rated as ESI levels 3–5 are considered non-high risk.
EMT: Emergency medical technician.

Figura 2: Recursos requeridos por los pacientes según su ESI [2].

Cada actividad lleva asignado un único tipo de recurso. En función de la prioridad, puede requerir más de una unidad de ese tipo de recurso, como ocurre con los ESI 1 y 2, que necesitan de 2 unidades del tipo de recurso enfermería (ver Figura 2). Los recursos de los que consta el SUH son [2]:

- Consulta (médico) para los ESI 1, 2 y 3.
- Consulta (médico asistente) para los ESI 4 y 5.
- Enfermería.
- Laboratorio.
- Rayos X.

- CT-Scan.
- Alta

El recurso de Alta es ficticio, únicamente se crea para asociarle un tipo de recurso a esta actividad, ya que no pueden crearse actividades que no tengan asignado ningún recurso.

Existen restricciones que definen la realización de las actividades que componen el PU, así como la correcta secuenciación. Se ha de respetar la precedencia entre las actividades, como se ha mencionado anteriormente. No obstante, las actividades que requieran de los recursos de laboratorio para el análisis de sangre y de Rayos X pueden ser realizadas de manera simultánea, ya que el laboratorio no requiere de la presencia del paciente. Por otro lado, la capacidad de cada tipo de recurso estará definida por las unidades de las que consta ese tipo de recurso exceptuando al recurso de laboratorio, cuya capacidad se supone infinita al tener la capacidad de realizar varios análisis de sangre en paralelo, sin que se vean afectadas las correspondientes actividades entre sí.

Cada recurso, por lo tanto, estará definido por el tipo de recurso al que pertenece, las unidades que componen ese tipo de recurso y el instante de disponibilidad inicial y final del recurso en cuestión. Esto último hace referencia a los horarios de trabajo definidos para cada recurso, pudiendo variar la composición del SUH en función de la franja horaria en la que se encuentre el paciente. Por último, todo recurso que requiera de una persona física (consultas, enfermería y radiodiagnóstico) llevará asociado un descanso obligatorio.

Con relación a los pacientes, los parámetros que lo definen son el nivel de prioridad asistencial, como ya se ha comentado, el PU asociado al ESI correspondiente, el TEPCOF y el instante de llegada. El recurso de consulta que le sea asignado a cada paciente en la actividad de primera consulta será el mismo que el de la segunda consulta, respetando así la continuidad asistencial.

4 METODOLOGÍA DE RESOLUCIÓN

El capítulo 4 contiene la definición de los algoritmos que se aplican en el proyecto realizado para la optimización del SUH, explicando la forma en la que se han aplicado al problema en el que se centra el proyecto. También se explica el código creado en el *decoding*, el cual lleva a cabo la correcta asignación de las actividades a los recursos evitando incompatibilidades.

En cuanto al algoritmo de optimización, se aplica el *Iterated Greedy* [1], siendo ésta una de las metaheurísticas más comunes en los entornos *Job-Shop* [5]. La metaheurística utilizada se aplica de dos formas distintas en función del tipo de representación de la solución que se utilice para la optimización. Un tipo de representación de la solución es por pacientes, construyéndose en este caso la secuencia insertando todo el conjunto de actividades que pertenecen al paciente en cuestión de manera consecutiva. En cambio, en el tipo de representación de la solución por actividades, la creación de la secuencia se realiza insertando una a una las actividades de los distintos PUs en el SUH, respetando precedencias y sucesiones de cada PU. El objetivo es comparar ambos tipos de representación de la solución y ver cuál de ellas ofrece mejores resultados en función del tiempo de computación establecido para realizar el análisis computacional.

4.1 Decoding

En el presente proyecto, además de realizar la programación de los algoritmos utilizados y de la creación de los datos de entrada, se ha desarrollado el código que lleva a cabo la asignación de las distintas actividades que contiene la secuencia de entrada a los recursos correspondientes que configuran el SUH, para concluir con el cálculo de las funciones objetivo. Los resultados obtenidos a partir de esta función creada permiten, además, la representación gráfica mediante un diagrama de Gantt, lo cual facilita el seguimiento sobre cómo afectan los distintos tipos de representación a las soluciones obtenidas.

Los datos de entrada que requiere el *decoding* son el instante de disponibilidad inicial de cada recurso, la secuencia de entrada (traducida a actividades si el tipo de representación de la solución es por pacientes), el tipo de recurso del que requiere cada actividad, el número de actividades totales, la configuración de los recursos que componen el SUH, los tiempos de proceso de cada actividad, la matriz de precedencias indica qué actividades se pueden realizar de manera simultánea), los PU (completos y/o parciales) de todos los pacientes que están en el SUH, y los TEPCOF asociados a cada paciente según su ESI.

La función recorre una por una las actividades según el orden establecido en la secuencia de entrada. Se comprueba, de todos los recursos del tipo indicado por el dato de entrada, cuál tiene disponible antes un hueco en el que se pueda realizar completamente la actividad. Con esta búsqueda se sigue la regla del *as soon as possible (ASAP),* realizándose una búsqueda del recurso del tipo que tiene asignada la correspondiente actividad que quede libre antes. De esta manera se asegura la minimización del tiempo total de estancia del paciente en el SUH. Para realizar dicha búsqueda, se recorre el recurso a lo largo del horizonte temporal, almacenando aquellos huecos disponibles a partir del instante inicial en el que la actividad correspondiente puede ser realizada, siendo dicho instante igual al de finalización de la actividad precedente. Tras esto, se compara con

el hueco encontrado en los otros recursos del mismo tipo con el fin de asignar aquel recurso con la disponibilidad más temprana. En el caso de que la actividad requiera de dos unidades del recurso correspondiente, como sucede con el recurso de enfermería para los niveles ESI 1 y 2, es necesario realizar el recorrido de forma paralela en dos de los recursos del mismo tipo, de manera que la actividad sea asignada en el mismo intervalo de tiempo para la realización simultánea por parte de ambos recursos. Si la actividad corresponde a la primera consulta por parte del médico o del médico asistente, según su ESI, se almacena el recurso del tipo consulta al que ha sido asignada con el objetivo de respetar la continuidad asistencial en la actividad de segunda consulta médica previa al alta que vendrá a posteriori, siempre que sea posible.

Una vez asignada la actividad al recurso, se almacena su instante de finalización en una matriz con número de filas igual al número de recursos totales del SUH y número de columnas igual al número total de actividades. De esta manera, se obtiene qué actividad ha sido asignada a qué recurso y cuándo ha sido completada.

Una vez recorrida toda la secuencia de entrada y realizada la asignación a los recursos, se lleva a cabo el cálculo de las funciones objetivo y del nivel de saturación resultante en cada tipo de recurso. La primera función objetivo calculada es el *tardiness* respecto al cumplimiento del TEPCOF. Para llevar a cabo este cálculo, en primer lugar se comprueba si la primera actividad del proceso de urgencia del correspondiente paciente es la de primera consulta. Si no es así, no se realiza el cálculo de la FO para dicho paciente debido a que ya se encontraba en el SUH al tener un PU parcialmente completado. Si no es así, para cada paciente se recorre su PU almacenando el instante de finalización de su actividad más temprana y que sea distinto de cero. Esto se hace debido a que, en el proceso de destrucción de la secuencia en la aplicación del algoritmo que se verá más adelante, es posible que se haya eliminado su primera actividad, lo cual no significa que ya haya sido realizada, si no que no ha sido secuenciada aún. Una vez finalizado el recorrido del PU de cada paciente, se comprueba si el inicio de la actividad más temprana es anterior al instante impuesto por el TEPCOF. Si no es así, se calcula el correspondiente *tardiness* ponderado en función del ESI del paciente, dándole más peso a aquellos pacientes más prioritarios. La segunda función objetivo corresponde al tiempo de estancia total del paciente en el SUH. De nuevo se realiza el recorrido de las actividades del PU de cada paciente, aunque en este caso se almacena el instante de finalización de la actividad más tardía, dando lugar al tiempo total de estancia en el SUH de cada paciente. Por último, se calcula el tiempo que ha sido ocupado cada recurso y se calcula la saturación de cada tipo de recurso, determinada por dicha ocupación, por el número de recursos que hay para el tipo de recurso correspondiente y del LOS para el que se está realizando la optimización.

Con esto finaliza la descripción de todas las funciones que se llevan a cabo en el *decoding,* siendo por tanto la base que posibilita los análisis llevados a cabo en el presente proyecto.

4.2 Tipos de representación de la solución

4.2.1 Representación de la solución por pacientes

La construcción de la secuencia se realiza por pacientes en primer lugar, traduciéndola posteriormente a las actividades que componen los PU de cada paciente. Para asegurar el cumplimiento de los descansos de los recursos que lo requieren, se crean pacientes ficticios con

una única actividad en su PU, la cual requiere del tipo de recurso que debe descansar. Habrá tantos pacientes ficticios como recursos que necesitan del correspondiente descanso.

La secuencia marca el orden en el que se realiza la asignación de los pacientes (con sus correspondientes actividades) a los recursos, asignando por completo el PU de un paciente antes de pasar al siguiente paciente. Un ejemplo de los distintos PU que puede haber en función de la prioridad del paciente se muestra en la Tabla 2.

Paciente	Prioridad	PU
1	1	ENF-CON-LAB-RAY-CTSCAN-ENF-CON-ALTA
2	4	ENF-CON.A-RAY-ENF-CON-ALTA
3	5	ENF-CON.A-ALTA
4	Ficticio	CON
5	Ficticio	ENF

Tabla 2: Ejemplos de PUs para distintas prioridades

Suponiendo que todos los pacientes ya se encuentran en el SUH y no están siendo atendidos en el momento de lanzar la optimización, es decir, que su instante de llegada es cero, se realiza la asignación para una secuencia propuesta, como puede ser [2,4,1,5,3].

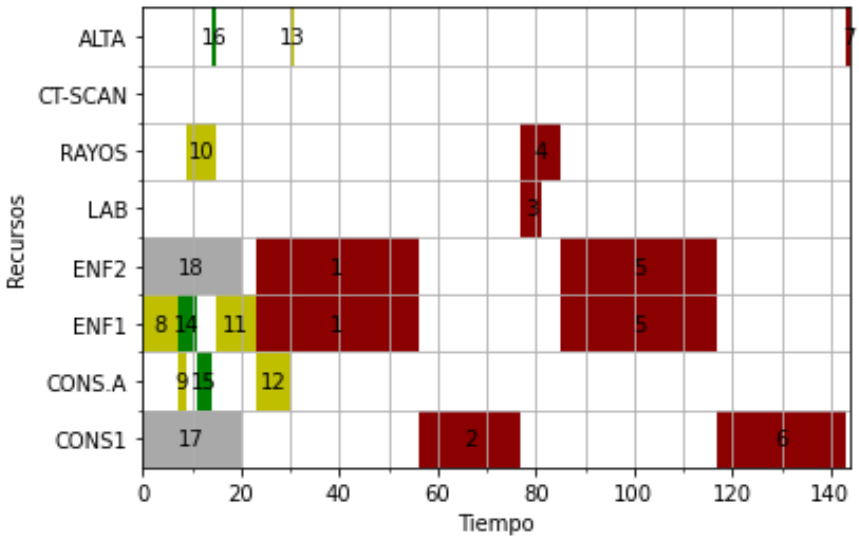

Figura 3: Diagrama de Gantt. Asignación de actividades a recursos según representación por pacientes

En este caso, los pacientes ficticios 4 y 5 con las actividades asignadas 17 y 18 respectivamente, corresponden a los descansos del único recurso de consulta y el segundo recurso del tipo enfermería, con una duración establecida de 20 minutos. Las actividades del tipo de recurso “alta” no tienen duración, ya que representan simplemente el instante en el que el paciente abandona el servicio de urgencias. En la Figura 3 se muestra cómo resultaría la asignación de las distintas

actividades según la representación anteriormente mencionada. Cabe destacar que los pacientes de ESI 1 y 2, al requerir de 2 enfermeros, han de esperar a que haya 2 recursos de este tipo disponible, buscando este instante lo antes posible.

En este diagrama se puede visualizar cómo se respeta la precedencia entre las actividades, no empezando cada actividad hasta que ha terminado la anterior, excepto en las actividades 4 y 3, que al ser de los tipos de recurso "laboratorio" y "rayos" respectivamente pueden ejecutarse de manera simultánea, siendo la más tardía de estas dos la que marque el comienzo de la actividad que las sucede. Por último, se observa la aplicación de la regla *ASAP* en la actividad 14, que es explicada más adelante. Esta actividad se asigna al primer recurso de enfermería al estar el segundo ya ocupado por el descanso, de manera que se realiza una búsqueda para detectar cuál de los dos se queda libre antes, siendo en este caso el primer recurso.

Los resultados obtenidos en este ejemplo sencillo serían:

- El *tardiness* ponderado respecto al TEPCOF es: 110
- El tiempo en el SUH es: 187
- La FO total es: 297

El único paciente que no es atendido antes del tiempo marcado por el TEPCOF asociado a su prioridad es el paciente 1, que es atendido por primera vez en el instante 23. El TEPCOF para esta prioridad es de 1 minuto, siendo el *tardiness* sería de 22 minutos que, multiplicándolo por un coeficiente asociado a su prioridad, da lugar a una FO de 110. El tiempo total de estancia es medido desde el instante en el que llegan (0 para todos los pacientes en este caso) hasta el instante en el que se ejecuta la actividad de tipo "alta". Con esto, el valor total de la FO es de 297, correspondiendo un 37% del valor al incumplimiento del TEPCOF. Con esto se cumple el objetivo por el cual se han añadido los coeficientes de peso al cálculo del *tardiness* en función de las prioridades, dar mayor importancia al incumplimiento de los pacientes prioritarios penalizando de manera significativa la FO total.

4.2.2 Representación de la solución por actividades

Si la construcción de la secuencia se realiza por actividades el proceso cambia. Se asigna un orden secuenciando actividad por actividad, siendo una de las restricciones el mantenimiento de las precedencias y de las sucesiones entre las distintas actividades del PU de un mismo paciente, lo cual hace más compleja la construcción de la secuencia y complica la obtención de mejores soluciones al llevarse a cabo iteraciones que no obtienen mejoras. Por ejemplo, nunca puede secuenciarse una actividad de alta antes que una de primera consulta de un mismo paciente. La diferencia en el resultado final respecto a la representación por pacientes radica en que pueden mezclarse actividades de distintos pacientes en la asignación a los recursos, lo que da mayor libertad a la hora de encontrar mejores valores de las FO.

Siguiendo el ejemplo del apartado anterior, el PU de cada paciente resultaría en el resultado que se muestra en la Tabla 3.

Paciente	Actividades						
1	1	2	3	4	5	6	7
2	8	9	10	11	12	13	
3	14	15	16				
4	17						
5	18						

Tabla 3: Actividades del proceso de urgencia

Un ejemplo de representación por actividades podría ser [1, 8, 9, 14, 2, 3 ,4 ,5, 6, 7, 10, 11, 17, 12, 13, 15, 16, 18]. Al mover la actividad 1 al principio de la secuenciación, se cumple el TEPCOF asignado al paciente 1. El resto de las actividades se mantienen excepto la 14, que también se adelanta. Esta secuenciación da como resultado el diagrama de Gantt de la Figura 4. Además, se obtienen los siguientes resultados:

El *tardiness* respecto al TEPCOF es: 0.

El tiempo en el SUH es: 223.

La FO total es: 223.

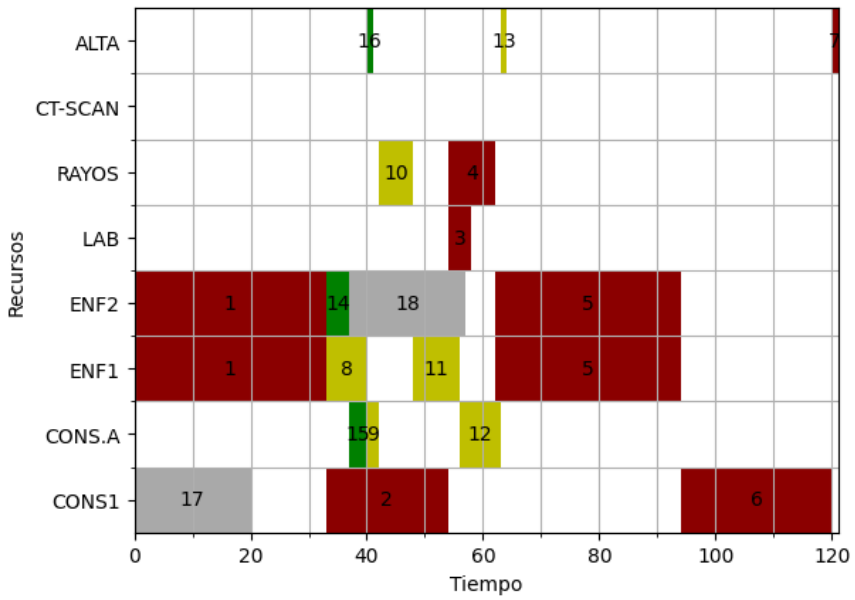

Figura 4: Diagrama de Gantt. Representación por actividades

Al asignar antes la actividad 1, el paciente 1 ve cumplido su TEPCOF al ser atendido en el instante 0, lo que mejora la FO1 al respetarse los TEPCOF de todos los pacientes. El SUH aumenta al retrasar la actividad 1 el resto de los procesos, sin embargo, el valor total de la FO es mejor debido a la influencia de la FO1.

4.3 Algoritmos de optimización

A continuación, se exponen los algoritmos usados para la optimización del problema propuesto. El algoritmo principal es el *Iterated Greedy* propuesto por [1]. El objetivo es obtener cuál de estos algoritmos ofrece un mejor resultado de las funciones objetivo en el mismo tiempo de computación, diferenciando a su vez la construcción de la secuencia en los dos tipos vistos en el apartado anterior: por pacientes y por actividades. La comparación final se realiza mediante un análisis computacional y el uso del indicador ARPD (*average relative percentage deviation*).

4.3.1 Iterated Greedy

El algoritmo *Iterated Greedy* (IG) genera distintas soluciones a base de la ejecución de varias iteraciones, en las que realiza, principalmente, dos procesos a la secuencia: destrucción de la secuencia original o propuesta y construcción de una nueva secuencia candidata para sustituir a la anterior, siempre que cumpla los requisitos impuestos. Estos requisitos vienen definidos por el criterio de aceptación, el cual suele ser si la función objetivo de la nueva secuencia candidata es mejor que la de la secuencia con mejor valor de la función objetivo, produciéndose en este caso la sustitución de la secuencia.

Para comenzar a aplicar el algoritmo, es necesario disponer de una primera solución sobre la que realizar las iteraciones. Mientras mejor sea esta secuencia inicial, mejores resultados aportará el algoritmo. Siguiendo la propuesta de [1], la solución inicial se construye utilizando la heurística NEH. Esta heurística es muy usada en entornos *Flowshop*, para la secuenciación de las actividades en las distintas máquinas [29].

Algorithm 2 NEH algorithm

Input:A K machines flow shop and a set $J = J_1, \ldots, J_n$ and each job j have the processing time $p_{j1}, \ldots, p_{jK}$ in K machines.
Step 1: Calculate the total processing time of each job by: $T(j) = \sum_{k=1}^{K} p_{jk}$
Construct the sequence $\pi = (\pi_1, \ldots, \pi_n)$ by sorting the jobs in descending order of $T(j)$
Step 2: Schedule the first two jobs π_1,π_2 and choose the best sequence π of the first two jobs
Step 3: Schedule other jobs
for $j = 3$ to n **do**
Test π_j in the different positions of the current sequence built and choose the best sequence with the minimum TT in all machines and update π_{NEH} of the optimal sequence.
end for
Output: π_{NEH}

Figura 5: Pasos fundamentales de la heurística NEH [29].

Los pasos que definen la heurística NEH son:

1. Para cada trabajo, que en el caso del presente proyecto corresponde a cada paciente, se calcula su tiempo total de proceso.

2. Se ordenan los pacientes en orden descendente de tiempo total de proceso. Se empieza a crear la secuencia insertando el primer paciente. Para el siguiente paciente, se introduce en

todas las posiciones posibles de la secuencia, calculando el valor de la función objetivo y eligiendo la posición en la que da un mejor resultado.

3. Se repite el proceso para los trabajos (pacientes) restantes.

Una vez aplicada la heurística NEH, se obtiene la solución de entrada para el IG. Esta heurística también se usa en la fase de construcción del algoritmo.

Tras la creación de la solución inicial, tiene lugar el proceso de destrucción. Para llevar a cabo la destrucción, se escogen *d* trabajos de forma aleatoria y sin repetición, los cuales son eliminados de la secuencia. De esta manera, se crean dos subsecuencias: la secuencia en la que se almacenan las actividades (o pacientes) extraídas (π_d) y la secuencia resultante de dicha eliminación (π_R).

En la fase de construcción, las actividades eliminadas (π_d) se insertan en π_R en el orden en el que fueron extraídas, una a una. En dicha inserción, se vuelve a aplicar el paso 2 de la heurística NEH, es decir, se inserta cada actividad (o paciente) en todas las posiciones posibles de π_R, eligiendo la mejor posición en función de la FO. Se reinsertan todas las actividades hasta que la subsecuencia π_d quede vacía. Tras esto, se obtiene una secuencia propuesta con su respectivo valor de la función objetivo, finalizando la ejecución del algoritmo denominado IG básico. Este proceso de destrucción y construcción se repite hasta que se cumpla un criterio de parada, que en este caso será fijado por un límite de tiempo de computación.

4.3.1.1 Búsqueda local

Al final de la etapa de construcción se puede añadir un procedimiento de búsqueda local, con el objetivo de mejorar las soluciones obtenidas en la etapa de construcción mediante la implementación de este algoritmo adicional (proceso de intensificación). Existen muchas alternativas en cuanto a las metaheurísticas que realizan este tipo de intensificación, siendo la utilizada en el presente proyecto la siguiente:

- **Búsqueda local iterada (ILS):** esta alternativa, la seleccionada para el presente proyecto, es más sencilla que la anterior, siendo también muy efectiva. La idea que define el ILS se basa en enfocar la búsqueda en un pequeño subconjunto de soluciones definido por aquellas que son óptimos locales [30].

  Partiendo de una solución inicial, procedente de la etapa de construcción del IG, se selecciona una actividad aleatoria y se realiza la búsqueda en las n posiciones vecinas. De nuevo, se almacena la secuencia con mejor solución y se sustituye por la inicial si presenta mejor resultado. Existen tres modos de llevar a cabo la búsqueda, en función del tipo de vecindad definido: intercambio adyacente, intercambio e inserción. Esta última es la usada en el presente proyecto.

- **Inserción**: se extrae una actividad de la secuencia inicial de N actividades y se reinserta en todas las posibles posiciones de la secuencia. En el caso de este proyecto, se define un tamaño de vecindad sobre el que realizar la inserción con el objetivo de mejorar el tiempo de computación requerido para hallar un óptimo local.

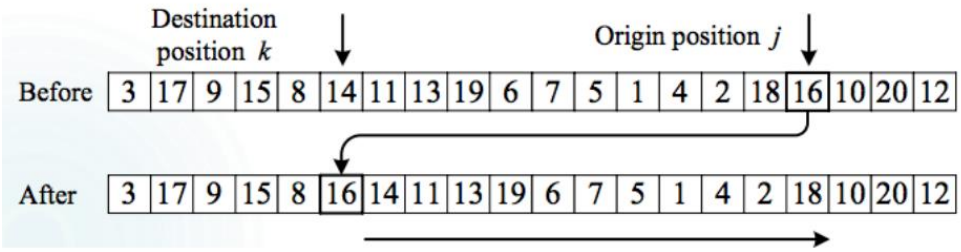

Figura 6: Inserción [30]

Para aumentar la diversificación y evitar caer en óptimos locales, se puede añadir un criterio de aceptación de la secuencia candidata para seleccionarla de cara a la nueva iteración, aunque no aporte un mejor valor de la FO.

```
procedure IterativeImprovement_Insertion (π)
    improve := true;
    while (improve = true) do
        improve := false;
        for i := 1 to n do
            remove a job k at random from π (without repetition)
            π' := best permutation obtained by inserting k in any possible positions of π;
            if C_max(π') < C_max(π) then
                π := π';
                improve := true;
            endif
        endfor
    endwhile
    return π
end
```

Figura 7: Búsqueda local iterada [1].

4.3.1.2 Recocido simulado

El recocido simulado es un método de diversificación que es usado usualmente como criterio de aceptación en los algoritmos voraces iterativos para evitar su estancamiento. Usar como criterio de aceptación únicamente el valor de la FO puede provocar que las iteraciones no sean útiles al no realizar las permutaciones adecuadas entre las distintas actividades de la secuencia, produciéndose así el estancamiento en un óptimo local. Para evitar esto, se añade un segundo criterio de aceptación que, si se cumple, activa la sustitución de la secuencia original por la candidata para variar el orden de la secuencia de las actividades, originando así nuevas permutaciones que puede aportar nuevos valores de la FO. Finalmente, si la mejor secuencia obtenida en las iteraciones sigue sin mejorar a la inicial, se adopta como óptima esta última.

El criterio de aceptación se basa en el coeficiente de temperatura, el cual se halla de la siguiente

manera:

$$Temperatura = T \cdot \frac{\sum_{j=1}^{m}\sum_{j=1}^{n} p_{ij}}{n \cdot m \cdot 10}$$

T es un parámetro que debe ser ajustado. *N* y *M* representan el número total de trabajos (pacientes o actividades) y el número de recursos disponibles respectivamente. P_{ij} es el tiempo de proceso del trabajo i en la máquina j. El algoritmo IG completo se muestra en la Figura 8.

En función del tipo de representación de la solución, es decir, si se trata de representación por pacientes o por actividades del PU, las actividades sobre las que se aplica el algoritmo corresponderán a los propios pacientes, realizándose la destrucción y construcción del paquete completo de actividades del PU, o a las actividades del propio PU.

La inclusión o no de la búsqueda local y del recocido simulado dará igual a un total de 4 algoritmos a estudiar, añadiendo además los dos tipos de representación:

- IG básico.
- IG con búsqueda local.
- IG con *Simulated Annealing.*
- IG con búsqueda local y S.A.

```
procedure IteratedGreedy_for_PFSP
    π := NEH_heuristic;
    π := IterativeImprovement_Insertion(π);
    π_b := π;
    while (termination criterion not satisfied) do
        π' := π;                                  % Destruction phase
        for i := 1 to d do
            π' := remove one job at random from π' and insert it in π'_R;
        endfor
        for i := 1 to d do                        % Construction phase
            π' := best permutation obtained by inserting job π_R(i) in all possible positions of π';
        endfor
        π'' := IterativeImprovement_Insertion(π');   % Local Search
        if C_max(π'') < C_max(π) then             % Acceptance Criterion
            π := π'';
            if C_max(π) < C_max(π_b) then         % check if new best permutation
                π_b := π;
            endif
        elseif (random ≤ exp{−(C_max(π'') − C_max(π))/Temperature}) then
            π := π'';
        endif
    endwhile
    return π_b
end
```

Figura 8: Algoritmo Iterated Greedy con búsqueda local y recocido simulado [1].

5 ANÁLISIS COMPUTACIONAL

A continuación, se realiza una breve presentación de las herramientas usadas para llevar a cabo el análisis de diferentes escenarios de gestión. El programa usado es Spyder, el cual es un entorno de desarrollo interactivo para el lenguaje Python. Se hace uso de las librerías que ofrece el entorno Python las cuales amplían en gran medida su funcionalidad.

Además, se explican los parámetros que definen los algoritmos utilizados, así como el valor tomado para realizar el análisis y los datos de entrada para realizar la optimización, con los procedimientos usados para su cálculo.

5.1 Herramienta utilizada

El desarrollo de todo el código referente a la optimización se ha realizado en el programa Spyder, el cual usa lenguaje Python. La ventaja que presenta este lenguaje radica en la gran variedad de librerías que se pueden importar al programa, estando implementadas perfectamente para poder ser utilizadas de manera simultánea.

El código creado para la optimización del problema planteado no se explica de forma detallada, sin embargo, se hace una breve explicación de la estructura que siguen los distintos códigos y de las funciones que cada uno realiza.

5.1.1 Librerías

En primer lugar, se recogen las librerías que se han utilizado en la optimización y se explican brevemente:

- **Matplotlib**: esta librería se usa para la representación de gráficas en 2D y 3D. Es de código abierto, lo que significa que se sigue optimizando su funcionamiento mediante la intervención de los distintos usuarios [31]. En el presente proyecto se ha utilizado para la representación de los diagramas de Gantt.

- **Numpy**: es una librería dedicada al cálculo numérico y al análisis de datos, sobre todo cuando es necesario el manejo de un gran volumen de datos. Esta librería agrega un mayor soporte para vectores y matrices, conteniendo además multitud de funciones matemáticas para operar con estos tipos de datos.[32]

- **Pandas**: es otra librería diseñada para el almacenamiento, manejo y manipulación de datos en Python. La clave de esta librería es que introduce dos objetos al programa, las series y los *Dataframes.* Este último es el que hace que esta librería sea útil en la optimización, ya que se puede entender como un conjunto de series del mismo tamaño, con índices comunes y que pueden ser almacenados juntos en un objeto tabular [33]. Este objeto es exportado a Excel para el almacenamiento de los resultados de las distintas instancias generadas.

5.1.2 Jerarquía del código

En la realización del código existen varios archivos .py, los cuales realizan las distintas funciones

requeridas para la creación de los datos del problema, la representación gráfica y la asignación a los recursos a lo largo del horizonte temporal, además de contener el código de los distintos algoritmos para la optimización en la secuenciación. A continuación, se mencionan las funciones creadas acompañadas de una breve explicación de la acción que desempeñan:

- **Asignación.py**: contiene el código referente al *decoding* explicado en apartados anteriores. Al finalizar la asignación de todos los recursos de la secuencia de entrada, realiza el cálculo de las dos funciones objetivos: incumplimiento del TEPCOF y tiempo de estancia total en el SUH.

- **Algoritmos.py**: crea las funciones referentes a la implementación de los distintos algoritmos a ejecutar. En total, contiene 9 algoritmos: el NEH para la creación de la solución inicial, el algoritmo IG básico, el algoritmo IG con búsqueda local, el algoritmo IG con recocido simulado y un último algoritmo juntando ambas variantes. Estos 4 últimos algoritmos se crean para los dos tipos de representación de la solución al haber diferencias en su código.

- **Búsqueda_local.py** y **sim_annealing.py**: contienen ambos algoritmos de intensificación.

- **Principal.py**: contiene el código referente a la creación de todos los datos de entrada: número de pacientes, prioridades de los pacientes, PU, tiempos de proceso, etc. Importa las funciones anteriores para su ejecución y registra los resultados de los distintos algoritmos, importándolos al final de la ejecución a un archivo Excel.

Además, se crea una función en **crear_gantt.py** para la representación del correspondiente diagrama principal resultado de la asignación de las actividades a los recursos según la secuencia dada.

5.2 Datos de entrada a la optimización

Para llevar a cabo la recreación de la jornada en un SUH para su posterior optimización, es necesario hacer una serie de suposiciones que faciliten la creación de los pacientes que entran al SUH junto con su prioridad, PU completo o parcialmente realizado, duración de las actividades de los distintos pacientes y recursos de los que dispone el hospital. Para ello, se seguirán los métodos aconsejados por Leonardo Bedoya-Valencia en su artículo *Evaluating alternative resource allocation in an emergency department using discrete event simulation* [2].

5.2.1 Generación de pacientes

Para la optimización del nivel operativo, en el presente proyecto, la generación de pacientes se limita a se ha seguido un método basado en la saturación de los distintos tipos de recurso que componen el SUH. Se estudian 3 escenarios: 50%, 75% y 100% de saturación máxima. De esta manera, para estos valores de saturación, se generan iterativamente pacientes, de uno en uno, hasta que la saturación de alguno de los recursos llegue al valor establecido. Para determinar la saturación tras generar un paciente es necesario conocer, en primer lugar, la prioridad de cada paciente para asignarle su PU, junto con los recursos que necesitan las actividades de dicho PU y las distintas duraciones de las actividades. Por último, se establece el horizonte temporal para el

que se va a mantener el nivel de saturación considerado. Este horizonte, que no afecta a la disponibilidad de los recursos, se establece igual al LOS medio extraído de [2], donde se calcula que el tiempo medio es de 140 minutos.

El realizar la optimización para un instante dado permite considerar el conjunto de pacientes conocido (problema estático), estando este conjunto compuesto por nuevos pacientes que llegan al SUH y los que ya se encuentran en él y que han podido ser atendidos parcialmente. Al suponer que todos los pacientes ya se encuentran en el SUH, su instante de llegada es cero. No obstante, para aquellos pacientes que están siendo atendidos en el instante en el que comienza la optimización y tienen parte de su PU realizado, se asigna un instante de llegada distinto de cero para garantizar que no avanzan en su PU hasta que la actividad que se les estaba realizando en el instante de inicio de la optimización ha sido finalizada.

Para asignar las distintas prioridades a cada paciente generado, el proyecto se basa en la proporción con la que cada ESI se asigna a los distintos pacientes en [2], ver Figura 9.

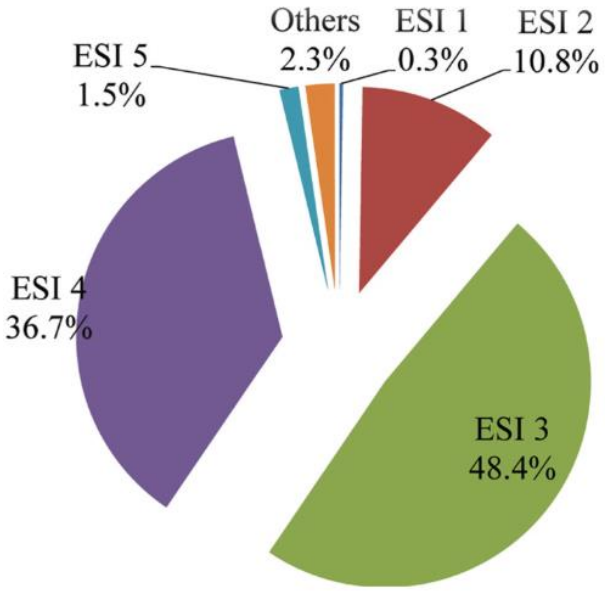

Figura 9: Porcentaje de pacientes según su ESI

Una vez asignado el PU correspondiente, se elige de manera aleatoria cuántas de esas actividades ya han sido realizadas debido a que se supone que el paciente puede estar o no en el SUH desde antes del instante de inicio de la optimización, pudiendo haber sido ya completadas algunas de sus actividades.

5.2.2 Generación de tiempos de proceso

La duración de las distintas actividades que componen un PU depende de múltiples factores, como se ha explicado en apartados anteriores. Uno de ellos es la prioridad asistencial asociada al paciente tratado ya que, en principio, un paciente de ESI 1 puede requerir más tiempo de consulta que un paciente de ESI 4. A su vez, para una misma prioridad, la duración de una misma actividad puede no tener la misma duración en los distintos pacientes. Esto demuestra el carácter estocástico que define el entorno del SUH.

Table 2. Patient processing times at the different stages based on Emergency Severity Index (ESI) levels.

Process	Type of patient				
	ESI 1	ESI 2	ESI 3	ESI 4	ESI 5
Sign-in	Triangular (3, 5, 10)	Triangular (3, 5, 10)	Triangular (3, 5, 10)	Triangular (3, 5, 10)	Triangular (3, 5, 10)
Triage	Triangular (0.5, 1, 1.5)	Triangular (2, 3, 5)	Triangular (5, 7, 10)	Triangular (5, 7, 10)	Triangular (5, 7, 10)
Registration	Triangular (3, 8, 12)	Triangular (3, 8, 12)	Triangular (3, 8, 12)	Triangular (3, 8, 12)	Triangular (3, 8, 12)
First visit (nurse)	Triangular (20, 30, 75)	Triangular (18, 28, 45)	Triangular (15, 22, 30)	Triangular (5, 10, 20)	Triangular (2, 4, 8)
First visit (P/PA)	Triangular (10, 20, 25)	Triangular (10, 20, 25)	Triangular (5, 15, 20)	Triangular (2, 3, 5)	Triangular (2, 3, 5)
Second visit (nurse)	Triangular (20, 30, 40)	Triangular (15, 20, 30)	Triangular (10, 15, 20)	Triangular (5, 8, 12)	-
Second visit (P/PA)	Triangular (20, 30, 40)	Triangular (15, 20, 25)	Triangular (8, 15, 18)	Triangular (5, 10, 15)	-
X-Ray	Triangular (3, 5, 15)	Triangular (5, 10, 20)	Triangular (5, 10, 20)	Triangular (2, 5, 10)	-
Blood	Triangular (3, 5, 15)	Triangular (3, 5, 15)	Triangular (3, 5, 15)	-	-
CT-Scan	Constant 20	Constant 15	Constant 15	-	-

P: physician; PA: physician assistant.

Figura 10: Tiempos de proceso en los distintos recursos según el ESI del paciente [2]

Para llevar a cabo el análisis computacional se aproximan las duraciones de las actividades según distribuciones triangulares como en [2]. Estas distribuciones tienen distintos parámetros de entrada en función de la actividad y de la prioridad asistencial, simulando así la aleatoriedad de un entorno real del servicio de urgencias. Los valores de estos parámetros se muestran en la Figura 10.

5.2.3 Recursos del servicio de urgencias

Debido a que ya puede haber pacientes que están siendo atendidos en el instante en el que se lanza la optimización, algunos de los recursos también pueden estar ocupados en el instante inicial por estos pacientes que ya se encuentran en el SUH. Para incluir esta no disponibilidad de algunos de los recursos se establece el instante de disponibilidad inicial como el instante en el que termina la actividad ficticia equivalente a la de un paciente que está siendo atendido en ese momento por el recurso en cuestión. La disponibilidad final se usará únicamente para el cálculo de la saturación del recurso, no limitando la asignación de las actividades más allá de ese instante.

Para llevar a cabo la optimización de la gestión de un SUH es necesario definir la configuración del propio servicio de urgencias. Esta composición viene definida por el número de recursos de los que consta cada tipo de recurso propio del SUH: médicos, médicos asistentes, enfermería, etc.

Esta composición del SUH varía en función de la franja horaria que se está analizando. Para el proyecto, sin pérdida de generalidad, la optimización se realiza en dos instantes de tiempo para analizar qué algoritmos responden mejor para una composición u otra (Tabla 4).

Tipos de recurso	**12:00 h**	**6:00 h**
Consulta médica	2	1
Consulta médico asistente	2	0
Enfermería	9	6
Laboratorio	1 (capacidad infinita)	1 (capacidad infinita)
Rayos	1	1
CT-Scan	1	1

Tabla 4: Cantidad de cada tipo de recurso según la franja horaria

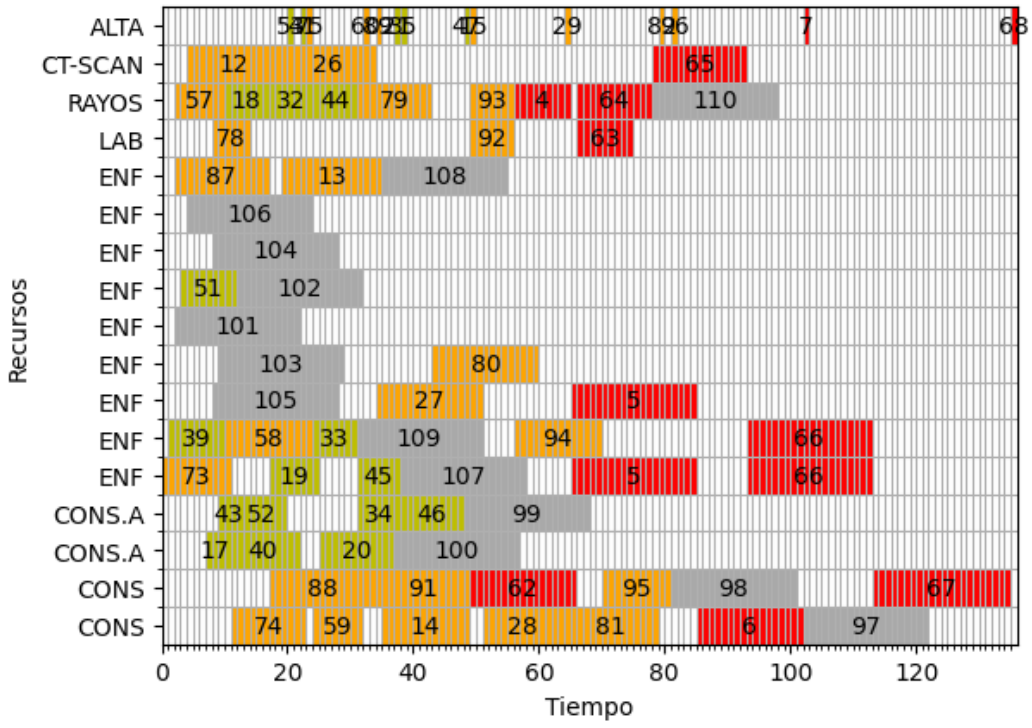

Figura 11: Diagrama de Gantt para saturación del 50%

En la Figura 11 se muestra el resultado de la representación por pacientes dada por el algoritmo NEH. Se han generado pacientes uno a uno hasta alcanzar la saturación del 50% en uno de los tipos de recurso de los que dispone el SUH. En este caso, esta saturación se alcanza en el tipo de recurso de Consulta al haberse generado un mayor número de pacientes de prioridades 2 y 3 que requieren de este tipo de recurso. Se puede observar que la primera actividad de algunos PU de los pacientes no corresponde a la primera atención por enfermería, la primera actividad del PU "teórico" de todo paciente. Por ejemplo, para el paciente 1 la primera actividad que se realiza es la 4, correspondiente al radiodiagnóstico. Esto es debido a que, como se ha mencionado anteriormente, en la optimización se generan pacientes cuyo PU ya ha sido completado parcialmente al estar desde antes del instante inicial en el SUH. En el caso del paciente 1, ya se habrían realizado las actividades 1, 2 y 3, correspondientes a la primera revisión por enfermería, la primera consulta y el análisis de sangre en el laboratorio. Como es lógico, el incumplimiento del TEPCOF no se calcula para aquellos pacientes cuya primera actividad ya ha sido realizada.

Cabe destacar el número de pacientes ficticios generados para asignar los descansos a los distintos recursos. Estas actividades de descanso son las de color gris, asignadas a todos los recursos humanos (es decir del tipo consulta, consulta de médico asistente, enfermería y rayos).

5.3 Calibración de los parámetros de los algoritmos

Para realizar la calibración se han generado 10 instancias para cada nivel de saturación considerado: 50%, 75% y 100%. La estructura del SUH en cuanto a los recursos disponibles es la correspondiente a la de las 12 horas, según lo expuesto en la Tabla 4. La comparación de resultados se realiza mediante un RPD (*Relative Percentage Deviation)* [1]:

$$RPD = \frac{Some_{sol} - Best_{sol}}{Best_{sol}} \cdot 100$$

Donde $Some_{sol}$ es a la solución obtenida por el algoritmo correspondiente para una instancia dada y $Best_{sol}$ es la mejor solución obtenida para dicha instancia con cualquier algoritmo aplicado. Tras esto se realiza el promedio del resultado obtenido por cada algoritmo (ARPD) y se compara con los demás algoritmos. El ARPD indica en tanto por ciento la desviación del resultado obtenido por el correspondiente algoritmo respecto a la mejor solución encontrada. Por tanto, el algoritmo que presente un menor valor ARPD es el que mejores resultados encuentra al problema planteado.

5.3.1 Tiempo de computación

El primer parámetro que hay que introducir en el algoritmo es el tiempo de computación. Este parámetro define cuánto tiempo se lleva el algoritmo realizando las distintas iteraciones de destrucción y construcción, arrastrando sucesivamente la secuencia que aportó una mejor solución en la iteración anterior. El valor del parámetro se establece siguiendo la regla propuesta por [1], que propone la siguiente fórmula para su cálculo:

$$tiempo = \frac{n \cdot m}{2} \cdot 60\, milisegundos$$

Siendo m el número de recursos disponibles y n el total de actividades a asignar a los distintos recursos, independientemente de si el tipo de representación es por pacientes o por actividades. En lugar de establecer la relación para 60 milisegundos, se amplía a 375 milisegundos para realizar una comparación más equilibrada con la representación por actividades, al requerir esta última de un mayor tiempo de computación para obtener mejores soluciones.

5.3.2 Parámetro de destrucción (*d*)

Dentro de la ejecución del algoritmo, el primer parámetro a fijar es *d*, el cual especifica el número de trabajos (pacientes o actividades, según el tipo de representación de la solución) a extraer de la secuencia inicial en la fase de destrucción. El valor se fija en *d*=4 para la representación por pacientes, siguiendo los resultados obtenidos por [1], que mejoran al fijar este valor.

En la representación de la solución por actividades se toma un valor de *d* distinto. Este tipo de representación se ve muy limitado por las precedencias, de manera que, a la hora de hacer la construcción, la posición en la que se reinserta cada actividad debe estar entre la actividad que la precede y la actividad sucesora. Por esto, se realizan varias ejecuciones para calibrar el valor de *d* eligiendo entre 10, 15, 20, 25 y *J*/2. Con esto se asegura que, como mínimo, se van a eliminar todas las actividades equivalentes a un PU para permitir una mayor libertad en la fase de construcción. Siguiendo el ejemplo del apartado 4.2.2, y con la secuencia propuesta [1, 8, 9, 14, 2, 3 ,4 ,5, 6, 7, 10, 11, 17, 12, 13, 15, 16, 18].

Paciente	Actividades						
1	1	2	3	4	5	6	7
2	8	9	10	11	12	13	
3	14	15	16				
4	17						
5	18						

Tabla 5: Ejemplo de proceso de urgencias

Si, por ejemplo, se extrajera la actividad 2, podría reinsertarse a partir de la posición 2 del vector, al estar su actividad precedente 1 secuenciada en la posición 1. Iterativamente se reinserta la actividad hasta la posición 5, la anterior a la de su actividad sucesora, la 3. Sin embargo, si se extrajera la actividad 4, no podría reinsertarse en otra posición que no sea la 7, al estar limitada por la actividad 3 y 5, secuenciadas justo antes y después.

5.3.3 Tamaño de la vecindad en la búsqueda local

En la aplicación del algoritmo con búsqueda local es necesario fijar el tamaño de la vecindad. Este tamaño define el número de posiciones vecinas, contando desde su posición original, con las que la actividad correspondiente realiza la inserción. Es decir, manteniendo como pivote la posición original de la actividad, en cuántas posiciones hacia adelante y hacia atrás se reinserta el trabajo correspondiente. Por la misma razón que se cambia el valor de *d* según el tipo de representación, en el algoritmo con búsqueda local también es necesario cambiar el tamaño de la vecindad. De nuevo se realizan varias ejecuciones para calibrar dicho parámetro, estableciendo los valores de 10, 15, 20 y de forma aleatoria entre 10 y *J*/2, siendo *J* el número total de actividades que hay que asignar a los distintos recursos del SUH.

5.3.4 Temperatura (*T*)

Al finalizar la construcción tiene lugar la aplicación del recocido simulado para asignar o no la secuencia candidata como nueva secuencia inicial, a pesar de que el resultado de su FO no sea mejor que el de la secuencia inicial anterior. Con la fórmula de la temperatura mostrada en el apartado 4.3.1.2, se introdujo el parámetro de temperatura inicial *T*. El resto de los factores que intervienen en el cálculo de la temperatura sirven para escalarla al tamaño del problema. El valor de *T* es 0.4, basado de nuevo la experimentación realizada por [1].

5.3.5 Resultados de la calibración

Como se ha mencionado, únicamente se calibran parámetros para la representación por actividades debido a las restricciones que presenta la propia secuenciación y al mayor número de trabajos que se introducen en el algoritmo en comparación con la representación por pacientes, donde dichos trabajos corresponden a los pacientes del SUH, siendo la dimensión de la secuencia menor que por actividades.

En primer lugar, para la calibración del parámetro de destrucción *d* únicamente se aplica la NEH para obtener la secuencia inicial y el algoritmo IG básico. Se calcula el ARPD entre los valores

propuestos del parámetro *d*, eligiendo aquel que presenta un ARPD menor, indicando que obtiene los mejores resultados. Los resultados obtenidos indican que el mejor valor del parámetro es para una destrucción de 10 actividades. En la Tabla 6 se muestran los resultados obtenidos en la calibración. Se observa que, a medida que aumenta el número de actividades destruidas, el ARPD obtenido aumenta, lo que significa que la distancia de las soluciones obtenidas para dicho valor del parámetro respecto a la mejor solución global obtenida es mayor. La principal explicación de este comportamiento es debido al menor número de iteraciones que se realizan, siendo en promedio de 72 para *d*=10 y de 37 para *d*=25.

Parámetro *d*	**IG básico**
10	**0,78%**
15	0,96%
20	1,86%
25	2,27%
J/2	2,20%

Tabla 6: Resultados de la calibración de *d*

El mismo procedimiento se sigue para la calibración del tamaño de la vecindad en el algoritmo IG con búsqueda local. En este caso, se ejecutan los algoritmos NEH y IG con búsqueda local, obteniendo de nuevo los valores del ARPD para las distintas instancias. Los resultados muestran que el mejor valor es el de 15 actividades.

Tamaño de la vecindad	**IG búsqueda local**
10	1,05%
15	**0,90%**
20	1,06%
Aleatorio {10, *J*/2}	2,57%

Tabla 7: Resultados de la calibración del tamaño de la vecindad

En este caso, los resultados obtenidos al aumentar el tamaño de la vecindad no siguen el mismo comportamiento que en el caso anterior. Existe un equilibrio entre un menor número de iteraciones y una mayor exploración de la vecindad.

6 RESULTADOS

Tras la calibración de los parámetros de los distintos algoritmos, se procede a la a la resolución de los escenarios considerados anteriormente del problema. Los escenarios de gestión analizados consideran dos composiciones de los recursos del SUH diferentes en función de la franja horaria en la que se centra el análisis: a las 12 y a las 6 horas. Para ambas composiciones, se generan 10 instancias para cada nivel de saturación (50%, 75% y 100%). Se aplican los 8 algoritmos creados, siendo 4 con representación por pacientes y los mismos con representación por actividades. Además, se aplica la NEH para partir de una buena solución inicial. Los 4 algoritmos que se estudian para cada tipo de representación son:

- IG básico.
- IG con búsqueda local.
- IG con *Simulated Annealing.*
- IG con búsqueda local y S.A.

6.1 Recursos de las 12:00 h.

En primer lugar, se realiza el análisis computacional para la composición del SUH correspondiente a la franja horaria de las 12 horas, contando de un total de 16 recursos compuestos por: 2 médicos, 2 médicos asistentes, 9 enfermeros, 1 laboratorio, 1 radiodiagnóstico y 1 CT- Scan. El tiempo de computación, establecido según lo expuesto en el apartado 5.3.1, es de aproximadamente 3 minutos por algoritmo para cada instancia para el primer nivel de saturación, de 5 minutos para el 75% de saturación de recursos y de 7 para el 100% de saturación. Las dimensiones promedio del problema para cada saturación, manteniendo fijos los 16 recursos totales, es de 57 actividades para el 50% de saturación, 75 actividades para el 75% y 94 para el 100%.

La primera comparación de los distintos algoritmos se realiza en función del número de mejores soluciones encontradas. En la Figura 12 se puede observar que, para el tiempo de computación considerado, la representación por pacientes supera a la de por actividades por un amplio margen en cuanto a número de mejores soluciones encontradas se refiere. En los algoritmos que consideran la representación por pacientes, el que mejores resultados obtiene es el *Iterated Greedy* con *Simulated Annealing,* seguido del IG con ambas intensificaciones. Resulta llamativo que el IG con ambas intensificaciones, búsqueda local y S.A., encuentra un menor número de mejores resultados que el que únicamente introduce el S.A. como intensificación. Esto es debido al mayor tiempo de computación que requiere la búsqueda local, lo que hace que se lleven a cabo un menor número de iteraciones que en el caso del algoritmo IG con la intensificación del S.A. únicamente. Con ello, se puede extraer la conclusión de que los algoritmos que consideran la representación por pacientes funcionan mejor para un mayor número de iteraciones realizadas que para una mayor dimensión de destrucción/exploración de pacientes.

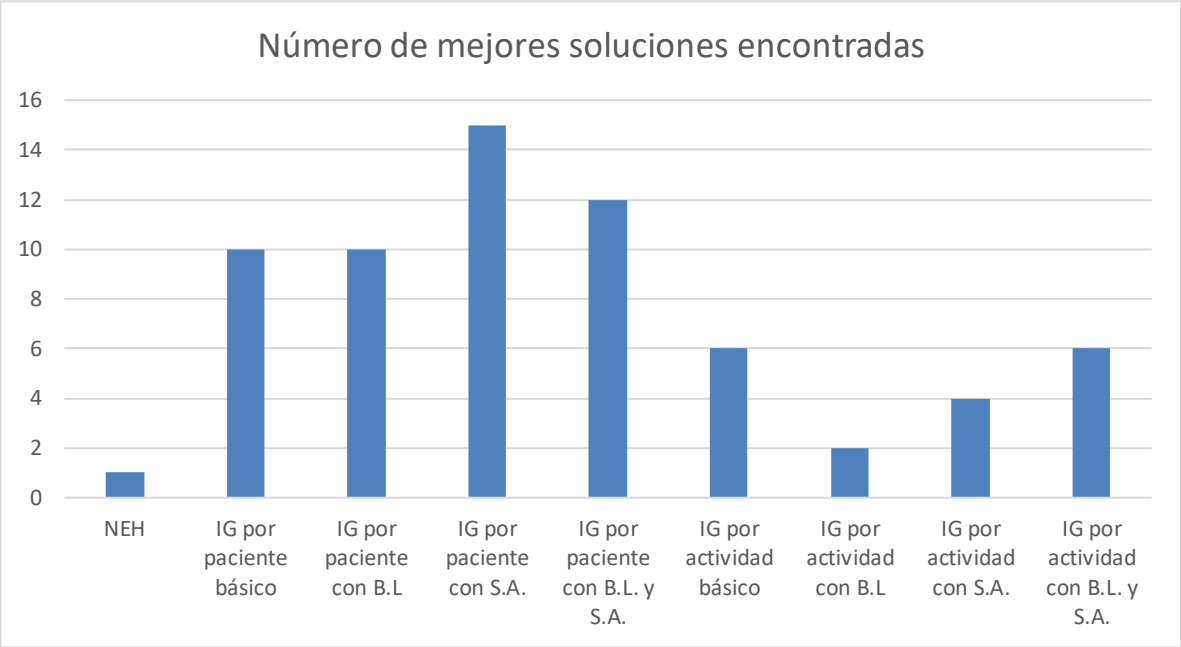

Figura 12: Número de mejores soluciones encontradas por algoritmo. Recursos de las 12 h.

En cuanto a la representación de la solución por actividades, el mayor número de mejores resultados para cada instancia encontrados lo obtiene el algoritmo IG básico igualado con el IG con ambas intensificaciones. En el caso de este tipo de representación, los resultados obtenidos se ven muy limitados por el número de iteraciones realizadas. Al ser la representación por actividades, la construcción de la solución es mucho más lenta que por pacientes y requiere de una serie de condiciones para que el algoritmo vaya evolucionando hacia una mejor solución. Por ejemplo, para que las actividades de un mismo PU vayan variando su posición sin que las limite la predecesora o la sucesora, estas mismas actividades deben ser extraídas en la fase de destrucción del algoritmo. Además, la primera actividad del PU de un paciente establece hasta dónde se puede insertar una actividad del PU, lo mismo ocurre con la última actividad del P.U. Es por esto por lo que, en el tiempo de computación establecido, las soluciones obtenidas son notablemente peores para este tipo de representación. Si, además, se introduce la diversificación por búsqueda local, las iteraciones realizadas disminuyen considerablemente, siendo los resultados obtenidos aún peores.

Para corroborar y apoyar las conclusiones obtenidas mediante el anterior análisis, se calcula el ARPD para todos los algoritmos en el conjunto de instancias establecido. Como se ha explicado anteriormente, mediante el cálculo del ARPD se obtiene la desviación promedio de cada algoritmo respecto a la mejor solución obtenida en cada instancia. De nuevo, la representación de la solución por pacientes obtiene los mejores resultados en cualquiera de sus algoritmos respecto a la representación por actividades. Se afirma que el IG con S.A. por pacientes es el mejor algoritmo de forma global con una desviación promedio del 0.68%, seguido por el IG con B.L (0.9%) y el IG con ambas intensificaciones (0.88%), ambas para la representación por pacientes. Respecto a la representación por actividades se puede deducir que todos los algoritmos se comportan de forma similar en cuanto a los resultados obtenidos con una desviación del 3.8% para el IG básico y con S.A. y del 3.6% para el IG con ambas intensificaciones. El IG con búsqueda local, muy limitado por el menor número de iteraciones que realiza, con un ARPD del 4.9% es el peor de los algoritmos para la representación por actividades. Los cálculos del ARPD para todos los algoritmos se pueden

ver en la Figura 13.

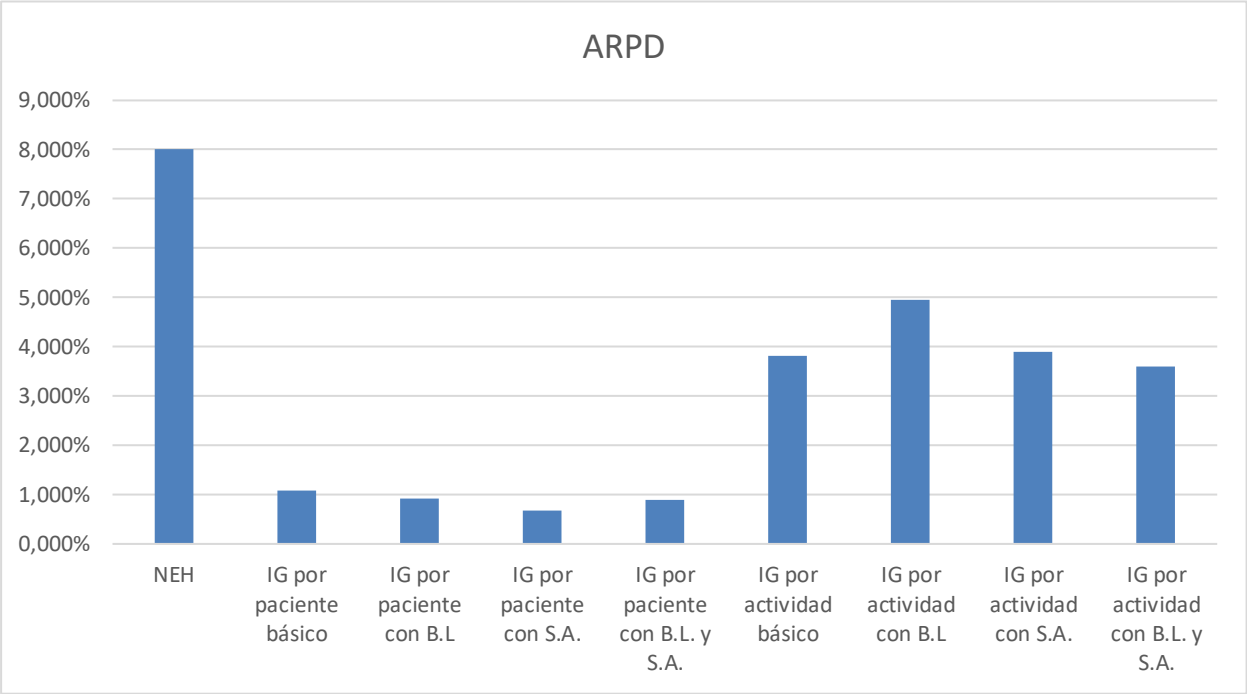

Figura 13: ARPD para 30 instancias. Recursos de las 12 h.

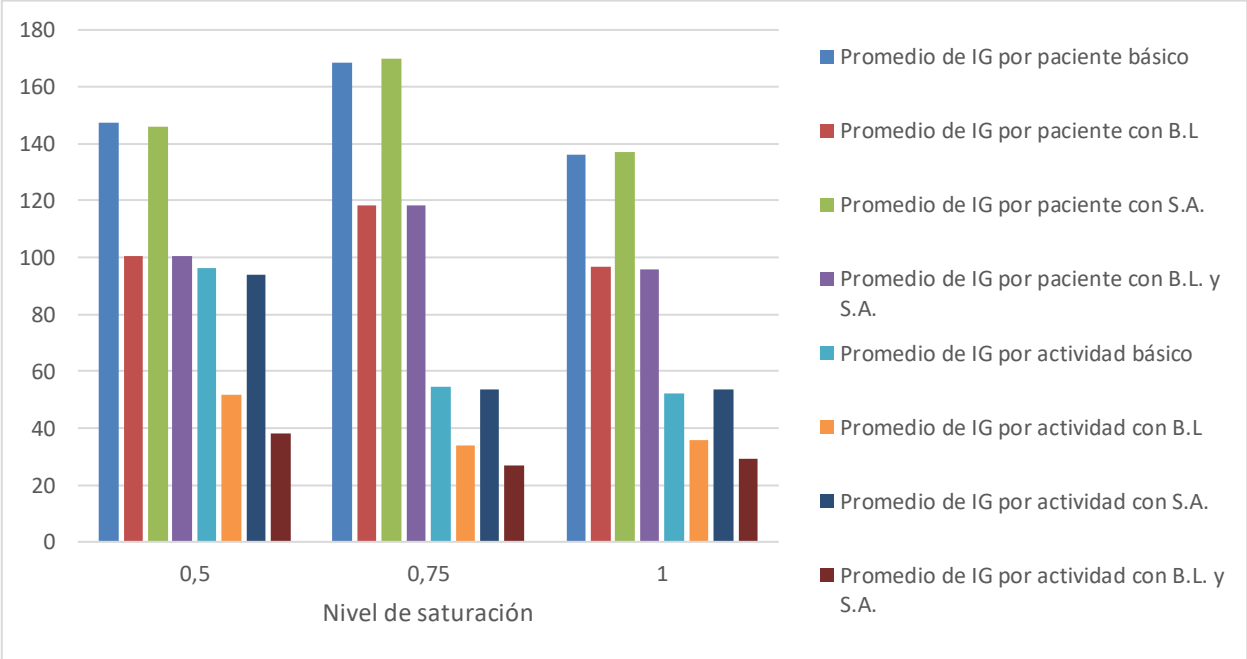

Figura 14: Promedio de iteraciones por algoritmo para cada nivel de saturación. Recursos de las 12 h.

Para apoyar los resultados obtenidos, en la Figura 14 se pueden observar el promedio de iteraciones que cada algoritmo realiza en cada nivel de saturación. Se puede ver que, excepto para el nivel del 50%, donde algunos de los algoritmos que consideran la representación por actividades se igualan

a otros que consideran la representación por pacientes, el número de iteraciones en la representación por actividades es aproximadamente la mitad de las que se realizan en el otro tipo de representación, lo que afecta en gran medida a los resultados obtenidos.

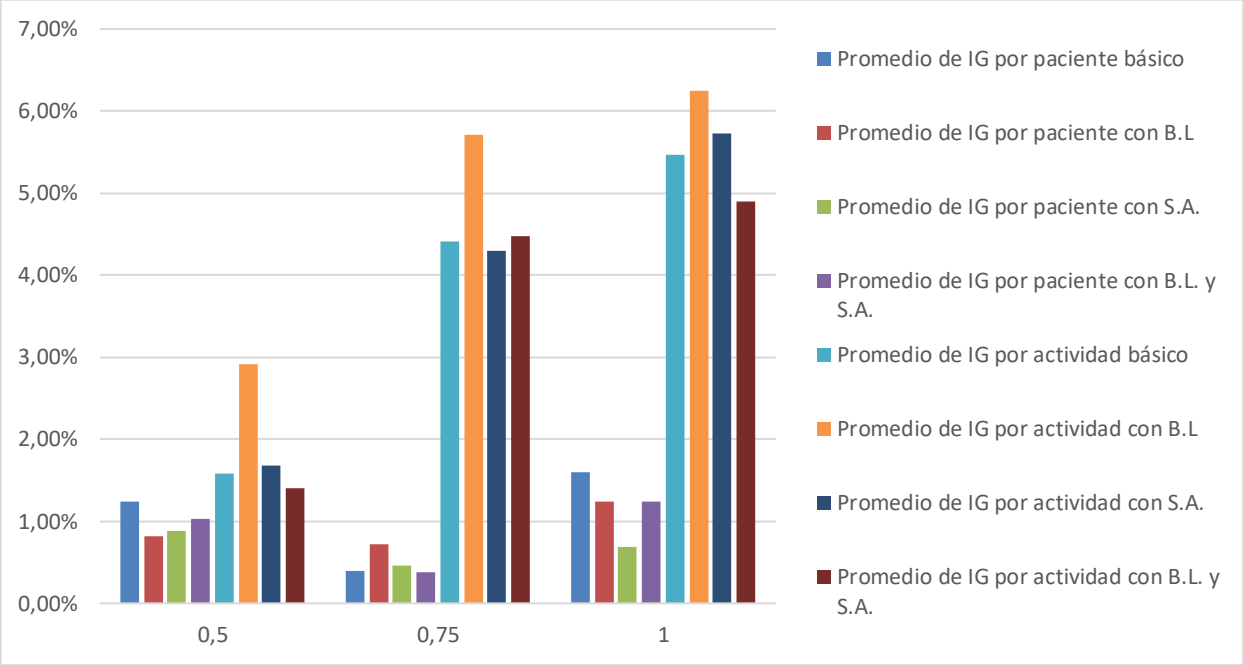

Figura 15: ARPD de cada algoritmo por nivel de saturación. Recursos de las 12 h.

Finalmente, para esta composición de recursos del SUH, se realiza una última computación aumentando el tiempo de ejecución y para un único nivel de saturación. En este caso, se comparan dos de los mejores algoritmos de cada tipo de representación de la solución con el objetivo de verificar que la representación por actividades, para un mayor tiempo de análisis computacional, acaba encontrando una mejor solución que la representación por pacientes. Esto es así ya que, según el modelado del problema, la mejor solución obtenida en la representación por actividades debe ser, como mínimo, igual a la obtenida en la representación por pacientes.

Para un tiempo de computación por algoritmo de 7 minutos aproximadamente y para una saturación del 75% (en el análisis anterior, para esta saturación el tiempo era de 4 minutos aprox.), los resultados para el algoritmo IG por pacientes con S.A. y del algoritmo IG por actividades básico se muestran en la Figura 16. Para hacer una comparación de los mejores algoritmos de cada tipo de representación según los resultados del ARPD anterior, también se ejecuta el IG por actividades con B.L y S.A. (Figura 16).

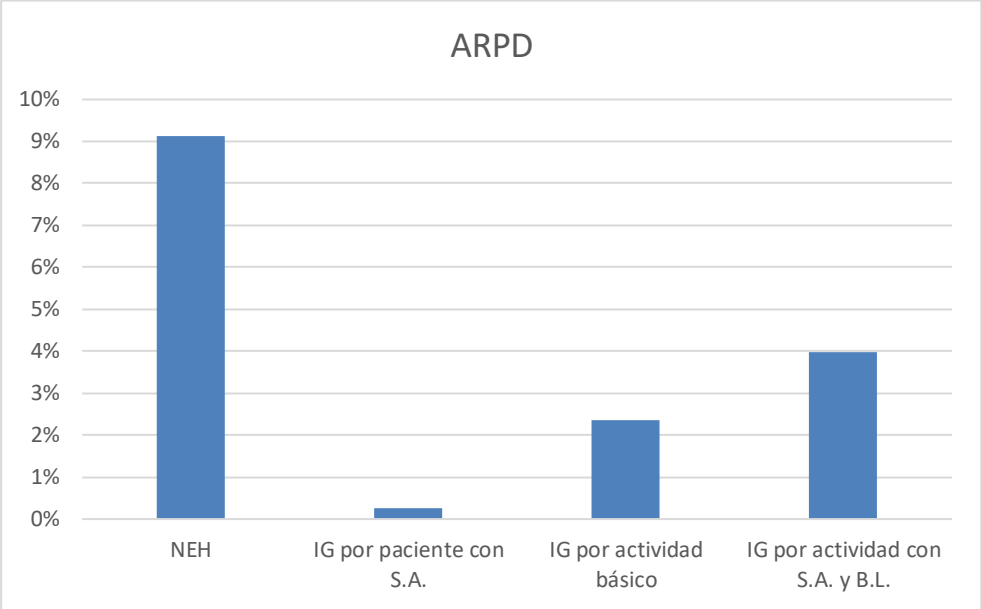

Figura 16: Comparación entre IG por paciente con S.A., IG por actividad con S.A. y B.L y IG básico, aumentando el tiempo de computación.

Con los resultados obtenidos aumentando el tiempo de computación aproximadamente el doble del establecido en las primeras optimizaciones realizadas, se pueden sacar dos conclusiones:

- En primer lugar, se observa que a pesar de que en el análisis computacional principal los algoritmos por actividades básico y con ambas intensificaciones empataban a número de mejores soluciones alcanzadas y, además, este último algoritmo mejoraba al anterior en el ARPD que tiene en cuenta el total de las soluciones encontradas, aumentando el tiempo de computación ocurre que el algoritmo IG por actividades básico presenta los mejores resultados para este tipo de representación. Esto es debido al factor principal que afecta al comportamiento de los algoritmos de representación por actividades, el número de iteraciones. Como se ha visto en la Figura 13, el algoritmo IG básico realiza aproximadamente el doble de iteraciones que el que contiene ambas intensificaciones.

- Al aumentar el tiempo de computación un 70% aproximadamente, la representación por actividades tiene una gran mejora al realizar un mayor número de iteraciones. Tanto es así que el ARPD obtenido en esta última ejecución es del 2.37% respecto al mejor algoritmo de la representación por pacientes, el IG con S.A, cuando en la ejecución principal (Figura 14) el ARPD tenía un valor del 3.6%. Si se siguiera aumentando el tiempo de computación y para un mayor número de instancias que permita concluir estadísticamente, se prevé que el tipo de representación de la solución por actividades acabe encontrando mejores soluciones que el tipo por pacientes.

6.2 Recursos de las 6:00 h.

A continuación, se resuelve el escenario de gestión para la distribución de recursos de las 6:00, volviendo al tiempo de computación establecido en la ejecución principal, es decir, con un factor de 375 milisegundos. El único dato de entrada que cambia respecto a la optimización del SUH a

las 12 h. es la composición de los recursos que se encuentran disponibles en esta franja horaria. El SUH, tal y como se expuso en apartados anteriores, consta de 1 médico, 6 enfermeros, 1 máquina de radiodiagnóstico, 1 laboratorio y un CT-Scan. Con la nueva composición de recursos, los distintos niveles de saturación máximos se alcanzarán para un menor número de pacientes en el SUH al haber menos recursos de cada tipo de recurso que compone el servicio de urgencias, siendo por tanto el problema de menor dimensión en cada saturación que con la composición de las 12 h. El número de actividades totales es de 30 para el 50% de saturación, 42 actividades para el 75% y 51 para el 100%.

En primer lugar, de nuevo se realiza la comparación de los algoritmos en función del número de mejores soluciones encontradas. Como se puede observar en la Figura 17, hay un cambio significativo respecto a la anterior franja horaria. Con esta composición de recursos, la representación por actividades supera a la de por pacientes en todos los algoritmos excepto en uno, el IG con búsqueda local, de nuevo castigado por la importante reducción de iteraciones a costa de una mayor exploración de la secuencia. El algoritmo que en más ocasiones encuentra el mejor resultado es el IG por actividades con S.A., haciéndolo en 22 de las 30 instancias resueltas; seguido del IG por actividades con ambas intensificaciones, con 19 ocasiones y el IG por actividades básico, con 18. En la representación por pacientes, el algoritmo que más veces alcanza estas soluciones es el IG básico, siendo el peor el IG con S.A., justo al contrario de lo que pasaba con la composición de recursos de las 12 h.

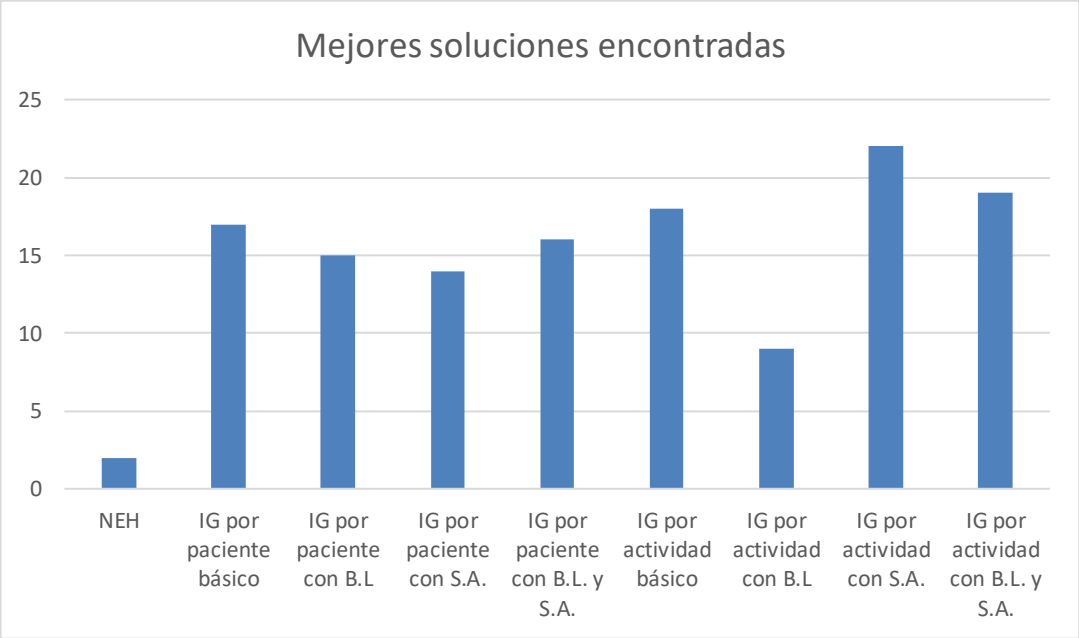

Figura 17: Número de mejores soluciones encontradas por algoritmo. Recursos de las 6 h.

Para seguir con el análisis se procede ahora a calcular el ARPD de cada algoritmo. En el cálculo de la desviación promedio de la mejor solución encontrada por cada algoritmo respecto a la mejor solución global, no se sigue la tendencia anterior. El ARPD muestra que los dos algoritmos con mejores resultados promedio son el algoritmo IG por pacientes con ambas intensificaciones y el algoritmo IG por actividades con S.A., con valores del 0.79 y 0.8% respectivamente. Comparándolo con los resultados obtenidos para la anterior composición de recursos, hay una

mejoría clara en cuanto al tipo de representación por actividades, influenciada por el menor número de iteraciones que es necesario realizar para alcanzar buenas soluciones debido a una menor dimensión del problema.

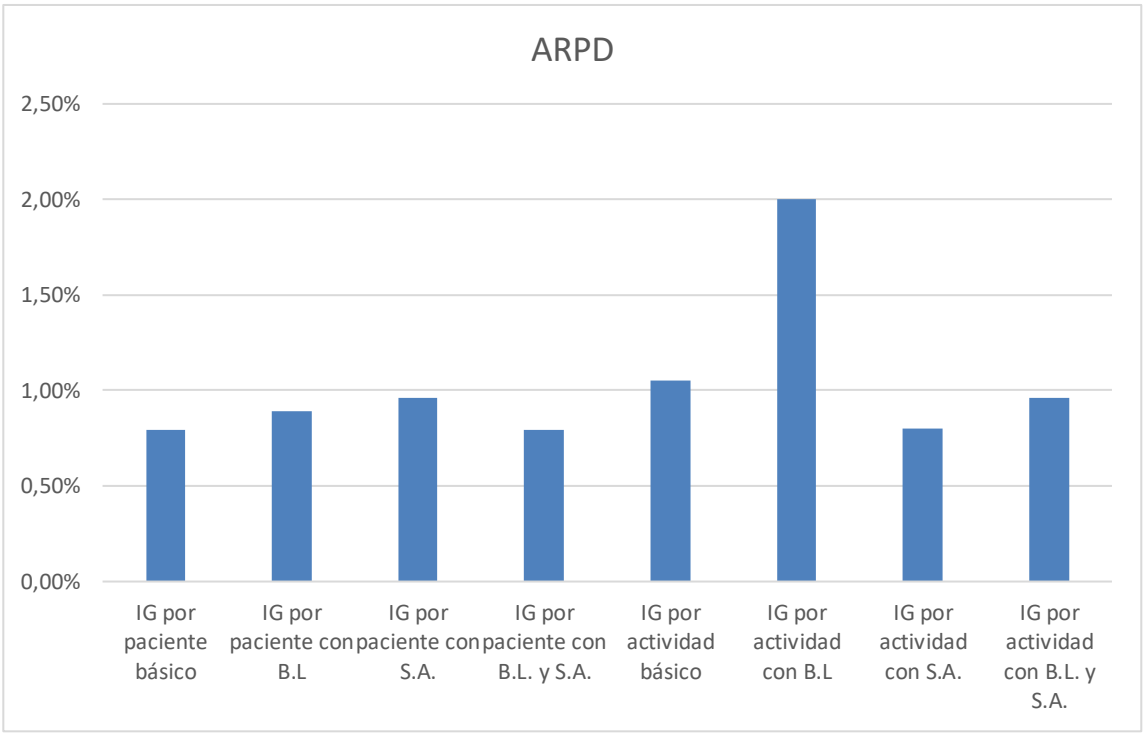

Figura 18: ARPD para 30 instancias. Recursos de las 6 h.

Analizando cada tipo de representación de la solución por separado, en el tipo de representación por pacientes el algoritmo que mejor ARPD presenta es, como se ha dicho antes, el IG con B.L. y S.A. igualado con el IG básico. El peor algoritmo para este tipo de representación con la presente composición de recursos es el IG con S.A., con un ARPD del 0.96%. De nuevo, hay un cambio respecto a la franja horaria de la 12 h., donde este algoritmo era el que mejor promedio de resultados presentaba de forma global. Respecto a la representación por actividades y en cuanto al cálculo del ARPD, el promedio de los algoritmos es peor en resultados que por pacientes. No obstante, los resultados son mucho más cercanos que en la franja horaria anterior. El peor algoritmo es, de nuevo, el IG con B.L, con una desviación del 2%, inferior a la mitad de la obtenida para las 12h., de casi el 5%. En este caso el IG básico empeora sus resultados, siendo el segundo en cuanto a peores resultados obtenidos se refiere. En promedio, la desviación de ambos tipos de representación es del 0.86% para la representación por pacientes y del 1.2% por actividades, siendo del 0.94% si se descartara el IG con B.L.

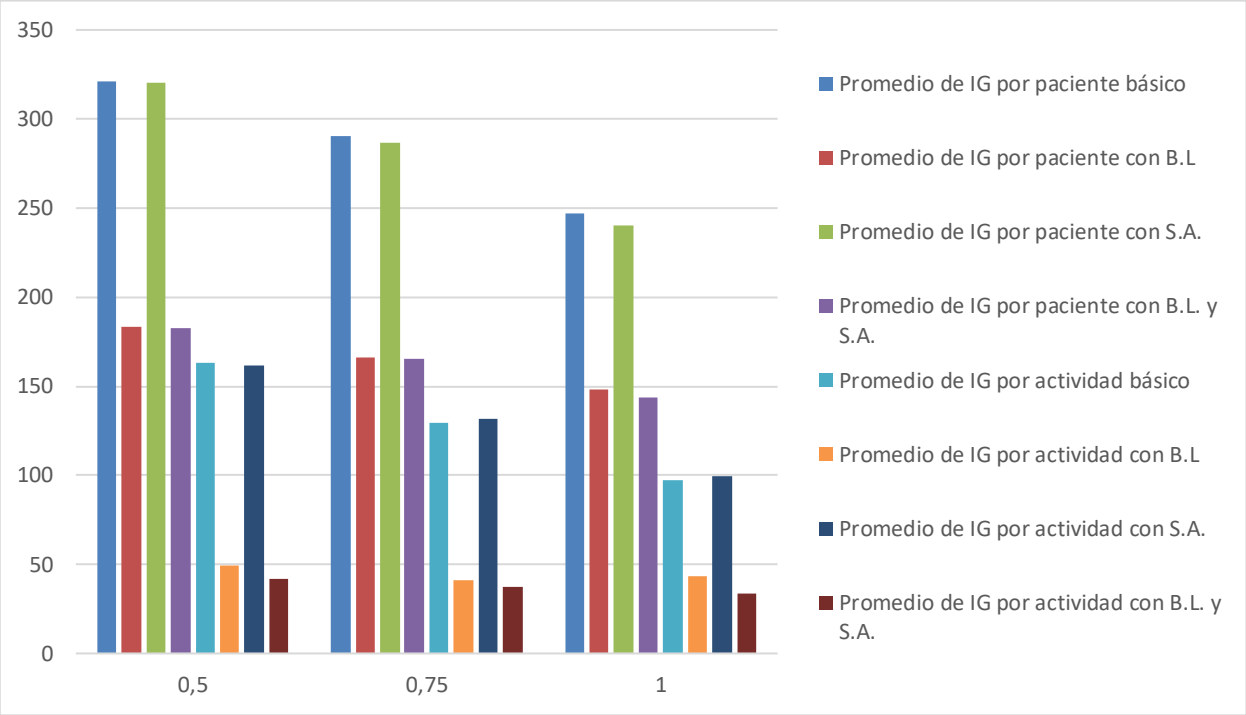

Figura 19: Promedio de iteraciones por algoritmo para cada nivel de saturación. Recursos de las 6 h.

En cuanto al número de iteraciones, se sigue la tendencia de la franja horaria de las 12 h. El número de iteraciones influye de manera notoria en los resultados promedio de los distintos algoritmos, siendo evidente la disminución del número de iteraciones realizadas a medida que aumenta la máxima saturación de los recursos y, con ello, el número de pacientes y la dimensión del problema. Con esta disminución de iteraciones los resultados alcanzados empeoran y, comparando los dos tipos de secuenciaciones, la representación por actividades disminuye el número de iteraciones llevadas a cabo más rápidamente que por pacientes a medida que aumenta la dimensión del problema, pasando de un promedio de 163 para el 50% de saturación, a 97 para el 100% en el algoritmo IG básico. Esta disminución es de un 40%, mucho más alta que la que sufre, por ejemplo, el IG por pacientes básico, del 23%. Este será el factor clave que influya en los resultados alcanzados por cada tipo de representación.

Por último, para concluir el análisis, se extraen los resultados del ARPD de cada algoritmo en función del nivel máximo de saturación, lo cual influye directamente en la cantidad de pacientes generados y, con ello, en la dimensión del problema. Tal y como se puede observar en la Figura 20, los algoritmos correspondientes al tipo de representación por actividades tienen mejor comportamiento para saturaciones menores. Esto es fundamentalmente por el menor número de iteraciones que requiere encontrar buenas soluciones en este tipo de representación de la solución al ser menor el número de pacientes. De hecho, para la saturación del 50% todos los algoritmos por actividades son mejores que los del tipo de representación por pacientes. Con la máxima saturación, la del 100%, los resultados obtenidos se invierten, presentando mejores soluciones los algoritmos por pacientes al sufrir una menor reducción de iteraciones y hacer una mayor permutación de actividades del PU con cada iteración.

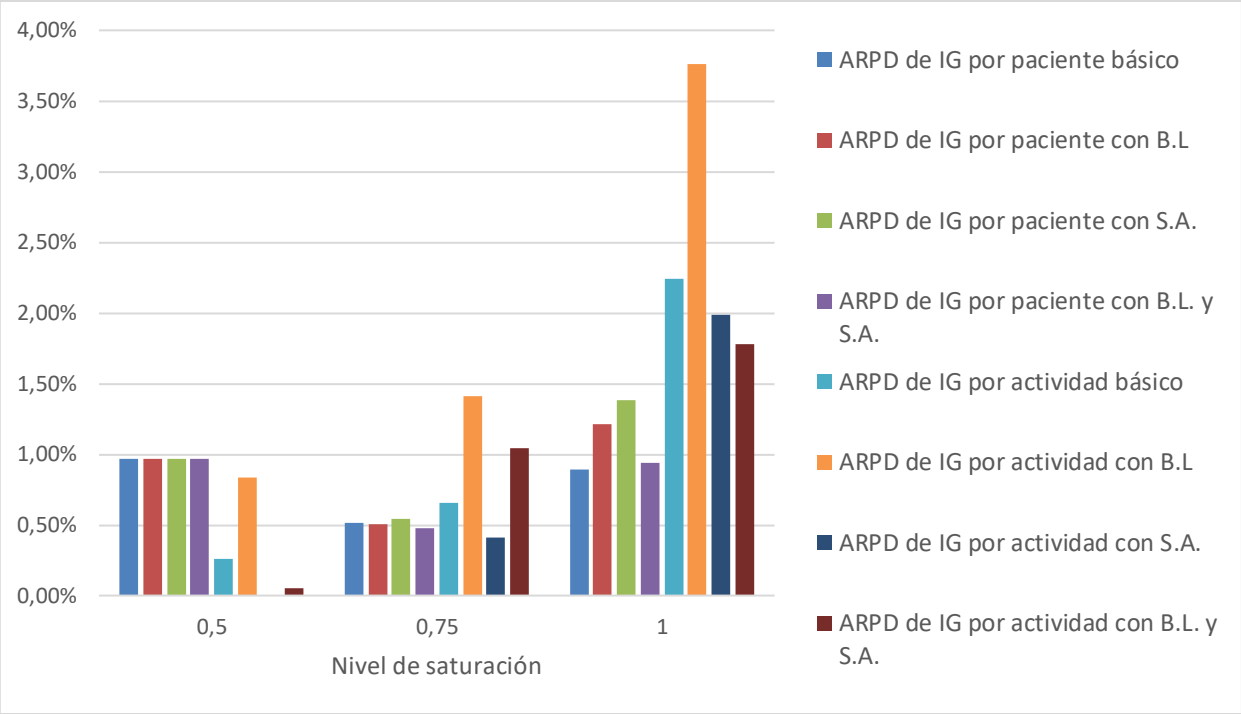

Figura 20: ARPD de cada algoritmo por nivel de saturación. Recursos de las 6 h.

Si se aumentara el tiempo de computación tal y como se ha hecho con la configuración de las 12h., los resultados de la representación por actividades mejorarían al aumentar el número de iteraciones realizadas de manera que, para un determinado tiempo de computación, las soluciones obtenidas por cualquier algoritmo de la representación por actividades serían mejores que las obtenidas por cualquier algoritmo que realice la representación por pacientes.

La conclusión que se extrae tras el análisis realizado es que, en primer lugar, la intensificación por búsqueda local no tiene un buen comportamiento para el tipo de representación por actividades al afectar negativamente al número de iteraciones realizadas, ya de por sí reducidas por la naturaleza de este tipo de representación. En cuanto al tipo de representación por pacientes, el algoritmo que mejores resultados obtiene en los distintos escenarios estudiados es, en promedio, el IG con ambas intensificaciones. Por último, comparando los dos tipos de representación, la representación de la solución por pacientes es sin duda la mejor opción si se quieren alcanzar buenas soluciones en un tiempo reducido y con un gran número de pacientes. Por otro lado, si se dispone de más tiempo para llevar a cabo la optimización o las dimensiones del problema son más o menos reducidas, la representación por actividades obtiene un mayor número de mejores soluciones y, además, mejores soluciones en promedio que la representación por pacientes.

7 CONCLUSIONES

El objetivo del proyecto realizado es mejorar la gestión del SUH mediante la secuenciación de las actividades asignadas a los distintos PUs de los pacientes que llegan al centro hospitalario. Dicha gestión corresponde al nivel operativo en la gestión de un SUH. Los objetivos planteados que facilitan la mejora mencionada se basan en el tiempo de estancia total de un paciente en el centro hospitalario y en el incumplimiento del TEPCOF asociado a la prioridad asistencial de cada paciente, el cual establece el tiempo máximo que dicho paciente puede esperar antes de ser atendido por primera vez.

Para llevar a cabo dichos objetivos se hace uso de metaheurísticas adaptadas del *Iterated Greedy* [1], el cual contiene además una búsqueda local y el *Simulated Annealing*. La combinación de estas metaheurísticas da lugar a cuatro algoritmos que llevan a cabo permutaciones en la secuenciación con el objetivo de alcanzar el mejor resultado posible. La diferenciación que se ha planteado en el presente proyecto ha sido la presentación de dos tipos de representación de la solución en función de cómo se asignan las actividades a los recursos del SUH. Los dos tipos de representación de la solución son:

- Representación de la solución por pacientes, en la que la asignación se realiza revisando todas las actividades que componen el PU de un determinado paciente y asignándolas en seguido a los recursos disponibles, siempre lo antes posible. De esta manera, cada paciente secuenciado lleva asociado un paquete de actividades que deben ser asignadas al completo antes de pasar al siguiente paciente.

- Representación de la solución por actividades, la novedad planteada en el presente proyecto. La construcción de la secuencia se realiza actividad por actividad, independientemente del paciente al que pertenezcan. Para realizar la asignación se crea un *decoding* que respete las restricciones de precedencia entre las distintas actividades, de manera que una actividad no puede ser asignada antes que su predecesora ni después que su sucesora.

Se ha llevado a cabo un análisis computacional para comparar ambos tipos de creación de la secuencia y, a su vez, las cuatro variantes de metaheurística. Para el análisis se han planteado dos escenarios distintos en función de la distinta composición de los recursos que componen el SUH: un escenario en la franja horaria de las 12 h. y otro en la de las 6 h., cada uno con los correspondientes recursos asociados. Para cada escenario, se consideran distintos niveles máximos de saturación de los recursos del servicio de urgencias: del 50, 75 y 100%, lo cual influye directamente en el número de pacientes que se atienden. Además, para cada saturación se generan 10 instancias, realizándose en total 60 instancias para la comparación de los dos tipos de representación de la solución.

Los resultados obtenidos han definido claramente cómo se comporta cada tipo de representación en los distintos escenarios. La secuencia construida por pacientes obtiene buenos resultados de forma general, independientemente del tamaño del problema definido por la cantidad de recursos que componen el SUH y el número de pacientes que llegan al centro hospitalario. Estableciendo un tiempo de computación máximo de 7 minutos por algoritmo, este tipo de secuencia encuentra

las mejores soluciones al problema planteado. En cuanto a la construcción de la secuencia por actividades, se ve claramente limitado por el tiempo de computación impuesto. Al realizar una mayor exploración en las actividades que componen la secuencia de asignación, las iteraciones que realiza dentro del tiempo establecido son aproximadamente un 50% de las que se realizan en el otro tipo de representación de la solución, lo cual afecta directamente a la calidad de las soluciones encontradas. Además, a medida que aumenta la dimensión del problema, las iteraciones realizadas en el tiempo de computación establecido disminuyen más rápidamente que en la representación por pacientes.

Las conclusiones obtenidas tras llevar a cabo la experimentación computacional es que, atendiendo únicamente a los dos tipos de representación, si el tiempo de computación es ajustado y la dimensión del problema es media o alta (75% o 100% de saturación), la secuencia que mejores resultados aporta es la construida por pacientes al no verse tan afectada por la reducción en las iteraciones realizadas. Esto es debido a la mayor variación en la asignación de las actividades que conlleva la alternancia en el orden de los pacientes, al cambiar el orden de todo el paquete de actividades que compone el PU de dichos pacientes (factibilidad de la secuencia del PU). Una vez llevado a cabo el análisis de los algoritmos propuestos, se han detectado las limitaciones en la representación por actividades que hacen que sea menos competitivo a medida que aumenta el tamaño del problema. La naturaleza del problema hace que muchas de las iteraciones que realiza dicha representación no aporten mejores resultados debido a las actividades extraídas en la etapa de destrucción del algoritmo. La obligación de respetar precedencias y sucesiones en la asignación provoca que, si se destruye una única actividad correspondiente al PU de un paciente y dicha actividad no es ni la primera ni la última de su PU, en la etapa de construcción únicamente se puede introducir la actividad destruida en la misma posición en la que se encontraba antes de extraerla, ya que su predecesora y su sucesora no han variado su posición. Esto hace que, en dicha iteración, se pierda una posible mejora asociada al cambio de posición de una de las d actividades destruidas, lo que conlleva que sean necesarias un mayor número de iteraciones para encontrar mejores soluciones. Sin embargo, si la dimensión del problema es pequeña o mediana (50% o 75%) o se dispone de más de 10 minutos para ejecutar el algoritmo elegido, la representación por actividades es claramente superior en los resultados obtenidos, obteniendo soluciones que no son factibles para el tipo de representación por pacientes debido a las restricciones en la asignación a los recursos.

Con esto, la elección de un tipo de representación de la solución u otro es, sobre todo, en función del tiempo del que se dispone para tomar decisiones en la gestión del SUH. No obstante, a medida que disminuye la dimensión del problema, este tiempo necesario para encontrar buenos resultados se iguala para ambos tipos de construcción de la secuencia.

Una mejora que se plantea para revisiones futuras del problema es partir de una secuencia inicial distinta a la aportada por la NEH (en este trabajo se parte en todas las variantes con la secuencia de pacientes obtenidas por la NEH). Al estar las actividades de un mismo PU secuenciadas consecutivamente en la secuencia inicial, en las iteraciones posteriores se hace más complicado la variación de las posiciones de las distintas actividades, siendo necesaria la destrucción de varias actividades de un mismo PU o de la primera y última actividad, ya que estas no tendrían predecesora ni sucesora respectivamente, abriendo el abanico de posiciones en las que más adelante se pueden insertar las otras actividades de ese mismo PU.

Por último, en cuanto al menor número de iteraciones que se realizan a medida que aumenta la dimensión del problema, se plantea no realizar la completa exploración de la secuencia por cada actividad extraída, sino insertarla en x posiciones aleatorias estudiando en cada inserción el resultado que se devuelve. Esta solución se ve de nuevo limitada por las precedencias y sucesiones, aunque a medida que avanza el tiempo de computación se hace más efectiva por la teórica evolución de la secuencia. Con esto se prevé que aumentaría el número de iteraciones llevadas a cabo, con la consecuente mejora en los resultados obtenidos.

Bibliografía

[1] R. Ruiz and T. Stützle, "A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem," *Eur J Oper Res*, vol. 177, no. 3, pp. 2033–2049, 2007, doi: 10.1016/j.ejor.2005.12.009.

[2] L. Bedoya-Valencia and E. Kirac, "Evaluating alternative resource allocation in an emergency department using discrete event simulation," *Simulation*, vol. 92, no. 12, pp. 1041–1051, 2016, doi: 10.1177/0037549716673150.

[3] R. Aringhieri, M. E. Bruni, S. Khodaparasti, and J. T. van Essen, "Emergency medical services and beyond: Addressing new challenges through a wide literature review," *Comput Oper Res*, vol. 78, pp. 349–368, 2017, doi: 10.1016/j.cor.2016.09.016.

[4] M. Masmoudi, B. Jarboui, and P. Siarry, *Operations research and simulation in healthcare*, 1st ed. 2021. Cham, Switzerland: Springer, 2021. doi: 10.1007/978-3-030-45223-0.

[5] D. Gómez Medina, "Algoritmos de optimización para el servicio de urgencias caso de estudio en el Hospital Universitario Virgen del Rocío : Trabajo Fin de Máster," Sevilla, 2021.

[6] Navonil. Mustafee, *Operational Research for Emergency Planning in Healthcare: Volume 1*, 1st ed. 2016. in OR Essentials. London: Palgrave Macmillan UK, 2016. doi: 10.1057/9781137535696.

[7] R. Silva López, "Simulación de un servicio de urgencias mediante Arena Trabajo Fin de Grado," Sevilla, 2022.

[8] K. Karboub, T. Mohamed, F. Moutaouakkil, D. Sofiene, and A. Dandache, *Emergency patient's arrivals management based on iot and discrete simulation using arena*, vol. 12293. 2020. doi: 10.1007/978-3-030-58008-7_19.

[9] E. Van Bockstal and B. Maenhout, "A study on the impact of prioritising emergency department arrivals on the patient waiting time," *Health Care Manag Sci*, vol. 22, no. 4, pp. 589–614, 2019, doi: 10.1007/s10729-018-9447-5.

[10] F. Zeinali, M. Mahootchi, and M. M. Sepehri, "Resource planning in the emergency departments: A simulation-based metamodeling approach," *Simul Model Pract Theory*, vol. 53, pp. 123–138, 2015, doi: 10.1016/j.simpat.2015.02.002.

[11] K. J. Cairns, A. H. Marshall, and F. Kee, "Using simulation to assess cardiac first-responder schemes exhibiting stochastic and spatial complexities," *Journal of the Operational Research Society*, vol. 62, no. 6, pp. 982–991, 2011, doi: 10.1057/jors.2010.27.

[12] M. Harzi, J.-F. Condotta, I. Nouaouri, and S. Krichen, "Using the hybrid ILS/VND method for solving the patients scheduling problem in emergency department: A case study," in *Procedia Computer Science*, 2018, pp. 733–742. doi: 10.1016/j.procS.2018.08.007.

[13] M. Yousefi and M. Yousefi, “Human resource allocation in an emergency department: A metamodel-based simulation optimization,” *Kybernetes*, vol. 49, no. 3, pp. 779–796, 2020, doi: 10.1108/K-12-2018-0675.

[14] S. Kiriş, N. Yüzügüllü, N. Ergün, and A. Alper Çevik, “A knowledge-based scheduling system for Emergency Departments,” *Knowl Based Syst*, vol. 23, no. 8, pp. 890–900, 2010, doi: 10.1016/j.knosys.2010.06.005.

[15] M. Harzi, J.-F. Condotta, I. Nouaouri, and S. Krichen, “Scheduling Patients in Emergency Department by Considering Material Resources,” in *Procedia Computer Science*, 2017, pp. 713–722. doi: 10.1016/j.procs.2017.08.153.

[16] F. Ajmi, S. B. Othman, H. Zgaya, S. Hammadi, and J.-M. Renard, *An innovative approach to jointly scheduling and assigning a consultation time to patients arriving in the emergency department*, vol. 245. 2017. doi: 10.3233/978-1-61499-830-3-989.

[17] A. Azadeh, M. Hosseinabadi Farahani, S. Torabzadeh, and M. Baghersad, “Scheduling prioritized patients in emergency department laboratories,” *Comput Methods Programs Biomed*, vol. 117, no. 2, pp. 61–70, 2014, doi: 10.1016/j.cmpb.2014.08.006.

[18] M. Harzi, J.-F. Condotta, I. Nouaouri, and S. Krichen, “Using the hybrid ILS/VND method for solving the patients scheduling problem in emergency department: A case study,” in *Procedia Computer Science*, 2018, pp. 733–742. doi: 10.1016/j.procS.2018.08.007.

[19] T. Alves de Queiroz, M. Iori, A. Kramer, and Y.-H. Kuo, *Scheduling of Patients in Emergency Departments with a Variable Neighborhood Search*, vol. 12559 LNCS. 2021. doi: 10.1007/978-3-030-69625-2_11.

[20] T. Alves de Queiroz, M. Iori, A. Kramer, and Y.-H. Kuo, “Dynamic scheduling of patients in emergency departments,” *Eur J Oper Res*, 2023, doi: 10.1016/j.ejor.2023.03.004.

[21] Y. Liu, T. Moyaux, G. Bouleux, and V. Cheutet, “Hybrid Simulation Modelling of Emergency Departments for Resource Scheduling,” *Journal of Simulation*, 2023, doi: 10.1080/17477778.2023.2187321.

[22] Z. Wang, R. Liu, and Z. Sun, “Physician Scheduling for Emergency Departments Under Time-Varying Demand and Patient Return,” *IEEE Transactions on Automation Science and Engineering*, vol. 20, no. 1, pp. 553–570, 2023, doi: 10.1109/TASE.2022.3163259.

[23] G. B. Yom-Tov and A. Mandelbaum, “Erlang-R: A time-varying queue with reentrant customers, in support of healthcare staffing,” *Manufacturing and Service Operations Management*, vol. 16, no. 2, pp. 283–299, 2014, doi: 10.1287/msom.2013.0474.

[24] D. Daldoul, I. Nouaouri, H. Bouchriha, and H. Allaoui, “Simulation-based optimisation approach to improve emergency department resource planning: A case study of Tunisian hospital,” *International Journal of Health Planning and Management*, 2022, doi: 10.1002/hpm.3499.

[25] B. PueblaMartínez and R. Vinader-Segura, *Ecosistema de una pandemia, COVID 19 : transformación mundial*. in Colección Conocimiento Contemporáneo ; 7. Madrid:

Dykinson, 2021.

[26] S. Bär, *Generic multi-agent reinforcement learning approach for flexible job-shop scheduling*. Wiesbaden: Springer Vieweg, 2022.

[27] Servicio de Salud de Castilla-La Mancha, "Decreto 45/2019, de 21 de mayo," 2019. Accessed: Jul. 13, 2023. [Online]. Available: https://docm.jccm.es/portaldocm/descargarArchivo.do?ruta=2019/05/29/pdf/2019_5139.pdf&tipo=rutaDocm

[28] Junta de Andalucía, "Protocolo de coordinación de asistencia hospitalaria," 2018.

[29] L. Amodeo, E.-G. Talbi, and F. Yalaoui, *Heuristics for optimization and learning*, 1st ed. 2021. in Studies in Computational Intelligence, 906. Cham, Switzerland: Springer, 2021. doi: 10.1007/978-3-030-58930-1.

[30] Á. León Martínez, "Algoritmo de búsqueda local iterada para la secuenciación en talleres de flujo y minimización de makespan Trabajo de Fin de Grado," Sevilla, 2018.

[31] J. Personat Moyano, "Desarrollo de una aplicación para el estudio de la superficie de suelo en almacenamientos Trabajo Fin de Grado," Sevilla, 2021.

[32] J. Santiago. Nolasco Valenzuela, *Python.* Madrid: RA-MA Editorial, 2018.

[33] C. Miller, *Hands-on data analysis with NumPy and Pandas : implement Python packages from data manipulation to processing*, 1st edition. Birmingham, UK ; Packt, 2016.

[34] Y. Liu, T. Moyaux, G. Bouleux, and V. Cheutet, "Hybrid Simulation Modelling of Emergency Departments for Resource Scheduling," *Journal of Simulation*, 2023, doi: 10.1080/17477778.2023.2187321.

Anexo

Creación de datos de entrada

```
import numpy as np
import asignacion
import gantt
import algoritmo
import pandas as pd
rg=np.random.default_rng(2)
resultados=np.zeros((0,10))
f_obj_1=np.zeros((0,10))
f_obj_2=np.zeros((0,10))
iteraciones=np.zeros((0,10))
inst=0
saturaciones=np.array([0.5,0.75,1])
for y in range(len(saturaciones)):
    print('Saturacion: ',saturaciones[y])
    for instancias in range(10): #10 instancias
        print('Instancia: ',instancias+1)
        resultados=np.insert(resultados, len(resultados), np.zeros(10),0)
        f_obj_1=np.insert(f_obj_1, len(f_obj_1), np.zeros(10),0)
        f_obj_2=np.insert(f_obj_2, len(f_obj_2), np.zeros(10),0)
        iteraciones=np.insert(iteraciones, len(iteraciones),
np.zeros(10),0)
        salir=0
        #Recursos [medico,medi.asis,enf,lab,ray,ct-scan,alta]
        R=np.array([1,2,3,3,3,3,3,3,4,5,6,7])
        disp_i=np.zeros(len(R),int) #instante a partir del que el recurso
i esta disponible, suponiendo 120 min de trabajo en asignacion
        for i in range(len(disp_i)-1):
            disp_i[i]=rg.integers(0,10) #asignarselo a pacientes
aleatorios
        LOS=140
        disp_f=np.full(len(R), LOS)
        #matriz precedencia recursos
        Y_r=np.zeros((7,7),int)
        for r in range(len(R)):
          for t in range(len(R)):
            if r==3 and t==4:
              Y_r[r][t]=1
            elif r==4 and t==3:
              Y_r[r][t]=1
        ficticios=0
        for i in range(len(R)):
            if R[i]==1 or R[i]==2 or R[i]==3 or R[i]==5:
                ficticios+=1
```

```
        I=ficticios
        p_i=np.full(I,6,int)
        max_sat=0
        t_j=np.zeros(I,int)
        Amax=8
        pu_copia=np.zeros((I,Amax),int)
        act=1
        pu_prima=np.zeros((I,Amax),int)
        ult_act=np.zeros(I,int)
        while max_sat<saturaciones[y]:
            if rg.uniform(0,1)<=0.01:
                p_i=np.insert(p_i, I-ficticios, 1)
            elif rg.uniform(0,1)<=0.12:
                p_i=np.insert(p_i, I-ficticios, 2)
            elif rg.uniform(0,1)<=0.60:
                p_i=np.insert(p_i, I-ficticios, 3)
            elif rg.uniform(0,1)<=0.97:
                p_i=np.insert(p_i, I-ficticios, 4)
            elif rg.uniform(0,1)<=1:
                p_i=np.insert(p_i, I-ficticios, 5)
            I+=1
            #prioridades
            tepcof=np.zeros(I,int)
            for i in range (I):
                if p_i[i]==1:
                    tepcof[i]=1
                elif p_i[i]==2:
                    tepcof[i]=10
                elif p_i[i]==3:
                    tepcof[i]=60
                elif p_i[i]==4:
                    tepcof[i]=120
                elif p_i[i]==5:
                    tepcof[i]=240
                elif p_i[i]==6:#para descansos
                    tepcof[i]=480
            #procesos de urgencia
            pu=pu_prima.copy()
            pu=np.insert(pu,I-ficticios-1,np.zeros(Amax,int),0)
            if act!=1:
                act=ult_act[I-ficticios-1]
            for i in range(I-ficticios-1,I):
                if p_i[i]==1:
                    if rg.integers(0,101)<=70:
                        for j in range(Amax):
                            pu[i][j]=act
                            act+=1
                    else:
                        for j in range(Amax-1):
```

```
                            pu[i][j]=act
                            act+=1
                if p_i[i]==2:
                    if rg.integers(0,101)<=70:
                        for j in range(Amax):
                            pu[i][j]=act
                            act+=1
                    else:
                        for j in range(Amax-1):
                            pu[i][j]=act
                            act+=1
                if p_i[i]==3:
                    if rg.integers(0,101)<=30:
                        for j in range(Amax):
                            pu[i][j]=act
                            act+=1
                    else:
                        for j in range(Amax-1):
                            pu[i][j]=act
                            act+=1
                if p_i[i]==4:
                    for j in range(Amax-2):
                        pu[i][j]=act
                        act+=1
                if p_i[i]==5:
                    for j in range(Amax-5):
                        pu[i][j]=act
                        act+=1
                if p_i[i]==6:
                    for j in range(1):
                        pu[i][j]=act
                        act+=1
            pu_prima=pu.copy()
            ult_act=np.zeros(I,int)
            for j in range(I):
                for i in range(Amax):
                    if pu[j][i]!=0:
                        ult_act[j]=pu[j][i]
            J=ult_act[I-1] #n° act
            a_j=np.zeros(J,int)
            for m in range(I):
                for k in range(J):
                    for n in range(Amax):
                        if (pu[m][n]==k+1):
                            a_j[k]=m+1 #para identificar a qué paciente
corresponde cada actividad
            #RECURSOS
            tipo_actividad_jr=np.zeros((J,2),int)
```

```
            desc=0
            for i in range(I):
                if p_i[i]==1:
                    tipo_actividad_jr[pu[i][0]-1][0]=3
                    tipo_actividad_jr[pu[i][0]-1][1]=2
                    tipo_actividad_jr[pu[i][1]-1][0]=1
                    tipo_actividad_jr[pu[i][1]-1][1]=1
                    tipo_actividad_jr[pu[i][2]-1][0]=4
                    tipo_actividad_jr[pu[i][2]-1][1]=1
                    tipo_actividad_jr[pu[i][3]-1][0]=5
                    tipo_actividad_jr[pu[i][3]-1][1]=1
                    if pu[i][-1]!=0:
                        tipo_actividad_jr[pu[i][4]-1][0]=6
                        tipo_actividad_jr[pu[i][4]-1][1]=1
                        tipo_actividad_jr[pu[i][5]-1][0]=3
                        tipo_actividad_jr[pu[i][5]-1][1]=2
                        tipo_actividad_jr[pu[i][6]-1][0]=1
                        tipo_actividad_jr[pu[i][6]-1][1]=1
                        tipo_actividad_jr[pu[i][7]-1][0]=7
                        tipo_actividad_jr[pu[i][7]-1][1]=1
                    else:
                        tipo_actividad_jr[pu[i][4]-1][0]=3
                        tipo_actividad_jr[pu[i][4]-1][1]=2
                        tipo_actividad_jr[pu[i][5]-1][0]=1
                        tipo_actividad_jr[pu[i][5]-1][1]=1
                        tipo_actividad_jr[pu[i][6]-1][0]=7
                        tipo_actividad_jr[pu[i][6]-1][1]=1
                if p_i[i]==2:
                    tipo_actividad_jr[pu[i][0]-1][0]=3
                    tipo_actividad_jr[pu[i][0]-1][1]=2
                    tipo_actividad_jr[pu[i][1]-1][0]=1
                    tipo_actividad_jr[pu[i][1]-1][1]=1
                    tipo_actividad_jr[pu[i][2]-1][0]=4
                    tipo_actividad_jr[pu[i][2]-1][1]=1
                    tipo_actividad_jr[pu[i][3]-1][0]=5
                    tipo_actividad_jr[pu[i][3]-1][1]=1
                    if pu[i][-1]!=0:
                        tipo_actividad_jr[pu[i][4]-1][0]=6
                        tipo_actividad_jr[pu[i][4]-1][1]=1
                        tipo_actividad_jr[pu[i][5]-1][0]=3
                        tipo_actividad_jr[pu[i][5]-1][1]=2
                        tipo_actividad_jr[pu[i][6]-1][0]=1
                        tipo_actividad_jr[pu[i][6]-1][1]=1
                        tipo_actividad_jr[pu[i][7]-1][0]=7
                        tipo_actividad_jr[pu[i][7]-1][1]=1
                    else:
                        tipo_actividad_jr[pu[i][4]-1][0]=3
                        tipo_actividad_jr[pu[i][4]-1][1]=2
                        tipo_actividad_jr[pu[i][5]-1][0]=1
```

```
                tipo_actividad_jr[pu[i][5]-1][1]=1
                tipo_actividad_jr[pu[i][6]-1][0]=7
                tipo_actividad_jr[pu[i][6]-1][1]=1
        if p_i[i]==3:
            tipo_actividad_jr[pu[i][0]-1][0]=3
            tipo_actividad_jr[pu[i][0]-1][1]=1
            tipo_actividad_jr[pu[i][1]-1][0]=1
            tipo_actividad_jr[pu[i][1]-1][1]=1
            tipo_actividad_jr[pu[i][2]-1][0]=4
            tipo_actividad_jr[pu[i][2]-1][1]=1
            tipo_actividad_jr[pu[i][3]-1][0]=5
            tipo_actividad_jr[pu[i][3]-1][1]=1
            if pu[i][-1]!=0:
                tipo_actividad_jr[pu[i][4]-1][0]=6
                tipo_actividad_jr[pu[i][4]-1][1]=1
                tipo_actividad_jr[pu[i][5]-1][0]=3
                tipo_actividad_jr[pu[i][5]-1][1]=1
                tipo_actividad_jr[pu[i][6]-1][0]=1
                tipo_actividad_jr[pu[i][6]-1][1]=1
                tipo_actividad_jr[pu[i][7]-1][0]=7
                tipo_actividad_jr[pu[i][7]-1][1]=1
            else:
                tipo_actividad_jr[pu[i][4]-1][0]=3
                tipo_actividad_jr[pu[i][4]-1][1]=1
                tipo_actividad_jr[pu[i][5]-1][0]=1
                tipo_actividad_jr[pu[i][5]-1][1]=1
                tipo_actividad_jr[pu[i][6]-1][0]=7
                tipo_actividad_jr[pu[i][6]-1][1]=1
        if p_i[i]==4:
            tipo_actividad_jr[pu[i][0]-1][0]=3
            tipo_actividad_jr[pu[i][0]-1][1]=1
            tipo_actividad_jr[pu[i][1]-1][0]=1
            tipo_actividad_jr[pu[i][1]-1][1]=1
            tipo_actividad_jr[pu[i][2]-1][0]=5
            tipo_actividad_jr[pu[i][2]-1][1]=1
            tipo_actividad_jr[pu[i][3]-1][0]=3
            tipo_actividad_jr[pu[i][3]-1][1]=1
            tipo_actividad_jr[pu[i][4]-1][0]=1
            tipo_actividad_jr[pu[i][4]-1][1]=1
            tipo_actividad_jr[pu[i][5]-1][0]=7
            tipo_actividad_jr[pu[i][5]-1][1]=1
        if p_i[i]==5:
            tipo_actividad_jr[pu[i][0]-1][0]=3
            tipo_actividad_jr[pu[i][0]-1][1]=1
            tipo_actividad_jr[pu[i][1]-1][0]=1
            tipo_actividad_jr[pu[i][1]-1][1]=1
            tipo_actividad_jr[pu[i][2]-1][0]=7
            tipo_actividad_jr[pu[i][2]-1][1]=1
```

```
                if p_i[i]==6:
                    if desc==0:
                        tipo_actividad_jr[pu[i][0]-1][0]=1
                        tipo_actividad_jr[pu[i][0]-1][1]=1
                        desc+=1
                    elif desc==1:
                        tipo_actividad_jr[pu[i][0]-1][0]=2
                        tipo_actividad_jr[pu[i][0]-1][1]=1
                        desc+=1
                    elif desc==2:
                        tipo_actividad_jr[pu[i][0]-1][0]=3
                        tipo_actividad_jr[pu[i][0]-1][1]=1
                        desc+=1
                    elif desc==3:
                        tipo_actividad_jr[pu[i][0]-1][0]=3
                        tipo_actividad_jr[pu[i][0]-1][1]=1
                        desc+=1
                    elif desc==4:
                        tipo_actividad_jr[pu[i][0]-1][0]=3
                        tipo_actividad_jr[pu[i][0]-1][1]=1
                        desc+=1
                    elif desc==5:
                        tipo_actividad_jr[pu[i][0]-1][0]=3
                        tipo_actividad_jr[pu[i][0]-1][1]=1
                        desc+=1
                    elif desc==6:
                        tipo_actividad_jr[pu[i][0]-1][0]=3
                        tipo_actividad_jr[pu[i][0]-1][1]=1
                        desc+=1
                    elif desc==7:
                        tipo_actividad_jr[pu[i][0]-1][0]=3
                        tipo_actividad_jr[pu[i][0]-1][1]=1
                        desc+=1
                    elif desc==8:
                        tipo_actividad_jr[pu[i][0]-1][0]=5
                        tipo_actividad_jr[pu[i][0]-1][1]=1
                        desc+=1

            for k in range(0,ult_act[I-ficticios-1]-pu[I-ficticios-
1][0]+1):
                t_j=np.insert(t_j,J-ficticios-1,0)
            i=I-ficticios-1
            if p_i[i]==1:
                t_j[pu[i][0]-1]=rg.triangular(20,30,75)
                t_j[pu[i][1]-1]=rg.triangular(10,20,25)
                t_j[pu[i][2]-1]=rg.triangular(3,5,15)
                t_j[pu[i][3]-1]=rg.triangular(3,5,15)
                if tipo_actividad_jr[pu[i][4]-1][0]==6:
                    t_j[pu[i][4]-1]=20
```

```
                t_j[pu[i][5]-1]=rg.triangular(20,30,40)
                t_j[pu[i][6]-1]=rg.triangular(20,30,40)
            else:
                t_j[pu[i][4]-1]=rg.triangular(20,30,40)
                t_j[pu[i][5]-1]=rg.triangular(20,30,40)
        if p_i[i]==2:
            t_j[pu[i][0]-1]=rg.triangular(18,28,45)
            t_j[pu[i][1]-1]=rg.triangular(10,20,25)
            t_j[pu[i][2]-1]=rg.triangular(3,5,15)
            t_j[pu[i][3]-1]=rg.triangular(5,10,20)
            if tipo_actividad_jr[pu[i][4]-1][0]==6:
                t_j[pu[i][4]-1]=15
                t_j[pu[i][5]-1]=rg.triangular(15,20,30)
                t_j[pu[i][6]-1]=rg.triangular(15,20,25)
            else:
                t_j[pu[i][4]-1]=rg.triangular(15,20,30)
                t_j[pu[i][5]-1]=rg.triangular(15,20,25)
        if p_i[i]==3:
            t_j[pu[i][0]-1]=rg.triangular(15,22,30)
            t_j[pu[i][1]-1]=rg.triangular(5,15,20)
            t_j[pu[i][2]-1]=rg.triangular(3,5,15)
            t_j[pu[i][3]-1]=rg.triangular(5,10,20)
            if tipo_actividad_jr[pu[i][4]-1][0]==6:
                t_j[pu[i][4]-1]=15
                t_j[pu[i][5]-1]=rg.triangular(10,15,20)
                t_j[pu[i][6]-1]=rg.triangular(8,15,18)
            else:
                t_j[pu[i][4]-1]=rg.triangular(10,15,20)
                t_j[pu[i][5]-1]=rg.triangular(8,15,18)
        if p_i[i]==4:
            t_j[pu[i][0]-1]=rg.triangular(5,10,20)
            t_j[pu[i][1]-1]=rg.triangular(2,3,5)
            t_j[pu[i][2]-1]=rg.triangular(2,5,10)
            t_j[pu[i][3]-1]=rg.triangular(5,8,12)
            t_j[pu[i][4]-1]=rg.triangular(5,10,15)
        if p_i[i]==5:
            t_j[pu[i][0]-1]=rg.triangular(2,4,8)
            t_j[pu[i][1]-1]=rg.triangular(2,3,5)
        for i in range(ficticios):
            t_j[J-1-i]=20 #descanso de 20 min

        i=I-ficticios-1
        pu_copia=np.insert(pu_copia,i,np.zeros(Amax,int),0)
        #se elige que avance tiene cada paciente
        inicio=rg.integers(0,ult_act[i]-1-pu[i][0])
        for j in range(inicio,Amax):
            pu_copia[i][j-inicio]=pu[i][j]
        for i in range(I-ficticios,I):
```

```
                pu_copia[i][0]=pu[i][0]
            r_i=np.zeros(I,int) # instante de llegada del paciente i
            for i in range(len(disp_i)):
                asignado=0
                for j in range(I-ficticios):
                    if tipo_actividad_jr[pu_copia[j][0]-2][0]==R[i] and
r_i[j]==0 and asignado==0:
                        r_i[j]=disp_i[i]
                        asignado=1
            pu=pu_copia.copy()

            sat=np.zeros(7) #n° tipo de recursos
            trab=np.zeros(7)
            disp=np.zeros(7)
            for i in range(I-ficticios):
                for j in range(Amax):
                    if pu[i][j]!=0 and pu[i][j]!=ult_act[i]:
                        trab[tipo_actividad_jr[pu[i][j]-1][0]-
1]+=t_j[pu[i][j]-1]
            for i in range(len(R)):
                for j in range(len(sat)):
                    if R[i]==j+1:
                        disp[j]+=disp_f[i]-disp_i[i]
            for i in range(len(sat)):
                sat[i]=trab[i]/disp[i]
            max_sat=np.amax(sat)
        print('La máxima saturación es ',max_sat,' en el recurso ',
np.where(sat==max_sat))

        secuencia_act,secuencia_paciente,C_rj,f_j,FO_p,seg,sat,FO1,FO2=alg
oritmo.neh(disp_i,disp_f,I,Amax,p_i, tipo_actividad_jr, r_i, J, R, a_j,
t_j, Y_r, pu,pu_prima, tepcof, ult_act)
        resultados[inst][0]=FO_p
        resultados[inst][-1]=saturaciones[y]
        f_obj_1[inst][0]=FO1
        f_obj_1[inst][-1]=saturaciones[y]
        f_obj_2[inst][0]=FO2
        f_obj_2[inst][-1]=saturaciones[y]
        iteraciones[inst][-1]=saturaciones[y]
        for i in range(I):
            for j in range(Amax):
                if pu[i][j]!=0:
                    if f_j[pu[i][j]-1]==0:
                        print()
                        print('ERROR AL ASIGNAR')
                        print()
                        print(pu[i][j])
                        salir=1
```

```
        C_rj,f_j,seg,sat,FO_p1,secuencia_act1,it,FO1,FO2=algoritmo.greedy_
pac(disp_i,disp_f,secuencia_paciente,secuencia_act,I,Amax,p_i,
tipo_actividad_jr, r_i, J, R, a_j, t_j, Y_r, pu,pu_prima, tepcof,
ult_act,FO_p)
        resultados[inst][1]=FO_p1
        f_obj_1[inst][1]=FO1
        f_obj_2[inst][1]=FO2
        iteraciones[inst][1]=it
        for i in range(I):
            for j in range(Amax):
                if pu[i][j]!=0:
                    if f_j[pu[i][j]-1]==0:
                        print()
                        print('ERROR AL ASIGNAR')
                        print()
                        print(pu[i][j])
                        salir=1
        C_rj,f_j,seg,sat,FO_p1,secuencia_act1,it,FO1,FO2=algoritmo.greedy_
bl_pac(disp_i,disp_f,secuencia_paciente,secuencia_act,I,Amax,p_i,
tipo_actividad_jr, r_i, J, R, a_j, t_j, Y_r, pu,pu_prima, tepcof,
ult_act,FO_p)
        resultados[inst][2]=FO_p1
        f_obj_1[inst][2]=FO1
        f_obj_2[inst][2]=FO2
        iteraciones[inst][2]=it
        for i in range(I):
            for j in range(Amax):
                if pu[i][j]!=0:
                    if f_j[pu[i][j]-1]==0:
                        print()
                        print('ERROR AL ASIGNAR')
                        print()
                        print(pu[i][j])
                        salir=1
        C_rj,f_j,seg,sat,FO_p1,secuencia_act1,it,FO1,FO2=algoritmo.greedy_
sa_pac(disp_i,disp_f,secuencia_paciente,secuencia_act,I,Amax,p_i,
tipo_actividad_jr, r_i, J, R, a_j, t_j, Y_r, pu,pu_prima, tepcof,
ult_act,FO_p)
        resultados[inst][3]=FO_p1
        f_obj_1[inst][3]=FO1
        f_obj_2[inst][3]=FO2
        iteraciones[inst][3]=it
        for i in range(I):
            for j in range(Amax):
                if pu[i][j]!=0:
                    if f_j[pu[i][j]-1]==0:
                        print()
                        print('ERROR AL ASIGNAR')
```

```
                        print()
                        print(pu[i][j])
                        salir=1
        C_rj,f_j,seg,sat,FO_p1,secuencia_act1,it,FO1,FO2=algoritmo.greedy_
bl_sa_pac(disp_i,disp_f,secuencia_paciente,secuencia_act,I,Amax,p_i,
tipo_actividad_jr, r_i, J, R, a_j, t_j, Y_r, pu,pu_prima, tepcof,
ult_act,FO_p)
        resultados[inst][4]=FO_p1
        f_obj_1[inst][4]=FO1
        f_obj_2[inst][4]=FO2
        iteraciones[inst][4]=it
        for i in range(I):
            for j in range(Amax):
                if pu[i][j]!=0:
                    if f_j[pu[i][j]-1]==0:
                        print()
                        print('ERROR AL ASIGNAR')
                        print()
                        print(pu[i][j])
                        salir=1
        C_rj,f_j,seg,sat,FO_p1,secuencia_act1,it,FO1,FO2=algoritmo.greedy_
act(disp_i,disp_f,I,secuencia_act,Amax,p_i, tipo_actividad_jr, r_i, J, R,
a_j, t_j, Y_r, pu,pu_prima, tepcof, ult_act,FO_p,secuencia_act)
        resultados[inst][5]=FO_p1
        f_obj_1[inst][5]=FO1
        f_obj_2[inst][5]=FO2
        iteraciones[inst][5]=it
        for i in range(I):
            for j in range(Amax):
                if pu[i][j]!=0:
                    if f_j[pu[i][j]-1]==0:
                        print()
                        print('ERROR AL ASIGNAR')
                        print()
                        print(pu[i][j])
                        salir=1
        C_rj,f_j,seg,sat,FO_p1,secuencia_act1,it,FO1,FO2=algoritmo.greedy_
bl_act(disp_i,disp_f,I,secuencia_act,Amax,p_i, tipo_actividad_jr, r_i, J,
R, a_j, t_j, Y_r, pu,pu_prima, tepcof, ult_act,FO_p,secuencia_act)
        resultados[inst][6]=FO_p1
        f_obj_1[inst][6]=FO1
        f_obj_2[inst][6]=FO2
        iteraciones[inst][6]=it
        for i in range(I):
            for j in range(Amax):
                if pu[i][j]!=0:
                    if f_j[pu[i][j]-1]==0:
                        print()
                        print('ERROR AL ASIGNAR')
```

```
                        print()
                        print(pu[i][j])
                        salir=1
        C_rj,f_j,seg,sat,FO_p1,secuencia_act1,it,FO1,FO2=algoritmo.greedy_
sa_act(disp_i,disp_f,I,secuencia_act,Amax,p_i, tipo_actividad_jr, r_i, J,
R, a_j, t_j, Y_r, pu,pu_prima, tepcof, ult_act,FO_p,secuencia_act)
        resultados[inst][7]=FO_p1
        f_obj_1[inst][7]=FO1
        f_obj_2[inst][7]=FO2
        iteraciones[inst][7]=it
        for i in range(I):
            for j in range(Amax):
                if pu[i][j]!=0:
                    if f_j[pu[i][j]-1]==0:
                        print()
                        print('ERROR AL ASIGNAR')
                        print()
                        print(pu[i][j])
                        salir=1
        C_rj,f_j,seg,sat,FO_p1,secuencia_act1,it,FO1,FO2=algoritmo.greedy_
bl_sa_act(disp_i,disp_f,I,secuencia_act,Amax,p_i, tipo_actividad_jr, r_i,
J, R, a_j, t_j, Y_r, pu,pu_prima, tepcof, ult_act,FO_p,secuencia_act)
        resultados[inst][8]=FO_p1
        f_obj_1[inst][8]=FO1
        f_obj_2[inst][8]=FO2
        iteraciones[inst][8]=it
        for i in range(I):
            for j in range(Amax):
                if pu[i][j]!=0:
                    if f_j[pu[i][j]-1]==0:
                        print()
                        print('ERROR AL ASIGNAR')
                        print()
                        print(pu[i][j])
                        salir=1
        inst+=1
        if salir==1:
            print('SALIDA POR ERROR')
            break
    if salir==1:
        print('SALIDA POR ERROR')
        break

col=['NEH','paciente basico','paciente busqueda local','paciente
s.a.','paciente s.a. y b.l.','act basico','act busqueda local','act
s.a.','act s.a. y b.l.','Saturacion']
data_null={}
df_null=pd.DataFrame(data_null)
```

```
df_null.to_excel('simulacion_recursos_6_act.xlsx',index=False)
writer=pd.ExcelWriter('simulacion_recursos_6_act.xlsx')
data = pd.DataFrame(resultados,None,col)
data2 = pd.DataFrame(iteraciones,None,col)
data3= pd.DataFrame(f_obj_1,None,col)
data4 = pd.DataFrame(f_obj_2,None,col)
data.to_excel(writer, sheet_name='resultados', index=False)
data2.to_excel(writer, sheet_name='iteraciones', index=False)
data3.to_excel(writer, sheet_name='FO1_tepcof', index=False)
data4.to_excel(writer, sheet_name='FO2', index=False)
writer.save()
writer.close()
```

Algoritmos

```
import numpy as np
import asignacion
import time
import busqueda_local
import sim_annealing
rg=np.random.default_rng(2)

#SECUENCIA INICIAL POR NEH con pacientes
def neh(disp_i,disp_f,I,Amax,p_i, tipo_actividad_jr, r_i, J, R, a_j, t_j,
Y_r, pu,pu_prima, tepcof, ult_act):
    n_act=0
    for i in range(I):
        for j in range(Amax):
            if pu[i][j]!=0:
                n_act+=1
    tiempo_total=np.zeros(I,int)
    secuencia_paciente=np.linspace(1,I,I,dtype=int)
    for j in range(I):
        suma=0
        for i in range(Amax):
            if pu[j][i]!=0:
                suma+=t_j[pu[j][i]-1]
        tiempo_total[j]=suma
    for k in range(I-1):
        for m in range(I-1):
            if (tiempo_total[secuencia_paciente[m]-
1])<(tiempo_total[secuencia_paciente[m+1]-1]):
                paciente=secuencia_paciente[m]
                secuencia_paciente=np.delete(secuencia_paciente, m)
                secuencia_paciente=np.insert(secuencia_paciente, m+1,
paciente)
    secuencia_prov=np.zeros(1,int)
    secuencia_prov[0]=secuencia_paciente[0]
```

```
    secuencia_aux=secuencia_prov.copy()
    FO_p1=0
    FO_p2=3000
    secuencia_inicial=np.zeros(2,int)
    for j in range(I):
        if j>0:
            for k in range(len(secuencia_prov)+1):
                secuencia_aux=np.insert(secuencia_prov, k,
secuencia_paciente[j])
                cont=0
                for i in range(len(secuencia_aux)):
                    for m in range(Amax):
                        if pu[secuencia_aux[i]-1][m]!=0:
                            cont+=1
                secuencia_act_aux=np.zeros(cont,int)
                w=0
                for i in range(len(secuencia_aux)):
                    for m in range(Amax):
                        if pu[secuencia_aux[i]-1][m]!=0:
                            secuencia_act_aux[w]=pu[secuencia_aux[i]-1][m]
                            w=w+1
                if k==0:
                    C_rj1,f_j1,FO11,FO21,seg,sat=asignacion.algoritmo(disp
_i,disp_f,p_i,secuencia_act_aux, tipo_actividad_jr, r_i, J, R, a_j, t_j,
Y_r, pu,pu_prima, tepcof, ult_act)
                    FO_p1=FO11+FO21
                    secuencia_inicial=secuencia_aux.copy()
                else:
                    C_rj2,f_j2,FO12,FO22,seg,sat=asignacion.algoritmo(disp
_i,disp_f,p_i,secuencia_act_aux, tipo_actividad_jr, r_i, J, R, a_j, t_j,
Y_r, pu,pu_prima, tepcof, ult_act)
                    FO_p2=FO12+FO22
                if FO_p2<FO_p1:
                    secuencia_inicial=secuencia_aux.copy()
                    FO_p1=FO_p2
            secuencia_prov=secuencia_inicial.copy()
            FO_p1=0
            FO_p2=3000
    secuencia_paciente=secuencia_prov.copy()
    secuencia_act=np.zeros(n_act,int)
    w=0
    for j in range(I):
        for k in range(Amax):
            if pu[secuencia_paciente[j]-1][k]!=0:
                secuencia_act[w]=pu[secuencia_paciente[j]-1][k]
                w=w+1
```

```
    C_rj,f_j,FO1,FO2,seg,sat=asignacion.algoritmo(disp_i,disp_f,p_i,
secuencia_act, tipo_actividad_jr, r_i, J, R, a_j, t_j, Y_r, pu,pu_prima,
tepcof, ult_act)
    FO_p=FO1+FO2
    return secuencia_act,secuencia_paciente,C_rj,f_j,FO_p,seg,sat,FO1,FO2


#GREEDY PACIENTES
def greedy_pac(disp_i,disp_f,secuencia_paciente,secuencia_act,I,Amax,p_i,
tipo_actividad_jr, r_i, J, R, a_j, t_j, Y_r, pu,pu_prima, tepcof,
ult_act,FO_p):
    n_act=0
    for i in range(I):
        for j in range(Amax):
            if pu[i][j]!=0:
                n_act+=1
    secuencia_gr=secuencia_paciente.copy()
    C_rj_G,f_j_G,FO1_G,FO2_G,seg_G,sat=asignacion.algoritmo(disp_i,disp_f,
p_i, secuencia_act, tipo_actividad_jr, r_i, J, R, a_j, t_j, Y_r,
pu,pu_prima, tepcof, ult_act)
    FO_G=FO2_G+FO1_G
    it=0
    crono_inicio=time.time()
    crono_parada=n_act*(len(R)-1)*375/(2*1000)
    while True:
        secuencia_prov=secuencia_gr.copy()
        #DESTRUCCION
        d=4 #parametro d a elegir en funcion de la mejor FO
        pi_d=np.zeros(d,int)
        for k in range(d):
            pos=rg.integers(len(secuencia_prov))
            #while p_i[secuencia_prov[pos]-1]==6:
                #pos=rg.integers(len(secuencia_prov))
            pi_d[k]=secuencia_prov[pos]
            secuencia_prov=np.delete(secuencia_prov, pos)
        #CONSTRUCCION
        secuencia_aux=secuencia_prov.copy()
        FO_p1=0
        FO_p2=3000
        secuencia_inicial=np.zeros(2,int)
        for j in range(d):
            for k in range(len(secuencia_prov)+1):
                secuencia_aux=np.insert(secuencia_prov, k, pi_d[j])
                cont=0
                for i in range(len(secuencia_aux)):
                    for m in range(Amax):
                        if pu[secuencia_aux[i]-1][m]!=0:
                            cont+=1
                secuencia_act_aux=np.zeros(cont,int)
```

```
                w=0
                for i in range(len(secuencia_aux)):
                    for m in range(Amax):
                        if pu[secuencia_aux[i]-1][m]!=0:
                            secuencia_act_aux[w]=pu[secuencia_aux[i]-1][m]
                            w=w+1
                if k==0:
                    C_rj1,f_j1,FO11,FO21,seg,sat=asignacion.algoritmo(disp
_i,disp_f,p_i,secuencia_act_aux, tipo_actividad_jr, r_i, J, R, a_j, t_j,
Y_r, pu,pu_prima, tepcof, ult_act)
                    FO_p1=FO11+FO21
                    secuencia_inicial=secuencia_aux.copy()
                else:
                    C_rj2,f_j2,FO12,FO22,seg,sat=asignacion.algoritmo(disp
_i,disp_f,p_i,secuencia_act_aux, tipo_actividad_jr, r_i, J, R, a_j, t_j,
Y_r, pu,pu_prima, tepcof, ult_act)
                    FO_p2=FO12+FO22
                if FO_p2<FO_p1:
                    secuencia_inicial=secuencia_aux.copy()
                    FO_p1=FO_p2
            secuencia_prov=secuencia_inicial.copy()
            FO_p1=0
            FO_p2=3000
        secuencia_act_prov=np.zeros(n_act,int)
        w=0
        for j in range(I):
            for k in range(Amax):
                if pu[secuencia_prov[j]-1][k]!=0:
                    secuencia_act_prov[w]=pu[secuencia_prov[j]-1][k]
                    w=w+1
        C_rj_G,f_j_G,FO1_G,FO2_G,seg_G,sat=asignacion.algoritmo(disp_i,dis
p_f,p_i, secuencia_act_prov, tipo_actividad_jr, r_i, J, R, a_j, t_j, Y_r,
pu,pu_prima, tepcof, ult_act)
        FO_G1=FO2_G+FO1_G
        #FIN CONSTRUCCION
        if FO_G1<FO_G:
            secuencia_gr=secuencia_prov.copy()
            FO_G=FO_G1
            if FO_G<FO_p:
                secuencia_paciente=secuencia_gr.copy()
                secuencia_act=secuencia_act_prov.copy()
                FO_p=FO_G
                C_rj,f_j,FO1,FO2,seg,sat=asignacion.algoritmo(disp_i,disp_
f,p_i, secuencia_act, tipo_actividad_jr, r_i, J, R, a_j, t_j, Y_r,
pu,pu_prima, tepcof, ult_act)
        it+=1
        if time.time()>(crono_inicio+crono_parada):
```

```
            C_rj,f_j,FO1,FO2,seg,sat=asignacion.algoritmo(disp_i,disp_f,p_
i, secuencia_act, tipo_actividad_jr, r_i, J, R, a_j, t_j, Y_r,
pu,pu_prima, tepcof, ult_act)
            FO_p=FO1+FO2

            break
    return C_rj,f_j,seg,sat,FO_p,secuencia_act,it,FO1,FO2

def
greedy_bl_pac(disp_i,disp_f,secuencia_paciente,secuencia_act,I,Amax,p_i,
tipo_actividad_jr, r_i, J, R, a_j, t_j, Y_r, pu,pu_prima, tepcof,
ult_act,FO_p):
    n_act=0
    for i in range(I):
        for j in range(Amax):
            if pu[i][j]!=0:
                n_act+=1
    secuencia_gr=secuencia_paciente.copy()
    C_rj_G,f_j_G,FO1_G,FO2_G,seg_G,sat=asignacion.algoritmo(disp_i,disp_f,
p_i, secuencia_act, tipo_actividad_jr, r_i, J, R, a_j, t_j, Y_r,
pu,pu_prima, tepcof, ult_act)
    FO_G=FO2_G+FO1_G
    it=0
    crono_inicio=time.time()
    crono_parada=n_act*(len(R)-1)*375/(2*1000)
    while True:
        secuencia_prov=secuencia_gr.copy()
        #DESTRUCCION
        d=4 #parametro d a elegir en funcion de la mejor FO
        pi_d=np.zeros(d,int)
        for k in range(d):
            pos=rg.integers(len(secuencia_prov))
            #while p_i[secuencia_prov[pos]-1]==6:
                #pos=rg.integers(len(secuencia_prov))
            pi_d[k]=secuencia_prov[pos]
            secuencia_prov=np.delete(secuencia_prov, pos)
        #CONSTRUCCION
        secuencia_aux=secuencia_prov.copy()
        FO_p1=0
        FO_p2=3000
        secuencia_inicial=np.zeros(2,int)
        for j in range(d):
            for k in range(len(secuencia_prov)+1):
                secuencia_aux=np.insert(secuencia_prov, k, pi_d[j])
                cont=0
                for i in range(len(secuencia_aux)):
                    for m in range(Amax):
                        if pu[secuencia_aux[i]-1][m]!=0:
                            cont+=1
```

```
                secuencia_act_aux=np.zeros(cont,int)
                w=0
                for i in range(len(secuencia_aux)):
                    for m in range(Amax):
                        if pu[secuencia_aux[i]-1][m]!=0:
                            secuencia_act_aux[w]=pu[secuencia_aux[i]-1][m]
                            w=w+1
                if k==0:
                    C_rj1,f_j1,FO11,FO21,seg,sat=asignacion.algoritmo(disp
_i,disp_f,p_i,secuencia_act_aux, tipo_actividad_jr, r_i, J, R, a_j, t_j,
Y_r, pu,pu_prima, tepcof, ult_act)
                    FO_p1=FO11+FO21
                    secuencia_inicial=secuencia_aux.copy()
                else:
                    C_rj2,f_j2,FO12,FO22,seg,sat=asignacion.algoritmo(disp
_i,disp_f,p_i,secuencia_act_aux, tipo_actividad_jr, r_i, J, R, a_j, t_j,
Y_r, pu,pu_prima, tepcof, ult_act)
                    FO_p2=FO12+FO22
                if FO_p2<FO_p1:
                    secuencia_inicial=secuencia_aux.copy()
                    FO_p1=FO_p2
            secuencia_prov=secuencia_inicial.copy()
            FO_p1=0
            FO_p2=3000
        secuencia_act_prov=np.zeros(n_act,int)
        w=0
        for j in range(I):
            for k in range(Amax):
                if pu[secuencia_prov[j]-1][k]!=0:
                    secuencia_act_prov[w]=pu[secuencia_prov[j]-1][k]
                    w=w+1
        C_rj_G,f_j_G,FO1_G,FO2_G,seg_G,sat=asignacion.algoritmo(disp_i,dis
p_f,p_i, secuencia_act_prov, tipo_actividad_jr, r_i, J, R, a_j, t_j, Y_r,
pu,pu_prima, tepcof, ult_act)
        FO_G1=FO2_G+FO1_G
        #FIN CONSTRUCCION
        # BUSQUEDA LOCAL
        secuencia_prov=busqueda_local.pacientes(disp_i,disp_f,secuencia_pr
ov, p_i, secuencia_act_prov, tipo_actividad_jr, r_i, J, R, a_j, t_j, Y_r,
pu,pu_prima, tepcof, ult_act)
        secuencia_act_prov=np.zeros(n_act,int)
        w=0
        for j in range(I):
            for k in range(Amax):
                if pu[secuencia_prov[j]-1][k]!=0:
                    secuencia_act_prov[w]=pu[secuencia_prov[j]-1][k]
                    w=w+1
```

```
        C_rj_G,f_j_G,FO1_G,FO2_G,seg_G,sat=asignacion.algoritmo(disp_i,dis
p_f,p_i, secuencia_act_prov, tipo_actividad_jr, r_i, J, R, a_j, t_j, Y_r,
pu,pu_prima, tepcof, ult_act)
        FO_G1=FO2_G+FO1_G
        #FIN BUSQUEDA
        if FO_G1<FO_G:
            secuencia_gr=secuencia_prov.copy()
            FO_G=FO_G1
            if FO_G<FO_p:
                secuencia_paciente=secuencia_gr.copy()
                secuencia_act=secuencia_act_prov.copy()
                FO_p=FO_G
                C_rj,f_j,FO1,FO2,seg,sat=asignacion.algoritmo(disp_i,disp_
f,p_i, secuencia_act, tipo_actividad_jr, r_i, J, R, a_j, t_j, Y_r,
pu,pu_prima, tepcof, ult_act)

        it+=1
        if time.time()>(crono_inicio+crono_parada):
            C_rj,f_j,FO1,FO2,seg,sat=asignacion.algoritmo(disp_i,disp_f,p_
i, secuencia_act, tipo_actividad_jr, r_i, J, R, a_j, t_j, Y_r,
pu,pu_prima, tepcof, ult_act)
            FO_p=FO1+FO2

            break
    return C_rj,f_j,seg,sat,FO_p,secuencia_act,it,FO1,FO2

def
greedy_sa_pac(disp_i,disp_f,secuencia_paciente,secuencia_act,I,Amax,p_i,
tipo_actividad_jr, r_i, J, R, a_j, t_j, Y_r, pu,pu_prima, tepcof,
ult_act,FO_p):
    n_act=0
    for i in range(I):
        for j in range(Amax):
            if pu[i][j]!=0:
                n_act+=1
    secuencia_gr=secuencia_paciente.copy()
    C_rj_G,f_j_G,FO1_G,FO2_G,seg_G,sat=asignacion.algoritmo(disp_i,disp_f,
p_i, secuencia_act, tipo_actividad_jr, r_i, J, R, a_j, t_j, Y_r,
pu,pu_prima, tepcof, ult_act)
    FO_G=FO2_G+FO1_G
    it=0
    crono_inicio=time.time()
    crono_parada=n_act*(len(R)-1)*375/(2*1000)
    while True:
        secuencia_prov=secuencia_gr.copy()
        #DESTRUCCION
        d=4 #parametro d a elegir en funcion de la mejor FO
        pi_d=np.zeros(d,int)
        for k in range(d):
```

```
            pos=rg.integers(len(secuencia_prov))
            #while p_i[secuencia_prov[pos]-1]==6:
                #pos=rg.integers(len(secuencia_prov))
            pi_d[k]=secuencia_prov[pos]
            secuencia_prov=np.delete(secuencia_prov, pos)
        #CONSTRUCCION
        secuencia_aux=secuencia_prov.copy()
        FO_p1=0
        FO_p2=3000
        secuencia_inicial=np.zeros(2,int)
        for j in range(d):
            for k in range(len(secuencia_prov)+1):
                secuencia_aux=np.insert(secuencia_prov, k, pi_d[j])
                cont=0
                for i in range(len(secuencia_aux)):
                    for m in range(Amax):
                        if pu[secuencia_aux[i]-1][m]!=0:
                            cont+=1
                secuencia_act_aux=np.zeros(cont,int)
                w=0
                for i in range(len(secuencia_aux)):
                    for m in range(Amax):
                        if pu[secuencia_aux[i]-1][m]!=0:
                            secuencia_act_aux[w]=pu[secuencia_aux[i]-1][m]
                            w=w+1
                if k==0:
                    C_rj1,f_j1,FO11,FO21,seg,sat=asignacion.algoritmo(disp
_i,disp_f,p_i,secuencia_act_aux, tipo_actividad_jr, r_i, J, R, a_j, t_j,
Y_r, pu,pu_prima, tepcof, ult_act)
                    FO_p1=FO11+FO21
                    secuencia_inicial=secuencia_aux.copy()
                else:
                    C_rj2,f_j2,FO12,FO22,seg,sat=asignacion.algoritmo(disp
_i,disp_f,p_i,secuencia_act_aux, tipo_actividad_jr, r_i, J, R, a_j, t_j,
Y_r, pu,pu_prima, tepcof, ult_act)
                    FO_p2=FO12+FO22
                if FO_p2<FO_p1:
                    secuencia_inicial=secuencia_aux.copy()
                    FO_p1=FO_p2
            secuencia_prov=secuencia_inicial.copy()
            FO_p1=0
            FO_p2=3000
        secuencia_act_prov=np.zeros(n_act,int)
        w=0
        for j in range(I):
            for k in range(Amax):
                if pu[secuencia_prov[j]-1][k]!=0:
                    secuencia_act_prov[w]=pu[secuencia_prov[j]-1][k]
```

```
                    w=w+1
        C_rj_G,f_j_G,FO1_G,FO2_G,seg_G,sat=asignacion.algoritmo(disp_i,dis
p_f,p_i, secuencia_act_prov, tipo_actividad_jr, r_i, J, R, a_j, t_j, Y_r,
pu,pu_prima, tepcof, ult_act)
        FO_G1=FO2_G+FO1_G
        #FIN CONSTRUCCION
        if FO_G1<FO_G:
            secuencia_gr=secuencia_prov.copy()
            FO_G=FO_G1
            if FO_G<FO_p:
                secuencia_paciente=secuencia_gr.copy()
                secuencia_act=secuencia_act_prov.copy()
                FO_p=FO_G
                C_rj,f_j,FO1,FO2,seg,sat=asignacion.algoritmo(disp_i,disp_
f,p_i, secuencia_act, tipo_actividad_jr, r_i, J, R, a_j, t_j, Y_r,
pu,pu_prima, tepcof, ult_act)

        #SIMULATED ANNEALING
        else:
            cambio=sim_annealing.recocido_paciente(I,R,it,FO_G,FO_G1,Amax,
pu,t_j)
            if cambio==1:
                secuencia_gr=secuencia_prov.copy()
                FO_G=FO_G1
        #FIN S.A.
        it+=1
        if time.time()>(crono_inicio+crono_parada):
            C_rj,f_j,FO1,FO2,seg,sat=asignacion.algoritmo(disp_i,disp_f,p_
i, secuencia_act, tipo_actividad_jr, r_i, J, R, a_j, t_j, Y_r,
pu,pu_prima, tepcof, ult_act)
            FO_p=FO1+FO2
            break
    return C_rj,f_j,seg,sat,FO_p,secuencia_act,it,FO1,FO2

def
greedy_bl_sa_pac(disp_i,disp_f,secuencia_paciente,secuencia_act,I,Amax,p_i
, tipo_actividad_jr, r_i, J, R, a_j, t_j, Y_r, pu,pu_prima, tepcof,
ult_act,FO_p):
    n_act=0
    for i in range(I):
        for j in range(Amax):
            if pu[i][j]!=0:
                n_act+=1
    secuencia_gr=secuencia_paciente.copy()
    C_rj_G,f_j_G,FO1_G,FO2_G,seg_G,sat=asignacion.algoritmo(disp_i,disp_f,
p_i, secuencia_act, tipo_actividad_jr, r_i, J, R, a_j, t_j, Y_r,
pu,pu_prima, tepcof, ult_act)
    FO_G=FO2_G+FO1_G
    it=0
```

```
    crono_inicio=time.time()
    crono_parada=n_act*(len(R)-1)*375/(2*1000)
    while True:
        secuencia_prov=secuencia_gr.copy()
        #DESTRUCCION
        d=4 #parametro d a elegir en funcion de la mejor FO
        pi_d=np.zeros(d,int)
        for k in range(d):
            pos=rg.integers(len(secuencia_prov))
            #while p_i[secuencia_prov[pos]-1]==6:
                #pos=rg.integers(len(secuencia_prov))
            pi_d[k]=secuencia_prov[pos]
            secuencia_prov=np.delete(secuencia_prov, pos)
        #CONSTRUCCION
        secuencia_aux=secuencia_prov.copy()
        FO_p1=0
        FO_p2=3000
        secuencia_inicial=np.zeros(2,int)
        for j in range(d):
            for k in range(len(secuencia_prov)+1):
                secuencia_aux=np.insert(secuencia_prov, k, pi_d[j])
                cont=0
                for i in range(len(secuencia_aux)):
                    for m in range(Amax):
                        if pu[secuencia_aux[i]-1][m]!=0:
                            cont+=1
                secuencia_act_aux=np.zeros(cont,int)
                w=0
                for i in range(len(secuencia_aux)):
                    for m in range(Amax):
                        if pu[secuencia_aux[i]-1][m]!=0:
                            secuencia_act_aux[w]=pu[secuencia_aux[i]-1][m]
                            w=w+1
                if k==0:
                    C_rj1,f_j1,FO11,FO21,seg,sat=asignacion.algoritmo(disp
_i,disp_f,p_i,secuencia_act_aux, tipo_actividad_jr, r_i, J, R, a_j, t_j,
Y_r, pu,pu_prima, tepcof, ult_act)
                    FO_p1=FO11+FO21
                    secuencia_inicial=secuencia_aux.copy()
                else:
                    C_rj2,f_j2,FO12,FO22,seg,sat=asignacion.algoritmo(disp
_i,disp_f,p_i,secuencia_act_aux, tipo_actividad_jr, r_i, J, R, a_j, t_j,
Y_r, pu,pu_prima, tepcof, ult_act)
                    FO_p2=FO12+FO22
                if FO_p2<FO_p1:
                    secuencia_inicial=secuencia_aux.copy()
                    FO_p1=FO_p2
            secuencia_prov=secuencia_inicial.copy()
```

```
            FO_p1=0
            FO_p2=3000
        secuencia_act_prov=np.zeros(n_act,int)
        w=0
        for j in range(I):
            for k in range(Amax):
                if pu[secuencia_prov[j]-1][k]!=0:
                    secuencia_act_prov[w]=pu[secuencia_prov[j]-1][k]
                    w=w+1
        C_rj_G,f_j_G,FO1_G,FO2_G,seg_G,sat=asignacion.algoritmo(disp_i,dis
p_f,p_i, secuencia_act_prov, tipo_actividad_jr, r_i, J, R, a_j, t_j, Y_r,
pu,pu_prima, tepcof, ult_act)
        FO_G1=FO2_G+FO1_G
        #FIN CONSTRUCCION
        # BUSQUEDA LOCAL
        secuencia_prov=busqueda_local.pacientes(disp_i,disp_f,secuencia_pr
ov, p_i, secuencia_act_prov, tipo_actividad_jr, r_i, J, R, a_j, t_j, Y_r,
pu,pu_prima, tepcof, ult_act)
        secuencia_act_prov=np.zeros(n_act,int)
        w=0
        for j in range(I):
            for k in range(Amax):
                if pu[secuencia_prov[j]-1][k]!=0:
                    secuencia_act_prov[w]=pu[secuencia_prov[j]-1][k]
                    w=w+1
        C_rj_G,f_j_G,FO1_G,FO2_G,seg_G,sat=asignacion.algoritmo(disp_i,dis
p_f,p_i, secuencia_act_prov, tipo_actividad_jr, r_i, J, R, a_j, t_j, Y_r,
pu,pu_prima, tepcof, ult_act)
        FO_G1=FO2_G+FO1_G
        #FIN BUSQUEDA
        if FO_G1<FO_G:
            secuencia_gr=secuencia_prov.copy()
            FO_G=FO_G1
            if FO_G<FO_p:
                secuencia_paciente=secuencia_gr.copy()
                secuencia_act=secuencia_act_prov.copy()
                FO_p=FO_G
                C_rj,f_j,FO1,FO2,seg,sat=asignacion.algoritmo(disp_i,disp_
f,p_i, secuencia_act, tipo_actividad_jr, r_i, J, R, a_j, t_j, Y_r,
pu,pu_prima, tepcof, ult_act)

        #SIMULATED ANNEALING
        else:
            cambio=sim_annealing.recocido_paciente(I,R,it,FO_G,FO_G1,Amax,
pu,t_j)
            if cambio==1:
                secuencia_gr=secuencia_prov.copy()
                FO_G=FO_G1
        #FIN S.A.
```

```
        it+=1
        if time.time()>(crono_inicio+crono_parada):
            C_rj,f_j,FO1,FO2,seg,sat=asignacion.algoritmo(disp_i,disp_f,p_
i, secuencia_act, tipo_actividad_jr, r_i, J, R, a_j, t_j, Y_r,
pu,pu_prima, tepcof, ult_act)
            FO_p=FO1+FO2
            break
    return C_rj,f_j,seg,sat,FO_p,secuencia_act,it,FO1,FO2

#GREEDY ACTIVIDADES
#se parte del NEH por pacientes
def greedy_act(disp_i,disp_f,I,secuencia_act_neh,Amax,p_i,
tipo_actividad_jr, r_i, J, R, a_j, t_j, Y_r, pu,pu_prima, tepcof,
ult_act,FO_p,secuencia_act):
    it=0
    n_act=0
    for i in range(I):
        for j in range(Amax):
            if pu[i][j]!=0:
                n_act+=1
    secuencia_gr=secuencia_act_neh.copy()
    C_rj,f_j,FO1,FO2,seg,sat=asignacion.algoritmo(disp_i,disp_f,p_i,
secuencia_gr, tipo_actividad_jr, r_i, J, R, a_j, t_j, Y_r, pu,pu_prima,
tepcof, ult_act)
    FO_G=FO1+FO2
    crono_inicio=time.time()
    crono_parada=n_act*(len(R)-1)*375/(2*1000)
    while True:
        secuencia_act_neh=secuencia_gr.copy()
        #DESTRUCCION
        d=10 #parametro a elegir
        pi_d=np.zeros(d,int)
        for k in range(d):
            pos=rg.integers(len(secuencia_act_neh))
            while p_i[a_j[secuencia_act_neh[pos]-1]-1]==6:
                pos=rg.integers(len(secuencia_act_neh))
            pi_d[k]=secuencia_act_neh[pos]
            secuencia_act_neh=np.delete(secuencia_act_neh, pos)
        #CONSTRUCCION
        FO_p1=0
        FO_p2=3000
        secuencia_inicial=np.zeros(2,int)
        secuencia_inicial=np.zeros(2,int)
        for j in range(d):
            act=0
            pos1=0
            act2=J
            pos2=len(secuencia_act_neh)+1
```

```
            for k in range(len(secuencia_act_neh)):
                if secuencia_act_neh[k]<pi_d[j] and
secuencia_act_neh[k]>=pu[a_j[pi_d[j]-1]-1][0]:
                    if secuencia_act_neh[k]>=act:
                        pos1=k+1
                        act=secuencia_act_neh[k]
                if secuencia_act_neh[k]>pi_d[j] and
secuencia_act_neh[k]<=ult_act[a_j[pi_d[j]-1]-1]:
                    if secuencia_act_neh[k]<=act2:
                      pos2=k+1
                      act2=secuencia_act_neh[k]
            for k in range(pos1,pos2,1):
                secuencia_act_aux=np.insert(secuencia_act_neh, k, pi_d[j])
                if k==pos1:
                    C_rj1,f_j1,FO11,FO21,seg,sat=asignacion.algoritmo(disp
_i,disp_f,p_i,secuencia_act_aux, tipo_actividad_jr, r_i, J, R, a_j, t_j,
Y_r, pu,pu_prima, tepcof, ult_act)
                    FO_p1=FO11+FO21
                    secuencia_inicial=secuencia_act_aux.copy()
                else:
                    C_rj2,f_j2,FO12,FO22,seg,sat=asignacion.algoritmo(disp
_i,disp_f,p_i,secuencia_act_aux, tipo_actividad_jr, r_i, J, R, a_j, t_j,
Y_r, pu,pu_prima, tepcof, ult_act)
                    FO_p2=FO12+FO22
                if t_j[pi_d[j]-1]==0:
                    if FO_p2<=FO_p1:
                        secuencia_inicial=secuencia_act_aux.copy()
                        FO_p1=FO_p2
                else:
                    if FO_p2<FO_p1:
                        secuencia_inicial=secuencia_act_aux.copy()
                        FO_p1=FO_p2
            secuencia_act_neh=secuencia_inicial.copy()
            FO_p1=0
            FO_p2=3000
        C_rj_G,f_j_G,FO1_G,FO2_G,seg_G,sat=asignacion.algoritmo(disp_i,dis
p_f,p_i, secuencia_act_neh, tipo_actividad_jr, r_i, J, R, a_j, t_j, Y_r,
pu,pu_prima, tepcof, ult_act)
        FO_G1=FO2_G+FO1_G
        #FIN CONSTRUCCION
        if FO_G1<=FO_G:
            secuencia_gr=secuencia_act_neh.copy()
            FO_G=FO_G1
            if FO_G<FO_p:
                secuencia_act=secuencia_gr.copy()
                FO_p=FO_G
                C_rj,f_j,FO1,FO2,seg,sat=asignacion.algoritmo(disp_i,disp_
f,p_i, secuencia_act, tipo_actividad_jr, r_i, J, R, a_j, t_j, Y_r,
pu,pu_prima, tepcof, ult_act)
```

```
        it+=1
        if time.time()>(crono_inicio+crono_parada):
            C_rj,f_j,FO1,FO2,seg,sat=asignacion.algoritmo(disp_i,disp_f,p_
i, secuencia_act, tipo_actividad_jr, r_i, J, R, a_j, t_j, Y_r,
pu,pu_prima, tepcof, ult_act)
            FO_p=FO1+FO2
            break
    return C_rj,f_j,seg,sat,FO_p,secuencia_act,it,FO1,FO2

def greedy_bl_act(disp_i,disp_f,I,secuencia_act_neh,Amax,p_i,
tipo_actividad_jr, r_i, J, R, a_j, t_j, Y_r, pu,pu_prima, tepcof,
ult_act,FO_p,secuencia_act):
    it=0
    n_act=0
    for i in range(I):
        for j in range(Amax):
            if pu[i][j]!=0:
                n_act+=1
    secuencia_gr=secuencia_act_neh.copy()
    C_rj,f_j,FO1,FO2,seg,sat=asignacion.algoritmo(disp_i,disp_f,p_i,
secuencia_gr, tipo_actividad_jr, r_i, J, R, a_j, t_j, Y_r, pu,pu_prima,
tepcof, ult_act)
    FO_G=FO1+FO2
    crono_inicio=time.time()
    crono_parada=n_act*(len(R)-1)*375/(2*1000)
    while True:
        secuencia_act_neh=secuencia_gr.copy()
        #DESTRUCCION
        d=10 #parametro a elegir
        pi_d=np.zeros(d,int)
        for k in range(d):
            pos=rg.integers(len(secuencia_act_neh))
            while p_i[a_j[secuencia_act_neh[pos]-1]-1]==6:
                pos=rg.integers(len(secuencia_act_neh))
            pi_d[k]=secuencia_act_neh[pos]
            secuencia_act_neh=np.delete(secuencia_act_neh, pos)
        #CONSTRUCCION
        FO_p1=0
        FO_p2=3000
        secuencia_inicial=np.zeros(2,int)
        secuencia_inicial=np.zeros(2,int)
        for j in range(d):
            act=0
            pos1=0
            act2=J
            pos2=len(secuencia_act_neh)+1
            for k in range(len(secuencia_act_neh)):
```

```
                if secuencia_act_neh[k]<pi_d[j] and
secuencia_act_neh[k]>=pu[a_j[pi_d[j]-1]-1][0]:
                    if secuencia_act_neh[k]>=act:
                        pos1=k+1
                        act=secuencia_act_neh[k]
                if secuencia_act_neh[k]>pi_d[j] and
secuencia_act_neh[k]<=ult_act[a_j[pi_d[j]-1]-1]:
                    if secuencia_act_neh[k]<=act2:
                      pos2=k+1
                      act2=secuencia_act_neh[k]
            for k in range(pos1,pos2,1):
                secuencia_act_aux=np.insert(secuencia_act_neh, k, pi_d[j])
                if k==pos1:
                    C_rj1,f_j1,FO11,FO21,seg,sat=asignacion.algoritmo(disp
_i,disp_f,p_i,secuencia_act_aux, tipo_actividad_jr, r_i, J, R, a_j, t_j,
Y_r, pu,pu_prima, tepcof, ult_act)
                    FO_p1=FO11+FO21
                    secuencia_inicial=secuencia_act_aux.copy()
                else:
                    C_rj2,f_j2,FO12,FO22,seg,sat=asignacion.algoritmo(disp
_i,disp_f,p_i,secuencia_act_aux, tipo_actividad_jr, r_i, J, R, a_j, t_j,
Y_r, pu,pu_prima, tepcof, ult_act)
                    FO_p2=FO12+FO22
                if t_j[pi_d[j]-1]==0:
                    if FO_p2<=FO_p1:
                        secuencia_inicial=secuencia_act_aux.copy()
                        FO_p1=FO_p2
                else:
                    if FO_p2<FO_p1:
                        secuencia_inicial=secuencia_act_aux.copy()
                        FO_p1=FO_p2
            secuencia_act_neh=secuencia_inicial.copy()
            FO_p1=0
            FO_p2=3000
        C_rj_G,f_j_G,FO1_G,FO2_G,seg_G,sat=asignacion.algoritmo(disp_i,dis
p_f,p_i, secuencia_act_neh, tipo_actividad_jr, r_i, J, R, a_j, t_j, Y_r,
pu,pu_prima, tepcof, ult_act)
        FO_G1=FO2_G+FO1_G
        #FIN CONSTRUCCION
        # BUSQUEDA LOCAL
        secuencia_act_neh=busqueda_local.actividades(disp_i,disp_f,p_i,
secuencia_act_neh, tipo_actividad_jr, r_i, J, R, a_j, t_j, Y_r,
pu,pu_prima, tepcof, ult_act)
        C_rj_G,f_j_G,FO1_G,FO2_G,seg_G,sat=asignacion.algoritmo(disp_i,dis
p_f,p_i, secuencia_act_neh, tipo_actividad_jr, r_i, J, R, a_j, t_j, Y_r,
pu,pu_prima, tepcof, ult_act)
        FO_G1=FO2_G+FO1_G
        # FIN BUSQUEDA
        if FO_G1<FO_G:
```

```
            secuencia_gr=secuencia_act_neh.copy()
            FO_G=FO_G1
            if FO_G<FO_p:
                secuencia_act=secuencia_gr.copy()
                FO_p=FO_G
                C_rj,f_j,FO1,FO2,seg,sat=asignacion.algoritmo(disp_i,disp_
f,p_i, secuencia_act, tipo_actividad_jr, r_i, J, R, a_j, t_j, Y_r,
pu,pu_prima, tepcof, ult_act)

        it+=1
        if time.time()>(crono_inicio+crono_parada):
            C_rj,f_j,FO1,FO2,seg,sat=asignacion.algoritmo(disp_i,disp_f,p_
i, secuencia_act, tipo_actividad_jr, r_i, J, R, a_j, t_j, Y_r,
pu,pu_prima, tepcof, ult_act)
            FO_p=FO1+FO2
            break
    return C_rj,f_j,seg,sat,FO_p,secuencia_act,it,FO1,FO2

def greedy_sa_act(disp_i,disp_f,I,secuencia_act_neh,Amax,p_i,
tipo_actividad_jr, r_i, J, R, a_j, t_j, Y_r, pu,pu_prima, tepcof,
ult_act,FO_p,secuencia_act):
    it=0
    n_act=0
    for i in range(I):
        for j in range(Amax):
            if pu[i][j]!=0:
                n_act+=1
    secuencia_gr=secuencia_act_neh.copy()
    C_rj,f_j,FO1,FO2,seg,sat=asignacion.algoritmo(disp_i,disp_f,p_i,
secuencia_gr, tipo_actividad_jr, r_i, J, R, a_j, t_j, Y_r, pu,pu_prima,
tepcof, ult_act)
    FO_G=FO1+FO2
    crono_inicio=time.time()
    crono_parada=n_act*(len(R)-1)*375/(2*1000)
    while True:
        secuencia_act_neh=secuencia_gr.copy()
        #DESTRUCCION
        d=10 #parametro a elegir
        pi_d=np.zeros(d,int)
        for k in range(d):
            pos=rg.integers(len(secuencia_act_neh))
            while p_i[a_j[secuencia_act_neh[pos]-1]-1]==6:
                pos=rg.integers(len(secuencia_act_neh))
            pi_d[k]=secuencia_act_neh[pos]
            secuencia_act_neh=np.delete(secuencia_act_neh, pos)
        #CONSTRUCCION
        FO_p1=0
        FO_p2=3000
```

```
        secuencia_inicial=np.zeros(2,int)
        secuencia_inicial=np.zeros(2,int)
        for j in range(d):
            act=0
            pos1=0
            act2=J
            pos2=len(secuencia_act_neh)+1
            for k in range(len(secuencia_act_neh)):
                if secuencia_act_neh[k]<pi_d[j] and
secuencia_act_neh[k]>=pu[a_j[pi_d[j]-1]-1][0]:
                    if secuencia_act_neh[k]>=act:
                        pos1=k+1
                        act=secuencia_act_neh[k]
                if secuencia_act_neh[k]>pi_d[j] and
secuencia_act_neh[k]<=ult_act[a_j[pi_d[j]-1]-1]:
                    if secuencia_act_neh[k]<=act2:
                      pos2=k+1
                      act2=secuencia_act_neh[k]
            for k in range(pos1,pos2,1):
                secuencia_act_aux=np.insert(secuencia_act_neh, k, pi_d[j])
                if k==pos1:
                    C_rj1,f_j1,FO11,FO21,seg,sat=asignacion.algoritmo(disp
_i,disp_f,p_i,secuencia_act_aux, tipo_actividad_jr, r_i, J, R, a_j, t_j,
Y_r, pu,pu_prima, tepcof, ult_act)
                    FO_p1=FO11+FO21
                    secuencia_inicial=secuencia_act_aux.copy()
                else:
                    C_rj2,f_j2,FO12,FO22,seg,sat=asignacion.algoritmo(disp
_i,disp_f,p_i,secuencia_act_aux, tipo_actividad_jr, r_i, J, R, a_j, t_j,
Y_r, pu,pu_prima, tepcof, ult_act)
                    FO_p2=FO12+FO22
                if t_j[pi_d[j]-1]==0:
                    if FO_p2<=FO_p1:
                        secuencia_inicial=secuencia_act_aux.copy()
                        FO_p1=FO_p2
                else:
                    if FO_p2<FO_p1:
                        secuencia_inicial=secuencia_act_aux.copy()
                        FO_p1=FO_p2
            secuencia_act_neh=secuencia_inicial.copy()
            FO_p1=0
            FO_p2=3000
        C_rj_G,f_j_G,FO1_G,FO2_G,seg_G,sat=asignacion.algoritmo(disp_i,dis
p_f,p_i, secuencia_act_neh, tipo_actividad_jr, r_i, J, R, a_j, t_j, Y_r,
pu,pu_prima, tepcof, ult_act)
        FO_G1=FO2_G+FO1_G
        #FIN CONSTRUCCION
        if FO_G1<FO_G:
            secuencia_gr=secuencia_act_neh.copy()
```

```
            FO_G=FO_G1
            if FO_G<FO_p:
                secuencia_act=secuencia_gr.copy()
                FO_p=FO_G
                C_rj,f_j,FO1,FO2,seg,sat=asignacion.algoritmo(disp_i,disp_
f,p_i, secuencia_act, tipo_actividad_jr, r_i, J, R, a_j, t_j, Y_r,
pu,pu_prima, tepcof, ult_act)

        #SIMULATED ANNEALING
        else:
            cambio=sim_annealing.recocido_act(I,R,it,FO_G,FO_G1,Amax,pu,t_
j)
            if cambio==1:
                secuencia_gr=secuencia_act_neh.copy()
                FO_G=FO_G1
        # FIN S.A
        it+=1
        if time.time()>(crono_inicio+crono_parada):
            C_rj,f_j,FO1,FO2,seg,sat=asignacion.algoritmo(disp_i,disp_f,p_
i, secuencia_act, tipo_actividad_jr, r_i, J, R, a_j, t_j, Y_r,
pu,pu_prima, tepcof, ult_act)
            FO_p=FO1+FO2
            break
    return C_rj,f_j,seg,sat,FO_p,secuencia_act,it,FO1,FO2

def greedy_bl_sa_act(disp_i,disp_f,I,secuencia_act_neh,Amax,p_i,
tipo_actividad_jr, r_i, J, R, a_j, t_j, Y_r, pu,pu_prima, tepcof,
ult_act,FO_p,secuencia_act):
    it=0
    n_act=0
    for i in range(I):
        for j in range(Amax):
            if pu[i][j]!=0:
                n_act+=1
    secuencia_gr=secuencia_act_neh.copy()
    C_rj,f_j,FO1,FO2,seg,sat=asignacion.algoritmo(disp_i,disp_f,p_i,
secuencia_gr, tipo_actividad_jr, r_i, J, R, a_j, t_j, Y_r, pu,pu_prima,
tepcof, ult_act)
    FO_G=FO1+FO2
    crono_inicio=time.time()
    crono_parada=n_act*(len(R)-1)*375/(2*1000)
    while True:
        secuencia_act_neh=secuencia_gr.copy()
        #DESTRUCCION
        #rg=np.random.default_rng(it)
        d=10 #parametro a elegir
        pi_d=np.zeros(d,int)
        for k in range(d):
```

```
            pos=rg.integers(len(secuencia_act_neh))
            while p_i[a_j[secuencia_act_neh[pos]-1]-1]==6:
                pos=rg.integers(len(secuencia_act_neh))
            pi_d[k]=secuencia_act_neh[pos]
            secuencia_act_neh=np.delete(secuencia_act_neh, pos)
        #CONSTRUCCION
        FO_p1=0
        FO_p2=3000
        secuencia_inicial=np.zeros(2,int)
        secuencia_inicial=np.zeros(2,int)
        for j in range(d):
            act=0
            pos1=0
            act2=J
            pos2=len(secuencia_act_neh)+1
            for k in range(len(secuencia_act_neh)):
                if secuencia_act_neh[k]<pi_d[j] and
secuencia_act_neh[k]>=pu[a_j[pi_d[j]-1]-1][0]:
                    if secuencia_act_neh[k]>=act:
                        pos1=k+1
                        act=secuencia_act_neh[k]
                if secuencia_act_neh[k]>pi_d[j] and
secuencia_act_neh[k]<=ult_act[a_j[pi_d[j]-1]-1]:
                    if secuencia_act_neh[k]<=act2:
                      pos2=k+1
                      act2=secuencia_act_neh[k]
            for k in range(pos1,pos2,1):
                secuencia_act_aux=np.insert(secuencia_act_neh, k, pi_d[j])
                if k==pos1:
                    C_rj1,f_j1,FO11,FO21,seg,sat=asignacion.algoritmo(disp
_i,disp_f,p_i,secuencia_act_aux, tipo_actividad_jr, r_i, J, R, a_j, t_j,
Y_r, pu,pu_prima, tepcof, ult_act)
                    FO_p1=FO11+FO21
                    secuencia_inicial=secuencia_act_aux.copy()
                else:
                    C_rj2,f_j2,FO12,FO22,seg,sat=asignacion.algoritmo(disp
_i,disp_f,p_i,secuencia_act_aux, tipo_actividad_jr, r_i, J, R, a_j, t_j,
Y_r, pu,pu_prima, tepcof, ult_act)
                    FO_p2=FO12+FO22
                if t_j[pi_d[j]-1]==0:
                    if FO_p2<=FO_p1:
                        secuencia_inicial=secuencia_act_aux.copy()
                        FO_p1=FO_p2
                else:
                    if FO_p2<FO_p1:
                        secuencia_inicial=secuencia_act_aux.copy()
                        FO_p1=FO_p2
            secuencia_act_neh=secuencia_inicial.copy()
            FO_p1=0
```

```
            FO_p2=3000
        C_rj_G,f_j_G,FO1_G,FO2_G,seg_G,sat=asignacion.algoritmo(disp_i,dis
p_f,p_i, secuencia_act_neh, tipo_actividad_jr, r_i, J, R, a_j, t_j, Y_r,
pu,pu_prima, tepcof, ult_act)
        FO_G1=FO2_G+FO1_G
        #FIN CONSTRUCCION
        # BUSQUEDA LOCAL
        secuencia_act_neh=busqueda_local.actividades(disp_i,disp_f,p_i,
secuencia_act_neh, tipo_actividad_jr, r_i, J, R, a_j, t_j, Y_r,
pu,pu_prima, tepcof, ult_act)
        C_rj_G,f_j_G,FO1_G,FO2_G,seg_G,sat=asignacion.algoritmo(disp_i,dis
p_f,p_i, secuencia_act_neh, tipo_actividad_jr, r_i, J, R, a_j, t_j, Y_r,
pu,pu_prima, tepcof, ult_act)
        FO_G1=FO2_G+FO1_G
        # FIN BUSQUEDA
        if FO_G1<FO_G:
            secuencia_gr=secuencia_act_neh.copy()
            FO_G=FO_G1
            if FO_G<FO_p:
                secuencia_act=secuencia_gr.copy()
                FO_p=FO_G
                C_rj,f_j,FO1,FO2,seg,sat=asignacion.algoritmo(disp_i,disp_
f,p_i, secuencia_act, tipo_actividad_jr, r_i, J, R, a_j, t_j, Y_r,
pu,pu_prima, tepcof, ult_act)

        #SIMULATED ANNEALING
        else:
            cambio=sim_annealing.recocido_act(I,R,it,FO_G,FO_G1,Amax,pu,t_
j)
            if cambio==1:
                secuencia_gr=secuencia_act_neh.copy()
                FO_G=FO_G1
        # FIN S.A
        it+=1
        if time.time()>(crono_inicio+crono_parada):
            C_rj,f_j,FO1,FO2,seg,sat=asignacion.algoritmo(disp_i,disp_f,p_
i, secuencia_act, tipo_actividad_jr, r_i, J, R, a_j, t_j, Y_r,
pu,pu_prima, tepcof, ult_act)
            FO_p=FO1+FO2
            break
    return C_rj,f_j,seg,sat,FO_p,secuencia_act,it,FO1,FO2
```

Búsqueda local

```
import asignacion
import numpy as np
rg=np.random.default_rng(2)
```

```
def actividades(disp_i,disp_f,p_i, secuencia_act, tipo_actividad_jr, r_i,
J, R, a_j, t_j, Y_r, pu,pu_prima, tepcof, ult_act):
    C_rj,f_j,FO1,FO2,seg,sat=asignacion.algoritmo(disp_i,disp_f,p_i,
secuencia_act, tipo_actividad_jr, r_i, J, R, a_j, t_j, Y_r, pu,pu_prima,
tepcof, ult_act)
    FO=FO1+FO2
    vecinos=12
    d=15
    mejora=1
    secuencia_aux=secuencia_act.copy()
    while mejora==1:
        mejora=0
        for i in range(d):
            sec_prima=secuencia_act.copy()
            pos=rg.integers(0,len(sec_prima))
            job=secuencia_act[pos]
            sec_prima=np.delete(sec_prima, pos)
            act=0
            pos1=0
            act2=J
            pos2=len(secuencia_act)+1
            for k in range(len(secuencia_act)):
                if secuencia_act[k]<job and secuencia_act[k]>=pu[a_j[job-
1]-1][0]:
                    if secuencia_act[k]>=act:
                        pos1=k+1
                        act=secuencia_act[k]
                if secuencia_act[k]>job and
secuencia_act[k]<=ult_act[a_j[job-1]-1]:
                    if secuencia_act[k]<=act2:
                        pos2=k
                        act2=secuencia_act[k]

            for j in range(pos-vecinos,pos+vecinos+1,1):
                if j>=0 and j>=pos1 and j<pos2 and j<=len(sec_prima):
                    sec_prov=np.insert(sec_prima, j, job)
                    C_rj1,f_j1,FO11,FO21,seg1,sat=asignacion.algoritmo(dis
p_i,disp_f,p_i, sec_prov, tipo_actividad_jr, r_i, J, R, a_j, t_j, Y_r,
pu,pu_prima, tepcof, ult_act)
                    FO_p=FO11+FO21

                    if FO>FO_p:
                        mejora=1
                        secuencia_aux=sec_prov.copy()
                        C_rj,f_j,FO1,FO2,seg,sat=asignacion.algoritmo(disp
_i,disp_f,p_i, secuencia_aux, tipo_actividad_jr, r_i, J, R, a_j, t_j, Y_r,
pu,pu_prima, tepcof, ult_act)
                        FO=FO1+FO2
        if mejora==1:
```

```
            secuencia_act=secuencia_aux.copy()
    return secuencia_aux

def pacientes(disp_i,disp_f,secuencia_paciente,p_i, secuencia_act,
tipo_actividad_jr, r_i, J, R, a_j, t_j, Y_r, pu,pu_prima, tepcof,
ult_act):
    C_rj,f_j,FO1,FO2,seg,sat=asignacion.algoritmo(disp_i,disp_f,p_i,
secuencia_act, tipo_actividad_jr, r_i, J, R, a_j, t_j, Y_r, pu,pu_prima,
tepcof, ult_act)
    FO=FO1+FO2
    vecinos=4
    d=4
    mejora=1
    secuencia_aux=secuencia_paciente.copy()
    while mejora==1:
        mejora=0
        for i in range(d):
            sec_prima=secuencia_paciente.copy()
            pos=rg.integers(0,len(sec_prima))
            job=secuencia_paciente[pos]
            sec_prima=np.delete(sec_prima, pos)
            for j in range(pos-vecinos,pos+vecinos+1,1):
                if j>=0 and j<=len(sec_prima):
                    sec_prov=np.insert(sec_prima, j, job)
                    w=0
                    for j in range(len(sec_prov)):
                        for k in range(8):#Amax
                            if pu[sec_prov[j]-1][k]!=0:
                                secuencia_act[w]=pu[sec_prov[j]-1][k]
                                w=w+1
                    C_rj1,f_j1,FO11,FO21,seg1,sat=asignacion.algoritmo(dis
p_i,disp_f,p_i, secuencia_act, tipo_actividad_jr, r_i, J, R, a_j, t_j,
Y_r, pu,pu_prima, tepcof, ult_act)
                    FO_p=FO11+FO21
                    if FO>FO_p:
                        mejora=1
                        secuencia_aux=sec_prov.copy()
                        C_rj,f_j,FO1,FO2,seg,sat=asignacion.algoritmo(disp
_i,disp_f,p_i, secuencia_act, tipo_actividad_jr, r_i, J, R, a_j, t_j, Y_r,
pu,pu_prima, tepcof, ult_act)
                        FO=FO1+FO2

        if mejora==1:
            secuencia_paciente=secuencia_aux.copy()

    return secuencia_aux
```

Simulated annealing

```
import numpy as np
import math

def recocido_paciente(I,R,it,FO_G,FO_G1,Amax,pu,t_j):
    tiempo_total=np.zeros(I,int)
    for j in range(I):
        suma=0
        for i in range(Amax):
            if pu[j][i]!=0:
                suma+=t_j[pu[j][i]-1]
        tiempo_total[j]=suma
    suma_tiempos=0
    cambio=0
    for i in range(I):
        suma_tiempos+=tiempo_total[i]
    N=len(R)
    T=0.4 #parametro a elegir
    Temp=T*suma_tiempos/(N*I*10)
    rg=np.random.default_rng(it)
    r=rg.uniform(0,1)
    if r<=math.exp(-(FO_G1-FO_G)/Temp):
        cambio=1
    return cambio

def recocido_act(I,R,it,FO_G,FO_G1,Amax,pu,t_j):
    tiempo_total=np.zeros(I,int)
    n_act=0
    for i in range(I):
        for j in range(Amax):
            if pu[i][j]!=0:
                n_act+=1
    for j in range(I):
        suma=0
        for i in range(Amax):
            if pu[j][i]!=0:
                suma+=t_j[pu[j][i]-1]
        tiempo_total[j]=suma
    suma_tiempos=0
    cambio=0
    for i in range(I):
        suma_tiempos+=tiempo_total[i]
    N=len(R)
    T=0.4 #parametro a elegir
    Temp=T*suma_tiempos/(N*n_act*10)
    rg=np.random.default_rng(it)
    r=rg.uniform(0,1)
    if r<=math.exp(-(FO_G1-FO_G)/Temp):
```

```
        cambio=1
    return cambio
```

Decoding

```
import numpy as np
def
algoritmo(disp_i,disp_f,p_i,secuencia_act,tipo_actividad_jr,r_i,J,R,a_j,t_
j,Y_r,pu,pu_prima,tepcof,ult_act):
    reservado=np.zeros(len(R),int)
    f_j=np.zeros(J,int) #tiempos finalización sin tener en cuenta el
recurso
    f_j_copia=np.zeros(J,int)
    C_rj=np.zeros((len(R),J),int)
    C_copia=np.zeros((len(R),J),int)
    seg=np.zeros((len(R),2),int)
    seg_copia=seg.copy()
    comienzo=0
    secuencia_paciente=np.array([a_j[secuencia_act[0]-1]])
    descansado=np.zeros(len(R),int)
    for p in range(len(secuencia_act)):
        descansado_copia=descansado.copy()
        descansar=0
        ok=1
        for m in range(len(secuencia_paciente)):
            if a_j[secuencia_act[p]-1]==secuencia_paciente[m]:
                ok=0
        if ok==1:
            secuencia_paciente=np.insert(secuencia_paciente,len(secuencia_
paciente),a_j[secuencia_act[p]-1])
        comienzo=0
        for t in range(p):
            if secuencia_act[t]<secuencia_act[p] and
secuencia_act[t]>=pu[a_j[secuencia_act[p]-1]-1][0]:
                if f_j[secuencia_act[t]-1]>=comienzo:
                    if Y_r[tipo_actividad_jr[secuencia_act[p]-1][0]-
1][tipo_actividad_jr[secuencia_act[t]-1][0]-1]!=1:
                        comienzo=f_j[secuencia_act[t]-1]
        if comienzo<r_i[a_j[secuencia_act[p]-1]-1]:
            comienzo=r_i[a_j[secuencia_act[p]-1]-1]
        if tipo_actividad_jr[secuencia_act[p]-1][1]==1:
            for m in range(len(R)):
                if R[m]==tipo_actividad_jr[secuencia_act[p]-1][0]:
                    if p_i[a_j[secuencia_act[p]-1]-1]==6:
                        descansar=1
                    recurso=m+1
                    if (reservado[m]==1 and p_i[a_j[secuencia_act[p]-1]-
1]>3 and p_i[a_j[secuencia_act[p]-1]-1]!=6):
```

```
                        True
                    else:
                        if secuencia_act[p]==pu[a_j[secuencia_act[p]-1]-
1][0]:
                            busqueda=1
                            if comienzo>=disp_i[recurso-1]:
                                inst=comienzo
                            else:
                                inst=disp_i[recurso-1]
                            if descansar==1 and descansado[recurso-1]==1:
                                True
                            else:
                                while busqueda==1 and
inst<=seg.shape[1]:
                                    if
seg.shape[1]<=(inst+t_j[secuencia_act[p]-1]):
                                        seg=np.insert(seg, seg.shape[1],
np.zeros((inst+t_j[secuencia_act[p]-1]-seg.shape[1],len(R)),int),1)
                                    vale=1
                                    if tipo_actividad_jr[secuencia_act[p]-
1][0]!=4:
                                        for k in
range(inst,inst+t_j[secuencia_act[p]-1],1):
                                            if seg[recurso-1][k]==0 and
vale==1:
                                                vale=1
                                                busqueda=0
                                            else:
                                                vale=0
                                                busqueda=1
                                                if seg[recurso-1][k]!=0:
                                                    inst=f_j[seg[recurso-
1][k]-1]
                                    else:
                                        vale=1
                                        busqueda=0
                                    if vale==1:
                                        if f_j[secuencia_act[p]-1]==0:
                                            for k in
range(inst,inst+t_j[secuencia_act[p]-1],1):
                                                seg[recurso-
1][k]=secuencia_act[p]
                                                C_rj[recurso-
1][secuencia_act[p]-1]=k+1
                                                f_j[secuencia_act[p]-
1]=C_rj[recurso-1][secuencia_act[p]-1]
                                                inst=k
                                                if descansar==1:
```

```
                                                    descansado[recurso-
1]=1
                                            elif inst<(f_j[secuencia_act[p]-
1]-t_j[secuencia_act[p]-1]):
                                                C_rj=C_copia.copy()
                                                f_j=f_j_copia.copy()
                                                seg=seg_copia.copy()
                                                descansado=descansado_copia.co
py()
                                                if
seg.shape[1]<=(inst+t_j[secuencia_act[p]-1]):
                                                    seg=np.insert(seg,
seg.shape[1], np.zeros((inst+t_j[secuencia_act[p]-1]-
seg.shape[1],len(R)),int),1)
                                                for k in
range(inst,inst+t_j[secuencia_act[p]-1],1):
                                                    seg[recurso-
1][k]=secuencia_act[p]
                                                    C_rj[recurso-
1][secuencia_act[p]-1]=k+1
                                                    f_j[secuencia_act[p]-
1]=C_rj[recurso-1][secuencia_act[p]-1]
                                                    inst=k
                                                    if descansar==1:
                                                        descansado[recurso-
1]=1
                                    if inst>=seg.shape[1] and busqueda==1 and
f_j[secuencia_act[p]-1]==0:
                                        seg=np.insert(seg, seg.shape[1],
np.zeros((inst+t_j[secuencia_act[p]-1]-seg.shape[1],len(R)),int),1)
                                        for k in
range(inst,inst+t_j[secuencia_act[p]-1],1):
                                            seg[recurso-1][k]=secuencia_act[p]
                                            C_rj[recurso-1][secuencia_act[p]-
1]=k+1
                                            f_j[secuencia_act[p]-
1]=C_rj[recurso-1][secuencia_act[p]-1]
                                            if descansar==1:
                                                descansado[recurso-1]=1
                            else: #si no es 1° act
                                busqueda=1
                                if comienzo>=disp_i[recurso-1]:
                                    inst=comienzo
                                else:
                                    inst=disp_i[recurso-1]
                                while busqueda==1 and inst<=seg.shape[1]:
                                    if
seg.shape[1]<=(inst+t_j[secuencia_act[p]-1]):
```

```
                                    seg=np.insert(seg, seg.shape[1],
np.zeros((inst+t_j[secuencia_act[p]-1]-seg.shape[1],len(R)),int),1)
                                vale=1
                                if tipo_actividad_jr[secuencia_act[p]-
1][0]!=4:
                                    for k in
range(inst,inst+t_j[secuencia_act[p]-1],1):
                                        if seg[recurso-1][k]==0 and
vale==1:
                                            vale=1
                                            busqueda=0
                                        else:
                                            vale=0
                                            busqueda=1
                                            if seg[recurso-1][k]!=0:
                                                inst=f_j[seg[recurso-
1][k]-1]
                                else:
                                    vale=1
                                    busqueda=0
                                if t_j[secuencia_act[p]-1]==0:
                                    vale=1
                                    C_rj[recurso-1][secuencia_act[p]-
1]=inst
                                    f_j[secuencia_act[p]-1]=C_rj[recurso-
1][secuencia_act[p]-1]
                                    busqueda=0
                                if vale==1:
                                    #CONT. ASISTENCIAL
                                    if (R[recurso-1]<=2 and C_rj[recurso-
1][pu_prima[a_j[secuencia_act[p]-1]-1][1]-1]!=0) or R[recurso-1]>2 or
f_j[pu_prima[a_j[secuencia_act[p]-1]-1][1]-1]==0:
                                        if f_j[secuencia_act[p]-1]==0:
                                            for k in
range(inst,inst+t_j[secuencia_act[p]-1],1):
                                                seg[recurso-
1][k]=secuencia_act[p]
                                                C_rj[recurso-
1][secuencia_act[p]-1]=k+1
                                                f_j[secuencia_act[p]-
1]=C_rj[recurso-1][secuencia_act[p]-1]
                                                inst=k
                                        elif inst<(f_j[secuencia_act[p]-
1]-t_j[secuencia_act[p]-1]):
                                            C_rj=C_copia.copy()
                                            f_j=f_j_copia.copy()
                                            seg=seg_copia.copy()
                                            if
seg.shape[1]<=(inst+t_j[secuencia_act[p]-1]):
```

```
                                                    seg=np.insert(seg,
seg.shape[1], np.zeros((inst+t_j[secuencia_act[p]-1]-
seg.shape[1],len(R)),int),1)
                                                for k in
range(inst,inst+t_j[secuencia_act[p]-1],1):
                                                    seg[recurso-
1][k]=secuencia_act[p]
                                                    C_rj[recurso-
1][secuencia_act[p]-1]=k+1
                                                    f_j[secuencia_act[p]-
1]=C_rj[recurso-1][secuencia_act[p]-1]
                                                    inst=k
                                if inst>=seg.shape[1] and busqueda==1 and
f_j[secuencia_act[p]-1]==0:
                                    #CONT ASISTENCIAL
                                    if (R[recurso-1]<=2 and C_rj[recurso-
1][pu_prima[a_j[secuencia_act[p]-1]-1][1]-1]!=0) or R[recurso-1]>2 or
f_j[pu_prima[a_j[secuencia_act[p]-1]-1][1]-1]==0:
                                        seg=np.insert(seg, seg.shape[1],
np.zeros((inst+t_j[secuencia_act[p]-1]-seg.shape[1],len(R)),int),1)
                                        for k in
range(inst,inst+t_j[secuencia_act[p]-1],1):
                                            seg[recurso-1][k]=secuencia_act[p]
                                            C_rj[recurso-1][secuencia_act[p]-
1]=k+1
                                            f_j[secuencia_act[p]-
1]=C_rj[recurso-1][secuencia_act[p]-1]

                            if m<len(R)-1:
                                if R[m]!=R[m+1]:
                                    seg_copia=seg.copy()
                                    C_copia=C_rj.copy()
                                    f_j_copia=f_j.copy()
                                    descansado_copia=descansado.copy()
                            elif m==len(R)-1:
                                seg_copia=seg.copy()
                                C_copia=C_rj.copy()
                                f_j_copia=f_j.copy()
                    else:  #solo se pueden necesitar 2 en ENF
                        asignado=0
                        recurso1=0
                        recurso2=0
                        nuevo_com=0
                        nuevo_com2=8*60
                        candidato=0
                        while asignado<tipo_actividad_jr[secuencia_act[p]-1][1]:
                            nuevo_com2=8*60
                            for m in range(len(R)):
```

```
                    if R[m]==tipo_actividad_jr[secuencia_act[p]-1][0] and
m!=recurso1-1:
                        recurso=m+1
                        busqueda=1
                        if comienzo>=disp_i[recurso-1]:
                            inst=comienzo
                        else:
                            inst=disp_i[recurso-1]
                        while busqueda==1 and inst<=seg.shape[1] and
asignado<tipo_actividad_jr[secuencia_act[p]-1][1]:  #seguie buscando mas
adelante el hueco
                            if seg.shape[1]<=(inst+t_j[secuencia_act[p]-
1]):
                                seg=np.insert(seg, seg.shape[1],
np.zeros((inst+t_j[secuencia_act[p]-1]-seg.shape[1],len(R)),int),1)
                            vale=1
                            for k in range(inst,inst+t_j[secuencia_act[p]-
1],1):
                                if seg[recurso-1][k]==0 and vale==1:
                                    vale=1
                                    busqueda=0
                                else:
                                    vale=0
                                    busqueda=1
                                    if seg[recurso-1][k]!=0:
                                        inst=f_j[seg[recurso-1][k]-1]
                            if vale==1:
                                if asignado==1:
                                    if inst==nuevo_com:
                                        asignado=2
                                        recurso2=recurso
                                    elif inst<nuevo_com:
                                        if nuevo_com<nuevo_com2:
                                            candidato=recurso1
                                            nuevo_com2=nuevo_com
                                        nuevo_com=inst
                                        recurso1=recurso
                                        asignado=1
                                    elif inst<nuevo_com2:
                                        nuevo_com2=inst
                                        candidato=recurso
                                else:
                                    nuevo_com=inst
                                    recurso1=recurso
                                    asignado=1
                        if inst>=seg.shape[1] and busqueda==1 and
f_j[secuencia_act[p]-1]==0:
                            if asignado==0:
                                nuevo_com=inst
```

```
                                recurso1=recurso
                                asignado=1
                            else:
                                if inst==nuevo_com:
                                    asignado=2
                                    recurso2=recurso
                                elif inst<nuevo_com:
                                    if nuevo_com<nuevo_com2:
                                        candidato=recurso1
                                        nuevo_com2=nuevo_com
                                    nuevo_com=inst
                                    recurso1=recurso
                                    asignado=1
                                elif inst<nuevo_com2:
                                    nuevo_com2=inst
                                    candidato=recurso
                if asignado<tipo_actividad_jr[secuencia_act[p]-1][1] and
asignado!=0:
                    asignado=1
                    recurso2=0
                    if nuevo_com2>nuevo_com:
                        nuevo_com=nuevo_com2
                        recurso1=candidato
                comienzo=nuevo_com
            inst=comienzo
            if seg.shape[1]<=(inst+t_j[secuencia_act[p]-1]):
                seg=np.insert(seg, seg.shape[1],
np.zeros((inst+t_j[secuencia_act[p]-1]-seg.shape[1],len(R)),int),1)
            vale=1
            for k in range(inst,inst+t_j[secuencia_act[p]-1],1):
                seg[recurso1-1][k]=secuencia_act[p]
                seg[recurso2-1][k]=secuencia_act[p]
                C_rj[recurso1-1][secuencia_act[p]-1]=k+1
                C_rj[recurso2-1][secuencia_act[p]-1]=k+1
                f_j[secuencia_act[p]-1]=C_rj[recurso1-1][secuencia_act[p]-
1]
                seg_copia=seg.copy()
                C_copia=C_rj.copy()
                f_j_copia=f_j.copy()

    FO1=0 #tardiness
    for j in range(len(secuencia_paciente)):
        at=seg.shape[1]
        if f_j[pu_prima[secuencia_paciente[j]-1][0]-1]!=0:
            for n in range(8): #Amax
                if pu[secuencia_paciente[j]-1][n]!=0:
                    if f_j[pu[secuencia_paciente[j]-1][n]-1]!=0:
                        if f_j[pu[secuencia_paciente[j]-1][n]-1]<at:
```

```
                            at=f_j[pu[secuencia_paciente[j]-1][n]-1]-
t_j[pu[secuencia_paciente[j]-1][n]-1]
            if (at-(tepcof[secuencia_paciente[j]-
1]+r_i[secuencia_paciente[j]-1]))>0:
                FO1+=(at-(tepcof[secuencia_paciente[j]-
1]+r_i[secuencia_paciente[j]-1]))*(6-p_i[secuencia_paciente[j]-1])
    FO2=0
    for j in range(len(secuencia_paciente)):
        if p_i[secuencia_paciente[j]-1]!=6:
            salida=0
            for n in range(8): #Amax
                if pu[secuencia_paciente[j]-1][n]!=0:
                    if f_j[pu[secuencia_paciente[j]-1][n]-1]!=0:
                        if f_j[pu[secuencia_paciente[j]-1][n]-1]>salida:
                          salida=f_j[pu[secuencia_paciente[j]-1][n]-1]
            FO2+=salida-r_i[secuencia_paciente[j]-1]
    ocu=np.zeros(len(R))
    for i in range(len(ocu)):
        for j in range(seg.shape[1]):
            if seg[i][j]!=0:
                if p_i[a_j[seg[i][j]-1]-1]!=6:
                    ocu[i]+=1
    sat=np.zeros(7) #7 tipos de recurso
    for i in range(len(sat)):
        trab=0
        minutos=0
        for j in range(len(R)):
            if R[j]==(i+1):
                trab+=ocu[j]
                minutos+=disp_f[j]-disp_i[j]
        sat[i]=trab/minutos
    return C_rj,f_j,FO1,FO2,seg,sat
```