Trabajo Fin de Máster

Máster en Ingeniería Industrial

Desarrollo de algoritmos aproximados para la secuenciación de tareas del personal sanitario en un SUH

Autor: Eva Moro Barroso

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 secuenciación de tareas del personal sanitario en un SUH

Autor:
Eva Moro Barroso

Tutor:
José Manuel Molina Pariente

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

Trabajo de fin de Máster: Desarrollo de algoritmos aproximados para la secuenciación de tareas del personal sanitario en un SUH

Autor: Eva Moro Barroso

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

A mi familia

A mis maestros

Resumen

En los Servicios Hospitalarios de Urgencia (SUHs) es de crucial importancia la toma de decisiones rápidas y certeras. Los pacientes deben ser atendidos con celeridad, atendiendo a su prioridad y a los recursos disponibles. Los tiempos de estancia (LOS) y de espera para la primera consulta facultativa (TBSPPA) de los pacientes que acuden al SUH son indicadores de la calidad de este servicio.

Por tanto, el objetivo de este trabajo es disminuir estos tiempos a través de una secuenciación más eficaz de las tareas del personal sanitario. Dichas tareas se corresponden con las actividades (visitas, pruebas de diagnóstico, etc.) que deben realizarse en los procesos de urgencia (PU) de los pacientes.

En el problema que se trata en este trabajo los pacientes solo serán atendidos si están asignados a un box, que solo podrá estar ocupado por un paciente a la vez, y permanecerán en él todo su PU.

Para encontrar la secuenciación que minimice dichos tiempos se proponen tres algoritmos metaheurísticos que son adaptados para que cumplan las restricciones y criterios del problema específico que se aborda. Estos algoritmos son tres: el Algoritmo Genético, el Recocido Simulado y la Búsqueda Tabú.

Se realiza una experimentación computacional para la cual se toman los datos extraídos del artículo de Bedoya-Valencia & Kirac (2016). En dicha experimentación se ejecutan y analizan 6 escenarios distintos en los que varían la combinación entre el nivel máximo de saturación del SUH y los recursos disponibles. Para cada uno de esos escenarios se generan 10 instancias que presentan datos de entrada diferentes. Los resultados obtenidos en cada escenario de la experimentación permiten evaluar y comparar el desempeño de los algoritmos implementados.

Por último, se concluye cuál de estos algoritmos aporta una mejor solución al problema de la secuenciación de tareas del personal sanitario en el SUH, siendo el tiempo de computación el mismo para todos los presentados.

Todo esto se realiza utilizando un programa implementado con el lenguaje de programación de Python.

Abstract

In emergency departments (ED) making quick and accurate decisions is of crucial importance. Patients must be treated immediately, taking into account their priority and the available resources. Length of stay (LOS) and Time To Be Seen By A Physician Or Physician Assistant (TBSPPA) for patients attending the ED are indicators of the quality of this service.

Therefore, the aim of this work is to reduce these times through a more effective sequencing of tasks for healthcare personnel. These tasks correspond to the activities (visits, diagnostic tests, etc.) that must be carried out in the emergency processes (EP) of patients.

In the problem addressed in this work, patients will only be attended if they are assigned to a treatment room, which can only be occupied by one patient at a time, and they will remain in it throughout all their EP.

To find the sequencing that minimizes these times, three metaheuristic algorithms are proposed and adapted to meet the constraints and criteria of the specific problem addressed. These algorithms are: Genetic Algorithm, Simulated Annealing and Tabu Search.

A computational experiment is carried out using data extracted from the article by Bedoya-Valencia & Kirac, (2016) . In this experiment, 6 different scenarios are executed and analyzed, varying the combination between the maximum saturation level of the ED and the available resources. For each of these scenarios, 10 instances are generated, presenting different input data. The results obtained in each scenario of the experiment allow for the evaluation and comparison of the performance of the implemented algorithms.

Finally, it is concluded which of these algorithms provides a better solution to the problem of sequencing healthcare personnel tasks in the ED, with the same computation time for all presented.

All of this is done using a program implemented in the Python programming language.

Índice

Resumen **ix**

Abstract **xi**

Índice **xiii**

Índice de Tablas **xv**

Índice de ilustraciones **xvii**

Notación **xix**

1 Introducción **1**
1.1 Contexto *1*
1.2 Presentación del problema *1*
1.3 Objetivos *2*
1.4 Sumario *3*

2 Estado del arte **4**

3 Descripción del problema **7**
3.1 Descripción del SUH *7*
3.2. Descripción del problema específico *9*

4 Metodología de resolución **11**
4.1. Decoding *11*
4.1.1 Descripción del *decoding* 11
4.1.2 Ejemplo de implementación 12
4.2. Algoritmos de optimización *15*
4.2.1 Recocido Simulado 15
4.2.2 Búsqueda Tabú 17
4.2.3 Algoritmo genético 19

5 Análisis computacional **21**
5.1 Experimentación (experimentacion.py) *21*
5.2 Algoritmos (algoritmos.py) *22*
5.3 Decoding (evaluación.py) *22*
5.4 Diagrama de Gantt (Gantt.py) *22*

6 Experimentación **23**
6.1 Implementación de la experimentación *23*
6.1.1 Número de pacientes 23
6.1.2 Generación de la prioridad 24
6.1.3 Procesos de urgencia 25
6.1.4 Duración de las actividades 26
6.1.5 Recursos disponibles 27
6.2 Resultados *27*
6.2.1 Criterios para la comparación de resultados. 28
6.2.2 Número de recursos disponibles de las 6 a.m. 28
6.2.3 Número de recursos disponibles de las 12 a.m. 35
6.3 Análisis de resultados *41*

7 Conclusiones **44**

Referencias **46**

Anexo A **50**

Anexo B **52**

Anexo C **55**

ÍNDICE DE TABLAS

Tabla 1. TBSPPA máximo en función de la prioridad o nivel ESI del paciente (Jiménez Murillo et al., 2019) 8
Tabla 2. Relación entre las actividades y los recursos que la realizan. 8
Tabla 3. Lista de recursos totales 13
Tabla 4. Lista de actividades y recursos de los PUs 13
Tabla 5. Relación de actividades y recursos 25
Tabla 6. Actividades y recursos del PU en función del nivel ESI. 26
Tabla 7. Recursos disponibles por tipología y horarios 27
Tabla 8. Recursos disponibles a las 6 a.m. 28
Tabla 9. Número de pacientes, actividades y recursos medio. Caso 1 29
Tabla 10. Número de pacientes, actividades y recursos medio. Caso 2 31
Tabla 11. Número de pacientes, actividades y recursos medio. Caso 3 33
Tabla 12. Recursos disponibles a las 12 a.m. 35
Tabla 13. Número de pacientes, actividades y recursos medio. Caso 4 35
Tabla 14. Número de pacientes, actividades y recursos medio. Caso 5 37
Tabla 15. Número de pacientes, actividades y recursos medio. Caso 6 39
Tabla 16. Valores de la RPD obtenidos por instancia del recocido simulado 50
Tabla 17. ARPD de cada escenario del recocido simulado 50
Tabla 18. Valores RPD obtenidos por instancia de la búsqueda tabú 51
Tabla 19. ARPD de cada escenario de la búsqueda tabú 51
Tabla 20. RDP obtenidos para las 10 instancias en el caso 1 52
Tabla 21. RDP obtenidos para las 10 instancias en el caso 2 52
Tabla 22. RPD obtenidos para las 10 instancias en el caso 3 53
Tabla 23. RDP obtenidos para las 10 instancias en el caso 4 53
Tabla 24. RDP obtenidos para las 10 instancias en el caso 5 53
Tabla 25. RDP obtenidos para las 10 instancias en el caso 6 54

ÍNDICE DE ILUSTRACIONES

Ilustración 1. Pasos de los pacientes en el SUH (Bedoya-Valencia & Kirac, 2016). 7
Ilustración 2. Secuenciación de tareas 14
Ilustración 3. Procedimiento de aplicación del Recocido Simulado (Delahaye et al., 2019) 16
Ilustración 4. Procedimiento búsqueda tabú (Velez & Montoya, 2007). 18
Ilustración 5. Pseudocódigo Algoritmo Genético Clásico. (Katoch et al., 2021) 19
Ilustración 6. Resultados de la simulación de (Bedoya-Valencia & Kirac, 2016) 24
Ilustración 7. Porcentaje de pacientes de acuerdo con su ESI (Bedoya-Valencia & Kirac, 2016) 24
Ilustración 8. Recursos requeridos por los pacientes según sus prioridades. Bedoya-Valencia & Kirac, 2016). 25
Ilustración 9. Tiempos de proceso de los pacientes en función del nivel ESI (Bedoya-Valencia & Kirac, 2016). 27
Ilustración 10. Horario de enfermeros, médicos asistentes y médicos (Bedoya-Valencia & Kirac, 2016) 27
Ilustración 11. Saturación de los recursos para este escenario. Caso 1 29
Ilustración 12. Saturación media de los recursos. Caso 1 30
Ilustración 13. Valores FO obtenidos para las 10 instancias. Caso 1 30
Ilustración 14. ARPD del caso 1 31
Ilustración 15. Saturación de los recursos para este escenario. Caso 2 31
Ilustración 16. Saturación media de los recursos. Caso 2 32
Ilustración 17. Valores FO obtenidos para las 10 instancias. Caso 2 32
Ilustración 18. ARPD del caso 2 33
Ilustración 19. Saturación de los recursos para este escenario. Caso 3 33
Ilustración 20. Saturación media de los recursos. Caso 3 34
Ilustración 21. Valores FO obtenidos para las 10 instancias. Caso 3 34
Ilustración 22. ARPD del caso 3 35
Ilustración 23. Saturación de los recursos para este escenario. Caso 4 36
Ilustración 24. Saturación media de los recursos. Caso 4 36
Ilustración 25. Valores FO obtenidos para las 10 instancias. Caso 4 37
Ilustración 26. ARPD del caso 4 37
Ilustración 27. Saturación de los recursos para este escenario. Caso 5 38
Ilustración 28. Saturación media de los recursos. Caso 5 38
Ilustración 29. Valores FO obtenidos para las 10 instancias. Caso 5 39
Ilustración 30. ARPD del caso 5 39
Ilustración 31. Saturación de los recursos para este escenario. Caso 6 40
Ilustración 32. Saturación media de los recursos. Caso 6 40
Ilustración 33. Valores FO obtenidos para las 10 instancias. Caso 6 41

Ilustración 34. ARPD del caso 6 41
Ilustración 35. Comparación de los ARPD de los algoritmos en los 6 escenarios 42
Ilustración 36. Evolución soluciones encontradas por recocido simulado 43

Notación

SUH	Servicio de Urgencia Hospitalario
LOS	*Length of Stay*
TBSPPA	*Time to Be Seen by a Physician or Physician Assistant*
ESI	*Emergency Severity Index*
FO	Función Objetivo
PU	Proceso de Urgencia
RPD	*Relative Percentage Deviation*

1 INTRODUCCIÓN

En este capítulo se proporciona una visión general del contexto en el que se enmarca el estudio y se hace una breve introducción al problema específico que se busca resolver, así como a los objetivos generales de este trabajo. Por último se ofrece un sumario de los contenidos de este documento.

1.1 Contexto

De todos los servicios que se ofrecen en un hospital, el Servicio de Urgencia Hospitalario (SUH) es uno de los más complejos dada la criticidad de las situaciones, la variedad de casos que se reciben y el tiempo limitado que tienen para atenderlos. Es por ello por lo que la optimización de este servicio ha sido ampliamente estudiada.

El SUH es un servicio dedicado a proporcionar atención especializada a todos los ciudadanos que necesiten o soliciten asistencia urgente. Tiene la capacidad de gestionar su ingreso en una unidad de hospitalización, trasladarlo a otro nivel de atención o dar el alta para regresar a su domicilio (Tudela & Deltell, 2015).

La medicina es una disciplina científica y, como tal, ha ido nutriéndose de los avances tecnológicos que le han permitido seguir desarrollándose.

Con el avance de los sistemas de información, también se ha incrementado la capacidad de procesamiento de los datos. Esto se debe tanto a la mejora de las tecnologías de hardware, como al perfeccionamiento de los algoritmos y técnicas de procesamiento de datos. Esta combinación ha permitido manejar un conjunto de datos mayor y más complejo de una manera más eficaz y eficiente.

Sin embargo, los SUHs tienen una naturaleza estocástica, es decir, están sujetos a la variabilidad e incertidumbre y siguen patrones probabilísticos (p.e. los tiempos de procesamiento, la demanda, los niveles de urgencia, etc., (Bahari et al., 2022)). Esto hace más complicada su optimización por lo que, actualmente, es objeto de numerosas investigaciones que pretenden mejorar su planificación y gestión (Eshghali et al., 2023; Fontes et al., 2023; J. Wang & Wang, 2023), aunque difieren mucho tanto en sus objetivos como en las estrategias de optimización utilizadas.

Además, la pandemia COVID-19 dejó en evidencia la realidad, y es que no estábamos preparados para esos niveles de saturación de los hospitales por lo que la gestión del SUH ha atraído aún más la atención de los hospitales (X. Hu et al., 2021).

1.2 Presentación del problema

En este proyecto se busca la secuenciación de tareas óptima que se realizan en la atención de los pacientes el SUH. Dicha optimalidad la define su función objetivo (FO), que es minimizar el *Length of Stay* (LOS) y el *Time Time to Be Seen by a Physician or Physician Assistant* (TBSPPA), o Tiempo de Espera de Primera Consulta Facultativa (TEPCOF), de los pacientes que se encuentran en ese instante en el SUH.

Para modelar este problema, en otros estudios se ha usado como analogía el problema del Job Shop flexible (Gómez Medina, 2021; López Garcelán, 2023). En estos trabajos los pacientes representan los trabajos y las actividades de su proceso de urgencia (PU) son las operaciones específicas que se realizan sobre ese trabajo.

Sin embargo, la principal diferencia de este estudio respecto a anteriores investigaciones es que, mientras que en otras son los pacientes los que se van moviendo por el SUH durante su PU (consultas, salas de rayos X, laboratorios, etc,), en este caso el paciente permanece durante toda su estancia en un box, y son los recursos humanos los que acuden a su box para realizarle las pruebas pertinentes.

Dado que, en la realidad, esta modalidad normalmente solo se aplica a los pacientes críticos, este trabajo introduce un innovador enfoque de funcionamiento para poder compararlo con la modalidad actual de los SUH, en la que los pacientes sí se desplazan a los recursos para ser atendidos.

La optimización del SUH puede realizarse a dos niveles: a nivel operativo y a nivel táctico. A nivel táctico, se centra en la planificación a medio plazo y en la implementación de estrategias que tendrán un impacto a lo largo del tiempo mientras que, a nivel operativo, está enfocada en acciones que pueden implementarse de inmediato para abordar situaciones y necesidades inmediatas.

El problema de la secuenciación de tareas del personal sanitario es a nivel operativo. Esto implica que se da preferencia a encontrar una solución factible en el mínimo tiempo posible, más aún en el servicio que se estudia en el que la respuesta debe ser inmediata.

Puesto que el problema es de naturaleza *NP-hard* (Vali et al., 2022), recurrimos a metaheurísticas como técnicas de optimización aproximadas para encontrar soluciones. Concretamente la búsqueda de soluciones se hará mediante tres metaheurísticas; algoritmo genético, búsqueda tabú y recocido simulado. De esta manera se pueden comparar los resultados obtenidos por cada una de ellas.

Como se ha comentado previamente, los SUH son altamente estocásticos lo cual dificulta mucho su modelado y optimación. Para poder reducir la variabilidad de este sistema, en este trabajo propone realizar una 'fotografía' al estado concreto en el que se encuentra el SUH. De esta manera si entran o salen pacientes en ese momento no se tendrán en cuenta en nuestra búsqueda de soluciones. Además, se considera que se conoce de antemano el PU del paciente que se estudia y los recursos disponibles.

Así pues, aunque tomar la fotografía es un evento determinista, el estado del sistema que se captura es en sí mismo estocástico ya que la duración de las actividades o la prioridad de los pacientes sigue dependiendo de una probabilidad.

Dichos datos son extraídos del artículo de Bedoya-Valencia & Kirac (2016) en la fase de experimentación, para aplicar los algoritmos propuestos a un problema cercano a la realidad.

En resumen, en este trabajo se resuelve el problema de la secuenciación de las tareas de los procesos de atención de los pacientes en el SUH de una manera innovadora, ya que los pacientes no se mueven de los boxes, y aplicando metaheurísticas buscando minimizar su estancia y su espera.

1.3 Objetivos

El objetivo principal de este trabajo es diseñar y desarrollar algoritmos de optimización utilizando diferentes técnicas metaheurísticas, realizando la descodificación de los datos proporcionados y una experimentación con diferentes hipótesis de saturación para encontrar la mejor secuenciación de tareas de los recursos del SUH.

Los objetivos específicos son los siguientes:

- Optimización de la asignación de tareas: Asignar las tareas y recursos eficientemente en función del nivel de prioridad de los pacientes.
- Minimizar el *Lenght of Stay* (LOS): Hace referencia al tiempo de estancia total que un paciente permanece en el SUH.
- Minimizar el *Time to Be Seen by a Physician or Physician Assistant* (TBSPPA): El TBSPPA es el tiempo que un paciente debe esperar desde su admisión hasta que es atendido por un médico o médico asistente.
- Adaptación a la saturación del SUH: Crear un algoritmo que se adapte a la variación del nivel de saturación del SUH modificando únicamente el valor de los datos de entrada.
- Validación y comparación de resultados: Evaluar la efectividad y eficiencia de las metaheurísticas planteadas comparándolas entre sí mediante la fase de experimentación propuesta.
- Garantizar el equilibrado de recursos: Evitar que unos recursos tengan mayor carga de recursos que otros del mismo tipo asignando las tareas al recurso cuya tarea anterior haya finalizado antes.

1.4 Sumario

El presente trabajo se estructura en siete capítulos y un anexo. A continuación, se detallan los contenidos de cada uno:

Capítulo 1: Introducción. Se proporciona una visión general del contexto en el que se enmarca el estudio y se hace una breve introducción al problema específico que se busca resolver junto a los objetivos generales de este trabajo.

Capítulo 2: Estado del arte. Contiene la revisión de la literatura que se centra en estudiar los problemas presentes en el SUH.

Capítulo 3: Descripción del problema. En este capítulo, primero se hace una descripción del SUHs y su funcionamiento general. Luego se centra en explicar detalladamente el problema específico que es el objeto de este trabajo.

Capítulo 4: Metodología de resolución. Incluye una descripción detallada de los algoritmos utilizados para resolver el problema descrito. Detalla el procedimiento de decoding que se ha realizado junto a los tres algortimos metaheurísticos implementados.

Capítulo 5: Análisis computacional. Incluye la descripción de la jerarquía del código de programación que se ha desarrollado utilizando Python para evaluar los algoritmos propuestos.

Capítulo 6: Experimentación. En este capítulo se explica como se ha llevado a cabo la evaluación de los algoritmos. Para ello, primero se define cómo se obtienen los datos necesarios para resolver el problema. Luego se proponen seis escenarios diferentes para analizar el comportamiento de los algoritmos diseñados y se comparan los resultados obtenidos.

Capítulo 7: Conclusiones. Se incluyen las conclusiones generales del proyecto y se proponen futuras líneas de investigación.

Anexo A: Se muestran los resultados de la calibración de los algoritmos metaheurísticos.

Anexo B: Se indican los datos que se obtienen al implementar los algoritmos en la experimentación.

Anexo C: Incluye el código Python desarrollado para resolver el problema.

2 ESTADO DEL ARTE

Todos los SUHs tienen una serie de características comunes, como la atención inmediata, la disponibilidad de 24 horas los 7 días de la semana, un área de triaje que prioriza a los pacientes según su gravedad y un equipo de multidisciplinar que debe ser altamente coordinado para dar una respuesta inmediata y en el menos tiempo posible (María & Virto, 2017). Sin embargo, son muchas las variantes que se encuentran a la hora de modelar y optimizar la realidad, dada la complejidad e importancia del problema que se trata.

En este apartado se recogen y analizan las investigaciones que se han centrado en el estudio de la optimización de los SUH. Éstas se pueden dividir en dos grandes bloques en función del problema que analizan, modelan y optimizan (Gómez Medina, 2021): a nivel táctico y a nivel operativo.

A nivel táctico los estudios se centran en la toma de decisiones y planificación a medio plazo o que afectan a un periodo de tiempo amplio. Buscan la mejora de los procesos existentes, resolver problemas recurrentes o hacer frente a nuevos desafíos. Encontramos artículos que analizan los factores que influencian el LOS de los pacientes en el SUH con diferentes niveles de urgencia y hacen propuestas de mejora, por ejemplo Chaou et al. (2016), otros que llevan a cabo un análisis del tiempo de espera de los pacientes (Bahari et al., 2022) y otros que proponen modelos de planificación de los recursos y sus demandas (Chin & Fleisher, 1998). Pero la mayoría de los estudios de este nivel se centran en planificar los horarios del personal.

Para la toma de decisiones descrita anteriormente, es de vital importancia la información de entrada a los enfoques de resolución. Para ello, el Machine Learning es el enfoque más popular en el análisis de datos, como soporte a la predicción diaria del flujo de pacientes (Hamzaoui et al., 2022) y predicción del LOS (Umoren et al., 2019), sirviendo como una herramienta de apoyo a la toma de decisiones para mejorar la planificación de recursos en el SUH (Etu et al., 2022). En la investigación de Eshghali et al. (2023), utilizan datos históricos y en tiempo real para predecir patrones de demanda y optimizar la asignación de personal y recursos.

Por otro lado, el data-mining es el método más comúnmente aplicado para extraer y descubrir patrones en el flujo de pacientes. En el estudio de Abourraja et al. (2022) el data-mining proporciona conocimiento sobre la capacidad disponible del SUH y descubre patrones en el flujo de pacientes que ayudan a establecer los cimientos del modelo de simulación. Este modelo se utiliza para evaluar diseños alternativos de flujos de trabajo buscando reducir los tiempos de espera en SUH.

Para poder obtener los datos necesarios es necesario conocer el estado del SUH en ese instante. Por ello, Waskito et al. (2022) implementan un sistema que detecta y rastrea la llegada y salida de pacientes, permitiendo al personal del hospital gestionar datos sobre la disponibilidad de la ED, el dispositivo de detección de pacientes o el ingreso de pacientes.

Respecto a los enfoques de resolución que encontramos a nivel táctitco, la planificación de los horarios del personal es el objetivo propuesto por (Rossetti et al., (n.d.), que presentan un modelo de simulación de los diferentes procesos y sus correspondientes flujos de actividades con Arena, que analiza las alternativas de la programación de los horarios del personal médico, y así comparar los resultados obtenidos. Al igual que ellos Evans et al. (1996) también utilizan modelos de simulación con el objetivo de evaluar varios horarios de los técnicos, médicos y enfermeros para evaluar la duración media de la estancia de los pacientes. Con ese mismo objetivo, Feng et al. (2017) desarrollan una metodología de simulación-optimización multiobjetivo que combina el algoritmo genético de clasificación no dominada II (NSGA II) con la asignación de presupuesto de cómputo multiobjetivo (MOCBA). Van Bockstal & Maenhout (2018) también abordan un problema multiobjetivo buscando minimizar el tiempo de espera de los pacientes mediante un algoritmo genético. Por otro lado, Zeinali et al. (2015) aplican metamodelos de red neuronal artificial y función de base radial junto con un modelo de simulación de eventos discretos para optimizar la configuración de los recursos y así mejorar el flujo de pacientes. Por último, X. Hu et al. (2021) utilizan la tecnología de gemelo digital para la planificación de los horarios de los recursos.

En los últimos años, las redes de Petri también han sido altamente populares, puesto que son una herramienta poderosa para modelar sistemas de eventos discretos. Wang (2023) las aplica con el objetivo de encontrar los

niveles de personal óptimos. De manera similar Wang & Wang (2023) las emplean para optimizar la asignación de recursos, pero considerando la minimización del tiempo de espera de los pacientes como FO.

Por lo tanto, a nivel táctico cobra gran importancia la planificación, nivelación y gestión de los recursos, pero también la evaluación de indicadores tales como la saturación del SUH, la gestión interna y el rendimiento del propio servicio de Urgencias (Tudela & Deltell, 2015).

Por otro lado, la aparición de los algoritmos por ordenador, que proporcionan una respuesta rápida y evaluar las situaciones en cada instante, ha permitido dar respuesta a los problemas que encontramos a nivel operativo. Las investigaciones que se centran en este nivel buscan resolver problemas a corto plazo y que necesitan una respuesta inmediata. Se centra en la gestión y operación eficiente de los recursos y la atención a los pacientes en tiempo real.

Los estudios a nivel operativo modelan el flujo de los pacientes en su PU, así como los recursos y el personal sanitario que necesitan durante su estancia en el SUH. Frecuentemente el problema que se aborda en este nivel es la secuenciación de tareas del personal sanitario y de los recursos disponibles.

Ese es el caso que se trata en este trabajo, sirviendo como referencia el artículo de Bedoya-Valencia & Kirac, (2016). En este estudio analizan las alternativas de asignación de los recursos con el objetivo de mejorar la eficiencia y demostrar como impactan estas estrategias. Para ello utilizan una simulación de eventos discretos, gracias a la cual representan los diferentes escenarios propuestos. Su modelo cuenta con pacientes con diferentes niveles de gravedad y que pueden ser tratados por varios recursos a la vez. Utilizan tres parámetros medir y comparar los efectos de los distintos escenarios propuestos; el LOS, el tiempo de espera de los pacientes antes de la primera visita por el médico TBSPPA y la utilización de recursos medio.

Así pues, comúnmente encontramos investigaciones que tienen como objetivo reducir el tiempo de espera de los pacientes (Duguay & Chetouane, 2016; Y. Hu et al., 2022; Kırış et al., 2010; J. Wang & Wang, 2023) y otras que comparan la optimización del LOS con la asignación de recursos (Bedoya-Valencia & Kirac, 2016; Raunak et al., 2009)

Sin embargo, también encontramos otros indicadores como la utilización media de los recursos o el TBSPPA, que es el tiempo transcurre pasa desde que el paciente es registrado con una prioridad en triaje, hasta que es atendido por primera vez en por un médico (Bedoya-Valencia & Kirac, 2016)

En cuanto a los algoritmos que encontramos, al igual que en el de Bedoya-Valencia & Kirac (2016) los enfoques basados en simulación son ampliamente utilizados en la evaluación del rendimiento y la planificación de distribución de los servicios de urgencias (Duguay & Chetouane, 2016; Raunak et al., 2009). Liu et al. (2022) en su estudio integran la simulación de eventos discretos con Agent-Based Simulation. Plantean tres estrategias diferentes para asignar los pacientes a los recursos del SUH estudiando pacientes con diferentes necesidades.

A pesar de la utilidad destacable de la simulación, en algunas aplicaciones el simple mapeo de ruta es suficiente, ya que la simulación requiere más habilidades analíticas y tiempo (Mould et al., 2016).

Por otro lado, Kırış et al. (2010) desarrollan una heurística constructiva para la optimización de la secuenciación de los pacientes, buscando minimizar su tiempo de espera y atendiendo a sus prioridades asistenciales. El algoritmo diseñado también se encarga de equilibrar las cargas de los recursos. Harzi et al., (2018) propone un algoritmo híbrido entre una búsqueda local iterada (ILS) y una búsqueda de vecindad variable (VND). En este caso incluye las principales actividades del proceso del paciente en el SUH: triaje, consulta, tratamiento y hospitalización. Alves de Queiroz et al. (2021) utilizan la metaheurística general variable neighborhood search (GVNS) combinada con greedy para minimizar el retraso ponderado del tiempo de espera que se asigna al paciente. La GVNS utiliza una estrategia de búsqueda en la que se exploran diferentes vecindarios alrededor de una solución actual para buscar mejoras. Z. Wang et al. (2023) proponen una técnica de linealización para formular el problema como un modelo de programación entera mixta (MIP) e implementa un algoritmo búsqueda tabú para optimizar la secuenciación de las consultas de un SUH, analizando el tiempo de espera de los pacientes.

También se estudia la optimización con modelos de programación lineal entera (MILP), como es el caso del estudio de Harzi et al. (2017). Sin embargo, se concluye que ese modelo solo es capaz de encontrar soluciones óptimas en un tiempo factible en problemas de dimensiones reducidas. Daldoul et al. (2022) crean un modelo de programación lineal entera mixta (MILP) estocástico que resuelven mediante el Sample Average Approximation centrándose en el dimensionamiento de las camas del SUH, la minimización de los tiempos de

espera y del LOS medio y el equilibrado de las cargas de trabajo.

Y. Hu et al. (2022) aplican las redes de Petri buscando evaluar la cola de espera media y el tiempo de espera de los pacientes.

Por otro lado, la metodología del Quality Function Deployment (QFD) también ha sido altamente utilizada (Adel et al., 2018; Bouzir & Benammou, 2020). El QFD es una metodología que tiene como objetivo asegurar que los requisitos y preferencias se integran de manera efectiva en el proceso de planificación. Para ello utilizan una matriz que se denomina Casa de la Calidad. El indicador de calidad es una vez más el tiempo de espera de los pacientes ya que reducirlo significa que ha mejorado el flujo de los pacientes.

Por último, encontramos investigaciones que, aplican el problema del Job Shop en el contexto de los hospitales, aunque no en el SUH. Vali et al. (2022) presenta un problema de Job Shop flexible de taller para optimizar el flujo de pacientes y minimizar la huella de carbono total, como FO, y aplicando un novedoso algoritmo de optimización metaheurístico como FO. Burdett & Kozan (2018) desarrollan algoritmos metaheurísticos híbridos y constructivos para resolver el problema de Job Shop con el objetivo de utilizar los espacios de tratamiento de manera efectiva, considerando a pacientes, camas, áreas de hospital y actividades de atención médica como trabajos, máquinas individuales, máquinas paralelas y operaciones, respectivamente.

Cabe destacar también que, a raíz de la pandemia de COVID-19**,** son numerosas las investigaciones que se centran en el estudio de SUH considerando esta variable (Etu et al., 2022; Hamzaoui et al., 2021, 2022).

3 DESCRIPCIÓN DEL PROBLEMA

En este capítulo, primero se hace una descripción del SUH y su funcionamiento general. Posteriormente se centra en explicar detalladamente el problema específico que es el objeto de este trabajo.

3.1 Descripción del SUH

Para poder entender el problema que se aborda en este trabajo es importante entender los pasos que sigue un paciente desde su llegada a SUH hasta su alta. En la ilustración 1 se puede observar dicho proceso.

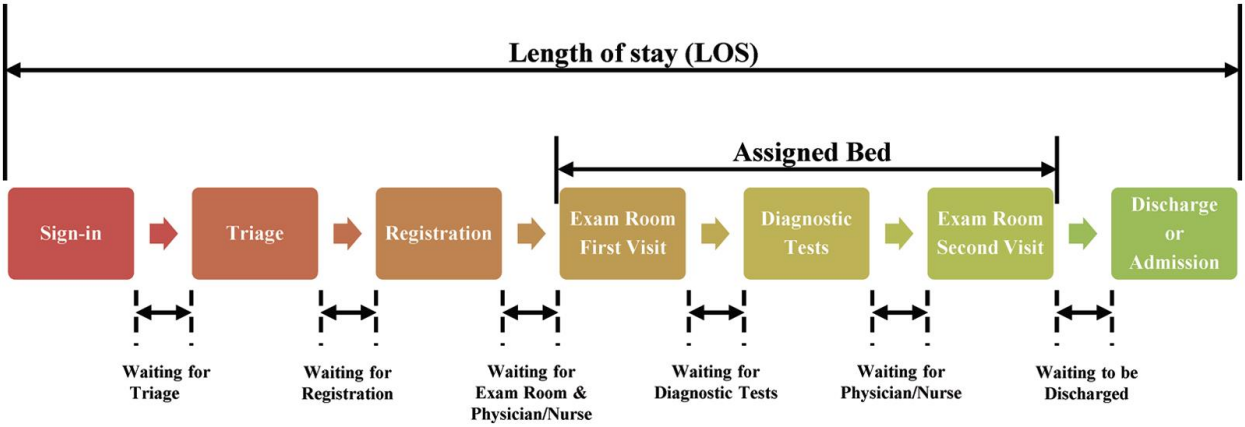

Ilustración 1. Pasos de los pacientes en el SUH (Bedoya-Valencia & Kirac, 2016).

Primero el paciente llega al SUH. La llegada de pacientes es totalmente aleatoria, por lo que habrá momentos en los que habrá suficientes recursos para todos los pacientes y otros en los que la que la demanda de atención médica superará la capacidad de los recursos disponibles. A ese concepto se le denomina saturación. Es interesante introducirlo ya que se tratará más adelante.

Tras la llegada, los pacientes pasan a un proceso de admisión. En primer lugar, el paciente es atendido por el personal de recepción que recogerá todos los datos del paciente y, posteriormente será atendido por una enfermera especializada. Esta profesional, junto a un médico, determinan la gravedad del paciente atendiendo a sus quejas y a otros elementos como la tensión arterial, el pulso, la temperatura y el dolor (Hospitaux universitaires de gèneve, n.d.).

A ese proceso de evaluación médica se le denomina triaje y según la gravedad se le asigna un nivel de ESI (*Emergency Severity Index*). La escala comprende valores del 1 al 5, siendo el ESI 1 el que categoriza a los pacientes más urgentes y el ESI 5 a los menos urgentes. La prioridad de atención de los pacientes está determinada por su nivel de ESI y su valor se corresponde con dicho nivel. Los pacientes de prioridad o nivel ESI 1 son aquellos cuyos los síntomas o signos que presenta sugieren un riesgo vital inminente (sepsis, infarto, ictus y trauma grave). Los pacientes categorizados como prioridad 2 son aquellos con sospecha de una patología aguda o exacerbación de una patología crónica, sin un riesgo vital o funcional inmediato pero que, si no se brinda esta atención, podrían evolucionar a prioridad 1. Aquellos a los que se les asigna la prioridad 3, son situaciones de urgencia relativas. Por último, los de prioridad 4 y 5 no requieren actuaciones urgentes. Los de ESI 4 pueden precisar de alguna actividad sanitaria pero los de ESI 5 se corresponden con pacientes cuyos problemas de salud no deben ser atendidos en el SUH (Álvarez Rueda et al., 2018).

El paciente que acude al SUH debe recibir una respuesta adecuada a la gravedad del problema que padece

(Álvarez Rueda et al., 2018).

Es por ello por lo que el nivel ESI asignado en el triaje determina también en el PU que seguirán, los recursos necesarios para su correcta atención y el tiempo máximo tolerable para proporcionarles atención (conocido como TBSPPA). Dicho TBSPPA máximo es definido por la Junta de Andalucía (Jiménez Murillo et al., 2019) y se muestra en la tabla 1.

ESI	Prioridad	TBSPPA máximo
ESI-1	1	0
ESI-2	2	15
ESI-3	3	60
ESI-4	4	100
ESI-5	5	120

Tabla 1. TBSPPA máximo en función de la prioridad o nivel ESI del paciente (Jiménez Murillo et al., 2019)

Una vez determinada la prioridad o nivel ESI del paciente, este deberá ser atendido en una consulta facultativa (médico o médico asistente) antes de alcanzar su TBSPPA máximo. Esta será la primera actividad del PU en el SUH de los pacientes. Dichos PU están formados por unas actividades que tienen asociadas unos recursos concretos y varían en función la prioridad del paciente. Esas actividades son las pruebas que se deben realizar a los pacientes y permiten diagnosticarlos y ofrecerles un remedio. Se muestran en la tabla 2.

Actividades	Recursos
Primera visita facultativa	Médicos o médicos asistentes
Primera visita enfermería	Enfermeros
Rayos- X	Máquinas rayos - X
Análisis de sangre	Laboratorios
TAC	Escáneres TC
Segunda visita facultativa	Médicos o médicos asistentes
Segunda visita de enfermería	Enfermeros
Alta	

Tabla 2. Relación entre las actividades y los recursos que la realizan.

La primera actividad del PU del paciente, como se ha mencionado previamente, es la consulta facultativa. En ella se determinan las pruebas de diagnóstico que serán necesarias. En las actividades de consulta facultativa, los pacientes con prioridad 1, 2 y 3 son atendidos por un médico, mientras que los pacientes con ESI nivel 4 y 5 son atendidos por un médico asistente

A continuación, los pacientes son atendidos por los enfermeros que preparan a los pacientes para dichas pruebas. Las pruebas que se realizan pueden ser una radiografía, un análisis de sangre, un TAC, etc. Una vez finalizadas se vuelven a ser atendidos en consulta facultativa y por los enfermeros. Concluido su PU el paciente será dado

de alta o admitido en el hospital (Bedoya-Valencia & Kirac, 2016). El alta es una actividad ficticia, como no requiere recurso no se tendrá en cuenta en este problema.

Como se ha explicado anteriormente, los PU varían en función de la prioridad o nivel ESI. Es decir, un paciente con nivel ESI 1 requerirá de todas las pruebas de diagnósticos mencionadas y, además, necesitarán ser atendidos por dos enfermeros de manera simultánea y un médico. Sin embargo, un paciente con nivel ESI 5 solo necesita atendido una vez por un médico asistente y un enfermero.

3.2. Descripción del problema específico

En el problema que se resuelve en este trabajo el paciente debe ser asignado a un box para poder ser atendido. Todas las consultas y pruebas que se le realizan serán en este box, es decir, son los recursos los que se desplazan en lugar del paciente.

Cada box es ocupado por un único paciente a la vez, y no es liberado hasta que el paciente finaliza su PU. Esto asegura que cada paciente reciba la atención y el espacio necesarios sin interrupciones ni cambios de ubicación durante su atención médica.

Para poder llevar a cabo esta hipótesis es necesario que los recursos sean portátiles, es decir, puedan ser trasladados de un box a otro con facilidad. Los recursos humanos (médicos, médicos asistentes y enfermeros) no son un inconveniente ya que se pueden trasladar donde son requeridos sin problemas. En cuanto a los otros tipos de recursos (escáneres TC, máquinas de Rayos X y laboratorios), encontramos varias marcas que han diseñado equipos que cumplen con esta especificación, incluyendo sistemas portátiles de análisis de sangre que serían equivalentes a los laboratorios.

La prioridad de los pacientes se considera estocástica ya que depende de su gravedad médica, de la necesidad de atención inmediata, del historial médico, de la edad del paciente, etc. Como se ha explicado previamente, en función de la prioridad del paciente, su PU requerirá de unas actividades específicas u otras, necesitarán más o menos recursos por actividad y variarán sus duraciones. Todos estos datos son difíciles de predecir por lo que para la evalución de los algoritmos utilizaremos los datos que nos proporciona Bedoya-Valencia & Kirac (2016). Así pues, para resolver el problema que se aborda se conocen las probabilidades asociadas a cada nivel de prioridad, los PU, los recursos que se requieren y la duración de las actividades. Estas últimas seguirán una distribución triangular.

En resumen, en los presentes algoritmos se considera que los pacientes están definidos por su nivel ESI o de prioridad asociado, el PU por dicho nivel ESI y la duración de actividades, los tipos de recursos y el número de recursos que las realizan por el PU.

Para poder simplificar el problema también es necesario definir una serie de suposiciones y límites del alcance. En este caso seguimos las propuestas por Bedoya-Valencia & Kirac (2016), que son:

- La disponibilidad de los recursos humanos varía a lo largo del día ya que están sujetos a unos horarios.
- No se tendrán en cuenta los tiempos de traslado de los recursos humanos entre los boxes, ya que se consideran despreciables en comparación con la duración de las actividades.
- Los equipos necesarios para realizar las pruebas de diagnóstico (Scáner TC, Rayos-X y laboratorio) estarán operativos las 24 horas del día.
- Los pacientes no dejarán el SUH sin haber realizado su PU completo.
- Una vez asignado un nivel de urgencia al paciente en el triaje este no cambiará durante su estancia en el SUH.

La FO buscará minimizar dos índices: el LOS total y el TBSPPA total. El LOS total será la suma de los tiempos de estancia de todos los pacientes que son procesados por los algoritmos. Por otro lado, el TBSPPA solo será positivo y, por tanto, penalizará la FO si se ha superado el TBSPPA máximo definido por la Junta de Andalucía y que se muestra en la Tabla 1. Además, se ponderará de manera que penalice más cuanta más prioridad tuviera el paciente.

La FO será menor cuanto mejor sea la secuenciación de las actividades de los PU de los pacientes que se asignan

a los recursos disponibles, respetando una serie de restricciones.

- No se podrá realizar una actividad si no hay recursos disponibles del tipo que se necesita para realizarlas.
- No se libera el box hasta que el paciente que lo ocupa finaliza su PU.
- Se debe respetar el orden de la secuenciación de actividades que definen el PU de un paciente, es decir, una actividad no podrá comenzar si no ha terminado la actividad que le precede.
- Solo se atenderá a un paciente cuando esté asignado a un box.
- Una vez asignada una tarea a un recurso se mantendrá asignada, así como se conservará su instante de inicio y finalización.

Además, se intenta que la asignación de tareas a recursos se haga de manera que los recursos del mismo tipo tengan una carga de trabajo equilibrada, asignando la tarea al recurso cuya actividad anterior haya finalizado antes.

4 METODOLOGÍA DE RESOLUCIÓN

Este capítulo incluye una descripción de los algoritmos que se aplican para resolver el problema de secuenciación de tareas en el SUH, explicando como se han implementado.

Se han aplicado tres de las metaheurísticas más utilizadas para resolver problemas de naturaleza *NP-Hard* (Vidal et al., 2013): el algoritmo genético, el recocido simulado y la búsqueda tabú. Cada uno de estos algoritmos buscan una solución al problema de manera independiente, para que sea posible evaluarlos individualmente y compararlos posteriormente.

Los algoritmos metaheurísticos se encargan de explorar una representación de la solución, que consiste en una lista ordenada de los pacientes que se encuentran en el SUH con sus prioridades asociadas, con el objetivo de encontrar la mejor solución posible al problema. Pero, para evaluarla, es necesario llevar a cabo un procedimiento llamado "*decoding*". Éste es el encargado de proporcionar un valor de la FO partiendo de la representación de la solución aportada por los algoritmos y unos datos iniciales. En este apartado también se detalla la descripción de este procedimiento y se proporciona un ejemplo para su mayor comprensión.

4.1. Decoding

4.1.1 Descripción del *decoding*

Para calcular el valor de la FO, el *decoding* determina la secuenciación de las tareas de los recursos del SUH. Esto lo consigue calculando el instante de inicio y finalización y el recurso que realiza cada una de las actividades de los PU de los pacientes de la representación de la solución, cumpliendo con las restricciones que imponen la secuenciación de actividades de los PU, la disponibilidad de los recursos, el orden de atención de los pacientes, etc.

Se parte de una representación de la solución. Ésta es una lista con los pacientes que deben atenderse y sus prioridades o nivel ESI asociadas.

Para controlar las actividades asignadas a cada recurso se han creado unas matrices, una por tipo recurso, que contienen una lista por cada recurso del mismo tipo y dentro de estas listas se almacenan los tiempos de inicio, finalización de las actividades que se asignan a ese recurso y paciente al que se le realiza. Por lo tanto, se diseñan 7 matrices: *t_BOX*, *t_P*, *t_N*, *t_PA*, *t_LAB*, *t_SCAN* y *t_RAY*.

Como se ha explicado anteriormente en el problema planteado los pacientes son asignados a un box durante todo el PU, y el resto de los recursos acuden al box a atenderlo. No se atienden los pacientes a menos que estén ocupando un box, por lo que si hay un recurso diferente al box ocupado por un paciente significa que dicho paciente ocupa también un box en ese instante.

Lo primero que se realiza es la comprobación de los recursos que se encuentran ocupados en el instante que se está examinando, distinguiendo entre dos grupos, los que son boxes y los que no. De cada recurso ocupado al inicio se conoce el tiempo restante ocupado, el tipo de recurso, el número de recursos de ese tipo necesario y el paciente que lo ocupa.

Para asignar los pacientes a un box, se toma la matriz de ese recurso y se asigna el paciente a uno de los boxes disponibles, indicando el instante de inicio y liberación del box. Para asegurar que los boxes ocupados en el instante de inicio no sean asignados a otros pacientes antes de que los pacientes que lo ocupan terminen su PU, se les asigna un tiempo de fin de ocupación muy superior a los tiempos medios de ocupación, ya que a priori se desconoce el tiempo que durará la estancia del paciente. Ese valor se sobrescribirá con el valor real una vez que son liberados.

De igual manera, se van asignando el resto de los tipos de los recursos que se encuentran ocupados al inicio, siendo el instante de inicio igual a cero, pero con la diferencia de que en este caso sí se conoce el tiempo de finalización de la tarea.

Una vez comprobado esto, se empiezan a asignar el resto de las actividades de los PU de los pacientes. Éstas se irán colocando siguiendo el orden de la representación de la solución. Es decir, si el orden de la solución es {paciente2, paciente3, paciente1}, primero se colocarán las actividades del paciente 2, a continuación, las del 3 y por último las del 1. Que se coloquen antes las actividades del PU de un paciente no implica necesariamente que estas vayan a realizarse antes que las del PU de otro, ya que puede ocurrir que entre las tareas que tiene asignadas un recurso haya un tiempo en el que esté disponible. A ese tiempo disponible es al que llamamos hueco. Por ello, al colocar las actividades del PU del paciente, se debe ir atendiendo a los huecos que pueden ir quedando disponibles tras colocar las actividades de los pacientes anteriores.

Para que una actividad pueda iniciarse tiene que cumplirse que la actividad anterior del PU de ese paciente haya finalizado y que los recursos que necesita la actividad actual no estén ocupados.

Para calcular la segunda restricción, se examinan las actividades ya asignadas al recurso que se necesita en la matriz del tipo de recurso que corresponda. Por ejemplo, si la actividad debe ser realizada por un médico se toma la matriz *t_P* y se calculan, en cada recurso de ese tipo, los huecos que hay disponibles entre actividades anteriormente asignadas. Posteriormente, se comprueba si es posible colocar la actividad ahí. Para ello el tamaño del hueco debe ser mayor que la duración de la actividad y debe iniciarse después del fin de la actividad anterior del PU del paciente. En caso positivo, se almacena ese hueco y en caso negativo pueden ocurrir dos cosas; que busque en el siguiente hueco, si lo hay, o que se asigne como última actividad del recurso.

Si hubiera más de un hueco disponible en más de un recurso del mismo tipo se tomará aquel cuya tarea anterior haya finalizado antes, para así equilibrar las tareas de los recursos.

En los casos en los que se requiere más de un recurso de un mismo tipo, el procedimiento calcula la asignación de manera análoga, pero buscando varios huecos disponibles simultáneos en lugar de uno.

Una vez asignadas todas las actividades de los pacientes, el valor de la FO estará definida por dos valores: la suma de los LOS de todos los pacientes más los TBSPPA en el caso de que sea superior al máximo establecido por la Junta de Andalucía (Jiménez Murillo et al., 2019).

El LOS es el tiempo total de estancia de los pacientes en el SUH. De esta manera, el LOS de cada uno de los pacientes es igual a la suma del tiempo que transcurre desde el instante que se evalúa hasta al momento de finalización de su última actividad, que se corresponderá también con el instante de liberación del box, más el tiempo que el paciente espera desde que llegó hasta el instante que se lanza el análisis. Este valor será menor cuanto mejor asignadas estén las actividades de sus PU a los recursos correspondientes, es decir, cuanto mejor sea su secuenciación.

Respecto al cálculo del TBSPPA de cada paciente, también es la suma de dos valores; el primero es el tiempo que ha estado esperando el paciente desde que llegó hasta el instante que se lanza el análisis y es uno de los datos de entrada del *decoding*. El segundo es el tiempo que transcurre desde que se inicia el análisis hasta que es atendido por primera vez por el médico o el médico asistente. El TBSPPA solo será positivo en el caso de que el paciente no se encuentre en mitad de su PU en el instante que se realiza el análisis, es decir, asignado a un box. Una vez obtenido el TBSPPA se compara con los valores máximos definidos por la Junta de Andalucía; si no lo supera no se tendrá en cuenta en el cálculo de la FO, pero si lo supera se ponderará en función de la prioridad del paciente, otorgando mayor importancia a los pacientes que requieren una atención más urgente, y se sumará al valor de la FO.

De esta manera, el *decoding* devuelve el valor de la FO calculado para dicha representación de la solución.

4.1.2 Ejemplo de implementación

Para una mayor comprensión, se hará una demostración de cómo se mostraría la secuenciación de tareas. Para ello se ha trasladado a un programa, diseñado con el lenguaje de programación de Python, siguiendo las directrices explicadas en el apartado anterior.

Partimos de la siguiente representación de la solución:

$$Representación\_solución = [\,('Pac1', 3), ('Pac3', 3), ('Pac2', 4), ('Pac4', 2),\,]$$

Se trata de lista de pacientes con sus prioridades asociadas.

Como se ha mencionado previamente, otros datos que se proporcionan son los recursos que se encuentran

ocupados en ese instante, en este caso serían:

$$Recursos\_ocupados = [[1000, 'BOX', 1, 'Pac1'], [1000, 'BOX', 1, 'Pac3'], [3, 'LAB', 1, 'Pac3']]$$

Cada array representa un recurso ocupado, siendo la primera posición del array el tiempo restante de ocupación, la segunda el tipo de recurso, la tercera el número de recursos necesarios de ese tipo y la cuarta el paciente que ocupa el recurso. Los boxes se ocupan con un valor que será bastante superior al LOS estándar de un paciente. El resto de los recursos, en este caso laboratorio, únicamente por el tiempo restante de la actividad.

Los recursos totales, independientemente de si están ocupados en el instante inicial, también se conocen y para este ejemplo son:

Tipo de recurso	Cantidad
Boxes (B)	2
Médicos (P)	1
Enfermeros (N)	6
Médico asistente (PA)	1
Laboratorios (LAB)	1
Máquinas de Rayos-X (RAY)	1
Escáneres CT (SCAN)	1

Tabla 3. Lista de recursos totales

También se proporcionan la matriz con las actividades, sus duraciones, el tipo de recurso que las realiza y el número de recursos necesarios que componen los PU de esos pacientes y que serían las siguientes:

Paciente	Prioridad	Actividades	Duración	Tipo de recursos	Número necesario
1	**3**	2º visita médico	16	P	1
		2º visita enfermera	18	N	1
3	**3**	Prueba rayos	12	RAY	1
		2º consulta médico	10	P	1
		2º consulta enfermera	17	N	1
2	**4**	1º visita médico asistente	4	PA	1
		1º visita enfermera	16	N	1
		Prueba rayos	7	RAY	1
		2º consulta médico asistente	8	PA	1
		2º consulta enfermera	9	N	1
4	**2**	1º visita médico	25	P	1
		1º visita enfermera	24	N	2
		Prueba laboratorio sangre	6	LAB	1
		Prueba rayos	10	RAY	1
		Prueba CT-Scan	15	SCAN	1
		2º consulta médico	17	P	1
		2º consulta enfermera	22	N	2

Tabla 4. Lista de actividades y recursos de los PUs

Por último, se proporcionan los tiempos de espera de los pacientes en el instante inicial, que respetando el orden de la representación de la solución sería:

$$tiempos\_esperados = [0, 0{,}1{,}14]$$

Los pacientes 1 y 3 están en mitad del proceso, luego sus tiempos de espera no interesan porque son anteriores al instante que se estudia.

La secuenciación de tareas calculada para este ejemplo se muestra en la Ilustración 2:

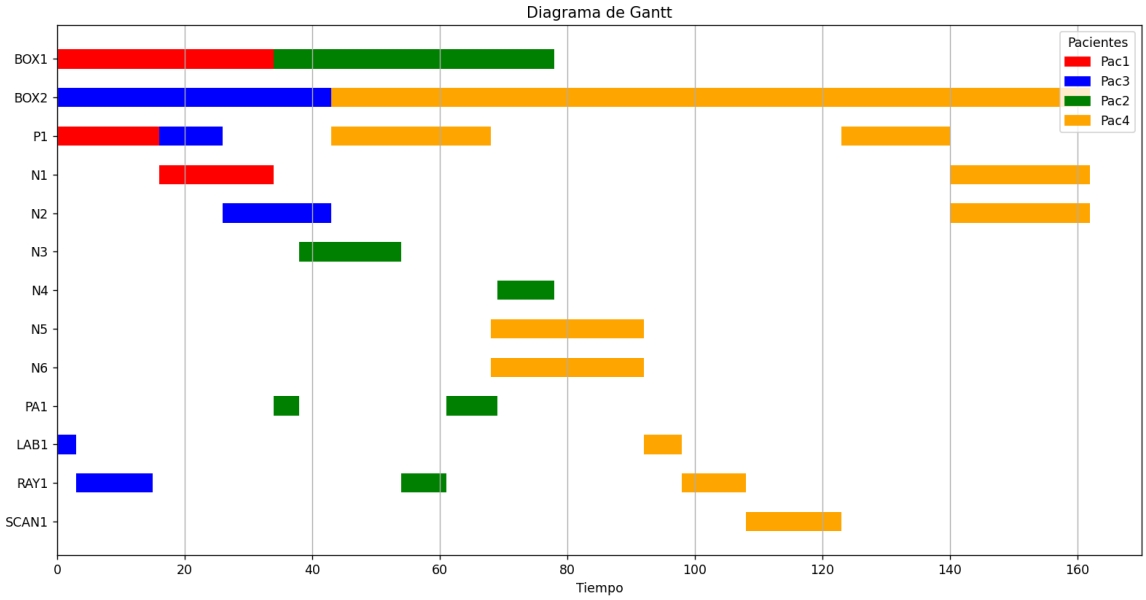

Ilustración 2. Secuenciación de tareas

Las actividades se van colocando siguiendo el orden de pacientes de la representación de la solución.

Al haber solo dos boxes y estar ambos ocupados por los pacientes 1 y 3 en el instante inicial, tienen que liberarse para poder ser utilizados por el resto de los pacientes. Es por ello por lo que las primeras tareas que se asignan a los recursos son las del paciente 1.

A continuación, se colocan las del paciente 3, comenzando por el laboratorio que era la actividad que se le estaba realizando al paciente en el momento del análisis. En este paciente también se puede observar que para poder ser atendido por el médico *(P1*, en la ilustración 2) debe esperar a que este sea liberado por el paciente 1 pese que su actividad anterior había finalizado antes.

El siguiente en asignarse es el paciente 2. Primero selecciona el box que se ha liberado antes, en este caso el *BOX 1* en la ilustración 2. Se asignan el resto de las actividades intentando que haya equilibrio en la carga de trabajo entre los recursos del mismo tipo.

Por último, se asigna el paciente 4 al box 2 y comienza su PU. En este caso necesita 2 recursos de enfermeros que trabajen simultáneamente, por lo que busca dos que estén libres a la vez para realizar esas actividades.

Otro aspecto que destacar de esta solución es cómo se ha logrado asignar las tareas a los recursos de manera equilibrada. Si observamos los recursos de tipo Enfermeros (*N*, en la ilustración 2), que en este caso hay 6, se puede comprobar que se ha priorizado la asignación de una tarea a aquellos que no hubieran realizado ninguna anteriormente o, en el caso de las últimas actividades del paciente 4, a aquellos que se habían liberado antes.

Como se ha explicado previamente, el cálculo de la FO se compone de la suma de dos valores; el LOS total y el TBSPPA total.

El LOS de cada paciente es la suma del tiempo que estuvo esperando antes del instante del análisis más el tiempo que transcurre hasta su alta, es decir, hasta que libera el box que ocupa. En el caso del paciente 1, su espera previa al instante del análisis es de 0 minutos por lo que el LOS tiene un valor de 37 minutos. El paciente 3 se calcula de manera similar y el resultado es de 42 minutos. El valor de los LOS de los pacientes 2 y 4 incluyen su espera previa, más el tiempo que han tenido que esperar a que se libere un box para ser atendidos, más el tiempo en el que se les realizan las pruebas de sus PU cuando son asignados a un box. Sus LOS son de 79 y 176

minutos respectivamente.

Por otro lado, respecto a los TBSPPA, como se ha explicado anteriormente esta variable solo tendrá valor si el paciente no se encontraba en mitad del PU, es decir, solo se contempla para los pacientes 2 y 4. Los datos de entrada indican que el paciente 2 acaba de llegar por lo que su tiempo de llegada será 1, siendo su límite máximo de TBSPPA de 100 minutos por su prioridad. A esto hay que sumarle que aún ha tenido que esperar 37 minutos más para ser atendido por un médico asistente, ya que no había boxes libres antes. Su espera total es de 38 minutos. El paciente 4 por su parte tiene un tiempo inicial de espera de 14 minutos, solo un minuto por debajo del límite máximo de TBSPPA establecido para su prioridad. Además, ha de esperar 42 minutos más a ser atendido por el médico, por lo que su espera total sería de 56 minutos. Sin embargo, en el paciente 2 al no superar el tiempo de espera el TBSPPA máximo, tiene un valor de TBSPPA en la FO de 0, es decir, no la penaliza. Por otro lado, el paciente 4 si ha superado su TBSPPA límite por lo que se penaliza dicho valor dándole más peso acorde a su prioridad siendo su valor penalizado de 224 minutos, y se suma a la FO.

En resumen, los resultados que se obtienen en este ejemplo son:

LOS_total= 334 minutos.

TBSPPA_total = 224 minutos.

FO = 558 minutos.

4.2. Algoritmos de optimización

Para la optimización del problema se han seleccionado tres algoritmos metaheurísticos de diversificación diferentes: el recocido simulado, la búsqueda tabú y el algoritmo genético.

Estos algoritmos reciben dos datos de entrada necesarios para su aplicación; el tiempo máximo de computación, que dependerá del tamaño del problema, y una representación de la solución inicial. Como se ha comentado anteriormente, la representación de la solución inicial es una lista ordenada de los pacientes que deben ser tratados y sus prioridades o nivel ESI. Los tres iteran sobre esa representación de la solución y seleccionan aquellas nuevas representaciones que minimizan el valor de la FO, hasta alcanzar el tiempo máximo de computación.

La evaluación de las nuevas representaciones de la solución que construyen estos algoritmos se realiza mediante el *decoding* explicado anteriormente. Si se varía el orden de la representación de la solución también cambia el orden en el que se irán colocando las tareas en el *decoding* por lo que la secuenciación de las tareas y la FO también serán diferentes.

4.2.1 Recocido Simulado

El Recocido Simulado es un algoritmo metaheurístico de diversificación que trata de mejorar la búsqueda local permitiendo en algunos momentos tomar soluciones peores y, de esta manera, tratar de escapar de lo óptimos locales (Delahaye et al., 2019). A su vez utiliza valores similares a otras soluciones para encontrar el resultado. Es uno de los algoritmos más utilizados ya que, si están bien parametrizados, es capaz de encontrar buenas soluciones en poco tiempo. Utiliza como metáfora los procesos que ocurren en el proceso metalúrgico del recocido, intentando simularlo, como su propio nombre indica. Los principios del Recocido Simulado se exponen en la ilustración 3.

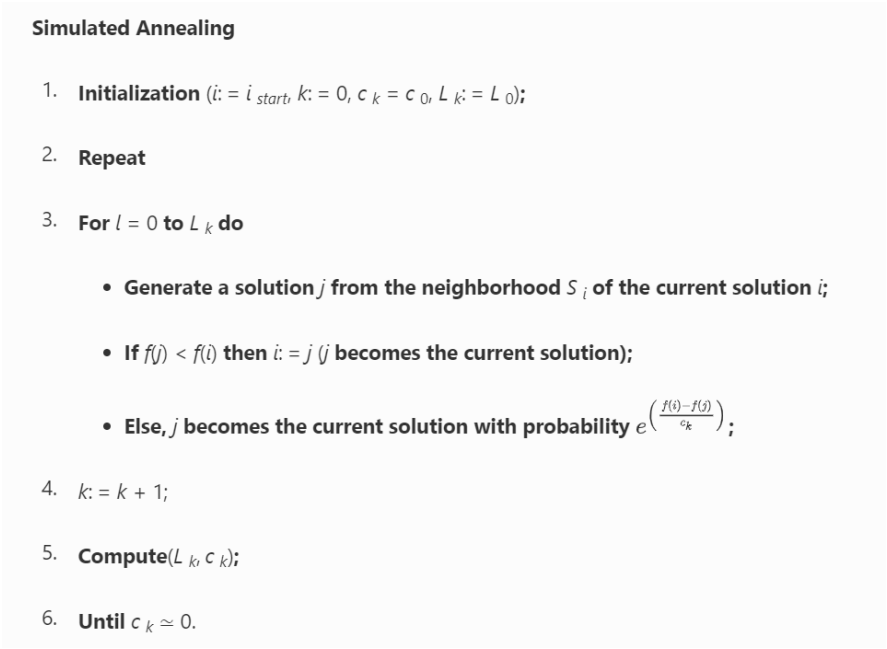

Ilustración 3. Procedimiento de aplicación del Recocido Simulado (Delahaye et al., 2019)

En este trabajo se ha adaptado tomando como base esos pasos de la siguiente manera:

1. Inicialización de Parámetros:

   Temperatura inicial (C_0).

   Temperatura final (C_k).

   Número de iteraciones en cada nivel de temperatura (L_k).

   Factor de disminución de temperatura (α).

2. Inicialización de Variables

   Se inicializa la temperatura actual con la temperatura máxima. (C_0)

   Se inicializa las listas de soluciones actuales y mejores (current_solution y best_solution).

   Se calcula el valor de la función de objetivo para la solución actual (current_LOS) y lo asigna como el mejor valor inicial (best_LOS).

3. Se ejecutará el código mientras no se alcance el tiempo límite de computación:
   i. Mientras no se alcance la temperatura mínima (C_k)
      a. Para un mismo nivel de temperatura: (L_k).
         1. Generación de la vecindad: Se crea una lista vecindad (S) que contiene la representación de la solución actual y todos los posibles vecinos generados a partir de esta. Estos se generan mediante el operador insertion, considerando todas las posibles permutaciones de intercambio.
         2. Se cambia el orden del vecindario para evitar repeticiones en la exploración.
         3. Se elige y evalúa un vecino aleatorio (j)
            a. Si la nueva selección es mejor que la de la actual se toma como solución actual. Si además es mejor que las de todas las vecindades evaluadas hasta el momento se toma como mejor solución
            b. Si no, se calcula la probabilidad de aceptación según la fórmula de probabilidad de Boltzmann y se compara con un número aleatorio (a). Si cumple la condición se toma como solución actual, aunque sea

peor.

ii. Se actualiza el valor de la temperatura operando con el Esquema de Cauchy modificado.

iii. Se comprueba el tiempo límite de computación

Al pseudocódigo de Delahaye et al. (2019) se le ha añadido un bucle adicional, que comprueba si se ha alcanzado el tiempo de computación. Además, se añade una función random que desordena la vecindad generada para evitar repeticiones.

Existen varias funciones de enfriamiento que se aplican para actualizar el valor de la temperatura. En el algoritmo propuesto se implementa el Esquema de Cauchy modificado (Cuberos et al., 2014):

$$C_{0+1} = C_0 / (1 + \alpha * C_0)$$

La velocidad de enfriamiento será menor cuanto más bajo sea el valor de la tasa de enfriamiento (α).

Una vez alcanzado el tiempo máximo de computación el algoritmo devuelve la mejor solución evaluada.

Para los valores de los parámetros se han tomado como referencia los propuestos por Dorgham et al., (2019):

- Temperatura inicial: $C_0 = 1000$
- Temperatura mínima: $C_k = 0{,}01$
- Tasa de enfriamiento: $\alpha = 0{,}94$

Para la elección del valor de Lk se han evaluado 10 instancias diferentes. En cada una de ellas se genera una representación de la solución de tamaño (*N*) y se ejecuta el algoritmo del recocido simulado en tres escenarios, todos con el mismo tiempo de computación, cambiando en cada escenario el valor de *Lk* entre los siguientes:

$Lk = [N/2, N, 2N]$

Tras realizar el análisis[1] se concluye que el valor con el que se obtienen mejores valores de la FO es $Lk = 2N$.

4.2.2 Búsqueda Tabú

La búsqueda Tabú es un procedimiento metaheurístico cuya principal característica es que evita reevaluar soluciones que ya han sido evaluadas antes, incluyéndolas en una lista tabú durante una serie de iteraciones.

La primera parte del algoritmo es similar a la del algoritmo simulado. Se impone un tiempo de computación máximo y se inicializan los valores de las soluciones. También se crean vecinos mediante el operador insertion.

La particularidad de este algoritmo reside en la lista tabú (Velez & Montoya, 2007). La lista tabú es una lista que se utiliza para mantener un registro de los movimientos recientes durante un período de tiempo específico. Los movimientos que se almacenan son los que permiten mejorar la solución actual. Esto evita que se repitan movimientos en un corto plazo. Sin embargo, se permite que un movimiento sea considerado si cumple con una condición especial llamada "criterio de aspiración". Esta condición generalmente se refiere a permitir un movimiento que se encuentra en la lista tabú si mejora la mejor solución hasta el momento. Encontramos otros criterios de aspiración, pero en este trabajo es ese el que se implementa.

Las bases del algoritmo son las siguiente:

[1] Los resultados obtenidos se muestran en la tabla 17 que se encuentra en el Anexo A.

Sea $x^* \in \Omega$ la mejor solución encontrada
Sea N_A el nivel de aspiración

Hacer $x^* \leftarrow x_0$
Mientras no se cumpla el criterio de parada:
  Seleccionar al azar el conjunto $V \subseteq N(x^*)$ de vecinos de x^*
  Sea $x' \in V$ la mejor solución en V
  Si el movimiento que generó x' no está en la lista Tabú
    Aceptar el movimiento haciendo $x^* \leftarrow x'$
    Actualizar la lista Tabú incorporando a la lista el movimiento que generó x'
    Actualizar N_A
  Sino
    Si se cumple el criterio de aspiración
      Aceptar el movimiento haciendo $x^* \leftarrow x'$
      Actualizar la lista Tabú incorporando a la lista el movimiento que generó x'
    Fin
  Fin
Fin

Ilustración 4. Procedimiento búsqueda tabú (Velez & Montoya, 2007).

La aplicación de este algoritmo en el presente estudio introduce algunos cambios al procedimiento presentado por Velez & Montoya (2007) . En el algoritmo implementado, el conjunto de vecinos *V* se selecciona evaluando todas las soluciones que se generan a partir de la solución actual y solo se añadirá a la vecindad aquellos que no se encuentran en la lista tabú o cumplen el criterio de aspiración, aunque se encuentre en la lista tabú. Una vez creada la vecindad *V* se selecciona de ella el vecino X' con menor valor de FO y se añade el cambio que lo ha generado a la lista tabú. Los pasos se detallan a continuación:

1. Se inicializa el número de iteraciones que un cambio será tabú (*tabú_size*).
2. Se inicializan la solución actual (X_0), su valor de la FO, el de la mejor solución conocida (X*), el valor de la FO de la mejor solución conocida y el de la lista tabú (*tabu_list*).
3. Mientras no se alcanza el tiempo límite de computación
    i. A partir de una solución actual (X_0):
        a. Se crean vecinos a partir de la solución actual mediante el operador insertion.
        b. Se evalúa cada vecino creado (X') y se almacena el cambio que lo ha generado.
        c. Se verifica si el intercambio de posiciones que genera X' no está en la lista tabú o si el valor de la solución vecina es mejor que el mejor valor conocido, aunque esté en la lista tabú (criterio de aspiración). Si se cumple la condición se agrega el vecino a la vecindad (*V*) y se guarda su valor de función de evaluación y se registra la posición de intercambio (swap).
    ii. Se selecciona el vecino con el menor valor de FO, y se actualiza la solución actual (X')
    iii. Si es mejor que la mejor solución histórica (X*) se toma ese valor como mejor solución.
    iv. Se actualiza la lista tabú (*tabu_list*) marcando la posición del cambio y disminuyendo los contadores de las demás posiciones.
    v. Se comprueba el tiempo de computación.

Para la elección del tamaño de la lista tabú se han evaluado 10 instancias diferentes. En cada una de ellas se genera una representación de la solución de tamaño (*N*) y se ejecuta el algoritmo de la búsqueda tabú en tres escenarios, todos con el mismo tiempo de computación, cambiando en cada escenario el tamaño de la lista tabú entre los siguientes:

Tabú_size = [*N*/2, *N*, 2*N*]

Con los resultados obtenidos[2] en esta experimentación se determina que el tamaño adecuado de la lista tabú es de 2*N*.

4.2.3 Algoritmo genético

Se trata de una metaheurística evolutiva que se basa en la teoría de evolución de las especies de Darwin, es decir, se basa en la premisa de que, entre un grupo de individuos en una población, solo los más adaptados sobrevivirán.

En el algoritmo genético, la población está compuesta por un conjunto de posibles soluciones, que se denominan individuos o cromosomas. A su vez estos individuos están formados por una secuencia de genes. En este trabajo los individuos son las representaciones de la solución y los genes los pacientes que la forman. Cada individuo tiene asociado un fitness, que representa el valor de la FO de esa solución. Se aplican una serie de operaciones para combinar los individuos y lograr una evolución en la población

En la ilustración 5 se describe el algoritmo genético clásico:

Input:
Population Size, n
Maximum number of iterations, MAX
Output:
Global best solution, Y_{bt}

begin
Generate initial population of n chromosomes $Y_i \ (i = 1, 2, ..., n)$
Set iteration counter $t = 0$
Compute the fitness value of each chromosomes
while $(t < MAX)$
Select a pair of chromosomes from initial population based on fitness
Apply crossover operation on selected pair with crossover probability
Apply mutation on the offspring with mutation probability
Replace old population with newly generated population
Increment the current iteration t by 1.
end while
return the best solution, Y_{bt}
end

Ilustración 5. Pseudocódigo Algoritmo Genético Clásico. (Katoch et al., 2021)

En este trabajo, se ha aplicado el algoritmo genético añadiendo una serie de cambios al algoritmo clásico descrito por Katoch et al. (2021). Se combina operadores genéticos (selección, cruce y mutación) con el operador insertion, el cual se utiliza para generar la población inicial. También se aplica el elitismo para preservar los individuos con mejor fitness.

El algoritmo implementado se explica a continuación:

1. Se crea una población inicial de tamaño *n* que contendrá la representación de la solución original y los individuos generados a partir de ésta aplicando el operador insertion. En cada ejecución del bucle, el operador insertion extrae a un paciente de su posición original y lo inserta en otra, permaneciendo el resto de los pacientes en el mismo orden. La generación de la población inicial se detiene cuando se alcanza el tamaño de la población *n*.

[2] Los resultados obtenidos se encuentran recogidos en el Anexo A

2. Mientras no se alcance el tiempo límite de computación:
    i. Bucle de tamaño la población (*n*).
        a. Selección. Se seleccionan dos padres por torneo. La estrategia de selección por torneo consiste en seleccionar varios individuos de la población al azar y, entre esos, elegir al que tiene mejor fitness. Por ello, primero se seleccionan de la población 3 individuos al azar. Esos 3 individuos son evaluados y se elige al que tiene mejor fitness, que será el primer padre. Luego se repite le proceso para seleccionar el segundo padre.
        b. Cruce: Si se da la probabilidad de cruce (*Pc*), se cruzan los dos padres creando un hijo. Si no el hijo será uno de los padres. Para realizar el cruce se elige un punto de cruce aleatorio. Este punto de cruce determina dónde se dividen la representación de la solución que se identifica como el primer padre. Para combinar los pacientes y generar el hijo, del primer padre toma los primeros pacientes hasta el punto de cruce. Del segundo padre toma, siguiendo el orden de esa representación de la solución, los pacientes que no están presente en los pacientes seleccionados del primer padre.
        c. Mutación Swap: Si se da la probabilidad de mutación (*Pm*) se intercambian de forma aleatoria dos pacientes del hijo generado.
        d. Elitismo: Se elige como descendiente el individuo de mejor fitness entre los padres seleccionados y el hijo generado. De esta manera aseguramos que el mejor elemento se mantenga en la nueva población.
        e. Se almacena el descendiente elegido para formar parte de una nueva población.
    ii. Cuando se consiguen tantos descendientes como individuos había en la población inicial de tamaño *n*, descendientes pasan a ser la nueva población.
    iii. Se toma como mejor solución el individuo que tenga mejor fitness de la nueva población. (*Ybt*)
    iv. Se comprueba el tiempo límite de computación.
    v. Si no se cumple criterio de parada se repite el algoritmo con la nueva población

En el algoritmo genético propuesto, tanto el cruce como la mutación solo se darán en el caso que se cumpla unas probabilidades independientes de cada uno. Por otra parte, en el elitismo propuesto se seleccionará como descendiente el que tenga mejor fitness comparando el hijo y los padres.

En un principio el algoritmo estaba implementado de manera que el mejor descendiente de la nueva población solo se tomara como mejor solución (Paso iii) si mejoraba el valor de la mejor solución hasta el momento. Sin embargo, esto restaba tiempo de computación para buscar nuevas soluciones y el algoritmo empeoraba.

Para la calibración de los parámetros se toman los valores propuestos por Gómez Medina, (2021) tras el análisis que realiza:

- Tamaño de la población: $n = N/2$
- Probabilidad de cruce: $Pc = 0{,}9$
- Probabilidad de mutación: $Pm = 0{,}1$

Siendo *N* el número de pacientes de la representación de la solución, es decir, la población estará formada por un número de individuos igual a la mitad de los pacientes que se encuentran en el SUH. En el caso que *N* sea impar, *n* es redondeado al entero superior.

5 ANÁLISIS COMPUTACIONAL

El código desarrollado para abordar el problema descrito y resolverlo está diseñado con el lenguaje de programación de Python, concretamente utilizando la herramienta de PyCharm.

La selección de Python como lenguaje de programación se ha debido, sobre todo, a su síntesis clara que hace sencilla su implementación. Además, cuenta con una extensa biblioteca estándar que proporciona una amplia gama de módulos y funciones que simplifican el código y aceleran el desarrollo. Aunque es cierto que no es el lenguaje con mayor rendimiento, para el problema que tratamos es suficiente.

La estructura del código se divide en 4 bloques: la experimentación, los algoritmos, el *decoding* y el diagrama de Gantt.

El código no se explica de manera detallada, pero si las librerías que se utilizan y las funciones que cumplen cada de los bloques.

5.1 Experimentación (experimentacion.py)

Es donde se extraen los datos globales y se generan los datos de entrada de los algoritmos. Estos son:

- La representación de la solución: una lista de los pacientes que se van a examinar con sus prioridades asociadas.
- Una matriz que contiene todas las actividades, los recursos que requieren y sus duraciones para cada paciente.
- Los tiempos de espera de los pacientes.
- Los recursos ocupados en el instante del análisis.
- Los recursos totales.

Este bloque se denomina experimentación porque se evalúan varios escenarios diferentes con el fin de analizar el problema con distintos datos de entrada, para ver cómo se comporta.

Para este bloque hemos requerido las siguientes librerías:

- **Xlrd:** se utiliza para leer archivos de Excel (*Leer Archivos Excel Con Python - Análisis y Decisión*, n.d.)Haga clic o pulse aquí para escribir texto.. En este caso sirve para extraer los datos iniciales que estaban almacenados en un Excel.
- **Numpy:** librería especializada en el cálculo numérico y el análisis de datos, especialmente para un gran volumen de datos (*NumPy : La Biblioteca de Python Más Utilizada En Data Science*, n.d.). Se ha utilizado para almacenar los datos extraídos del Excel.
- **Random:** implementa generadores de números pseudoaleatorios para varias distribuciones (*La Biblioteca Estándar de Python — Documentación de Python - 3.12.0*, n.d.). Se ha utilizado para generar probabilidades aleatorias.
- **Math:** Este módulo proporciona acceso a las funciones matemáticas definidas en el estándar de C(*La Biblioteca Estándar de Python — Documentación de Python - 3.12.0*, n.d.). Se ha ulizado para redondear al entero superior

Este bloque se explicará detalladamente en el capítulo 6.

5.2 Algoritmos (algoritmos.py)

En este bloque se encuentran los tres algoritmos metaheurísticos que se aplican para encontrar las mejores soluciones aproximadas y que han sido explicados en el capítulo 4. Se han programado los siguientes:

- Algoritmo genético
- Búsqueda tabú
- Recocido Simulado

Cada algoritmo se aplica de manera individual para poder comparar los valores de las soluciones que hallan cada uno. Dichos algoritmos se ejecutan hasta alcanzar el criterio de parada, que es temporal e igual para todos, y será definido más adelante.

Las librerías necesarias en este bloque son las siguientes:

- **Numpy:** en este caso se ha utilizado para crear la lista tabú del algoritmo de la búsqueda tabú.
- **Random:** se ha utilizado sobre todo en el algoritmo genético.
- **Copy:** crean enlaces entre un objetivo y un objeto. Para colecciones que son mutables o que contienen elementos mutables, a veces se necesita una copia para que uno pueda cambiar una copia sin cambiar la otra (*La Biblioteca Estándar de Python — Documentación de Python - 3.12.0*, n.d.). Se utiliza en el recocido simulado para copiar la solución inicial.
- **Math:** se utiliza en el recocido simulado para hallar la probabilidad de que se acepte la solución candidata como actual, aunque sea peor.
- **Time:** Este módulo proporciona varias funciones relacionadas con el tiempo (*La Biblioteca Estándar de Python — Documentación de Python - 3.12.0*, n.d.). Se utiliza para establecer el criterio de parada de los algoritmos, que depende del tiempo transcurrido

5.3 *Decoding* (evaluación.py)

Este bloque también ha sido explicado en detalle en el capítulo 4. El objetivo del *decoding* es determinar el instante de inicio y finalización de las actividades del PU de los pacientes que son asignadas a los recursos para poder evaluar la representación de la solución proporcionada por los algoritmos y hallar la secuenciación de las tareas en cada recurso que definirán la FO.

El *decoding* devuelve el valor de la FO para dicha representación de la solución, que es lo que necesitamos para seleccionar la mejor solución.

No se han utilizado librerías en este bloque.

5.4 Diagrama de Gantt (Gantt.py)

Este bloque permite visualizar el diagrama de Gantt que muestra como quedaría la secuenciación de las actividades una vez seleccionada la mejor solución.

Para ello vuelve a hacer el mismo *decoding* anterior, con la única diferencia de que en este caso devuelve las matrices por cada tipo de recurso que contienen las actividades que se han asignado a cada uno, con su instante de inicio, finalización y el paciente a la que se le realiza.

En este bloque solo se ha requerido una librería: **matplotlib.pyplot**: Proporciona una forma implícita, similar a MATLAB, de hacer gráficos (*Matplotlib.Pyplot — Matplotlib 3.5.3 Documentation*, n.d.). Se ha utilizado para construir gráficamente el diagrama de Gantt

6 EXPERIMENTACIÓN

Como se ha explicado anteriormente, los algoritmos diseñados requieren de unos datos de entradas que deben estar almacenados con un formato específico. En esta sección se muestra como a partir de los datos que proporciona una hoja de Excel se formulan todos los datos necesarios. Posteriormente se evalúan diferentes escenarios y se analizan los resultados obtenidos para comprobar cómo se comportan en los distintos escenarios en los que podrían implementarse. Se han utilizado los datos proporcionados por Bedoya-Valencia & Kirac 82016).

Se presentan 6 escenarios diferentes en los que variarán los siguientes factores:

- El nivel de saturación máxima del SUH que se desea. Se proponen 3 circunstancias: 75%, 100% y 150 % de saturación.
- Los recursos disponibles en función del instante que se analiza. Se estudiarán dos horarios estáticos; las 12 a.m y las 6 a.m

6.1 Implementación de la experimentación

Se trata de la resolución de un problema estático y que cada vez que se ejecuta nos encontramos en un instante t=0 en el que los algoritmos metaheurísticos conocen el número de pacientes, sus prioridades, sus PU, la duración de las actividades y los recursos disponibles.

Los datos de los recursos disponibles, los PU en función de la prioridad y la duración de las actividades que se desarrollan en dichos PU y los recursos requeridos para ello están contenidos en un archivo de Excel. Así pues, el primer paso de la experimentación consiste en leer ese archivo y almacenar todos los datos en el código.

Una vez obtenidos los datos, se irán generando pacientes de manera iterativa hasta alcanzar la saturación máxima establecida siguiendo la siguiente metodología:

1. Se genera un paciente y su prioridad.
2. Se define su PU y la duración de las actividades que lo componen, que dependerá de la prioridad y de si ya se encuentra en mitad de su proceso o aún no lo ha iniciado.
3. Se evalúa la saturación de todos los recursos que participan en el SUH, que depende del número de recursos disponibles.

A continuación, se detalla cómo se realiza cada uno de esos pasos y se tratan los datos extraídos para obtener los datos de entrada necesarios para ejecutar los algoritmos metaheurísticos.

6.1.1 Número de pacientes

El número de pacientes que se evaluarán en estos algoritmos dependerá del nivel de saturación máxima del SUH que se estudia. La saturación del SUH es el nivel se utilización de los recursos que hay disponibles en ese momento. Así pues, se plantean 3 niveles diferentes de saturación máxima: del 75%, del 100% y del 150%. Con esto se podrá evaluar el rendimiento de los algoritmos de una forma más completa.

Estos niveles de saturación se miden en el intervalo de tiempo representado por el $LOS_{medio.}$. Es decir que en el intervalo que transcurre desde el instante en el que se lanza el algoritmo hasta que alcanza el LOS_{medio} no puede sobrepasarse el nivel de saturación para ningún tipo de recursos en el SUH. Este valor se extrae de los resultados de la simulación del artículo de Bedoya-Valencia & Kirac (2016). Esta tabla, entre otros datos, muestra el LOS_{medio} en función del nivel ESI.

Table 3. Results of the validation run of the developed model.

	ESI level				
	1	2	3	4	5
Simulation average LOS (hrs.)	2.53	3.15	2.81	1.54	1.30
Real average LOS (hrs.)	3.15[a]	3.14	2.86	1.49	1.31
Real maximum LOS (hrs.)	3.15[a]	13.95	26.87	9.40	4.00
Simulation average time to be seen by a P/PA (hrs.)	0.05	1.10	1.21	1.00	1.23
Real average time to be seen by a P/PA (hrs.)	0.03[a]	1.14	1.27	1.03	1.25

[a]Corresponding to one patient during the period of time analyzed.
ESI: Emergency Severity Index; LOS: length-of-stay; P: physician; PA: physician assistant.

Ilustración 6. Resultados de la simulación de (Bedoya-Valencia & Kirac, 2016)

Así pues, tomando los datos de la segunda fila se calcula el LOS_{medio} que es de 143.4 minutos.

La saturación se calcula para cada tipo de recurso (j) y es la suma los tiempos de las actividades del PU del paciente en las que interviene dicho recurso multiplicado por el número de recursos de ese tipo que necesita la actividad.

El valor máximo que no pueden alcanzar es directamente proporcional al LOS_{medio}, al nivel de saturación que se evalúa y al número de recursos disponibles de ese tipo (j) de la siguiente manera:

$$Saturación_{máxima} = Nivel_{saturación} * LOS_{medio} * Recursos\_disponibles_j$$

Si la saturación de algún tipo de recurso excede la máxima establecida, se paran las iteraciones, y se obtiene la representación de la solución inicial. En caso contrario, se repiten esos pasos de nuevo generando un nuevo paciente con su prioridad.

6.1.2 Generación de la prioridad

La prioridad o valor de urgencia es un valor estocástico y, por tanto, se establece según una probabilidad. En la ilustración 7 se muestra el porcentaje de pacientes tratados en el SUH para cada nivel de ESI.

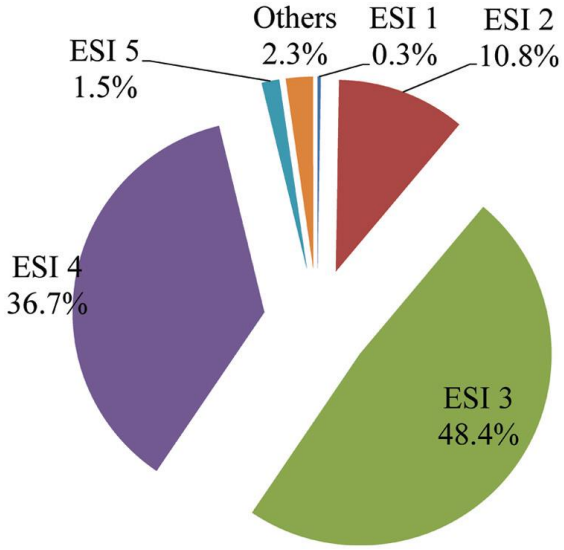

Ilustración 7. Porcentaje de pacientes de acuerdo con su ESI (Bedoya-Valencia & Kirac, 2016)

La prioridad de los pacientes se les asignará de manera aleatoria atendiendo a esos pesos.

6.1.3 Procesos de urgencia

Los PU son las actividades que realizan los recursos sobre los pacientes desde que ingresa hasta que se deja el SUH. Los PU están condicionados por dos factores, la prioridad y la situación en la que se encuentra el paciente en el instante de la foto. Al ejecutar el algoritmo los pacientes pueden encontrarse en situaciones diferentes de su PU.

Las actividades y recursos asociados que pueden componer el PU del paciente son las siguientes:

Actividades	Tipos de recursos
Visita médica	Médico (P)
Visita médico asistente	Médico asistente (PA)
Visita enfermero	Enfermero (N)
Prueba laboratorio sangre	Laboratorio (LAB)
Prueba Rayos-X	Máquina Rayos-X (RAY)
Prueba escáner CT	Escáner CT (SCAN)

Tabla 5. Relación de actividades y recursos

Según la prioridad, los recursos necesarios son los que establecen Bedoya-Valencia & Kirac (2016) en su artículo y que se muestran en la ilustración 8:

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.

Ilustración 8. Recursos requeridos por los pacientes según sus prioridades. Bedoya-Valencia & Kirac, 2016).

En la experimentación se excluye el servicio EMT (Equipos de Emergencia Médica), ya que son equipos que se encargan atender a pacientes que llegan críticos en ambulancia y que quizás ni entren en el SUH y vayan directamente a cuidados intensivos.

En cuanto a los pacientes de nivel ESI 1-2, destacar que necesitarán ser atendidos por dos enfermeros simultáneamente. Por último, hay que señalar que los pacientes con nivel ESI 4-5 no serán atendidos los médicos, sino por los médicos asistentes.

Así pues, las actividades y los recursos asociados que componen los PU en función de la prioridad del paciente son las siguientes:

Prioridad	Actividades	Tipos de recursos	Cantidad
1	1º visita médico	P	1
	1º visita enfermero	N	2
	Prueba laboratorio sangre	LAB	1
	Prueba Rayos-X	RAY	1
	Prueba escáner CT	SCAN	1
	2º consulta médico	P	1
	2º consulta enfermero	N	2
2	1º visita médico	P	1
	1º visita enfermero	N	2

	Prueba laboratorio sangre	LAB	1
	Prueba Rayos-X	RAY	1
	Prueba escáner CT	SCAN	1
	2º consulta médico	P	1
	2º consulta enfermero	N	2
3	1º visita médico	P	1
	1º visita enfermero	N	1
	Prueba laboratorio sangre	LAB	1
	Prueba Rayos-X	RAY	1
	Prueba escáner CT	SCAN	1
	2º consulta médico	P	1
	2º consulta enfermero	N	1
4	1º visita médico	PA	1
	1º visita enfermero	N	1
	Prueba Rayos-X	RAY	1
	2º consulta médico asistente	PA	1
	2º consulta enfermero	N	1
5	Visita médico asistente	PA	1
	Visita enfermero	N	1

Tabla 6. Actividades y recursos del PU en función del nivel ESI.

Cabe observar que el recurso del escáner CT será necesario y, por tanto, la actividad de CT-Scan se realizará o no atendiendo a la probabilidad que se muestra en la Ilustración 8 para pacientes de nivel ESI 1-3.

Una vez hallada la prioridad se conocen las actividades que tendría su PU de inicio a fin. Pero el paciente podrá encontrarse en 4 situaciones en el momento en el instante que se analiza:

1. El paciente acaba de llegar al SUH.
2. El paciente ha llegado al SUH antes de la foto, pero aún no ha sido atendido.
3. El paciente está en el PU siendo atendido en el SUH, pero el único recurso que está ocupando es el box.
4. El paciente está en el PU y está siendo atendido por un recurso en ese instante.

Si el paciente está en las situaciones 3 o 4 su TBSPPA será 0. En caso contrario será distinto de 0. Si se encuentra en la situación 2 no sabemos cuánto tiempo puede llevar esperando el paciente por lo que se generará su TBSPPA aleatoriamente de una lista de valores que contendrán un minuto menos de los límites según el artículo de (Jiménez Murillo et al., 2019).

Esos 4 estados serán generados aleatoriamente, al igual que el punto del PU el que se encuentra el paciente en las situaciones 3 y 4. Únicamente habrá una restricción para esos dos últimos casos, que establece que en ese instante no puede haber más recursos ocupados de los que hay disponibles. De esta forma si por ejemplo solo hay dos médicos y ya se han generado dos pacientes que están siendo atendidos por ellos en el instante inicial, si hay un tercero que coincide se le asignará otra actividad aleatoria que sí tenga recursos disponibles.

De esta manera obtendremos los PU específicos de cada paciente.

6.1.4 Duración de las actividades

La duración de las actividades también es estocástica y se establece de acuerdo con el artículo de Bedoya-Valencia & Kirac (2016) . El artículo modela estas duraciones con una distribución triangular y serán diferentes en función de la prioridad del paciente. Se muestra en la tabla de la ilustración 9:

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.

Ilustración 9. Tiempos de proceso de los pacientes en función del nivel ESI (Bedoya-Valencia & Kirac, 2016).

Para cada paciente se generarán unas duraciones distintas en cada una de sus actividades.

6.1.5 Recursos disponibles

En la publicación de Bedoya-Valencia & Kirac (2016) se define la disponibilidad de los recursos humanos que depende de la franja horaria que se analiza. Se puede observar en la ilustración 10:

Table 4. Schedule in the base case for nurses, physicians, and physician assistants.

Shift	Schedule	Type and number of resources		
		Nurses	PAs	Physicians
1	7 a.m.–7 p.m.	6	0	1
2	9 a.m.–5 p.m.	0	1	1
3	9 a.m.–9 p.m.	1	0	0
4	11 a.m.–11 p.m.	1	1	0
5	5 p.m.–1 a.m.	0	1	1
6	7 p.m.–7 a.m.	6	0	1
7	12 a.m.–12 p.m.	1	0	0
8	3 p.m.–3 a.m.	1	0	0

PAs: physician assistants.

Ilustración 10. Horario de enfermeros, médicos asistentes y médicos (Bedoya-Valencia & Kirac, 2016)

En cuanto al resto de recursos, considera que no se necesita personal para realizar las actividades de Escáner TC, Rayos-X y Laboratorio y estarán operativos las 24 horas del día. También define que habrá 27 boxes disponibles todo el día.

Para este trabajo se van a estudiar dos instantes estáticos: las 6 a.m. y las 12 a.m. Puesto que a las 6 a.m. no hay médicos asistentes y tendremos pacientes que deberán ser atendidos por ellos se considera que contamos con uno en este horario.

De esta manera los recursos disponibles en cada uno de esos instantes son:

Recursos disponibles	***Boxes (B)***	***Médicos (P)***	***Enfermeros (N)***	***Médicos asistentes (PA)***	***Laboratorios (LAB)***	***Máquina Rayos-X (RAY)***	***Escáner CT (CT-Scan)***
6 a.m.	27	1	6	1	1	1	1
12 a.m.	27	2	9	2	1	1	1

Tabla 7. Recursos disponibles por tipología y horarios

6.2 Resultados

Una vez se ha explicado cómo se obtienen los datos y se realiza la fase de experimentación, se procede a exponer los resultados obtenidos en cada uno de los 6 escenarios diferentes que se evalúan. En cada escenario se considera distintas hipótesis de niveles de saturación y recursos disponibles:

- Número de recursos disponibles de las 6 a.m. y nivel de saturación del 75%.
- Número de recursos disponibles de las 6 a.m. y nivel de saturación del 100%.
- Número de recursos disponibles de las 6 a.m. y nivel de saturación del 150%.
- Número de recursos disponibles de las 12 a.m. y nivel de saturación del 75%.
- Número de recursos disponibles de las 12 a.m. y nivel de saturación del 100%.
- Número de recursos disponibles de las 12 a.m. y nivel de saturación del 150%.

Para cada uno de estos escenarios se generan 10 instancias. Cada instancia contará con una generación de datos aleatoria diferente.

Para poder comparar los resultados de los algoritmos metaheurísticos es necesario que el criterio de detención de la búsqueda de mejor solución sea el mismo para los tres. Además de este criterio, en este apartado se explica el que se ha utilizado para determinar el algoritmo con mejor rendimiento.

6.2.1 Criterios para la comparación de resultados.

Como se ha explicado anteriormente, los algoritmos metaheurísticos tienen un criterio de parada temporal. Esto quiere decir que el algoritmo realizará iteraciones para buscar nuevas soluciones mientras no se alcance ese tiempo. Para establecer el valor del parámetro se sigue la siguiente fórmula:

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

En el que *n* es el total de actividades a asignar y *m* el número de recursos disponibles.

Al emplear un criterio de finalización uniforme para todos los algoritmos, se posibilita la comparación para determinar cuál ofrece el mejor rendimiento en ese intervalo temporal. Es importante tener presente que nos enfrentamos a un problema operativo en el cual la velocidad de respuesta se considera una prioridad.

Dicha comparación se halla mediante un RPD (Relative Percentage Deviation, Ruiz & Stützle, 2007):

$$RPD = \frac{Solution - Best_{sol}}{Best_{sol}} * 100$$

Donde *Solution* es, para una instancia dada, la solución obtenida por ese algoritmo y $Best_{sol}$ es la mejor solución obtenida de entre los tres algoritmos implementados. Con esto se calcula el ARPD. El ARPD se calcula haciendo el promedio del RPD obtenido por cada algoritmo e indica en tanto por ciento la desviación del resultado obtenido por el correspondiente algoritmo respecto a la mejor solución encontrada. El algoritmo que presente un menor valor ARPD es el que mejores resultados encuentra al problema planteado.

Los valores hallados de las mejores soluciones y los RPD de los algoritmos se muestran en el Anexo B.

6.2.2 Número de recursos disponibles de las 6 a.m.

Recordamos que los recursos disponibles en este instante son los siguientes:

Recursos disponibles	*Boxes (B)*	*Médicos (P)*	*Enfermeros (N)*	*Médicos asistentes (PA)*	*Laboratorios (LAB)*	*Máquina Rayos-X (RAY)*	*Escáner CT (CT-Scan)*
6 a.m.	27	1	6	1	1	1	1

Tabla 8. Recursos disponibles a las 6 a.m.

Es decir, contamos con 38 recursos disponibles para resolver el problema.

A continuación, se exponen los distintos niveles de saturación que se van a estudiar.

6.2.2.1 Caso 1: Nivel de saturación del 75%

En este caso la dimensión del problema es la siguiente:

Número pacientes medio	7,5
Número actividades medio	31,9
Tiempo límite de computación de los algoritmos (seg)	36,37

Tabla 9. Número de pacientes, actividades y recursos medio. Caso 1

Es decir, se atienden una media de 7,5 pacientes en el SUH. El promedio de las actividades totales de sus PU es de cerca de 32 actividades y el tiempo que tienen los algoritmos para encontrar la mejor solución es de 36 segundos.

En la Ilustración 11, la primera columna de todos los recursos (representada en azul en la Ilustración 11) determina la saturación máxima que no se debe sobrepasar en este escenario, y que es el criterio de parada de la generación de los pacientes. El resto de las columnas representan las 10 instancias que se realizan en cada escenario. El recurso box no se ha representado en esta gráfica para mayor claridad de los datos. Por lo tanto, esta gráfica permite observar en cada instancia qué tipo de recurso son los más saturados.

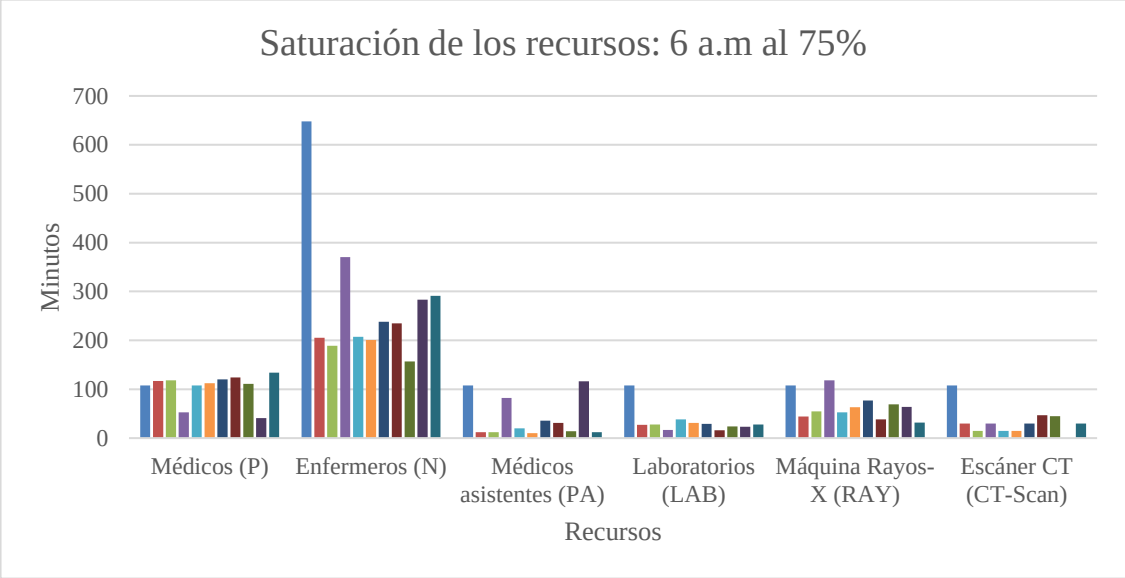

Ilustración 11. Saturación de los recursos para este escenario. Caso 1

En cuanto a los recursos, el médico es el primero en alcanzar el nivel de saturación máximo admitido, superándolo en casi todas las instancias. En una de las instancias, la novena, es el médico asistente quien lo supera. Esto tiene sentido porque ambos recursos son excluyentes, es decir, un paciente con médico en el PU no tiene médico asistente, y viceversa. Esa instancia generaría pocas prioridades 1-3 (que requieren médico) y muchas 4-5 (que requieren médico asistente). Además del médico, otro recurso bastante saturado es la máquina de Rayos-X

Esto se puede contrastar con la ilustración 12, donde se evidencia que el recurso más saturado de media es el médico, seguido por la máquina de Rayos X.

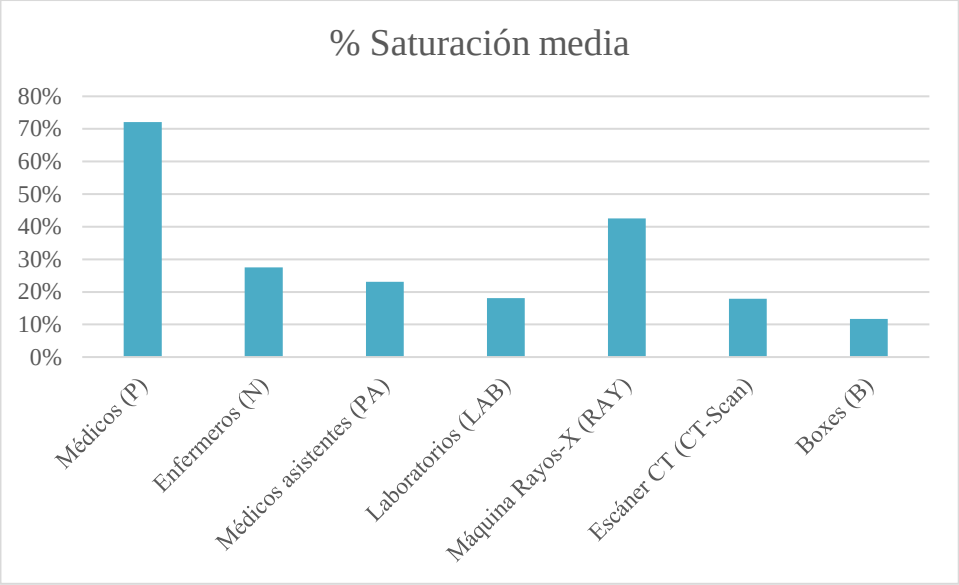

Ilustración 12. Saturación media de los recursos. Caso 1

En cuanto a los valores de las mejores soluciones halladas, en este escenario observamos poca variación entre los algoritmos de búsqueda.

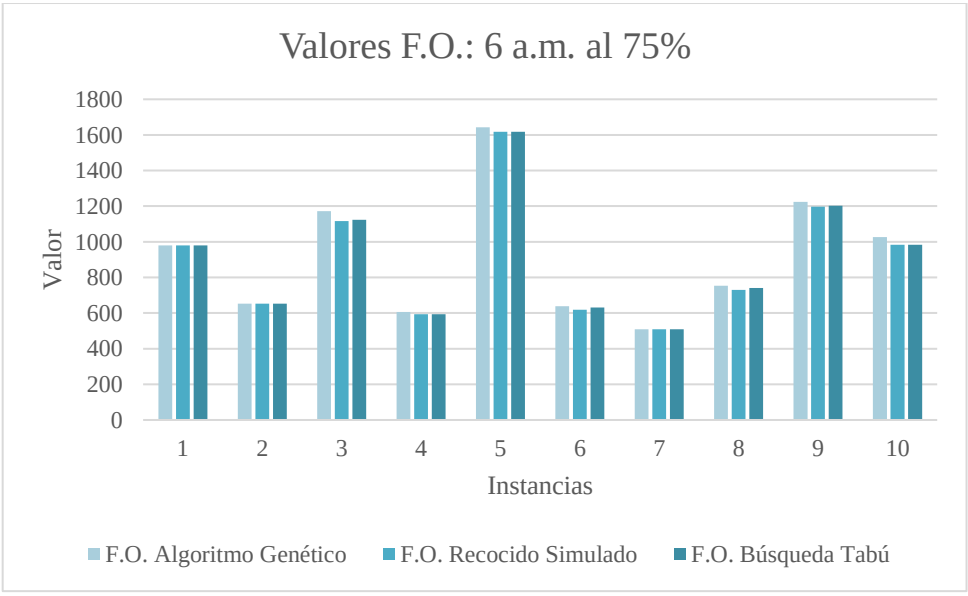

Ilustración 13. Valores FO obtenidos para las 10 instancias. Caso 1

Para poder comparar los valores con mayor claridad se ha calculado el ARPD, cuyos resultados se muestran en la ilustración 14. El recocido simulado es el que encuentra la mejor solución, seguido por la búsqueda tabú que presenta menos de un 1% de desviación. El algoritmo genético presenta mayor desviación, pero poco significativa.

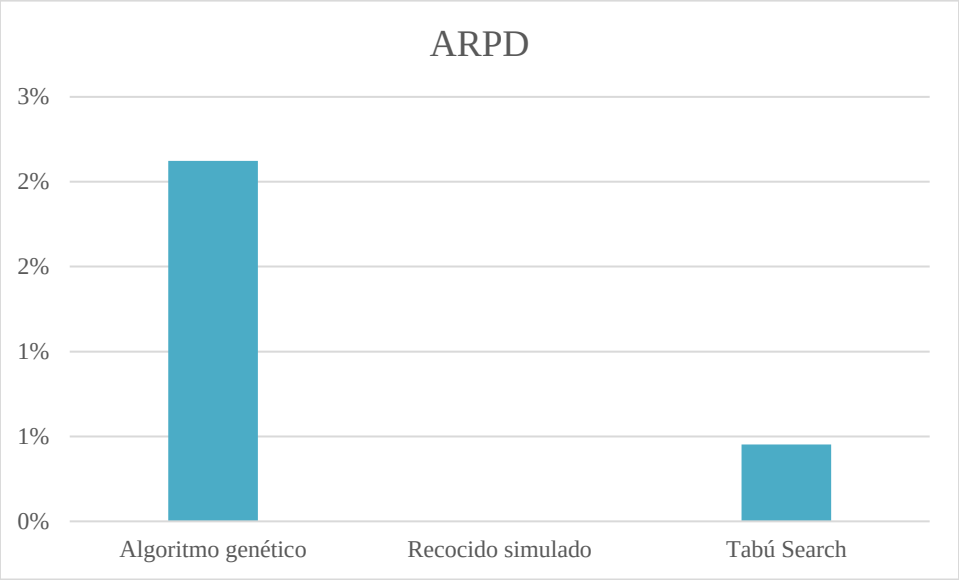

Ilustración 14. ARPD del caso 1

6.2.2.2 Caso 2: Nivel de saturación del 100%

En este escenario se atienden 4 pacientes más que en el caso anterior. Al aumentar el número medio de actividades también ha aumentado el tiempo límite de computación.

Número pacientes medio	11,7
Número actividades medio	48,5
Tiempo límite de ejecución de los algoritmos (seg)	55,29

Tabla 10. Número de pacientes, actividades y recursos medio. Caso 2

En este escenario es el médico el que supera el nivel de saturación máximo en todas las instancias, como se puede observar en la gráfica de la Ilustración 15, presentando una diferencia considerable respecto a la saturación de otros recursos. También se puede observar que el recurso enfermeros es el que tiene mayor utilización, pero al haber tantos recursos de ese tipo no supone una restricción.

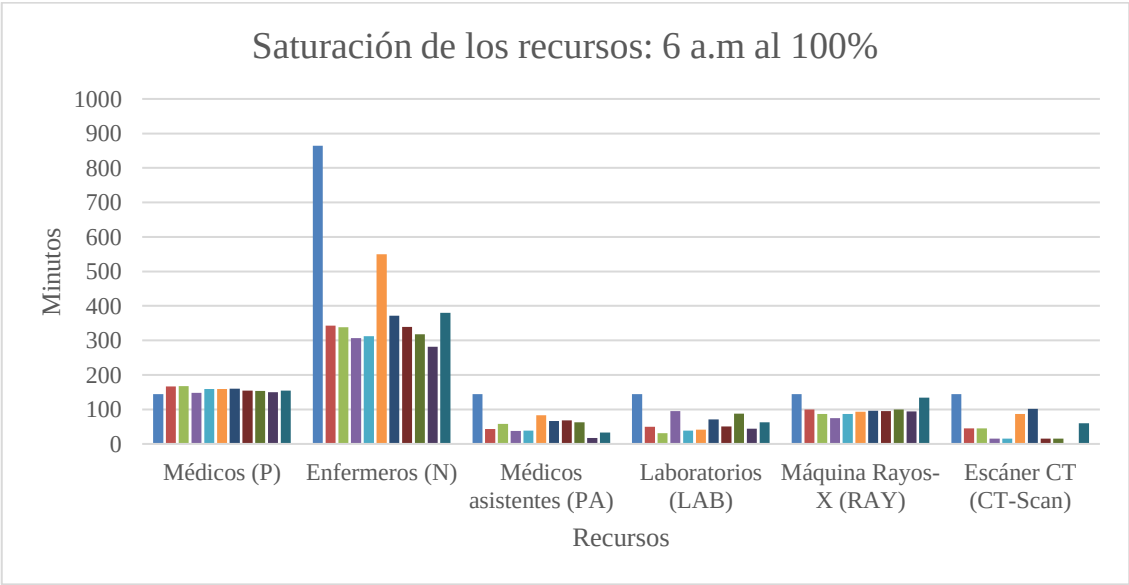

Ilustración 15. Saturación de los recursos para este escenario. Caso 2

En la Ilustración 16 se verifica que el médico es claramente el cuello de botella en este escenario, ya que alcanza la saturación máxima del 100% establecida.

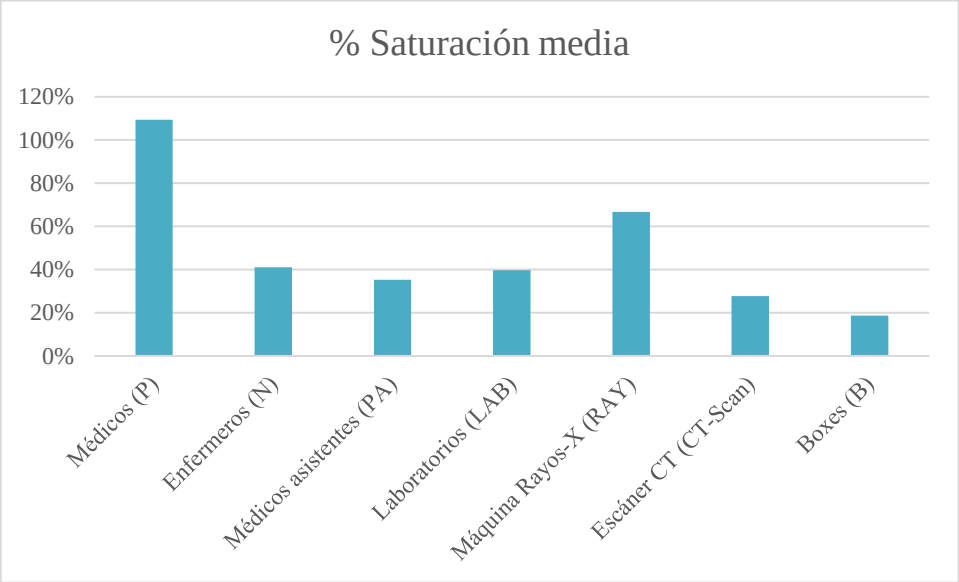

Ilustración 16. Saturación media de los recursos. Caso 2

En cuanto a los valores de la FO en las diferentes instancias, en la Ilustración 17 empezamos a ver una ligera diferencia entre los valores del algoritmo genético en comparación con la búsqueda tabú y el recocido simulado, siendo este último el que ofrece mejores soluciones una vez más.

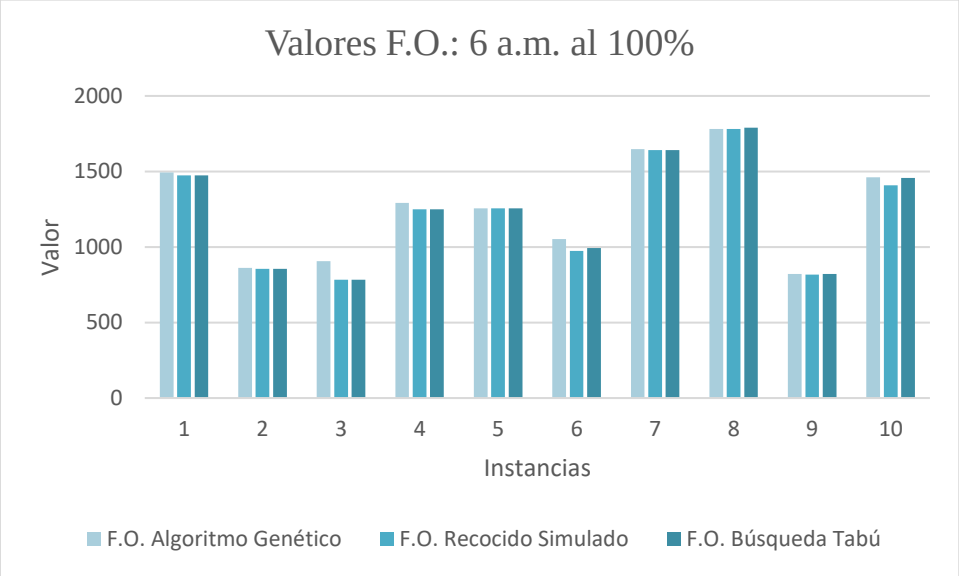

Ilustración 17. Valores FO obtenidos para las 10 instancias. Caso 2

En la Ilustración 18 se puede observar que el ARPD de la búsqueda tabú apenas han empeorado respecto al caso 1, sin embargo, el algoritmo genético si ha aumentado su desviación en un 1%.

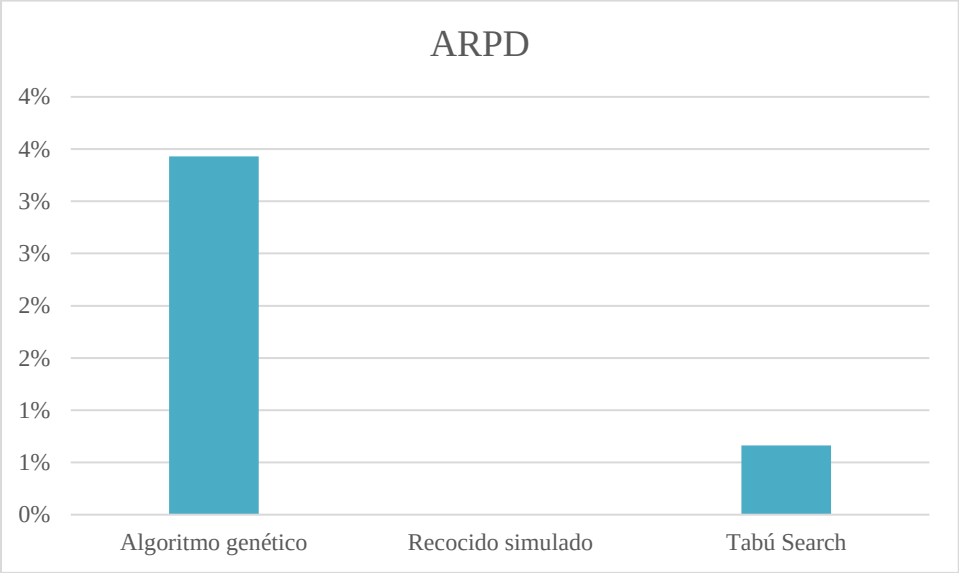

Ilustración 18. ARPD del caso 2

6.2.2.3 Caso 3: Nivel de saturación del 150%

Al tratarse de algoritmos aproximados podemos probar como se comportaría si la saturación del SUH superase el 100%. Las dimensiones del problema en ese caso serían:

Número pacientes medio	16,1
Número actividades medio	61
Tiempo límite de computación de los algoritmos (seg)	69,54

Tabla 11. Número de pacientes, actividades y recursos medio. Caso 3

En cuanto a la saturación, encontramos 2 cuellos de botella, el recurso médico y el recurso máquina de rayos, superándose el nivel de saturación máximo en ambos recursos en varias de las instancias.

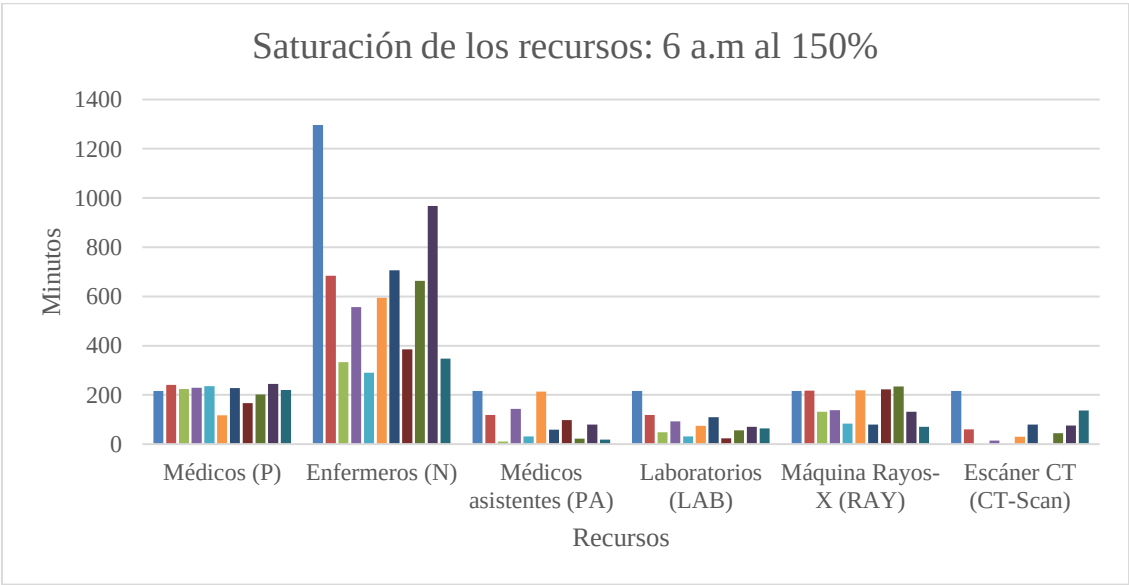

Ilustración 19. Saturación de los recursos para este escenario. Caso 3

Sin embargo, en la gráfica de la Ilustración 20 se muestra que el recurso más saturado en este escenario son los

médicos.

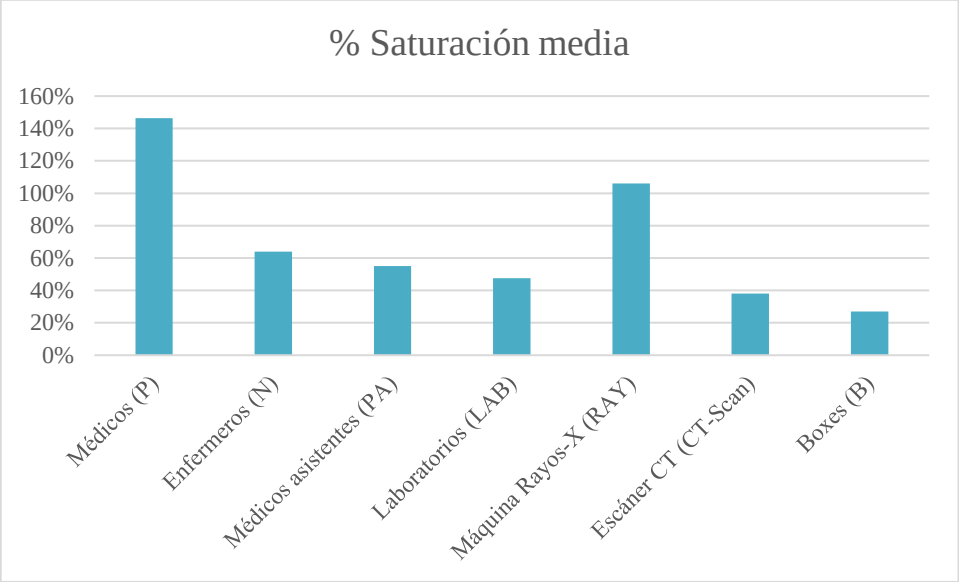

Ilustración 20. Saturación media de los recursos. Caso 3

Respecto a los valores de las funciones objetivos halladas, una vez más es el recocido simulado el que alcanza las mejores soluciones como se puede observar en la Ilustración 21.

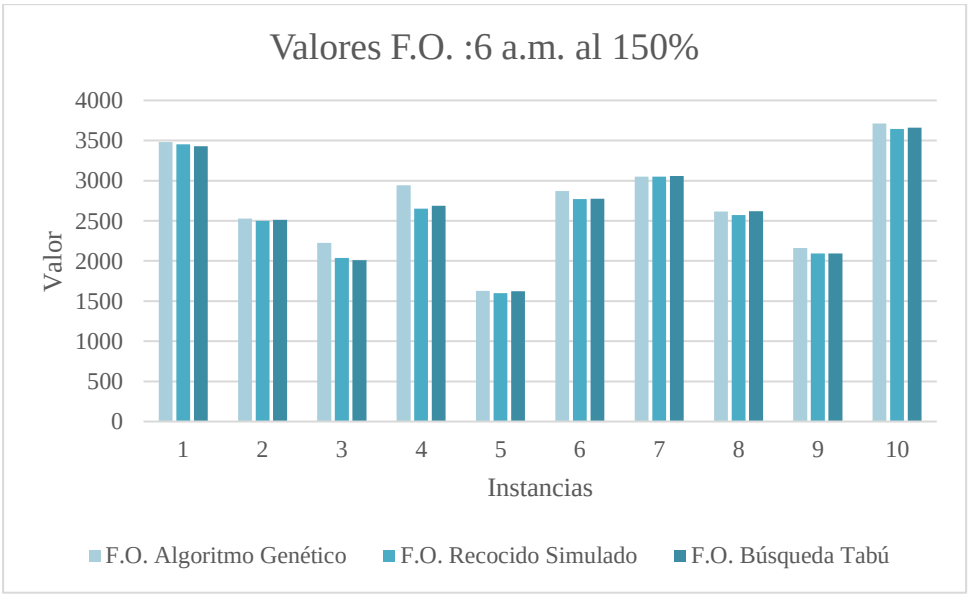

Ilustración 21. Valores FO obtenidos para las 10 instancias. Caso 3

Sin embargo, en la Ilustración 22 se observa que, aunque ha aumentado las dimensiones del problema, las desviaciones del algoritmo genético y de la búsqueda tabú apenas ha variado.

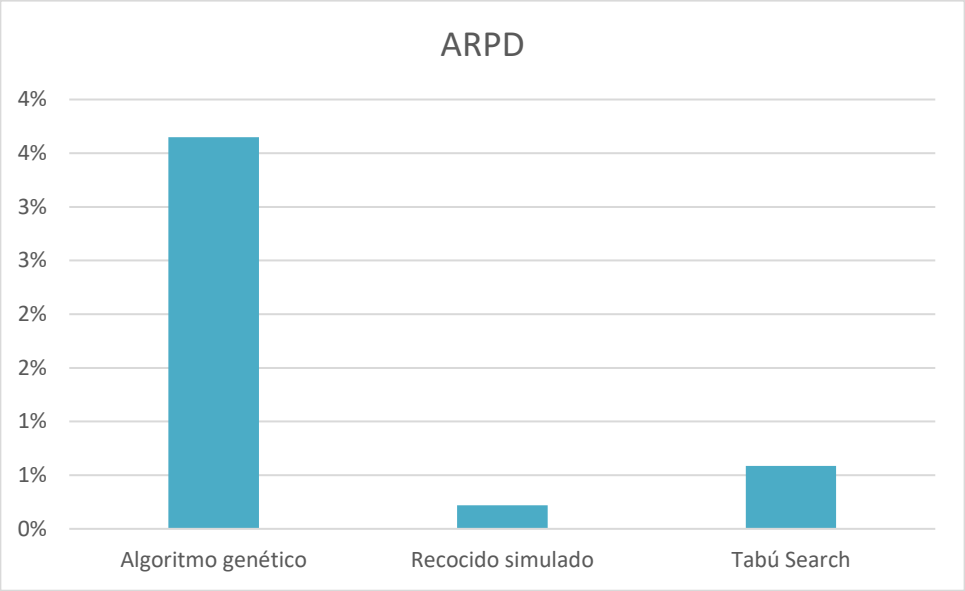

Ilustración 22. ARPD del caso 3

6.2.3 Número de recursos disponibles de las 12 a.m.

Los recursos disponibles en este instante son los siguientes:

Recursos disponibles	*Boxes (B)*	*Médicos (P)*	*Enfermeros (N)*	*Médicos asistentes (PA)*	*Laboratorios (LAB)*	*Máquina Rayos-X (RAY)*	*Escáner CT (CT-Scan)*
12 a.m.	27	2	9	2	1	1	1

Tabla 12. Recursos disponibles a las 12 a.m.

Los recursos de médicos, y médicos asistentes se han duplicado. Los enfermeros también han aumentado en dos unidades, sin embargo, las cantidades de los recursos no humanos no han sido modificadas. El número de recursos total disponibles es 43.

6.2.3.1 Caso 4: Nivel de saturación del 75%

Al haber aumentado el número de recursos, pese a ser la misma saturación, las dimensiones del problema han aumentado considerablemente respecto al caso 1, atendiéndose al doble de pacientes en el SUH. Sin embargo, hay poca diferencia respecto al caso 3 estudiado antes. La mayor diferencia es que ha aumentado el tiempo de computación, ya que es directamente proporcional al número de las actividades que se secuencian y a los recursos disponibles

Número pacientes medio	15,9
Número actividades medio	66,5
Tiempo límite de computación de los algoritmos (seg)	83,79

Tabla 13. Número de pacientes, actividades y recursos medio. Caso 4

Al igual que en el caso 3, en la gráfica de la Ilustración 23 se pueden observar dos cuellos de botellas. Los médicos y la máquina de Rayos-X.

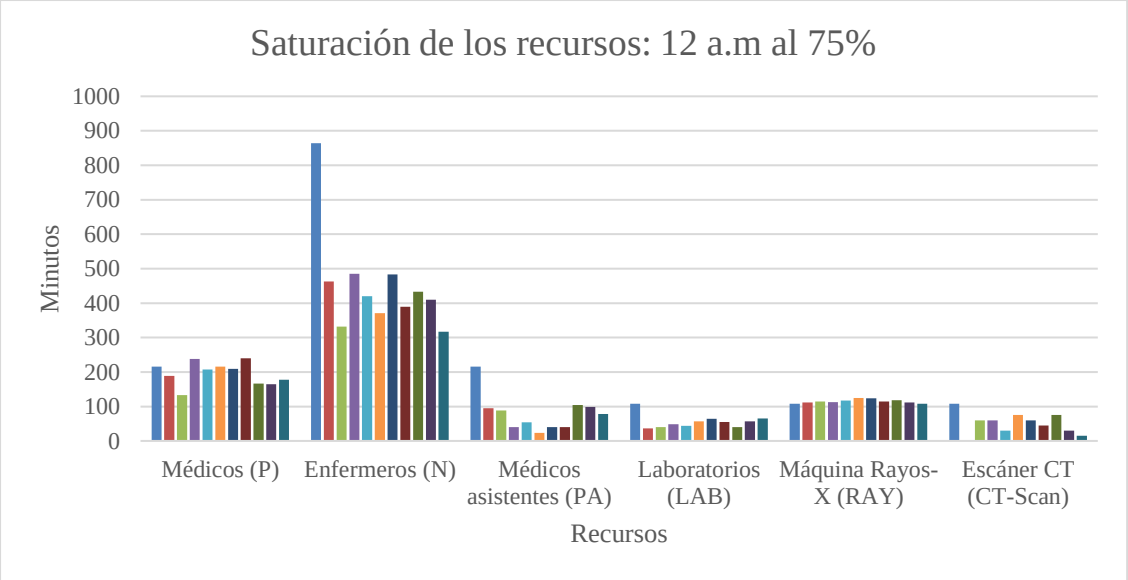

Ilustración 23. Saturación de los recursos para este escenario. Caso 4

Sin embargo, en la mayoría de los casos en los que se satura el médico, también se satura la máquina de Rayos-X, mientras que el caso contrario no ocurre. Esto se puede ver con claridad en la gráfica de la Ilustración 24, dónde los médicos no rozan el 70% se saturación y la máquina de Rayos sobrepasan el 75%.

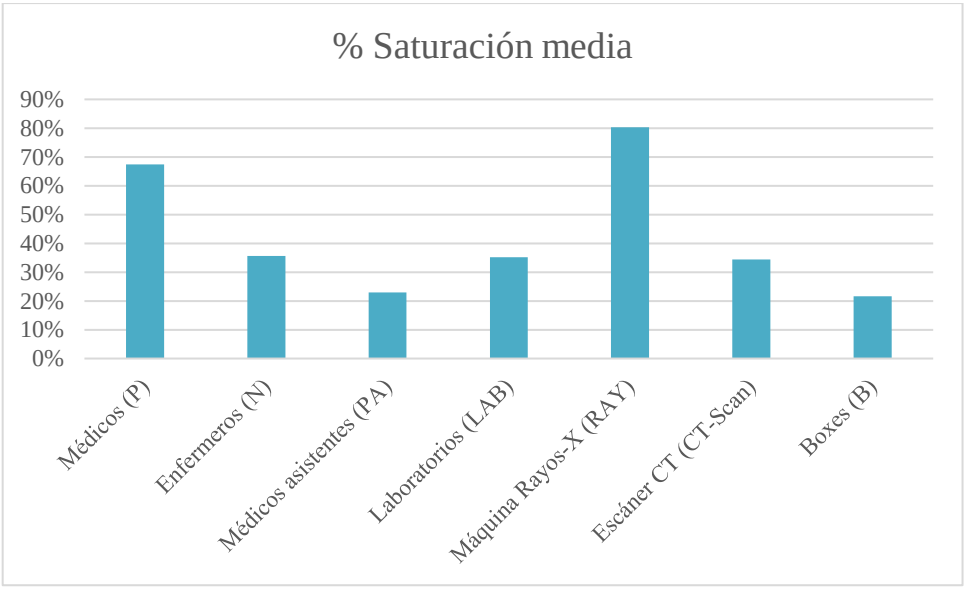

Ilustración 24. Saturación media de los recursos. Caso 4

En cuanto a los valores de la FO, el recocido simulado es el que vuelve a obtener mejores soluciones en casi todas las instancias, seguido de la búsqueda tabú.

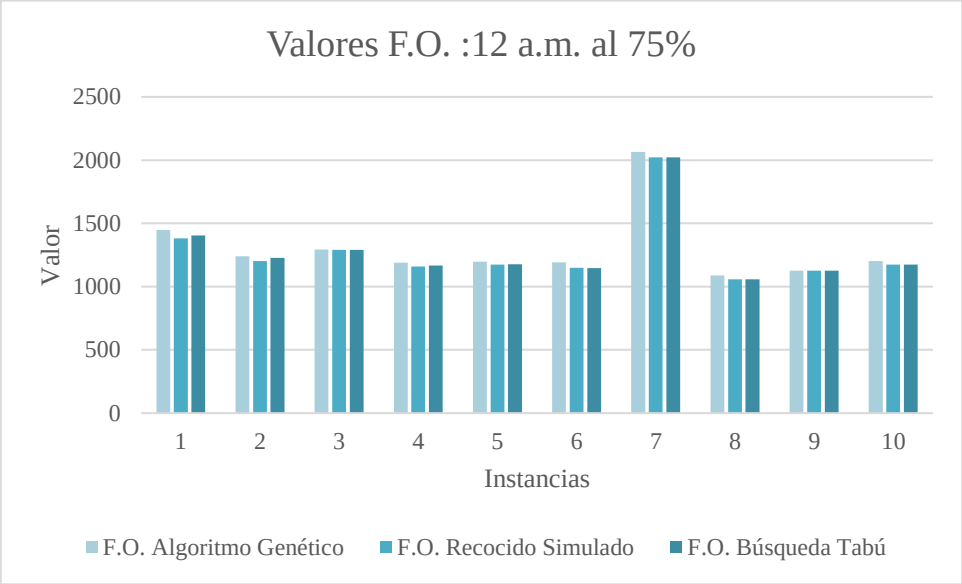

Ilustración 25. Valores FO obtenidos para las 10 instancias. Caso 4

Sin embargo, en la gráfica de la Ilustración 26, se ve como el algoritmo genético disminuye su desviación respecto a la mejor solución. Esto es destacable ya que como se ha dicho al principio prácticamente la única diferencia que hay en este escenario respecto al caso 3 es que ha aumentado el tiempo de computación, y esto ha permitido al algoritmo genético mejorar su rendimiento.

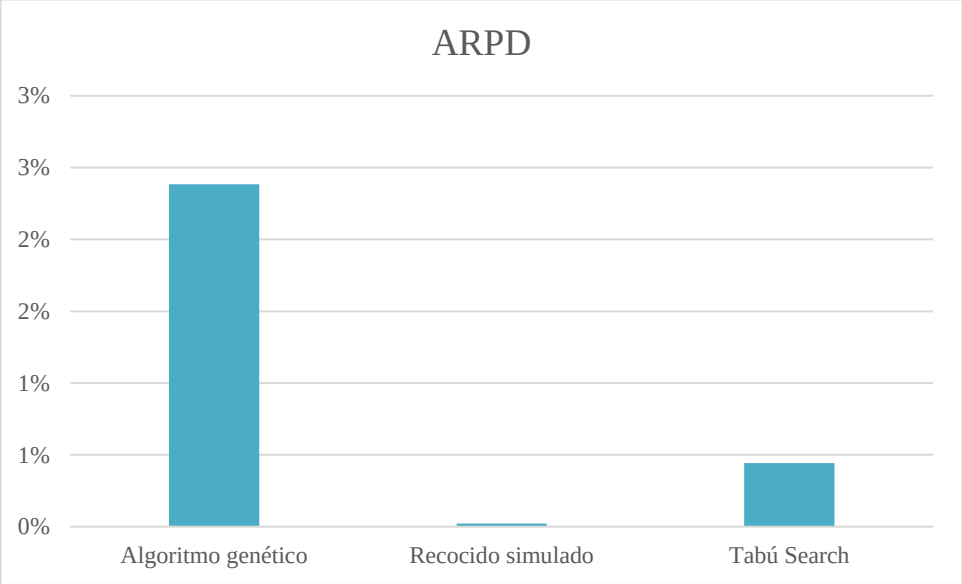

Ilustración 26. ARPD del caso 4

6.2.3.2 Caso 5: Nivel de saturación del 100%

A continuación, se muestran los datos que se han generado en este escenario.

Número pacientes medio	18,1
Número actividades medo	69,6
Tiempo límite de computación de los algoritmos (seg)	100,62

Tabla 14. Número de pacientes, actividades y recursos medio. Caso 5

En este escenario, vuelve a ser el recurso de máquinas de rayos el que supera el nivel de saturación en la mayoría de las instancias, seguidos del recurso médicos. El recurso enfermeros también tiene una alta utilización, pero al haber tantos recursos disponibles no es un cuello de botella y está poco saturado.

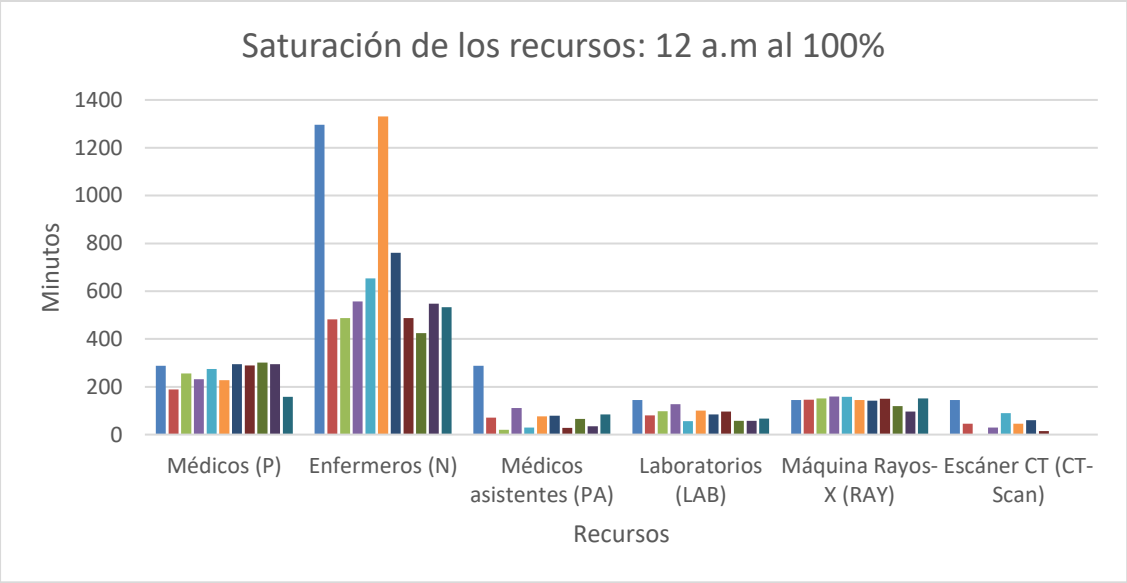

Ilustración 27. Saturación de los recursos para este escenario. Caso 5

Respecto a la situación anterior se empieza a ver con más claridad que el médico no es cuello de botella en los escenarios de las 12 de la mañana, ya que en ese horario se cuentan con dos recursos disponibles de ese tipo.

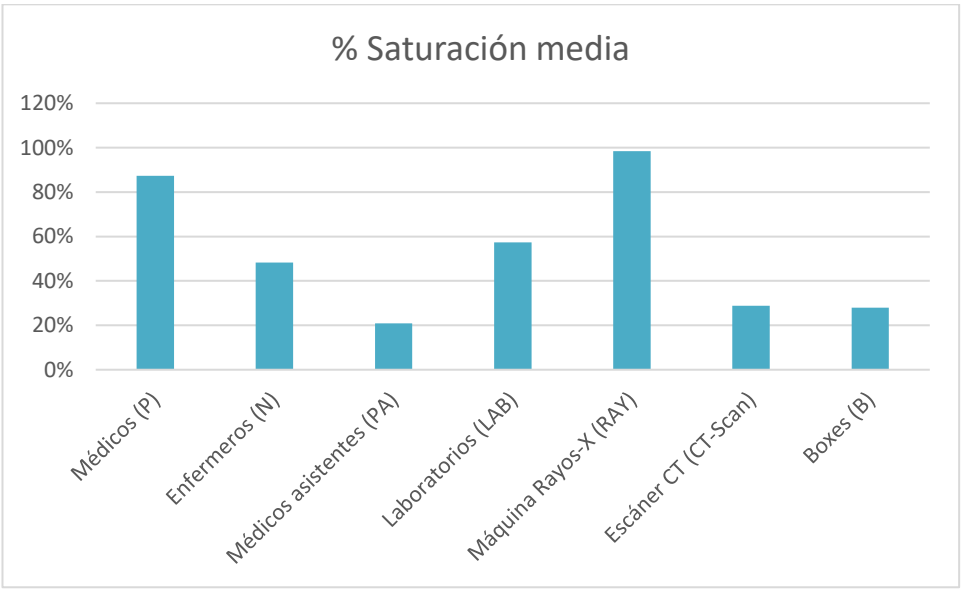

Ilustración 28. Saturación media de los recursos. Caso 5

Por otro lado, los valores de la FO obtenidos por los algoritmos siguen siendo parecidos en todas las instancias.

Volviendo a ser el recocido simulado el que obtiene mejor solución, seguido por la búsqueda tabú.

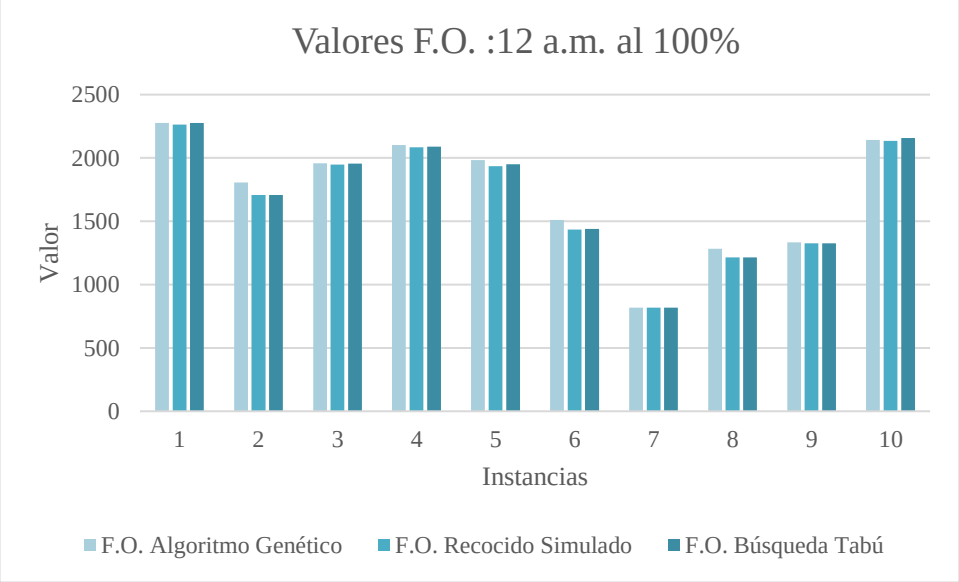

Ilustración 29. Valores FO obtenidos para las 10 instancias. Caso 5

Si se observa la Ilustración 30 se observa que las desviaciones del algoritmo genético y de la búsqueda tabú se mantiene respecto al caso 4, ya que ambos casos tienen dimensiones similares.

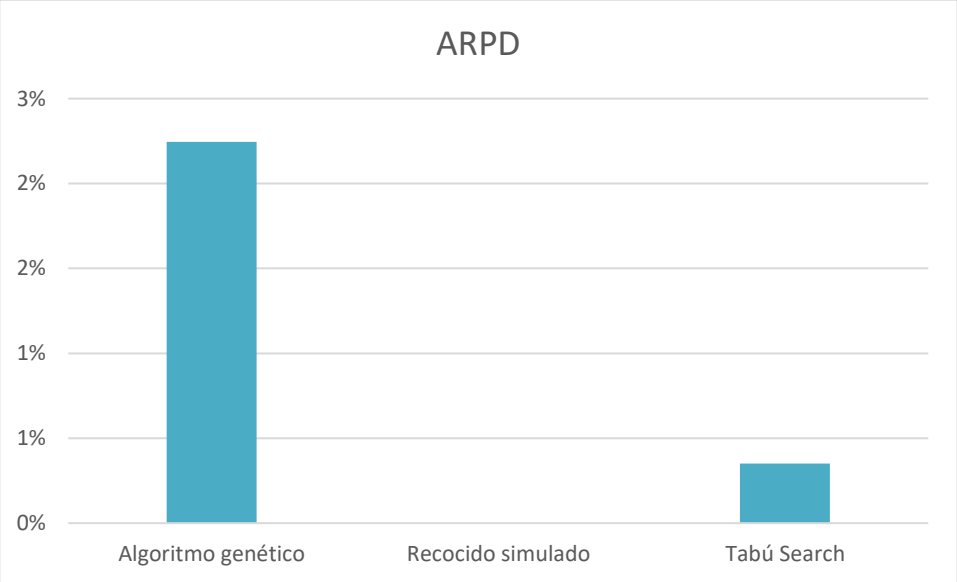

Ilustración 30. ARPD del caso 5

6.2.3.3 Caso 6: Nivel de saturación del 150%

En este escenario se calcula que la dimensión del problema es la siguiente:

Número pacientes medio	25,5
Número actividades medio	102,8
Tiempo límite de computación de los algoritmos (seg)	121,26

Tabla 15. Número de pacientes, actividades y recursos medio. Caso 6

En la ilustración 31 se observa cómo, de nuevo, la máquina de Rayos-X es el que alcanza antes el nivel de

saturación máximo, en casi todas las instancias.

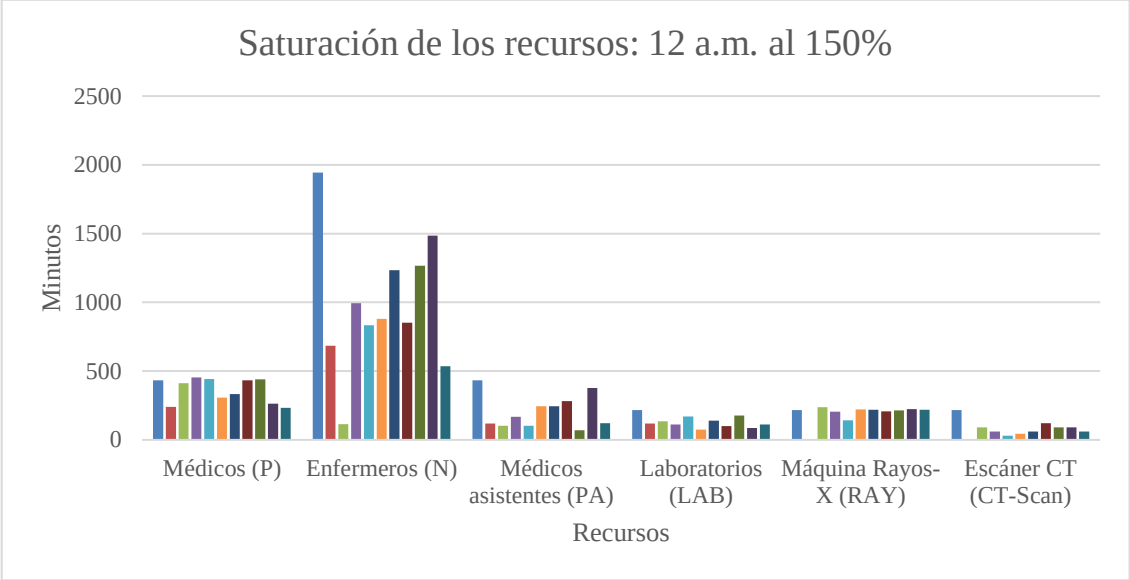

Ilustración 31. Saturación de los recursos para este escenario. Caso 6

En la Ilustración 32 se corrobora que el recurso de máquina de Rayos-X está altamente saturado, seguido por el recurso médico.

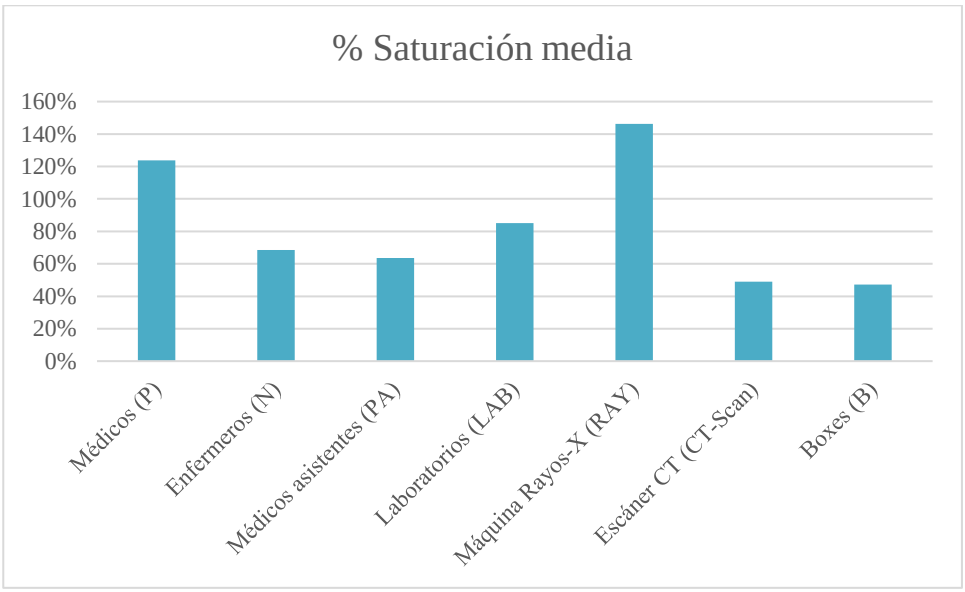

Ilustración 32. Saturación media de los recursos. Caso 6

Respecto a la gráfica de la Ilustración 33, se observa de nuevo poca diferencia entre las mejores soluciones encontradas por el recocido simulado y las encontradas por la búsqueda tabú.

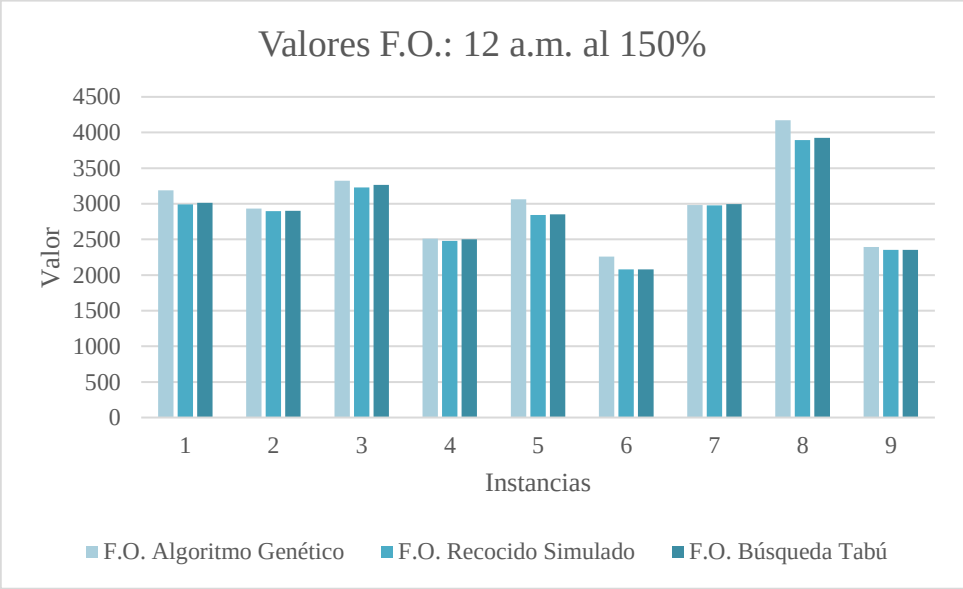

Ilustración 33. Valores FO obtenidos para las 10 instancias. Caso 6

Esa diferencia se puede corroborar en la gráfica de la Ilustración 34 del ARPD, donde se observa que el ARPD del Búsqueda tabú es muy cercano al 1%, mientras que el del algoritmo genético ha empeorado al aumentar las dimensiones del problema.

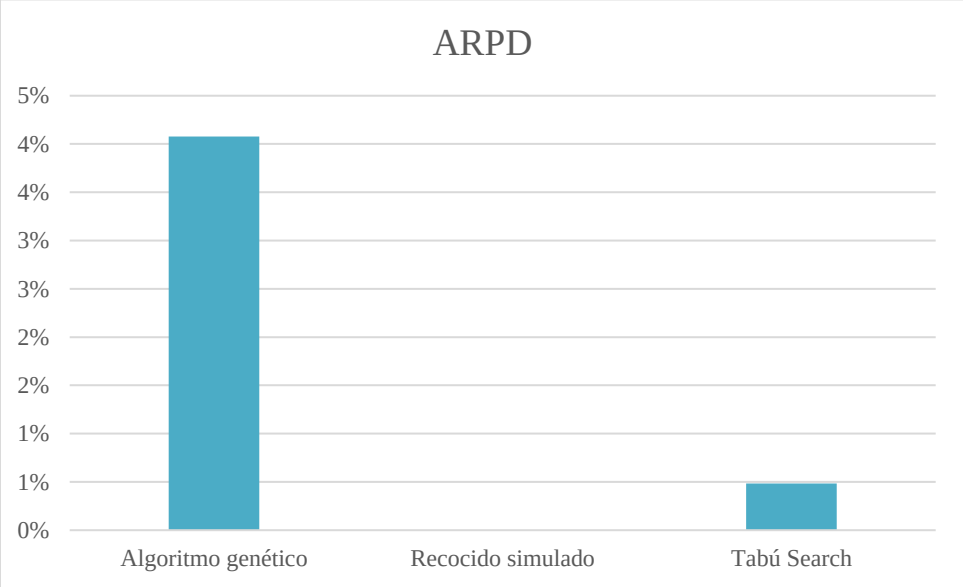

Ilustración 34. ARPD del caso 6

6.3 Análisis de resultados

Los resultados obtenidos en esta experimentación son solo un ejemplo del alcance y potencial que tienen los algoritmos que se proporcionan en este trabajo.

Al resolverse mediante algoritmos aproximados, se puede obtener una respuesta factible en pocos minutos. Haciendo el promedio de los pacientes cuyos PU se secuencian y los tiempos de computación en los 6 escenarios, se halla que se tratan una media de 16 pacientes en 78 segundos de computación, es decir, cada algoritmo metaheurístico propuesto tarda alrededor de 5 segundos por paciente que se atiende, en encontrar una buena

secuenciación de las tareas a asignar a los recursos. Aunque para conocer cuánto se acerca la solución que proporciona a la solución óptima de este problema sería necesario compararlo con una metodología de resolución exacta, que solo encontrarían soluciones exactas a problemas muy pequeños y alejados de la realidad, se sabe por otros estudios anteriores (Dorgham et al., 2019; Y.-Y. Feng et al., 2017; Fontes et al., 2023; Gómez Medina, 2021; Katoch et al., 2021; Moyano Personat, 2023; Ruiz & Stützle, 2007) que un algoritmo con buenas heurísticas puede ofrecer soluciones bastante cercanas a la óptima y en menos tiempo.

De los tres algoritmos que se comparan en este trabajo, claramente el de recocido simulado es el que está encontrando mejores soluciones, independientemente de las dimensiones del problema, como se puede observar en la Ilustración 35. La búsqueda tabú también mantiene un rendimiento similar en todos los escenarios y, aunque presenta mayor desviación que el recocido simulado, ésta no alcanza el 1%. Por último, el algoritmo genético por lo general empeora a medida que aumenta la saturación del SUH, escenarios en los que aumenta el tamaño del problema a resolver. Igualmente, presenta poca desviación respecto a los otros algoritmos propuestos.

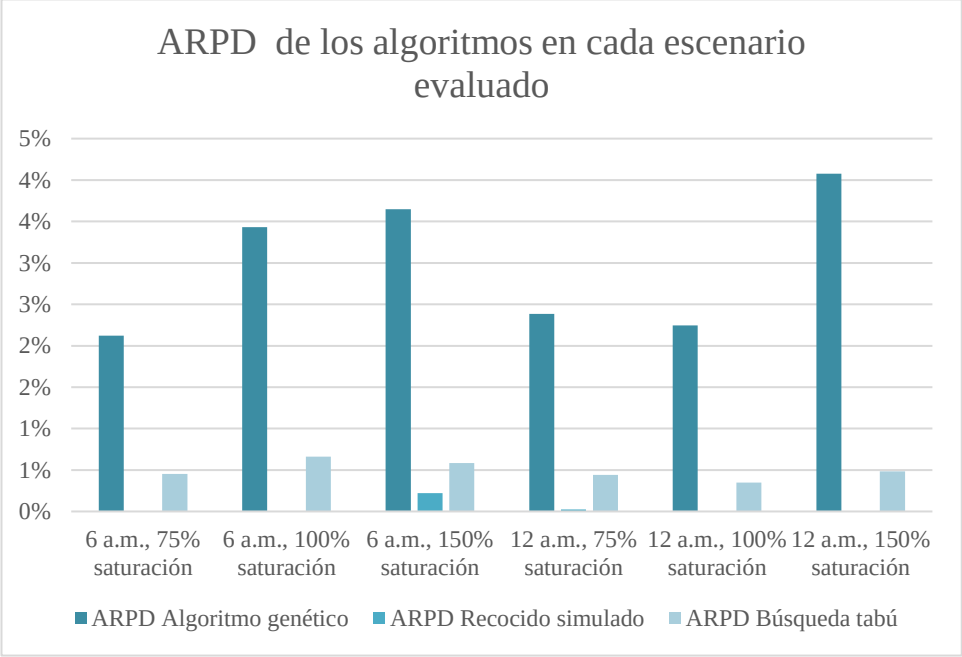

Ilustración 35. Comparación de los ARPD de los algoritmos en los 6 escenarios

El dominio del recocido simulado puede deberse a varios motivos:

- El Algoritmo Genético y el Búsqueda tabú necesitan mayor tiempo de computación para alcanzar la solución del Recocido Simulado.
- La calibración del Algoritmo Genético y del Búsqueda tabú es peor que la del Recocido Simulado.
- El recocido simulado da muy buena respuesta a problemas que tienen varios óptimos locales, sin embargo, el Búsqueda tabú y el Algoritmo genético una vez obtienen una solución subóptima es difícil que salgan de ese óptimo local, para ello es necesario tener una buena diversidad de soluciones. Esta capacidad de mejora del recocido simulado se puede observar en la Ilustración 36, que es una ampliación de la gráfica de evolución de las soluciones del recocido simulado. En ella se observa como toma un valor de solución que empeora el valor de la FO para poder salir del óptimo local, y encontrar un nuevo óptimo.

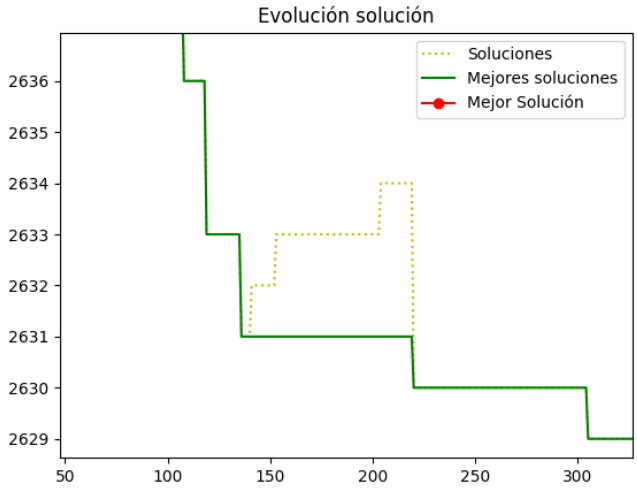

Ilustración 36. Evolución soluciones encontradas por recocido simulado

Por otro lado, a nivel táctico estos algoritmos también podrían ser útil para corregir los horarios de los recursos. Se puede observar que siempre va a haber un recurso, que será el cuello de botella respecto a la saturación del SUH, por lo que aumentar la disponibilidad de ese recurso no evitará que se alcance la saturación del SUH. Sin embargo, sí puede equilibrarse la saturación media del resto de recursos. Por ejemplo, en todos los casos estudiados la saturación del recurso 'Enfermeros' está muy por debajo del límite, por lo que se podría reducir el número de recursos disponibles en esos horarios.

Respecto al objetivo operacional, es importante volver a destacar que la velocidad de respuesta es una prioridad en el SUH, por lo que con los algoritmos aproximados encontramos una solución adecuada al problema que se trata, obteniendo una secuenciación de las tareas de los recursos en el SUH factible en pocos minutos

7 CONCLUSIONES

El objetivo de este trabajo era mejorar la secuenciación de tareas de los recursos en el SUH para reducir el LOS y el TBSPPA de los pacientes, aplicando algoritmos aproximados para tal fin. Las tareas de los recursos que se encargan de asignar y secuenciar dichos algoritmos son las actividades (visitas médicas, pruebas de diagnóstico, etc.) que componen los PU de los pacientes que acuden al SUH. Tanto los PU como los TBSPPA máximos varían en función de la prioridad, y la duración de las actividades será diferente para cada paciente, ya que siguen una distribución triangular.

El problema que se resuelve presenta una característica principal, que consiste en que los pacientes una vez que son asignados a un box, no se desplazan por el SUH, sino que son los recursos los que acuden al box para realizar las pruebas (actividades) del PU del paciente.

Puesto que se trata de un problema *NP-hard*, se propone alcanzar el mejor resultado posible mediante tres algoritmos metaheurísticos diferentes que son adaptados al problema: el algoritmo genético, el recocido simulado y la búsqueda tabú.

Para evaluar las soluciones que proponen los algoritmos metaheursíticos ha sido necesario realizar un proceso de *decoding*. Dicho proceso parte de una representación de la solución, compuesta por los pacientes que se encuentran en el SUH y sus prioridades asociadas, y devuelve un valor de la FO que es la suma del LOS y del TBSPPA, en el caso que el TBSPPA supere el máximo, de todos los pacientes que componen la solución. El valor de la FO será menor cuanto mejor sea la secuenciación de las tareas de los recursos que realiza el proceso de *decoding*. La secuenciación se realiza cumpliendo una serie de restricciones.

Inicialmente, se ha realizado una revisión detallada de la literatura científica para comprender los trabajos previos relacionados con este problema. Posteriormente, se ha presentado y descrito el problema concreto que se aborda. En el capítulo de metodología de resolución se describen los pasos seguidos para obtener la mejor solución de cada algoritmo a partir de unos datos de entrada. Para poder resolver el problema, se ha aplicado a un código de programación diseñado con Python del cual también se proporciona una explicación en el apartado de análisis computacional.

Por último, se llevado a cabo una experimentación en la que se explica primero cómo se obtienen los datos que alimentan los algoritmos y posteriormente se evalúan dichos algoritmos en 6 escenarios diferentes, dependientes de la saturación del SUH, y del horario del personal del hospital. Los niveles de saturación que se evalúan son 3: del 75%, del 100% y del 150%, y las franjas horarias dos: las 12 a.m. y las 6 a.m., en las cuales varía el número de recursos disponibles. Las combinaciones de esas casuísticas generan los 6 escenarios propuestos. En función del escenario variará el número de pacientes que se atienen en el SUH. Para cada uno de esos escenarios se han realizado 10 instancias, registrando los datos obtenidos para su comparación posterior.

La comparación del rendimiento entre los algoritmos se hace mediante el promedio del RPD (ARPD) de las 10 instancias que se evalúan en los 6 escenarios propuestos. También se comparan como varía la capacidad del código al modificar la saturación y los recursos disponibles.

Por último, se han comparado los resultados obtenidos para sacar unas conclusiones las cuales se exponen a continuación:

- Los algoritmos aproximados son una herramienta muy útil para resolver el problema de la secuenciación de tareas en el SUH ya que dan una respuesta rápida y factible al problema, de aproximadamente 5 segundos que se atiende.
- De los tres algoritmos diseñados, el Recocido Simulado es el que da mejor respuesta en todos los escenarios, seguido por el Búsqueda tabú (1% ARPD de media) y siendo el Algoritmo genético el que presenta una mayor desviación (4% ARPD de media). Sin embargo, el Recocido Simulado va perdiendo eficacia a medida que aumenta las dimensiones del problema.
- La secuenciación de tareas obtiene buenos resultados independientemente del escenario que se evalúa, respetando las restricciones y equilibrando las asignaciones de tareas a los recursos del mismo tipo.

- Con estos algoritmos, además de secuenciar las tareas y equilibrar la asignación de éstas, también se puede equilibrar la saturación de los recursos, comparando la saturación media de cada tipo de recursos con el nivel de saturación máxima que se evalúa en ese escenario.

Para futuras investigaciones, sería interesante considerar asignar tareas teniendo en cuenta qué profesional atiende al paciente en su segunda visita, en lugar de priorizar la distribución equitativa de tareas entre recursos del mismo tipo. Esto se debe a que en la realidad si un médico, por ejemplo, atiende a un paciente en una primera visita, es probable que sea el mismo médico quien lo atienda en visitas posteriores.

Por otro lado, puesto que la tecnología no para de avanzar, otro enfoque podría ser aplicar a este problema técnicas de inteligencia artificial para abordar este desafío de manera más efectiva. Una de esas técnicas es el aprendizaje automático (Machine Learning), que pueden analizar grandes conjuntos de datos en tiempo real. Esos datos permitirían determinar de forma más precisa la mejor asignación de pacientes a los recursos y mejorar la secuenciación de tareas aplicando algoritmos como los propuestos.

REFERENCIAS

Abourraja, M. N., Lethvall, S., Marzano, L., Falk, N., Raghothama, J., Boodaghian, A., Adam, A., Darwich, S., & Meijer, S. (2022). *A DATA-DRIVEN DISCRETE EVENT SIMULATION MODEL TO IMPROVE EMERGENCY DEPARTMENT LOGISTICS*. https://doi.org/10.1109/WSC57314.2022.10015465

Adel, H., Wahed, M. A., & Saleh, N. (2018). *A novel Approach for Improving Patient Flow in Emergency Department*.

Álvarez Rueda, J. M., Ávila Rodríguez, F. J., & Caballero García, A. (2018). *PROTOCOLOS DE COORDINACIÓN DE LA ASISTENCIA EXTRAHOSPITALARIA URGENTE Y EMERGENTE DEL SISTEMA SANITARIO PÚBLICO DE ANDALUCÍA*.

Alves de Queiroz, T., Iori, M., Kramer, A., & Kuo, y. (2021). Scheduling of Patients in Emergency Departments with a Variable Neighborhood Search. In *Proceedings*. http://www.springer.com/series/7407

Bahari, A., Aboueljinane, L., & Lebbar, M. (2022). Analysis of Patients Boarding Time in Emergency Departments: A Discrete Event Simulation Approach. *2022 IEEE 14th International Conference of Logistics and Supply Chain Management, LOGISTIQUA 2022*. https://doi.org/10.1109/LOGISTIQUA55056.2022.9938052

Bedoya-Valencia, L., & Kirac, E. (2016). Evaluating alternative resource allocation in an emergency department using discrete event simulation. *Simulation*, *92*(12), 1041–1051. https://doi.org/10.1177/0037549716673150

Bouzir, A., & Benammou, S. (2020). *Towards a New Approach for Developing Patient Flow in Emergency Department: Sahloul and Hached Hospitals Case Study Imen HAMZAOUI*. https://doi.org/10.1109/LOGISTIQUA49782.2020.9353899

Burdett, R. L., & Kozan, E. (2018). An integrated approach for scheduling health care activities in a hospital. *European Journal of Operational Research*, *264*(2), 756–773. https://doi.org/10.1016/J.EJOR.2017.06.051

Chaou, C. H., Chiu, T. F., Yen, A. M. F., Ng, C. J., & Chen, H. H. (2016). Analyzing Factors Affecting Emergency Department Length of Stay-Using a Competing Risk-accelerated Failure Time Model. *Medicine*, *95*(14). https://doi.org/10.1097/MD.0000000000003263

Chin, L., & Fleisher, G. (1998). Planning model of resource utilization in an academic pediatric emergency department. *Pediatric Emergency Care*, *14*(1), 4–9. https://doi.org/10.1097/00006565-199802000-00002

Cuberos, M., Tutor, G., & Santana, A. E. (2014). *Heurística de Recocido Simulado para la resolución del problema del acarreo terrestre*.

Daldoul, D., Nouaouri, I., Bouchriha, H., & Allaoui, H. (2022). *Simulation-based optimisation approach to improve emergency department resource planning: A case study of Tunisian hospital*.

Delahaye, D., Chaimatanan, S., & Mongeau, M. (2019). Simulated Annealing: From Basics to Applications. In M. Gendreau & J.-Y. Potvin (Eds.), *Handbook of Metaheuristics* (pp. 1–35). Springer International Publishing. https://doi.org/10.1007/978-3-319-91086-4_1

Dorgham, K., Nouaouri, I., Ben-Romdhane, H., & Krichen, S. (2019). A Hybrid Simulated Annealing Approach for the Patient Bed Assignment Problem. *Procedia Computer Science*, *159*, 408–417. https://doi.org/https://doi.org/10.1016/j.procs.2019.09.195

Duguay, C., & Chetouane, F. (2016). Modeling and Improving Emergency Department Systems using Discrete Event Simulation. *Http://Dx.Doi.Org/10.1177/0037549707083111*, *83*(4), 311–320. https://doi.org/10.1177/0037549707083111

Eshghali, M., Kannan, D., Salmanzadeh-Meydani, N., & Esmaieeli Sikaroudi, A. M. (2023). Machine learning based integrated scheduling and rescheduling for elective and emergency patients in the operating theatre. *Annals of Operations Research*. https://doi.org/10.1007/s10479-023-05168-x

Etu, E. E., Monplaisir, L., Arslanturk, S., Masoud, S., Aguwa, C., Markevych, I., & Miller, J. (2022). Prediction of Length of Stay in the Emergency Department for COVID-19 Patients: A Machine Learning Approach. *IEEE Access*, *10*, 42229–42237. https://doi.org/10.1109/ACCESS.2022.3168045

Evans, G. W., Gor, T. B., & Unger, E. (1996). *A Simulation Model for Evaluating Personnel Schedules in a Hospital Emergency Department*. https://repository.lib.ncsu.edu/handle/1840.4/6888

Feng, Y. Y., Wu, I. C., & Chen, T. L. (2017). Stochastic resource allocation in emergency departments with a multi-objective simulation optimization algorithm. *Health Care Management Science*, *20*(1), 55–75. https://doi.org/10.1007/s10729-015-9335-1

Feng, Y.-Y., Wu, & I.-C., & Chen, T.-L. (2017). *Stochastic resource allocation in emergency departments with a multi-objective simulation optimization algorithm*. https://doi.org/10.1007/s10729-015-9335-1

Fontes, D. B. M. M., Homayouni, S. M., & Gonçalves, J. F. (2023). A hybrid particle swarm optimization and simulated annealing algorithm for the job shop scheduling problem with transport resources. *European Journal of Operational Research*, *306*(3), 1140–1157. https://doi.org/https://doi.org/10.1016/j.ejor.2022.09.006

Gómez Medina, D. (2021). *Algoritmos de Optimización para el servicio de urgencias: Caso de estudio en el Hospital Universitario Virgen del Rocío*.

Hamzaoui, I., Bouzir, A., & Benammou, S. (2021). Towards A Fuzzy Rule-Based Approach for Patient Flow Management in Emergency Department During the COVID-19: Sahloul Hospital Case Study, Tunisia. *2021 International Conference on Engineering and Emerging Technologies (ICEET)*, 1–5. https://doi.org/10.1109/ICEET53442.2021.9659696

Hamzaoui, I., Bouzir, A., & Benammou, S. (2022). *A Comparative Machine Learning Approaches for Patient Flow Forecasting in an Emergency Department during the COVID-19*. https://doi.org/10.1109/LOGISTIQUA55056.2022.9938025

Harzi, M., Condotta, J.-F., Nouaouri, I., & Krichen, S. (2017). Scheduling Patients in Emergency Department by Considering Material Resources. *Procedia Computer Science*, *112*, 713–722. https://doi.org/https://doi.org/10.1016/j.procs.2017.08.153

Harzi, M., Condotta, J.-F., Nouaouri, I., & Krichen, S. (2018). Using the hybrid ILS/VND method for solving the patients scheduling problem in emergency department: a case study. *Procedia Computer Science*, *126*, 733–742. https://doi.org/https://doi.org/10.1016/j.procs.2018.08.007

Hospitaux universitaires de gèneve. (n.d.). *La admisión en el servicio de urgencias*. www.urgences-ge.ch

Hu, X., Cao, H., Shi, J., Dai, Y., & Dai, W. (2021). *Study of hospital emergency resource scheduling based on digital twin technology*. https://doi.org/10.1109/ICIBA52610.2021.9688239

Hu, Y., Wang, J., & Liu, G. (2022). *Identify Bottlenecks of Patient Flow in Emergency Departments*. https://doi.org/10.1109/CASE49997.2022.9926432

Jiménez Murillo, L., Díaz Borrego, J., & López Serrato, M. (2019). *Plan de Mejora de los Servicios de Urgencias de Hospital del Sistema Sanitario Público de Andalucía*.

Katoch, S., Chauhan, S. S., & Kumar, V. (2021). A review on genetic algorithm: past, present, and future. *Multimedia Tools and Applications*, *80*(5), 8091–8126. https://doi.org/10.1007/s11042-020-10139-6

Kırış, Ş., Yüzügüllü, N., Ergün, N., & Alper Çevik, A. (2010). A knowledge-based scheduling system for Emergency Departments. *Knowledge-Based Systems*, *23*(8), 890–900. https://doi.org/https://doi.org/10.1016/j.knosys.2010.06.005

La biblioteca estándar de Python — documentación de Python - 3.12.0. (n.d.). https://docs.python.org/es/3/library/index.html

Leer archivos Excel con Python - Análisis y Decisión. (n.d.). https://analisisydecision.es/leer-archivos-excel-con-python/

Liu, Y., Moyaux, T., Bouleux, G., & Cheutet, V. (2022). Hybrid Simulation Modelling of Emergency Departments for Resource Scheduling. *Journal of Simulation*, 1–16.

https://doi.org/10.1080/17477778.2023.2187321

López Garcelán, J. (2023). *Desarrollo de modelos de programación lineal para la gestión del flujo de pacientes en un servicio de urgencias hospitalario*.

María, A., & Virto, M. (2017). *SERVICIOS DE URGENCIAS HOSPITALARIAS: INFLUENCIA DE LA CARACTERIZACIÓN DE LOS PACIENTES Y SU PROCESO ASISTENCIAL DURANTE SU TIEMPO DE PERMANENCIA*.

matplotlib.pyplot — Matplotlib 3.5.3 documentation. (n.d.). https://matplotlib.org/3.5.3/api/_as_gen/matplotlib.pyplot.html

Mould, G., Bowers, J., Dewar, C., & McGugan, E. (2016). Assessing the Impact of Systems Modeling in the Redesign of an Emergency Department. *Operational Research for Emergency Planning in Healthcare: Volume 2*, 31–47. https://doi.org/10.1007/978-1-137-57328-5_3

Moyano Personat, J. (2023). *Desarrollo de algoritmos aproximados para la resolución del problema de gestión del flujo de pacientes en un servicio de urgencia hospitalario*.

NumPy : La biblioteca de Python más utilizada en Data Science. (n.d.). https://datascientest.com/es/numpy-la-biblioteca-python

Raunak, M., Osterweil, L., Wise, A., Clarke, L., & Henneman, P. (2009). Simulating patient flow through an emergency department using process-driven discrete event simulation. *Proceedings of the 2009 ICSE Workshop on Software Engineering in Health Care, SEHC 2009*, 73–83. https://doi.org/10.1109/SEHC.2009.5069608

Rossetti, M. D., Trzcinski, G. F., & Syverud, S. A. (n.d.). *EMERGENCY DEPARTMENT SIMULATION AND DETERMINATION OF OPTIMAL ATTENDING PHYSICIAN STAFFING SCHEDULES*.

Ruiz, R., & Stützle, T. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, *44*, 2033–2049. https://doi.org/10.1016/j.ejor.2005.12.009

Tudela, P., & Deltell, J. M. M. (2015). La saturación en los servicios de urgencias hospitalarios. *Emergencias: Revista de La Sociedad Española de Medicina de Urgencias y Emergencias, ISSN 1137-6821, Vol. 27, Nº. 2 (Abril), 2015, Págs. 113-120*, *27*(2), 113–120. https://dialnet.unirioja.es/servlet/articulo?codigo=5388440&info=resumen&idioma=ENG

Umoren, I., Udonyah, K., & Isong, E. (2019). *A Computational Intelligence Framework for Length of Stay Prediction in Emergency Healthcare Services Department*.

Vali, M., Salimifard, K., Gandomi, A. H., & Chaussalet, T. J. (2022). *Application of job shop scheduling approach in green patient flow optimization using a hybrid swarm intelligence*. https://doi.org/10.1016/j.cie.2022.108603

Van Bockstal, E., & Maenhout, B. (2018). *A study on the impact of prioritising emergency department arrivals on the patient waiting time*.

Velez, M. C., & Montoya, J. A. (2007). METAHEURISTICOS: UNA ALTERNATIVA PARA LA SOLUCION DE PROBLEMAS COMBINATORIOS EN ADMINISTRACION DE OPERACIONES. *Revista EIA*, 99–115. http://www.scielo.org.co/scielo.php?script=sci_arttext&pid=S1794-12372007000200009&nrm=iso

Vidal, A., Carpente, M. L., & Casas Méndez, B. (2013). *Algoritmos heurísticos en optimización*.

Wang, J. (2023). Patient Flow Modeling and Optimal Staffing for Emergency Departments: A Petri Net Approach. *IEEE TRANSACTIONS ON COMPUTATIONAL SOCIAL SYSTEMS*, *10*(4). https://doi.org/10.1109/TCSS.2022.3186249

Wang, J., & Wang, J. (2023). Real-Time Adaptive Allocation of Emergency Department Resources and Performance Simulation Based on Stochastic Timed Petri Nets. *IEEE Transactions on Computational Social Systems*, *10*(4), 1986–1996. https://doi.org/10.1109/TCSS.2023.3266501

Wang, Z., Liu, R., & Sun, Z. (2023). Physician Scheduling for Emergency Departments Under Time-Varying

Demand and Patient Return. *IEEE Transactions on Automation Science and Engineering*, *20*(1), 553–570. https://doi.org/10.1109/TASE.2022.3163259

Waskito, A. A., Arifin, A., & Nuh, M. (2022). *Optimization of Emergency Department Bed Availabilty using Patient Detection System*. https://doi.org/10.1109/ISITIA56226.2022.9855365

Zeinali, F., Mahootchi, M., & Sepehri, M. M. (2015). Resource planning in the emergency departments: A simulation-based metamodeling approach. *Simulation Modelling Practice and Theory*, *53*, 123–138. https://doi.org/https://doi.org/10.1016/j.simpat.2015.02.002

ANEXO A

- Resultados de la calibración del algoritmo del recocido Simulado.

Instancias	N	FO Lk= N/2	FO Lk= N	FO Lk=2N	Mejor solución	RPD N/2	RPD N	RPD 2*N
1	12	1357	1391	1357	1357	0%	3%	0%
2	9	1349	1349	1349	1349	0%	0%	0%
3	13	1846	1846	1858	1846	0%	0%	1%
4	14	1189	1190	1188	1188	0%	0%	0%
5	15	2162	2156	2154	2154	0%	0%	0%
6	8	1636	1636	1636	1636	0%	0%	0%
7	16	3249	3252	3255	3249	0%	0%	0%
8	9	1006	998	998	998	1%	0%	0%
9	11	1597	1602	1581	1581	1%	1%	0%
10	13	2009	2009	2009	2009	0%	0%	0%

Tabla 16. Valores de la RPD obtenidos por instancia del recocido simulado

	Lk = N/2	***LK = N***	***LK = 2*N***
ARPD	0,23%	0,42%	0,08%

Tabla 17. ARPD de cada escenario del recocido simulado

- Resultados de la calibración de la Búsqueda Tabú.

Instancias	Número pacientes	FO Tabú_size = N/2	FO Tabú_size = N	FO Tabú_size =2N	Mejor solución	RPD N/2	RPD N	RPD 2*N
1	23	2144	2144	2132	2132	1%	1%	0%
2	19	2132	2132	2132	2132	0%	0%	0%
3	18	1173	1167	1165	1165	1%	0%	0%
4	11	2898	2886	2886	2886	0%	0%	0%
5	11	1920	1920	1920	1920	0%	0%	0%
6	9	1542	1542	1542	1542	0%	0%	0%
7	15	1811	1811	1811	1811	0%	0%	0%
8	22	3034	3034	3034	3034	0%	0%	0%
9	20	2044	2044	2044	2044	0%	0%	0%
10	21	3417	3356	3356	3356	2%	0%	0%

Tabla 18. Valores RPD obtenidos por instancia de la búsqueda tabú

	Tabú_size = N/2	***Tabú_size = N***	***Tabú_size = 2*N***
ARPD	0,35%	0,07%	0,00%

Tabla 19. ARPD de cada escenario de la búsqueda tabú

ANEXO B

Instancias	F.O. Algoritmo Genético	F.O. Recocido Simulado	F.O. Búsqueda Tabú	Mejor solución	RPD Algoritmo genético	RPD Recocido simulado	RPD Búsqueda Tabú
1	979	979	979	979	0%	0%	0%
2	653	653	653	653	0%	0%	0%
3	1172	1116	1124	1116	5%	0%	1%
4	606	594	594	594	2%	0%	0%
5	1642	1618	1618	1618	1%	0%	0%
6	638	619	632	619	3%	0%	2%
7	510	510	510	510	0%	0%	0%
8	753	731	741	731	3%	0%	1%
9	1225	1198	1202	1198	2%	0%	0%
10	1026	983	983	983	4%	0%	0%

Tabla 20. RDP obtenidos para las 10 instancias en el caso 1

Instancias	F.O. Algoritmo Genético	F.O. Recocido Simulado	F.O. Búsqueda Tabú	Mejor solución	RPD Algoritmo genético	RPD Recocido simulado	RPD Búsqueda Tabú
1	1494	1473	1473	1473	1%	0%	0%
2	862	855	855	855	1%	0%	0%
3	907	783	783	783	16%	0%	0%
4	1292	1249	1249	1249	3%	0%	0%
5	1256	1256	1256	1256	0%	0%	0%
6	1053	973	994	973	8%	0%	2%
7	1647	1642	1642	1642	0%	0%	0%
8	1781	1781	1790	1781	0%	0%	1%
9	821	817	821	817	0%	0%	0%
10	1461	1408	1457	1408	4%	0%	3%

Tabla 21. RDP obtenidos para las 10 instancias en el caso 2

Instancias	F.O. Algoritmo Genético	F.O. Recocido Simulado	F.O. Búsqueda Tabú	Mejor solución	RPD Algoritmo genético	RPD Recocido simulado	RPD Búsqueda Tabú
1	3480	3455	3429	3429	1%	1%	0%
2	2527	2502	2512	2502	1%	0%	0%
3	2227	2037	2010	2010	11%	1%	0%
4	2942	2650	2687	2650	11%	0%	1%
5	1628	1601	1622	1601	2%	0%	1%
6	2871	2770	2774	2770	4%	0%	0%
7	3052	3050	3058	3050	0%	0%	0%
8	2617	2573	2622	2573	2%	0%	2%
9	2160	2095	2093	2093	3%	0%	0%
10	3712	3644	3660	3644	2%	0%	0%

Tabla 22. RPD obtenidos para las 10 instancias en el caso 3

Instancias	F.O. Algoritmo Genético	F.O. Recocido Simulado	F.O. Búsqueda Tabú	Mejor solución	RPD Algoritmo genético	RPD Recocido simulado	RPD Búsqueda Tabú
1	1447	1381	1404	1381	5%	0%	2%
2	1240	1202	1227	1202	3%	0%	2%
3	1292	1290	1290	1290	0%	0%	0%
4	1189	1160	1166	1160	3%	0%	1%
5	1196	1174	1176	1174	2%	0%	0%
6	1193	1152	1146	1146	4%	1%	0%
7	2064	2022	2021	2021	2%	0%	0%
8	1089	1059	1059	1059	3%	0%	0%
9	1126	1126	1126	1126	0%	0%	0%
10	1201	1174	1174	1174	2%	0%	0%

Tabla 23. RDP obtenidos para las 10 instancias en el caso 4

Instancias	F.O. Algoritmo Genético	F.O. Recocido Simulado	F.O. Búsqueda Tabú	Mejor solución	RPD Algoritmo genético	RPD Recocido simulado	RPD Búsqueda Tabú
1	2277	2263	2275	2263	1%	0%	1%
2	1806	1707	1707	1707	6%	0%	0%
3	1959	1947	1954	1947	1%	0%	0%
4	2103	2083	2089	2083	1%	0%	0%
5	1984	1934	1950	1934	3%	0%	1%
6	1511	1434	1440	1434	5%	0%	0%
7	817	817	817	817	0%	0%	0%
8	1284	1216	1216	1216	6%	0%	0%
9	1333	1326	1326	1326	1%	0%	0%
10	2143	2135	2158	2135	0%	0%	1%

Tabla 24. RDP obtenidos para las 10 instancias en el caso 5

Instancias	F.O. Algoritmo Genético	F.O. Recocido Simulado	F.O. Búsqueda Tabú	Mejor solución	RPD Algoritmo genético	RPD Recocido simulado	RPD Búsqueda Tabú
1	3190	2992	3012	2992	7%	0%	1%
2	2934	2895	2903	2895	1%	0%	0%
3	3321	3229	3263	3229	3%	0%	1%
4	2511	2477	2502	2477	1%	0%	1%
5	3061	2842	2850	2842	8%	0%	0%
6	2259	2077	2078	2077	9%	0%	0%
7	2984	2977	2996	2977	0%	0%	1%
8	4170	3892	3925	3892	7%	0%	1%
9	2395	2351	2351	2351	2%	0%	0%
10	2332	2267	2267	2267	3%	0%	0%

Tabla 25. RDP obtenidos para las 10 instancias en el caso 6

ANEXO C

Experimentación.py

```
import numpy as np
import xlrd
import random
import math
import algoritmos
import Gantt

def read_file_excel(filename):
    """función utilizada para extraer de un archivo excel los datos de duraciones de las actividades
    y recursos utilizados por dichas actividades según la prioridad de los pacientes"""
    """[tiempo minimo,tiempo medio,tiempo máximo,TIPO DE RECURSO,Número recursos necesarios]"""
    """Ejemplo datos de salida: ['20.0' '30.0' '75.0' 'P' '1.0']['3.0' '5.0' '15.0' 'LAB' '1.0']['3.0' '5.0' '15.0' 'RAY'
'1.0']['20.0' '20.0' '20.0' 'SCAN' '1.0']
    ['20.0' '30.0' '40.0' 'P' '1.0']['20.0' '30.0' '40.0' 'N' '2.0']"""

    wb = xlrd.open_workbook(filename)
    activities = wb.sheet_by_index(1)
    resources = wb.sheet_by_index(0)
    recursos=[]
    actividades = []
    for fila in range(1,activities.nrows):
        columnas2 = []
        for columna in range(3,8):
            columnas2.append(activities.cell_value(fila,columna))
        actividades.append(columnas2)
    df1 = np.array(actividades)

    for fila in range(1,resources.nrows):
        for columna in range(1,8):
            recursos.append(int(resources.cell_value(fila,columna))) #numero recursos
disponibles=[Boxes(B),Physician(P),Physsician assistant(PA),Nurses (N),LAB,RAY,CT-Scan]

    return df1,recursos

def calculate_time_by_activity(prioridad,general_data):
    """En esta funcion se recibe como argumentos los pacientes y las prioridades asociadas a cada uno junto con los
datos extraídos del excel
    y se devuelve los tiempos medios y los recursos utilizados por cada actividad que requiere cada paciente según su
prioridad"""
    """Ejemplo valores entrada:  [tiempo minimo,tiempo medio,tiempo máximo,TIPO DE RECURSO,Número
recursos necesarios] y [('Pac1', 1), ('Pac2', 2), ('Pac3', 3), ('Pac4', 4)] """
    """Ejemplo valores salida: [[5, 'PA', 1], [9, 'N', 1], [6, 'RAY', 1], [11, 'PA', 1], [9, 'N', 1]]"""
```

```
    probabilidad = random.random()  #genera una probabilidad que se utilizará para decidir si es necesaria o no alguna actividad
    total_time=[] #recoge los timepos medios de todas las actividades para cada paciente

    #primero calcular los tiempos de proceso a partir de las triangulares de todas las actividades para cada paciente
    for act in general_data:
        t=math.ceil(random.triangular(float(act[0]),float(act[2]),float(act[1])))
        time_for_activity=[t,act[3],int(float(act[4]))] #[duración media, recurso utilizado, numero de recursos necesarios]
        total_time.append(time_for_activity)

    #luego tomar solo las actividades que están en el proceso de emergencia del paciente
    if prioridad==1:
        J=total_time[0:4]
        if probabilidad <= 0.7: #Verificar si se realiza la actividad o no CTscan
            J.append(total_time[4])
        J.extend(total_time[5:7])
    elif prioridad==2:
        J=total_time[7:11]
        if probabilidad <= 0.7: #Verificar si se realiza la actividad o no CTscan
            J.append(total_time[11])
        J.extend(total_time[12:14])
    elif prioridad==3:
        J=total_time[14:18]
        if probabilidad <= 0.3: #Verificar si se realiza la actividad o no CTscan
            J.append(total_time[18])
        J.extend(total_time[19:21])
    elif prioridad==4:
        J=total_time[21:26]
    elif prioridad==5:
        J=total_time[26:28]


    return J

def experimentacion(general_data,recursos_disponibles):
    """Esta función se tiene como finalidad generar los datos de entrada de los algoritmos y los ejecuta para evluar su rendimiento"""

    #inicialización de parámetros""
    random.seed(13)
    numero_pacientes=0
    numero_actividades=0
    numero_recursos=0
    LOS_medio=((3.15+3.14+2.86+1.49+1.31)*60)/5
    t_ocupado=1000
    tiempos_espera=[10,14,59,99,119] #tiempos que en el instante de inicio puede llevar esperando el paciente
    sat=1.5 #% de saturación de saturación del SUH
    nivel_saturacion=math.ceil(sat*LOS_medio) #saturación de saturación del SUH en minutos
    FO_genetico=[]
```

```
    FO_recocido=[]
    FO_tabu=[]
    total_recursos=sum(recursos_disponibles)
    recursos_nombres=['BOX','P','N','PA','LAB','RAY','SCAN']
    dic_recursos_disponibles=(dict(list(zip(recursos_nombres,recursos_disponibles)))) #diccionario de los recursos iniciales disponibles que dependen del horario
    #6 a.m
    #recursos_disponibles=[27,1,6,1,1,1,1] #[BOX,P,N,PA,LAB,RAY,SCAN]
    #12 a.m
    #recursos_disponibles=[27,2,9,2,1,1,1] #[BOX,P,N,PA,LAB,RAY,SCAN]

    #se generan 10 instancias
    for instancia in range(10):
        tipos_recursos_ocupados = {'BOX':0,'P': 0, 'N': 0, 'PA': 0, 'LAB': 0, 'RAY': 0, 'SCAN': 0}
        sat_recursos=[0 for _ in range(7)]
        tiempos_esperados=[]
        recursos_ocupados=[]
        datos=[]
        flag=1
        pac=0
        prioridades=[]
        nombres=[]
        actividades_instancia=0

        #se generan pacientes con sus prioridades y procesos de urgencia hasta alcanzar el nivel de saturación
        while flag==1:
            pac+=1
            #se genera la prioridad y las duraciones de su proceso de urgencia
            prioridad_paciente=generate_priority()
            duracion_actividades=calculate_time_by_activity(prioridad_paciente,general_data)

            #se genera la probabilidad aleatoria que define es qué situación se encuentra el paciente e ese instante
            if tipos_recursos_ocupados['BOX']<recursos_disponibles[0]: #si no están ocupados el maximo de box puede estar en cualquiera de las 4 situacioines
                probabilidad_estado=random.random()
            else: #si están todos ocupados solo puede estar en la situacion 1 o en la 2
                probabilidad_estado=random.uniform(0, 0.5)

            #se pueden dar 4 situaciones en las que puede estar ese paciente en el proceso de la foto
            #situación 1: paciente que llega en el instante analizado
            if probabilidad_estado<=0.25:
                tiempos_esperados.append(1)
                proceso=duracion_actividades

            #situación 2: paciente que llega antes del instante que se analiza pero aun no ha sido atendido
            elif probabilidad_estado>0.25 and probabilidad_estado<=0.5:
                if prioridad_paciente ==1: #se le asigna la máxima espera para que se le de la máxima prioridad
                    tiempos_esperados.append(119)
                else: #si no es de nivel ESI 1 se le asigna una espera random
                    random_espera = random.randint(0, 4)
```

```
            tiempos_esperados.append(tiempos_espera[random_espera])
        proceso=duracion_actividades

    #situación 3:paciente que en el instante analizado se encuentra en el proceso de urgencia pero sin ocupar recurso (únicamente el box)
    elif probabilidad_estado>0.5 and probabilidad_estado<=0.75:
        tipos_recursos_ocupados['BOX']+=1
        tiempos_esperados.append(0) #el tiempo inicial esperado será 0 porque no se cuenta para la evaluación
        punto_proceso= random.randint(1, len(duracion_actividades)-1) #se genera un numero aleatorio que determina en que punto del PU está el paciente
        proceso=duracion_actividades[punto_proceso:]
        box_ocupado=[t_ocupado,'BOX',1,f'Pac{pac}'] #se ocupa un box con el paciente
        recursos_ocupados.append(box_ocupado)

    #situación 4:paciente que en el instante analizado se encuentra en el proceso de urgencia y ocupando recursos
    else:
        tiempos_esperados.append(0) #el tiempo inicial esperado será 0 porque no se cuenta para la evaluación
        proceso=[]
        tipos_recursos_ocupados['BOX']+=1
        for _ in range(7): #evitamos que no se generen mas recursos ocupados de los que hay disponibles
            punto_proceso= random.randint(1, len(duracion_actividades)-1)
            if tipos_recursos_ocupados[duracion_actividades[punto_proceso][1]]<dic_recursos_disponibles[duracion_actividades[punto_proceso][1]]:
                break

        tipos_recursos_ocupados[duracion_actividades[punto_proceso][1]]+=duracion_actividades[punto_proceso][2]

        actividad_en_curso=[math.ceil((duracion_actividades[punto_proceso][0])/2),duracion_actividades[punto_proceso][1],duracion_actividades[punto_proceso][2]]
        actividad_en_curso.append(f'Pac{pac}')
        proceso.extend(duracion_actividades[punto_proceso+1:])
        box_ocupado=[t_ocupado,'BOX',1,f'Pac{pac}']
        recursos_ocupados.append(box_ocupado)
        recursos_ocupados.append(actividad_en_curso)

    #se calcula la saturación de los recursos examinando las actividades del PU de los pacientes:
    for act in proceso:
        sat_recursos[0]+=act[0] #saturacion del box
        if act[1]=='P':
            sat_recursos[1]+=act[0]*act[2] #saturacion del recurso médico
        elif act[1]=='N':
            sat_recursos[2]+=act[0]*act[2] #saturacion del recurso enfermero
        elif act[1]=='PA':
            sat_recursos[3]+=act[0]*act[2] #saturacion del recurso médico asistente
        elif act[1]=='LAB':
            sat_recursos[4]+=act[0]*act[2] #saturacion del recurso laboratorio
        elif act[1]=='RAY':
            sat_recursos[5]+=act[0]*act[2] #saturacion del recurso rayos
```

```
        elif act[1]=='SCAN':
            sat_recursos[6]+=act[0]*act[2] #saturacion del recurso CT-SCAN

    #se suma la saturación de los recursos que se encuentran ocupados:
    for rec in recursos_ocupados:
        if rec[1]=='P':
            sat_recursos[1]+=rec[0]*rec[2]  #saturacion del recurso médico
            sat_recursos[0]+=rec[0]#saturacion del box
        elif rec[1]=='N':
            sat_recursos[2]+=rec[0]*rec[2] #saturacion del recurso enfermero
            sat_recursos[0]+=rec[0]#saturacion del box
        elif rec[1]=='PA':
            sat_recursos[3]+=rec[0]*rec[2] #saturacion del recurso médico asistente
            sat_recursos[0]+=rec[0]#saturacion del box
        elif rec[1]=='LAB':
            sat_recursos[4]+=rec[0]*rec[2] #saturacion del recurso laboratorio
            sat_recursos[0]+=rec[0]#saturacion del box
        elif rec[1]=='RAY':
            sat_recursos[5]+=rec[0]*rec[2] #saturacion del recurso rayos
            sat_recursos[0]+=rec[0]#saturacion del box
        elif rec[1]=='SCAN':
            sat_recursos[6]+=rec[0]*rec[2] #saturacion del recurso CT-SCAN
            sat_recursos[0]+=rec[0]#saturacion del box

    #se evalua si algún recursos ha superado la saturación máxima
    for j in range(len(sat_recursos)):
        if(sat_recursos[j]>=nivel_saturacion*recursos_disponibles[j]):

            flag=0
            break

    datos.append(proceso)
    prioridades.append(prioridad_paciente)
    nombres.append(f'Pac{pac}')


solucion_inicial = list(zip(nombres, prioridades))

numero_pacientes+=pac

for proc in datos:
    for act in proc:
        numero_recursos+=act[2]
        numero_actividades+=1
        actividades_instancia+=1

#se calcula el tiempo máximo de ejecución de los algoritmos
tiempo_limite_milisegundos =(actividades_instancia*total_recursos)*60/2
tiempo_limite=tiempo_limite_milisegundos/1000
```

```
        #se calcula la mejor solución a partir de la solucion inicial y los datos proporcionados
        #algoritmo genético

best_solution_genetico,best_FO_genetico=algoritmos.algoritmo_genetico(solucion_inicial,datos,recursos_disponibles,tiempo_limite,tiempos_esperados,recursos_ocupados)

        #simulate annealing

best_solution_recocido,best_FO_recocido=algoritmos.simulated_annealing(solucion_inicial,datos,recursos_disponibles,tiempo_limite,tiempos_esperados,recursos_ocupados)

        #tabu search
        best_solution_tabu,
best_FO_tabu=algoritmos.tabu_search(solucion_inicial,datos,recursos_disponibles,tiempo_limite,tiempos_esperados,recursos_ocupados)

        #se almacenan los datos de las instancias
        FO_genetico.append(best_FO_genetico)
        FO_recocido.append(best_FO_recocido)
        FO_tabu.append(best_FO_tabu)


    return best_solution_recocido,datos,recursos_ocupados


def generate_priority():

    prioridades=[1,2,3,4,5]
    prior=random.choices(prioridades,weights=[0.3,10.8,48.4,36.7,1.5])[0]

    return prior

if __name__ == "__main__":
    archivo = 'C:/Users/evita/Documents/MASTER/TFM/DATOS.xls'
    datos_iniciales,recursos_disponibles = read_file_excel(archivo) #lectura y extraccion de los datos del fichero
    best_solution,datos,recursos_ocupados=experimentacion(datos_iniciales,recursos_disponibles)

    #representacion de la mejor solucion

t_P,t_PA,t_N,t_RAY,t_LAB,t_SCAN,t_BOX=Gantt.calculate_Gantt(best_solution,datos,recursos_disponibles,recursos_ocupados)


    # Impresión y visualización de resultados
    Gantt.diagrama_de_gantt(best_solution,t_P,t_PA,t_N,t_RAY,t_LAB,t_SCAN,t_BOX,recursos_disponibles)
#dibujar Gantt con los datos de la mejor solución
```

Algoritmos.py

```
import numpy as np
import random
import copy
import math
import time
import evaluacion

def
algoritmo_genetico(solucion_inicial,tiempo_total,recursos_disponibles,tiempo_maximo,tiempos_esperados,recur
sos_ocupados):

   tiempo_inicio=time.time()
   Pc=0.9
   Pm=0.1
   P=math.ceil(len(solucion_inicial)/2)


   #Inicialización: se crea una poblacion inicial aplicando el operador insertion
   poblacion=[solucion_inicial]
   for i in range(P):
      for j in range (i+1,len(solucion_inicial)):
         nueva_solucion=solucion_inicial[0:i] + [solucion_inicial[j]] + solucion_inicial[i+1:j] + [solucion_inicial[i]] +
solucion_inicial[j+1:]
         poblacion.append(nueva_solucion)

   #bucle de generaciones de soluciones
   while True:
      descendientes=[]
      probabilidad_cruce=random.random()
      probabilidad_mutacion=random.random()

      for _ in range(P):
         evaluados=[]
         #seleccion basada en torneo: se seleccionan las soluciones con mejor fitness de una muestra aleatoria de la
poblacion
         padres=[]
         for _ in range(2): #queremos 2 padres
            torneo= random.sample(poblacion,3) #se seleccionan 3 padres al azar y de esos uno
            padres.append(min(torneo,  key=lambda solucion: evaluacion.evaluate_solution(solucion,
tiempo_total,recursos_disponibles,tiempos_esperados,recursos_ocupados)))
         evaluados.extend(padres)
         #cruce de un punto: si se da la probabilidad de cruce se cruza
         padre1, padre2 = random.sample(padres, 2)
         if probabilidad_cruce<=Pc:
            punto_cruce = random.randint(1, len(padre1)-1)
            hijo = padre1[:punto_cruce] + [x for x in padre2 if x not in padre1[:punto_cruce]]

         else:
```

```
            hijo=padre1

        #mutación aleatoria:Si se da la probabilidad de mutación se intercambian dos valores
        if probabilidad_mutacion<=Pm:
            indice1, indice2 = random.sample(range(len(hijo)), 2)
            hijo[indice1], hijo[indice2] = hijo[indice2], hijo[indice1]
        evaluados.extend([hijo])

        #elitismo:selecciono el que tenga mejor fitness entre los padres y el hijo
        descendiente=min(evaluados,  key=lambda solucion: evaluacion.evaluate_solution(solucion,
tiempo_total,recursos_disponibles,tiempos_esperados,recursos_ocupados))
        descendientes.extend([descendiente])

    poblacion=descendientes
    best_solution = min(poblacion, key=lambda solucion: evaluacion.evaluate_solution(solucion,
tiempo_total,recursos_disponibles,tiempos_esperados,recursos_ocupados))
    best_fitness =
evaluacion.evaluate_solution(best_solution,tiempo_total,recursos_disponibles,tiempos_esperados,recursos_ocup
ados)


    #se comprueba el tiempo transcurrido
    tiempo_actual = time.time()
    tiempo_transcurrido = tiempo_actual - tiempo_inicio
    if tiempo_transcurrido>=tiempo_maximo:
        break

  return best_solution, best_fitness

def
tabu_search(initial_solution,tiempo_total,recursos_disponibles,tiempo_maximo,tiempos_esperados,recursos_ocu
pados):

  tiempo_inicio = time.time()
  tabu_size = 2*len(initial_solution) #iteraciones en las que un valor será tabú

  #se inicializa la solucion actual, la lista tabú y la mejor solucion
  current_solution=initial_solution

current_LOS=evaluacion.evaluate_solution(current_solution,tiempo_total,recursos_disponibles,tiempos_esperad
os,recursos_ocupados)
  best_solution=current_solution
  best_LOS=current_LOS
  tabu_list = np.zeros([len(current_solution), len(current_solution)], dtype="int") #matriz con tantos 0 por linea y
columna como pacientes
  #se crea una lista de vecinos a partir de la solicuon actual

  while True:
    swap = [] #recoje los dos valores de las posiciones que se intercambian
    neighbourhood=[current_solution]
```

```
    neighbourhood_value=[current_LOS]
    for i in range(len(current_solution)):
      for j in range (i+1,len(current_solution)):
        neighbour=current_solution[0:i] + [current_solution[j]] + current_solution[i+1:j] + [current_solution[i]] +
current_solution[j+1:] #(insertion operation) los vecinos se calculan insertando el valor i en las diferentes
posiciones de la solucion y manteniendo el resto de valores en su posicion
        value = evaluacion.evaluate_solution(neighbour,
tiempo_total,recursos_disponibles,tiempos_esperados,recursos_ocupados) #evaluo el vecino
        ti=initial_solution.index(current_solution[i]) #posicion de los pacientes intercambiados
        tj=initial_solution.index(current_solution[j])

        if tabu_list[ti][tj] == 0 or value < best_LOS: #si las posiciones intercambiadas no estan en la lista tabo o el
valor es menor
          neighbourhood.append(neighbour)
          neighbourhood_value.append(value)
          swap.append([ti, tj]) #guardo las posiciones intercambiadas

    #anteriormente se crea la vecinadad reducida excluyendo los que tiene valores en el tabu list+
    #en el siguiente paso según tomo el vecindario con menor neighbourhood value
    current_LOS, swap, current_solution=min(zip(neighbourhood_value, swap, neighbourhood))


    if current_LOS < best_LOS: #si es mejor que la mejor solucion historica
      best_solution=current_solution[:]
      best_LOS=current_LOS


    #editamos lista tabú
    tabu_list[swap[0]][swap[1]]=tabu_size+1 #posicion del cambio en el tabu list
    for i in range(len(tabu_list)):
      for j in range(len(tabu_list)):
        if tabu_list[i][j]>0:
          tabu_list[i][j]=tabu_list[i][j]-1

    #se comprueba el tiempo transcurrido
    tiempo_actual = time.time()
    tiempo_transcurrido = tiempo_actual - tiempo_inicio
    if tiempo_transcurrido>=tiempo_maximo:
      break

  return best_solution, best_LOS

def
simulated_annealing(initial_solution,tiempo_total,recursos_disponibles,tiempo_maximo,tiempos_esperados,recur
sos_ocupados): #utilizamos el recocido simulado para generar soluciones

  tiempo_inicio=time.time()
  #parámetros
  max_temp = 1000 #temperatura inicial
  min_temp = 0.01 #temperatura final que marca el final del algoritmo
```

```
    L = 2*len(initial_solution) #iteraciones que estaremos en cada nivel de temperatura
    alpha=0.94 #indica como vamos a ir bajando la temperatura

    #inicializa variables
    temp = max_temp

    #toma una solucion inicial e inicializa el valor de la mejor solucion con ella
    current_solution=initial_solution #orden de los pacientes

current_LOS=evaluacion.evaluate_solution(current_solution,tiempo_total,recursos_disponibles,tiempos_esperados,recursos_ocupados) #devuelve tiempo de espera total de los pacientes
    best_LOS= current_LOS #guardará el mejor valor vecindarios generados en todas las temp evaluadas
    best_solution = copy.deepcopy(current_solution)


    while True:
        while temp > min_temp:
            for i in range(L):#veces que se calculan neighbourhoods
                neighbourhood=[current_solution]
                for i in range(len(current_solution)): #para cada iteracion L genero la combinacion de la vecindad entera y le doy un orden aleatorio
                    for j in range (i+1,len(current_solution)):
                        neighbour=current_solution[0:i] + [current_solution[j]] + current_solution[i+1:j] + [current_solution[i]] + current_solution[j+1:] #(insertion operation) Esta vecindad se construye considerando todas las posibles permutaciones de intercambio
                        neighbourhood.append(neighbour)
                order=list(range(len(neighbourhood)))
                random.shuffle(order) #mejorar la exploración del espacio de búsqueda y evita repeticiones de exploración.

                #elijo un vecino al azar y ahora lo evaluo
                for i in range(len(neighbourhood)):
                    new_LOS =evaluacion.evaluate_solution(neighbourhood[order[i]],tiempo_total,recursos_disponibles,tiempos_esperados,recursos_ocupados)

                    #calculo la diferencia entre ambas FO
                    delta = new_LOS - current_LOS
                    if delta<0: #si es menor que cero la nueva solucion es mejor que la estoy y esa solucion pasa a ser la solucion actual
                        current_LOS= new_LOS
                        current_solution = copy.copy(neighbourhood[order[i]])
                        if current_LOS < best_LOS: #se compara si tambien es la mejor solucion de todos los vecindarios generados en todas las temp evaluadas hasta el momento
                            best_LOS = current_LOS
                            best_solution = copy.copy(neighbourhood[order[i]])
                        break
                    else:
                        #en caso de que sea mayor genero al azar un numero y veo si ese numero es menor que la expresion matemática de aceptar el cambio
```

```
                a=random.random()
                if a < math.exp(-delta/temp): #math.exp(-delta/temp) es la probabilidad de que se acepte la solucion candidata como actual aunque sea peor
                    #si se da la condicion acepto el cambio y paso a la siguiente temperatura y si no pues paso a la siguiente iteracion dentro de mi  temperatura
                    current_LOS = new_LOS
                    current_solution = copy.copy(neighbourhood[order[i]])
                    break


        temp = temp / (1+alpha * temp)
    #se comprueba el tiempo transcurrido
    tiempo_actual = time.time()
    tiempo_transcurrido = tiempo_actual - tiempo_inicio
    if tiempo_transcurrido>=tiempo_maximo:
        break

  return best_solution,best_LOS
```

Evaluación.py

```
def evaluate_solution (Pacientes,Tiempos,recursos_disponibles,tiempos_esperados,recursos_ocupados):
  """Esta función es la encargada de evaluar las soluciones generadas. Recibe como argumento la matriz cuyas filas son los pacientes
   y las columnas las actividades que necesita cada uno y las va ordenando de manera que que minimice el LOS"""


  total_LOS=0
  total_TBSPPA=0
  tbox= [0] * recursos_disponibles[0]

  #se inicializa por cada tipo de recurso una lista de listas que contendrá tantas sublistas como recursos de cada tipo haya
  # y contendra por cada uno el tiempo de inicio, finalizacion y paciente que lo utilza
  t_P=lista_recursos(recursos_disponibles[1]) #Ejemplo: t_P=[[[0,0,pac],[0,14,pac]][[0,0,pac],[14,18,pac]]
  t_N=lista_recursos(recursos_disponibles[2])
  t_PA=lista_recursos(recursos_disponibles[3])
  t_LAB=lista_recursos(recursos_disponibles[4])
  t_RAY=lista_recursos(recursos_disponibles[5])
  t_SCAN=lista_recursos(recursos_disponibles[6])
  t_BOX=lista_recursos(recursos_disponibles[0])
  actividades_inicio_fin=[]

  tipos_recursos = {'BOX': t_BOX,'P': t_P, 'N': t_N, 'PA': t_PA, 'LAB': t_LAB, 'RAY': t_RAY, 'SCAN': t_SCAN}
  n=0
  for i in range(len(recursos_ocupados)):
    tipo = recursos_ocupados[i][1]
    if tipo=='BOX':
      tbox[n ]=1000
```

```
            n+=1
        else:
            for j in range(recursos_ocupados[i][2]): #numero de recursos que necesitan ocuparse
                for k in range(len(tipos_recursos[tipo])):
                    if len(tipos_recursos[tipo][k])<2:
                        tipos_recursos[tipo][k].append([0, recursos_ocupados[i][0], recursos_ocupados[i][3]])
                        break

    #calculamos tiempo de inicio, fin y recurso para cada paciente utilizado según cada actividad para calcular el gantt
    for i,paciente in enumerate(Pacientes):
        t_activity=min(tbox)
        posicion_minimo=tbox.index(t_activity)
        count=0
        inicio_box=t_activity
        for rec in recursos_ocupados:
            if rec[1]=='BOX':
                count+=1
                if rec[3]==paciente[0]:
                    t_activity=0
                    posicion_minimo=count-1
                    inicio_box=t_activity
            elif rec[3]==paciente[0]:
                t_activity=rec[0]
                LOS_inicio=rec[0]


        indice_paciente=int(paciente[0][3:])-1 #sacamos el indice del paciente para la matriz de tiempos
        actividades_por_paciente=Tiempos[indice_paciente] #extraemos las actividades de la matriz de tiempos de ese paciente
        actividades_inicio_fin_por_paciente=[]

        if tiempos_esperados[indice_paciente] != 0: #si son pacientes que acaban de llegar o que llevaban tiempo esperando
            TBSPPA=tiempos_esperados[indice_paciente]+t_activity
        else:
            TBSPPA=tiempos_esperados[indice_paciente]

        for act in actividades_por_paciente:
            if act[1]=='P':
                pos_actividad=calculate_time_by_resource(t_P,act,t_activity) #se llama a una función que calcula el tiempo en el que se libera el recurs necesitado, pos_actividad=[recurso,pos,tinit,tfin,pac]
                for posicion in pos_actividad:
                    nuevo_valor=[posicion[2],posicion[3],paciente[0]]
                    if posicion[1]>=len(t_P[posicion[0]]):
                        t_P[posicion[0]].append(nuevo_valor)
                    else:
                        t_P[posicion[0]].insert(posicion[1],nuevo_valor)
            elif act[1]=='N':
                pos_actividad=calculate_time_by_resource(t_N,act,t_activity)
```

```
            for posicion in pos_actividad:
                nuevo_valor=[posicion[2],posicion[3],paciente[0]]
                if posicion[1]>=len(t_N[posicion[0]]):
                    t_N[posicion[0]].append(nuevo_valor)
                else:
                    t_N[posicion[0]].insert(posicion[1],nuevo_valor)
        elif act[1]=='PA':
            pos_actividad=calculate_time_by_resource(t_PA,act,t_activity) #se llama a una función que calcula el tiempo en el que se libera el recurso necesitado, pos_actividad=[[recurso,pos,tinit,tfin]]
            for posicion in pos_actividad:
                nuevo_valor=[posicion[2],posicion[3],paciente[0]]
                if posicion[1]>=len(t_PA[posicion[0]]):
                    t_PA[posicion[0]].append(nuevo_valor)
                else:
                    t_PA[posicion[0]].insert(posicion[1],nuevo_valor)
        elif act[1]=='LAB':
            pos_actividad=calculate_time_by_resource(t_LAB,act,t_activity)
            for posicion in pos_actividad:
                nuevo_valor=[posicion[2],posicion[3],paciente[0]]
                if posicion[1]>=len(t_LAB[posicion[0]]):
                    t_LAB[posicion[0]].append(nuevo_valor)
                else:
                    t_LAB[posicion[0]].insert(posicion[1],nuevo_valor)
        elif act[1]=='RAY':
            pos_actividad=calculate_time_by_resource(t_RAY,act,t_activity)
            for posicion in pos_actividad:
                nuevo_valor=[posicion[2],posicion[3],paciente[0]]
                if posicion[1]>=len(t_RAY[posicion[0]]):
                    t_RAY[posicion[0]].append(nuevo_valor)
                else:
                    t_RAY[posicion[0]].insert(posicion[1],nuevo_valor)
        elif act[1]=='SCAN':
            pos_actividad=calculate_time_by_resource(t_SCAN,act,t_activity)
            for posicion in pos_actividad:
                nuevo_valor=[posicion[2],posicion[3],paciente[0]]
                if posicion[1]>=len(t_SCAN[posicion[0]]):
                    t_SCAN[posicion[0]].append(nuevo_valor)
                else:
                    t_SCAN[posicion[0]].insert(posicion[1],nuevo_valor)
        t_activity=posicion[3]

    if len(actividades_por_paciente)>0:
        LOS=posicion[3]+tiempos_esperados[indice_paciente]
        fin_box=posicion[3]
    else: #si solo hay una actividad en el PU del paciente
        LOS=LOS_inicio
        fin_box=LOS_inicio


    t_BOX[posicion_minimo].append([inicio_box, fin_box, paciente[0]])
```

```
        actividades_inicio_fin.append(actividades_inicio_fin_por_paciente) #recoge los procesos de urgencia de todos los pacientes
        tbox[posicion_minimo]=fin_box

        TBSPPA_penalizado=Penalizacion_por_tiempo_esperado(TBSPPA,paciente)#penalizamos si el paciente ha estado esperando más del TBSPPA max

        total_LOS+=LOS
        total_TBSPPA+=TBSPPA_penalizado

    funcion_objetivo=total_LOS+total_TBSPPA #buscamos min los LOS totales y los TBSPPA penalizado


    return funcion_objetivo

def Penalizacion_por_tiempo_esperado(waiting_time,paciente):
    """función que penaliza el tiempo de espera de los pacientes"""

    tiempo_max_TBSPPA=[0,15,60,100,120] #tiempo máximo en el que el paciente debe ser atendido en la primera consulta facultatica [P1,P2,P3,P4,P5]
    prior=paciente[1]

    if tiempo_max_TBSPPA[prior-1]-waiting_time<0: #si ha estado esperando más de lo máximo se penaliza
        penalizacion= waiting_time*(6-prior)
    else:
        penalizacion=0

    return penalizacion

def lista_recursos(numero_recursos):
    """función que inicializa tantas sublistas como recursos de cada tipo haya"""

    lista_recursos = [[[0, 0, 0]] for _ in range(numero_recursos)]
    return lista_recursos

def calculate_time_by_resource(t_recursos,act,t_activity):
    """"calcula el momento en el que se libera el recurso necesitado.
    Recibe como argumentos el array con la ocupación de cada tipo de recurso hasta el momento, la actividad (duración,recurso necesario,numero de recursos necesarios),
    el tiempo de finalización de la actividad anterior y el paciente.
    Devuelve el momento de finalización de la actividad, el nuevo_valor (tinit,tfin,paciente),el recurso en caso de que haya mas de uno del mismo tipo y la posicion
    t_recursos=[[[0, 0, 0], [14, 57, 'Pac2'], [57, 124, 'Pac1']], [[0, 0, 0], [13, 33, 'Pac3'], [55, 72, 'Pac3'], [93, 115, 'Pac2'], [160, 193, 'Pac1']]] cada sublista es un recurso del mismo tipo que contiene
    el momento de inicio, el de finalizacion y el paciente que requiere ese recurso"""
    posicion_huecos=[] #almacena la posición en la que se encontraría el hueco
    posicion_actividad=[]

    for recurso in range(len(t_recursos)):
```

```
    for i in range(len(t_recursos[recurso])):
        t_hueco_por_recurso = t_recursos[recurso][i][0] - t_recursos[recurso][i-1][1] #tiempo de inicio de utilización del recurso anterior menos el final de la utilizacion del mismo recurso posterior
        t_hueco_por_actividad=t_recursos[recurso][i][0] -t_activity #tiempo de inicio de utilización del recurso menos el final de la actividad anterior del proceso
        t_hueco=min(t_hueco_por_recurso,t_hueco_por_actividad) #de ambos el hueco disponible será el más restrictivo
        t_inicio_hueco=t_recursos[recurso][i-1][1] #el hueco empieza cuando termina la actividad anterior de ese recurso
        t_final_hueco=t_recursos[recurso][i][0] #el hueco termina cuando empieza la actividad
        if t_hueco >= act[0]: #si el hueco es mayor que la duracion de la actividad que se evalua se toma esa posición
            posicion = (recurso, i,t_inicio_hueco,t_final_hueco)
            posicion_huecos.append(posicion)
    #haya o no huecos entre actividades, se añade también un hueco al final del array
    posicion_huecos.append((recurso, len(t_recursos[recurso]),t_recursos[recurso][-1][1],float('inf'))) #(posicion recurso en el array,indice posicion actividad,tfin hueco por la ultima act del array),tinicio hueco que será infinito porque no hay más actividades

  """Dependiendo del numero de recursos del mismo tipo que se necesiten se hallará la pos de la actividad de diferente forma"""
  posicion_mas_cercana = min(posicion_huecos, key=lambda pos: pos[2]) #se selecciona la posicion en la que se colocará el recurso cogiendo aquella en la que la act anterior que finalice antes
  posicion_huecos.remove(posicion_mas_cercana)
  if act[2] ==2: #si se necesitan dos recursos o más
    pos_valida=[]
    for _ in range(len(posicion_huecos)):
      for hueco in posicion_huecos:
        if posicion_mas_cercana[2]<=hueco[3] and posicion_mas_cercana[3]>=hueco[2]: #si el momento de inicio del hueco es menor que el de finalizacion de otro y el de final mayor que el de inicio de otro
          tinicio_hueco_doble=max(posicion_mas_cercana[2],hueco[2])
          tfin_hueco_doble=min(posicion_mas_cercana[3],hueco[3])
          t_hueco_doble=tfin_hueco_doble-tinicio_hueco_doble
          if t_hueco_doble>=act[0]:
            tinit=max(t_activity,tinicio_hueco_doble)
            pos_2=(hueco[0], hueco[1],tinit,tinit+act[0])
            pos_valida.append(pos_2)
      if pos_valida: #si hay un hueco gemelo válido en esa posicion paro de buscar
        break
      else: #si no actualizo la posicion mas cercana con la siguiente mas cercana de la lista de huecos
        posicion_mas_cercana=min(posicion_huecos, key=lambda pos: pos[2])
        posicion_huecos.remove(posicion_mas_cercana)


    posicion_mas_cercana2 = min(pos_valida, key=lambda pos: (pos[2],pos[1]))
    pos_1=(posicion_mas_cercana[0],
posicion_mas_cercana[1],posicion_mas_cercana2[2],posicion_mas_cercana2[3])
    posicion_actividad.append(posicion_mas_cercana2)
    posicion_actividad.append(pos_1)
```

```
    else: #si solo necesita un recurso
        recurso, pos,t_inicio_hueco,t_final_hueco = posicion_mas_cercana
        tiempo_recurso=t_recursos[recurso][pos-1][1]
        tinit=max(t_activity,tiempo_recurso) #calculamos el momento en el que empieza la actividad
        tfin=tinit+act[0] #calculamos tiempo finalizacion
        pos=(recurso,pos,tinit,tfin)
        posicion_actividad.append(pos)

    return posicion_actividad
```

Gantt.py

```
import matplotlib.pyplot as plt

def calculate_Gantt(Pacientes,Tiempos,recursos_disponibles,recursos_ocupados):
    """esta función es gemela a la función evaluate_solution, la unica diferencia es que en este caso devuelve los datos necesarios para dibujar el gantt,
    es decir, devuelve un array por cada tipo de recursos que recoge los tiempos de inicio, fin y paciente que utiliza cada uno"""

    tbox= [0] * recursos_disponibles[0]
    #se inicializa por cada tipo de recurso una lista de listas que contendrá tantas sublistas como recursos de cada tipo haya
    # y contendra por cada uno el tiempo de inicio, finalizacion y paciente que lo utilza
    t_P=lista_recursos(recursos_disponibles[1]) #Ejemplo: t_P=[[[0,0,pac],[0,14,pac]][[0,0,pac],[14,18,pac]]
    t_N=lista_recursos(recursos_disponibles[2])
    t_PA=lista_recursos(recursos_disponibles[3])
    t_LAB=lista_recursos(recursos_disponibles[4])
    t_RAY=lista_recursos(recursos_disponibles[5])
    t_SCAN=lista_recursos(recursos_disponibles[6])
    t_BOX=lista_recursos(recursos_disponibles[0])
    actividades_inicio_fin=[]

    tipos_recursos = {'BOX':t_BOX,'P': t_P, 'N': t_N, 'PA': t_PA, 'LAB': t_LAB, 'RAY': t_RAY, 'SCAN': t_SCAN}

    n=0
    for i in range(len(recursos_ocupados)):
        tipo = recursos_ocupados[i][1]
        if tipo=='BOX':
            tbox[n]=1000
            n+=1
        else:
            for j in range(recursos_ocupados[i][2]): #numero de recursos que necesitan ocuparse
                for k in range(len(tipos_recursos[tipo])):
                    if len(tipos_recursos[tipo][k])<2:
                        tipos_recursos[tipo][k].append([0, recursos_ocupados[i][0], recursos_ocupados[i][3]])
                        break
```

```
  #calculamos tiempo de inicio, fin y recurso para cada paciente utilizado según cada actividad para calcular el gantt
  for i,paciente in enumerate(Pacientes):
    t_activity=min(tbox)
    posicion_minimo=tbox.index(t_activity)
    count=0
    inicio_box=t_activity
    for rec in recursos_ocupados:
      if rec[1]=='BOX':
        count+=1
        if rec[3]==paciente[0]:
          t_activity=0
          posicion_minimo=count-1
          inicio_box=0
      elif rec[3]==paciente[0]:
        t_activity=rec[0]
        LOS_inicio=rec[0]
        inicio_box=0

    indice_paciente=int(paciente[0][3:])-1 #sacamos el indice del paciente para la matriz de tiempos
    actividades_por_paciente=Tiempos[indice_paciente] #extraemos las actividades de la matriz de tiempos de ese paciente
    actividades_inicio_fin_por_paciente=[]


    for act in actividades_por_paciente:
      if act[1]=='P':
        pos_actividad=calculate_time_by_resource(t_P,act,t_activity) #se llama a una función que calcula el tiempo en el que se libera el recurs necesitado, pos_actividad=[recurso,pos,tinit,tfin,pac]
        for posicion in pos_actividad:
          nuevo_valor=[posicion[2],posicion[3],paciente[0]]
          if posicion[1]>=len(t_P[posicion[0]]):
            t_P[posicion[0]].append(nuevo_valor)
          else:
            t_P[posicion[0]].insert(posicion[1],nuevo_valor)
      elif act[1]=='N':
        pos_actividad=calculate_time_by_resource(t_N,act,t_activity)
        for posicion in pos_actividad:
          nuevo_valor=[posicion[2],posicion[3],paciente[0]]
          if posicion[1]>=len(t_N[posicion[0]]):
            t_N[posicion[0]].append(nuevo_valor)
          else:
            t_N[posicion[0]].insert(posicion[1],nuevo_valor)
      elif act[1]=='PA':
        pos_actividad=calculate_time_by_resource(t_PA,act,t_activity) #se llama a una función que calcula el tiempo en el que se libera el recurs necesitado, pos_actividad=[[recurso,pos,tinit,tfin]]
        for posicion in pos_actividad:
          nuevo_valor=[posicion[2],posicion[3],paciente[0]]
          if posicion[1]>=len(t_PA[posicion[0]]):
            t_PA[posicion[0]].append(nuevo_valor)
```

```
                else:
                    t_PA[posicion[0]].insert(posicion[1],nuevo_valor)
            elif act[1]=='LAB':
                pos_actividad=calculate_time_by_resource(t_LAB,act,t_activity)
                for posicion in pos_actividad:
                    nuevo_valor=[posicion[2],posicion[3],paciente[0]]
                    if posicion[1]>=len(t_LAB[posicion[0]]):
                        t_LAB[posicion[0]].append(nuevo_valor)
                    else:
                        t_LAB[posicion[0]].insert(posicion[1],nuevo_valor)
            elif act[1]=='RAY':
                pos_actividad=calculate_time_by_resource(t_RAY,act,t_activity)
                for posicion in pos_actividad:
                    nuevo_valor=[posicion[2],posicion[3],paciente[0]]
                    if posicion[1]>=len(t_RAY[posicion[0]]):
                        t_RAY[posicion[0]].append(nuevo_valor)
                    else:
                        t_RAY[posicion[0]].insert(posicion[1],nuevo_valor)
            elif act[1]=='SCAN':
                pos_actividad=calculate_time_by_resource(t_SCAN,act,t_activity)
                for posicion in pos_actividad:
                    nuevo_valor=[posicion[2],posicion[3],paciente[0]]
                    if posicion[1]>=len(t_SCAN[posicion[0]]):
                        t_SCAN[posicion[0]].append(nuevo_valor)
                    else:
                        t_SCAN[posicion[0]].insert(posicion[1],nuevo_valor)
            t_activity=posicion[3]
        if len(actividades_por_paciente)>0:
            LOS=posicion[3]
        else:
            LOS=LOS_inicio
        t_BOX[posicion_minimo].append([inicio_box, LOS, paciente[0]])
        actividades_inicio_fin.append(actividades_inicio_fin_por_paciente) #recoge los procesos de urgencia de todos
los pacientes
        tbox[posicion_minimo]=LOS



    return t_P,t_PA,t_N,t_RAY,t_LAB,t_SCAN,t_BOX


def diagrama_de_gantt(Pacientes,t_P,t_PA,t_N,t_RAY,t_LAB,t_SCAN,t_BOX,recursos_disponibles):
    """Esta función es la encargada de mostrar el diagrama de Gantt de la secuenciación de las tareas de los
recursos"""

    fig, ax = plt.subplots()

    # Lista de colores disponibles
    colores_disponibles = ['red', 'blue', 'green', 'orange', 'yellow', 'purple', 'pink', 'brown', 'black',
    'gray', 'cyan', 'magenta', 'lime', 'teal', 'indigo', 'maroon', 'gold', 'silver', 'olive',
```

```
'navy', 'aquamarine', 'coral', 'sienna', 'violet', 'turquoise', 'salmon', 'orchid', 'khaki',
'lavender', 'crimson', 'fuchsia', 'chartreuse', 'beige', 'peru', 'thistle', 'cyan', 'slategray',
'darkorange', 'royalblue', 'darkgreen', 'deeppink', 'burlywood', 'indianred', 'midnightblue',
'darkviolet', 'darkslategray', 'lightcoral', 'powderblue', 'darkkhaki', 'palegreen', 'seashell',
'cadetblue', 'darkorchid', 'mediumpurple', 'orangered', 'darkolivegreen', 'limegreen', 'deepskyblue',
'darkturquoise', 'saddlebrown', 'tomato', 'dodgerblue', 'lightseagreen', 'hotpink', 'forestgreen',
'darkgoldenrod', 'turquoise', 'springgreen', 'darkslateblue', 'darkred', 'greenyellow', 'plum',
'lightpink', 'slateblue', 'mediumslateblue', 'firebrick', 'chocolate', 'darkcyan', 'lavenderblush',
'tan', 'oldlace', 'rosybrown', 'papayawhip', 'linen', 'mistyrose', 'aliceblue', 'ghostwhite',
'whitesmoke', 'honeydew', 'mintcream', 'azure', 'lightcyan', 'lightyellow', 'lightgoldenrodyellow']


patient_colors = {}
# Asignar colores a los pacientes
for i, paciente in enumerate(Pacientes):
    if i < len(colores_disponibles):
        color_asignado = colores_disponibles[i]
    else:
        # Si se acaban los colores reiniciamos la lista
        color_asignado = colores_disponibles[i % len(colores_disponibles)]
    # Asignar el color al paciente en el diccionario
    patient_colors[paciente[0]] = color_asignado

count=0
for i, resource_data in enumerate(t_BOX):
    for activity in resource_data[1:]:
        start_time, end_time, patient = activity
        duration = end_time - start_time
        ax.barh(count+i, duration, left=start_time, height=0.5, color=patient_colors[patient])
count=len(t_BOX)
for i, resource_data in enumerate(t_P):
    for activity in resource_data[1:]:
        start_time, end_time, patient = activity
        duration = end_time - start_time
        ax.barh(count+i, duration, left=start_time, height=0.5, color=patient_colors[patient])

count=len(t_BOX)+len(t_P)

for i, resource_data in enumerate(t_N):
    for activity in resource_data[1:]:
        start_time, end_time, patient = activity
        duration = end_time - start_time
        ax.barh(count+i, duration, left=start_time, height=0.5, color=patient_colors[patient])

count=len(t_BOX)+len(t_P)+len(t_N)

for i, resource_data in enumerate(t_PA):
    for activity in resource_data[1:]:
        start_time, end_time, patient = activity
        duration = end_time - start_time
```

```
        ax.barh(count+i, duration, left=start_time, height=0.5, color=patient_colors[patient])



count=len(t_BOX)+len(t_P)+len(t_N)+len(t_PA)
for i, resource_data in enumerate(t_LAB):
    for activity in resource_data[1:]:
        start_time, end_time, patient = activity
        duration = end_time - start_time
        ax.barh(count+i, duration, left=start_time, height=0.5, color=patient_colors[patient])


count=len(t_BOX)+len(t_P)+len(t_N)+len(t_PA)+len(t_LAB)
for i, resource_data in enumerate(t_RAY):
    for activity in resource_data[1:]:
        start_time, end_time, patient = activity
        duration = end_time - start_time
        ax.barh(count+i, duration, left=start_time, height=0.5, color=patient_colors[patient])

count=len(t_BOX)+len(t_P)+len(t_N)+len(t_PA)+len(t_LAB)+len(t_RAY)

for i, resource_data in enumerate(t_SCAN):
    for activity in resource_data[1:]:
        start_time, end_time, patient = activity
        duration = end_time - start_time
        ax.barh(count+i, duration, left=start_time, height=0.5, color=patient_colors[patient])


nombres_recursos=['BOX','P','N','PA','LAB', 'RAY', 'SCAN']
y_labels=[]

for i, cantidad in enumerate(recursos_disponibles):
    for j in range(1, cantidad+1):
        y_labels.append(f'{nombres_recursos[i]}{j}')


y_positions = list(range(len(y_labels)))

legend_labels = [f'{paciente}' for paciente in patient_colors.keys()]
legend_handles = [plt.Rectangle((0, 0), 1, 1, color=patient_colors[patient]) for patient in patient_colors.keys()]
ax.legend(legend_handles, legend_labels, loc='upper right', title='Pacientes')
ax.set_yticks(y_positions)
ax.set_yticklabels(y_labels)

ax.invert_yaxis()

ax.set_xlabel('Tiempo')
ax.set_title('Diagrama de Gantt')
plt.grid(axis='x')
```

```
    plt.show()

def lista_recursos(numero_recursos):
    """función que inicializa tantas sublistas como recursos de cada tipo haya"""

    lista_recursos = [[[0, 0, 0]] for _ in range(numero_recursos)]
    return lista_recursos

def calculate_time_by_resource(t_recursos,act,t_activity):
    """"calcula el momento en el que se libera el recurso necesitado.
    Recibe como argumentos el array con la ocupación de cada tipo de recurso hasta el momento, la actividad (duración,recurso necesario,numero de recursos necesarios),
    el tiempo de finalización de la actividad anterior y el paciente.
    Devuelve el momento de finalización de la actividad, el nuevo_valor (tinit,tfin,paciente),el recurso en caso de que haya mas de uno del mismo tipo y la posicion
    t_recursos=[[[[0, 0, 0], [14, 57, 'Pac2'], [57, 124, 'Pac1']], [[0, 0, 0], [13, 33, 'Pac3'], [55, 72, 'Pac3'], [93, 115, 'Pac2'], [160, 193, 'Pac1']]] cada sublista es un recurso del mismo tipo que contiene
    el momento de inicio, el de finalizacion y el paciente que requiere ese recurso"""
    posicion_huecos=[] #almacena la posición en la que se encontraría el hueco
    posicion_actividad=[]

    for recurso in range(len(t_recursos)):
        for i in range(len(t_recursos[recurso])):
            t_hueco_por_recurso = t_recursos[recurso][i][0] - t_recursos[recurso][i-1][1] #tiempo de inicio de utilización del recurso anterior menos el final de la utilizacion del mismo recurso posterior
            t_hueco_por_actividad=t_recursos[recurso][i][0] -t_activity #tiempo de inicio de utilización del recurso menos el final de la actividad anterior del proceso
            t_hueco=min(t_hueco_por_recurso,t_hueco_por_actividad) #de ambos el hueco disponible será el más restrictivo
            t_inicio_hueco=t_recursos[recurso][i-1][1] #el hueco empieza cuando termina la actividad anterior de ese recurso
            t_final_hueco=t_recursos[recurso][i][0] #el hueco termina cuando empieza la actividad
            if t_hueco >= act[0]: #si el hueco es mayor que la duracion de la actividad que se evalua se toma esa posición
                posicion = (recurso, i,t_inicio_hueco,t_final_hueco)
                posicion_huecos.append(posicion)
        #haya o no huecos entre actividades, se añade también un hueco al final del array
        posicion_huecos.append((recurso, len(t_recursos[recurso]),t_recursos[recurso][-1][1],float('inf'))) #(posicion recurso en el array,indice posicion actividad,tfin hueco por la ultima act del array),tinicio hueco que será infinito porque no hay más actividades

    """Dependiendo del numero de recursos del mismo tipo que se necesiten se hallará la pos de la actividad de diferente forma"""
    posicion_mas_cercana = min(posicion_huecos, key=lambda pos: pos[2]) #se selecciona la posicion en la que se colocará el recurso cogiendo aquella en la que la act anterior que finalice antes
    posicion_huecos.remove(posicion_mas_cercana)
    if act[2] ==2: #si se necesitan dos recursos
        pos_valida=[]
        for _ in range(len(posicion_huecos)):
            for hueco in posicion_huecos:
```

```
            if posicion_mas_cercana[2]<=hueco[3] and posicion_mas_cercana[3]>=hueco[2]: #si el momento de inicio
del hueco es menor que el de finalizacion de otro y el de final mayor que el de inicio de otro
                tinicio_hueco_doble=max(posicion_mas_cercana[2],hueco[2])
                tfin_hueco_doble=min(posicion_mas_cercana[3],hueco[3])
                t_hueco_doble=tfin_hueco_doble-tinicio_hueco_doble
                if t_hueco_doble>=act[0]:
                    tinit=max(t_activity,tinicio_hueco_doble)
                    pos_2=(hueco[0], hueco[1],tinit,tinit+act[0])
                    pos_valida.append(pos_2)
        if pos_valida: #si hay un hueco gemelo válido en esa posicion paro de buscar
            break
        else: #si no actualizo la posicion mas cercana con la siguiente mas cercana de la lista de huecos
            posicion_mas_cercana=min(posicion_huecos, key=lambda pos: pos[2])
            posicion_huecos.remove(posicion_mas_cercana)

    posicion_mas_cercana2 = min(pos_valida, key=lambda pos: (pos[2],pos[1]))
    pos_1=(posicion_mas_cercana[0],
posicion_mas_cercana[1],posicion_mas_cercana2[2],posicion_mas_cercana2[3])
    posicion_actividad.append(posicion_mas_cercana2)
    posicion_actividad.append(pos_1)

  else: #si solo necesita un recurso
    recurso, pos,t_inicio_hueco,t_final_hueco = posicion_mas_cercana
    tiempo_recurso=t_recursos[recurso][pos-1][1]
    tinit=max(t_activity,tiempo_recurso) #calculamos el momento en el que empieza la actividad
    tfin=tinit+act[0] #calculamos tiempo finalizacion
    pos=(recurso,pos,tinit,tfin)
    posicion_actividad.append(pos)

  return posicion_actividad
```