Trabajo Fin de Grado

Ingeniería de las Tecnologías Industriales

Optimización quirúrgica para maximización de pacientes operados mediante algoritmos de evolución diferencial

Autor: Agustín Rascón Castañeda

Tutor: Víctor Fernández-Viagas Escudero

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

Sevilla, 2024

Trabajo Fin de Grado
Ingeniería de las Tecnologías Industriales

Optimización quirúrgica para maximización de pacientes operados mediante algoritmos de evolución diferencial

Autor:

Agustín Rascón Castañeda

Tutor:

Víctor Fernández-Viagas Escudero

Profesor titular

Dpto. de Organización Industrial y Gestión de Empresas I

Escuela Técnica Superior de Ingeniería

Universidad de Sevilla

Sevilla, 2024

Trabajo de Fin de Grado: Optimización quirúrgica para maximización de pacientes operados mediante algoritmos de evolución diferencial

Autor: Agustín Rascón Castañeda

Tutor: Víctor Fernández-Viagas Escudero

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

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2024

El Secretario del Tribunal

Agradecimientos

A mi madre y padre, por la educación, la paciencia y el inquebrantable apoyo que siempre me han brindado para poder lograr todas mis metas.

A mi abuela, cuyo orgullo por mi progreso ha sido el motor que me ha impulsado durante estos años.

A mi pareja y compañera de vida, quien me ha acompañado de cerca durante este camino y día tras día ha sido la fuerza que me ha alentado a nunca rendirme.

A mis amigos, con quienes he crecido, compartiendo ilusiones, ambiciones, dudas ,momentos, risas y alegrías.

Por último, a mi tutor, por su constante implicación en enseñarme y guiarme a lo largo de estos meses.

Gracias.

Agustín Rascón Castañeda

Sevilla, 2024

Resumen

El objetivo de este Trabajo de Fin de Grado es abordar el problema real de las crecientes listas de espera que enfrentan los pacientes en el sistema sanitario, específicamente en un hospital de referencia a nivel andaluz.

Este problema se resuelve mediante métodos de optimización, en los que se busca que los pacientes en lista de espera, y su cirujano, se asignen a los quirófanos disponibles, en la fecha que serán operados. La planificación, se realiza con el objetivo de maximizar el número de pacientes operados durante el horizonte temporal de una semana, buscando disminuir las listas de espera, aumentando la eficiencia y rendimiento de los recursos disponibles. Para ello, se desarrolla el algoritmo metaheurístico de Evolución Diferencial y variaciones de este. Todos los algoritmos se ejecutarán ante una batería de instancias que simulan los recursos de los que dispone la unidad sanitaria, obteniendo así los resultados de cada algoritmo y el mejor método para obtener la secuencia en la que los pacientes en lista de espera deben ser operados para maximizar las operaciones quirúrgicas.

El desarrollo de los algoritmos se realiza mediante programación en Python, usando Excel para almacenar datos de manera auxiliar.

Abstract

The aim of this paper is to address the significant issue of increasing waiting lists experienced by patients within the healthcare system, specifically in a major hospital at the regional level in Andalusia.

The problem is solved using optimization methods, where the goal is to assign patients on the waiting list, along with their surgeons, to the available operating rooms on the date they are to be operated on. he planning is done with the aim of maximizing the number of patients operated on within a one-week time horizon, seeking to reduce waiting lists by increasing the efficiency and performance of available resources. To achieve this, the Differential Evolution metaheuristic algorithm and its variations are developed. All algorithms will be tested against a set of instances that simulate the resources available to the healthcare unit, thus obtaining the results of each algorithm and determining the best method for scheduling the sequence in which waiting list patients should be operated on to maximize surgical operations.

The development of the algorithms is done using Python programming, with Excel employed to store data as an auxiliary tool.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

1.1 Programación de la producción

La programación de la producción se puede entender como el proceso de toma de decisiones basadas en un análisis previo para establecer un conjunto de operaciones que permitan realizar un producto final a partir de los recursos disponibles (Framiñan et al., 2014). Todo proyecto, de producción o servicio, se basa en este proceso, que busca ser lo más eficiente posible para alcanzar los objetivos establecidos, minimizando los costes y maximizando el uso de los recursos.

En este proceso productivo hay que reconocer unos conceptos básicos (Framiñan et al., 2014):

- Las unidades o productos en los que la actividad se puede dividir se conocen como "trabajos", el cual es completamente dependiente del contexto en el que se sitúe el proceso.
- Los recursos, los cuales en el ámbito de la manufactura son reconocidos como "máquinas", y los cuales forman parte de las operaciones para transformar o actuar sobre los trabajos. Su forma también varía en función del ambiente que se encuentren.
- El programa de la producción o "schedule" resulta de la asignación de las máquinas para la transformación en un conjunto de trabajos establecido en una escala temporal, tal y como se muestra en la ***Figura 1-1***,y será el principal fin del problema (Framiñan et al., 2014). Esto es diferente a la "secuencia" la cual corresponde a una permutación de los trabajos en el orden que deben ser procesados en las máquinas (Pinedo, 2016).

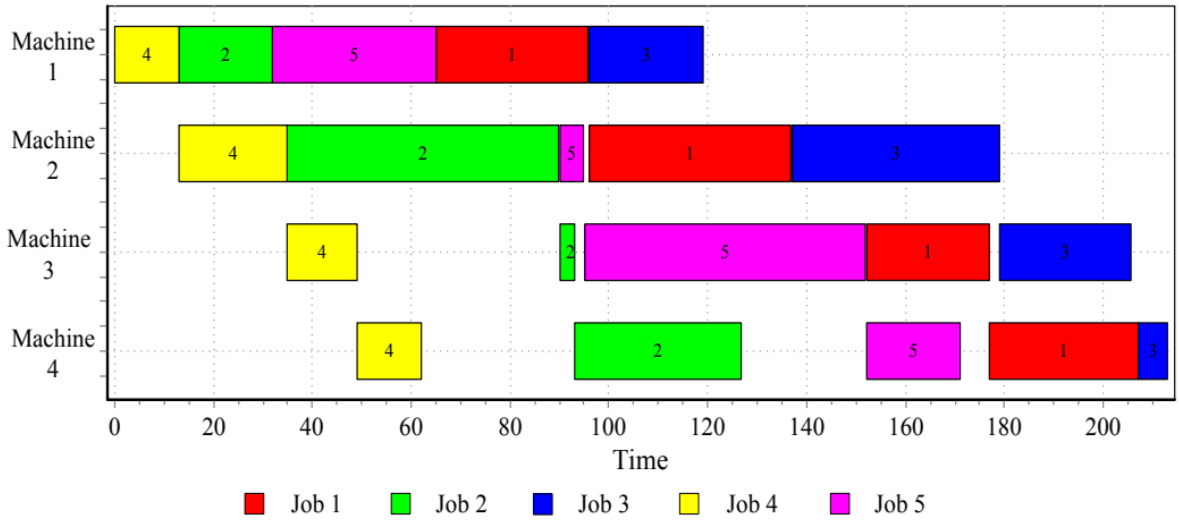

Figura 1-1*: Diagrama de Gantt* (Framiñan et al., 2014)

Un problema de programación de la producción se puede dividir en tres principales parámetros: El entorno de la máquina, las características del proceso y sus restricciones y el objetivo a optimizar (Pinedo, 2016).

Cabe destacar el dinamismo y variabilidad de los procesos productivos, caracterizados por varios factores, como las propiedades y necesidades de cada máquina, el tipo de automatización establecido, el factor humano involucrado o las propiedades de distintos materiales (Pinedo, 2005). Todas estas posibles variables llevan a que cada situación sea única y necesite de su propia programación de la producción y de ahí, el constante crecimiento en la creación de modelos y algoritmos que permitan adaptarse a la realidad presentada.

1.1.1 Técnicas de optimización

La optimización de la programación determinista y metaheurística son estrategias distintas para poder resolver estos problemas. La optimización determinista es un enfoque en el cual se buscan soluciones óptimas para un problema dado, sin la influencia de la aleatoriedad. Sus componentes son predecibles, y no están sujetos a variabilidad estocástica, esto significa, que los resultados se obtienen de manera sistemática y reproducible (Acharya et al., 2022). Sin embargo, debido a la exhaustiva exploración de posibles soluciones, el tiempo de cómputo tiende a ser elevado, ya que implica evaluar cada opción de manera completa para encontrar la mejor solución posible (Acharya et al., 2022). Las metaheurísticas en cambio, no garantizan que la solución converja en el óptimo global, esto es debido a que la mayoría tienden a implementar algún tipo de optimización estocástica usando variables aleatorias para encontrar la solución, obteniendo un conjunto de soluciones factibles (Blum & Roli, 2003) y requiriendo por ello, de mucho menos esfuerzo computacional.

Es importante tener en cuenta en el proceso de la toma de decisiones, el equilibrio entre el valor resultante de obtener una solución óptima y el tiempo requerido para conseguirla. Es por ello que una de estas soluciones factibles, pese a no ser la óptima, puede conllevar mayores beneficios desde un punto de vista global. También es relevante considerar que existen variedad de problemas que no pueden ser resueltos con métodos exactos, las metaheurísticas son una opción muy útil para variedad de situaciones (Azhir et al., 2019).

Entre la multitud de técnicas de optimización metaheurísticas, muchas de ellas basadas en aspectos biológicos y naturales, cabe destacar la Evolución Diferencial, la cual pertenece a la familia de algoritmos genéticos y utiliza métodos poblacionales. Los algoritmos genéticos son modelos computacionales, inspirados en los principios moleculares y genéticos de la biología, y aunque sea un modelado simplificado, son capaces de evolucionar hacia complejas e interesantes estructuras (Forrest, 1996). Estas estructuras llamadas "individuos", representan la solución del problema. Al utilizar también inspiración en la teoría de herencia genética de la población, los individuos finales del algoritmo serán los más adecuados para las características y condiciones establecidas inicialmente en el problema. En la ***Figura 1-2***, se muestra el funcionamiento genérico de un algoritmo genético.

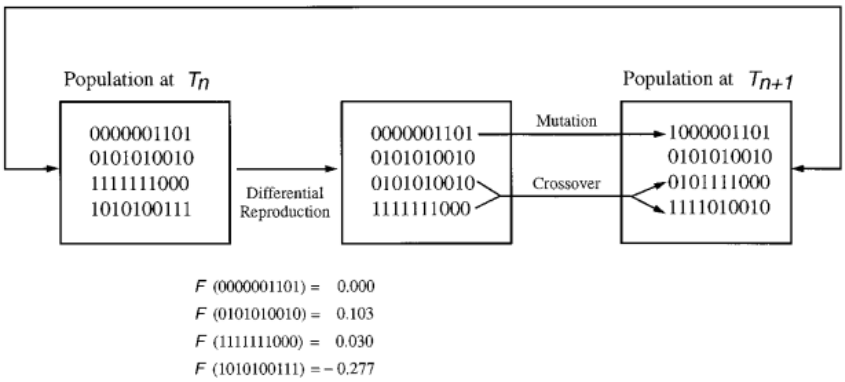

Figura 1-2: *Algoritmo genético* (Forrest, 1996).

Los algoritmos que emplean métodos poblacionales lo hacen debido a la estrecha relación entre herencia genética y población. En general, estos algoritmos comienzan con una población inicial conformada por individuos generados de manera aleatoria, o derivados de otros métodos. Esta población inicial es luego sometida a modificaciones, mutaciones y cruces entre individuos de la población, dando lugar a nuevos descendientes con pequeñas variaciones respecto a sus progenitores. Como resultado de estas variaciones, algunos individuos se destacan como soluciones más eficientes al problema que otros (Forrest, 1996). Este proceso se conoce como selección, en el cual se favorece a los individuos que representan una mejor solución mediante un sesgo predefinido. Estos individuos seleccionados son entonces integrados a la población, mientras que aquellos con un rendimiento inferior son descartados.

Con este ciclo de evaluación, selección, y operaciones genéticas, se van generando poblaciones de individuos, ligeramente distintos a sus progenitores y progresivamente mejores bajo el criterio de una función establecida que interese mejorar (Forrest, 1996). Existen múltiples de maneras de aplicar estos principios genéticos y poblacionales, muchas de ellas aplicando principios naturales y biológicos con conceptos estadísticos y combinatorios. Para este trabajo se destaca la Evolución Diferencial, cuyo diagrama de flujo genérico se muestra en la ***Figura 1-3***.

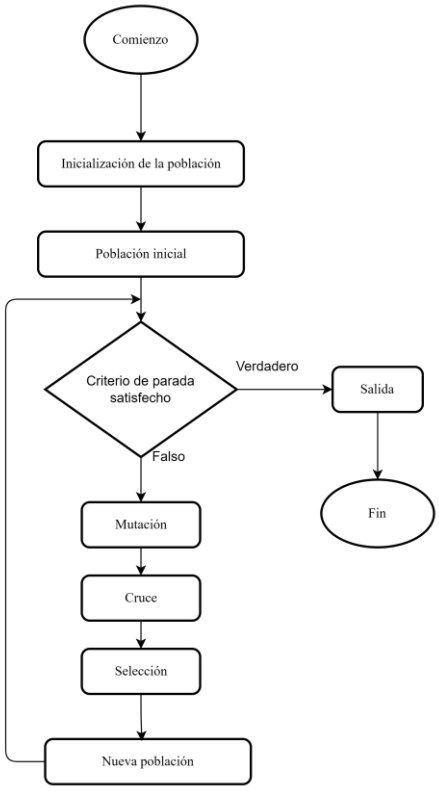

Figura 1-3: *Diagrama de flujo de un algoritmo de Evolución diferencial genérico.*

La Evolución Diferencial en lugar de perturbar las soluciones mediante vectores de diferencia escalados, como es común en otros algoritmos de la misma familia, obtiene soluciones generando descendientes a través de la recombinación de soluciones existentes, siempre que se cumplan ciertas condiciones, siendo sustituidas en la solución aquellos individuos que sean superados por algún descendiente (Ahmad et al., 2022). Es considerado uno de los algoritmos más robusto y simple, debido a que la búsqueda es regida por pocos parámetros como pueden ser el factor de escala o la ratio de cruce (Ahmad et al., 2022).

1.2 Contexto sanitario

La optimización de la producción es un campo vital en la gestión empresarial, donde constantemente se busca maximizar los recursos y minimizar los costes. Sin embargo, este desafío no es exclusivo de este ámbito, de hecho, se extiende a sectores cruciales como el sanitario, donde la eficacia en la asignación de recursos y la gestión del tiempo son fundamentales para garantizar la atención oportuna y adecuada a los pacientes. En este contexto, las técnicas de optimización adquieren una relevancia aún mayor, ya que pueden marcar la diferencia entre una atención médica eficiente y una que no lo sea. En este sentido, la gestión de las listas de espera emerge como un aspecto crítico. Reducir estas listas y aumentar la cantidad de pacientes operados no solo mejora la calidad de vida de quienes requieren atención médica, sino que también alivia la presión sobre los sistemas de salud.

Las listas de espera quirúrgicas indican el lapso que los pacientes aguardan desde que se les prescribe una cirugía hasta su realización, siendo uno de los principales desafíos en sistemas de salud de carácter público. Estas listas muestran un incremento anual en su extensión (***Figura 1-4***), principalmente por la creciente demanda de intervenciones quirúrgicas. Este aumento se origina por diversos factores:

- Un envejecimiento progresivo de la población, impulsado por un constante aumento en la esperanza de vida, ha sido señalado como un factor determinante (Ministerio de Sanidad, 2023). A medida que la edad avanza, se incrementa el número de patologías, generando así una mayor demanda de asistencia médica y quirúrgica (*Cirugía ambulatoria en pacientes mayores de 65 años: nuestra experiencia | Revista Española de Geriatría y Gerontología*, s. f.)
- Una normalización de las operaciones quirúrgicas por parte de la sociedad, la cual, en busca de una mayor calidad de vida y mediante su confianza en el sistema médico, está más dispuesta que nunca a someterse a cirugía (Barajas-Gamboa, 2019).
- El desarrollo de nuevos métodos de diagnósticos ha permitido la detección precoz de patologías y su tratamiento (Gerges et al., 2010; Ministerio de Sanidad, 2023). Esto lleva a la aparición de novedosas operaciones quirúrgicas y, por lo tanto, mayor demanda inducida por una nueva parte de la población.

Esta complejidad creciente y el incremento proporcional de sus costes, obligan a reestructurar la organización sanitaria, con el fin de adecuar los recursos a las demandas y garantizar el mantenimiento de la calidad asistencial, la eficiencia clínica y la sostenibilidad (Gómez-Ríos et al., 2019). Con ello, la optimización de la programación sanitaria ha recibido una mayor atención con el objetivo de dar un servicio a un coste asequible por parte del sistema médico, principalmente por el alto coste y limitación de recursos de los hospitales (Abdalkareem et al., 2021).

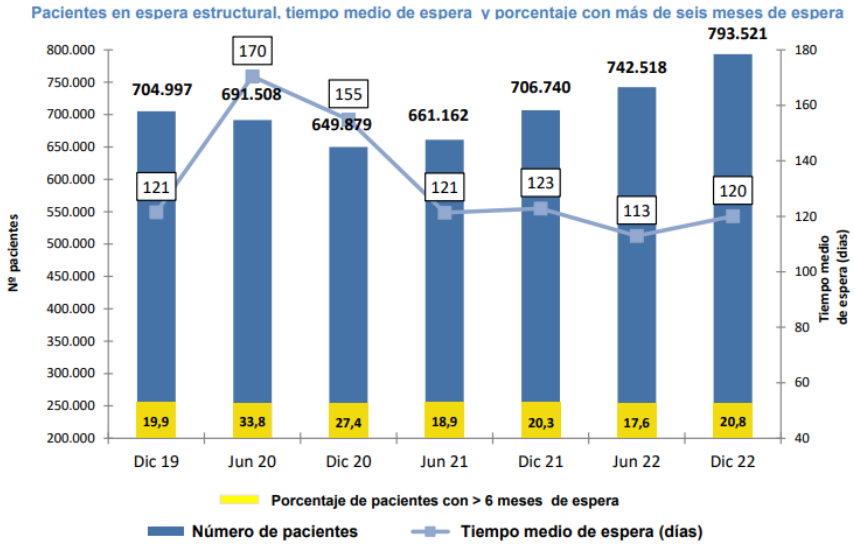

Figura 1-4: *Pacientes en lista de espera y tiempo medio de espera. (Ministerio de Sanidad, 2023)*

La complejidad de la planificación quirúrgica viene dada por el hecho de coordinar a multitud de profesionales, cuyos intereses a veces entran en conflicto, para obtener la máxima eficacia posible en búsqueda de objetivos comunes. Una diversidad de procedimientos, algunos urgente y otros programados, con prioridades relativas, dependencia de otras unidades sanitarias con su propia capacidad limitada, y la escasa disponibilidad de quirófanos, hacen tener a la planificación quirúrgica una gran trascendencia en la gestión sanitaria y sus parámetros de eficiencia (Gómez-Ríos et al., 2019). El rendimiento del quirófano, bajo el cual son juzgados durante la toma de decisiones de mayor jerarquía, depende en gran medida de la cantidad de casos realizados, en paralelo con la gestión de la lista de espera y la organización del personal.

La programación de operaciones quirúrgicas tiene la posibilidad de reducir los tiempos de espera y facilitar el acceso y calidad de los servicios médicos, de ahí el rol fundamental que toma para los hospitales que buscan mejorar su servicio y reducir sus costes (Abdalkareem et al., 2021). Una gestión eficaz de las operaciones quirúrgicas no solo mejora las listas de espera y la satisfacción de los pacientes, sino que también puede reducir el mayor coste hospitalario asociado a los procedimientos quirúrgicos: los salarios y los gastos relacionados con los quirófanos (Dexter et al., 1999). Es fundamental, por lo tanto, el establecimiento de reglas de programación quirúrgica, la generación de cronogramas para las operaciones y una asignación eficiente del personal involucrado.

En el contexto de la sanidad española, la gestión de los quirófanos ha experimentado una evolución significativa a lo largo del tiempo, siendo dividida en tres niveles de decisión principales: operacional, táctica y estratégica (Cardoen et al., 2010). La gestión estratégica se centra en planificar las variables relacionadas con la cantidad y el tipo de cirugías que se llevarán a cabo en un horizonte de planificación semanal (Molina-Pariente, Fernández-Viagas, et al., 2015). La gestión operacional se encarga de establecer el programa maestro, que incluye fechas, quirófanos y tiempos para cada cirugía. Este nivel se divide en dos etapas: la programación anticipada, que implica asignar un quirófano y una fecha a cada cirugía pendiente; y la programación de asignación, que implica definir una secuencia de cirugías para cada quirófano durante cada día del horizonte de planificación (J.M. & J.B., 1978).

Desde una falta de organización definida, hasta la instauración del cargo de director de quirófanos, este progreso es dirigido por la necesidad de mejorar la eficiencia operativa en los hospitales (Muñoz Alameda & Macario, 2017), de ahí, la importancia en la introducción de técnicas de optimización en la programación quirúrgica, que busca mantener la eficiencia del programa de operaciones ante diversos desafíos, como urgencias, cancelaciones y cambios de recursos y pacientes (Molina-Pariente, Hans, et al., 2015).

1.3 Alcance del trabajo

El presente Trabajo de Fin de Grado tiene como objetivo proponer un algoritmo capaz de resolver eficientemente el problema de las listas de espera en el sistema sanitario, para el hospital de referencia. Este proyecto se centra en la optimización de la asignación de pacientes y cirujanos a los quirófanos disponibles, con el fin de maximizar el número de operaciones realizadas en una semana como solución a las largas listas de espera.

Para llevar a cabo este proyecto, se desarrollarán distintas variaciones del algoritmo de Evolución Diferencial, las cuales se ejecutarán ante diversas instancias, que representan los recursos y limitaciones del hospital. Estas instancias permitirán evaluar la eficacia de los algoritmos desarrollados, identificando la mejor estrategia para reducir las listas de espera y mejorar el rendimiento de los recursos quirúrgicos. El análisis de los resultados se llevará a cabo mediante la comparación de los valores de la función objetivo para cada algoritmo en cada instancia. Para realizar esta comparación de manera efectiva, se utilizará el porcentaje de desviación promedio relativa (ARPD). Este método de análisis permite evaluar de manera cuantitativa y objetiva el rendimiento de las distintas variaciones del algoritmo, facilitando la identificación de la configuración que ofrece los mejores resultados en términos de eficiencia y por lo tanto una mayor reducción de listas de espera mediante maximización de las operaciones quirúrgicas realizadas.

Por último, los resultados obtenidos y las conclusiones derivadas de este estudio podrían servir como modelo para mejorar la gestión de quirófanos en otros centros de salud, contribuyendo a la mejora general del servicio sanitario mediante la programación de las operaciones resultante del algoritmo de mejor rendimiento.

2 DESCRIPCIÓN DEL PROBLEMA

2.1 Entorno

Para gestionar los quirófanos, se utilizan dos principales estrategias: la estrategia de programación de bloques modificada, donde los cirujanos han sido asignados previamente una ventana temporal en las que realizara las operaciones y la estrategia de programación abierta, donde se asignan los quirófanos a los cirujanos (Molina-Pariente, Fernández-Viagas, et al., 2015). En este trabajo se utiliza la política de programación abierta debido a su mayor flexibilidad, ya que esta política abarca todas las soluciones de la política de programación por bloques, lo que la hace más adaptable y eficiente para la planificación y programación de las salas de operaciones (Fei, H., Chu, C., & Meskens, N., 2009).

La unidad de especialidad del hospital de referencia dispone de 14 cirujanos y 4 quirófanos multifuncionales para hacer cirugías de urgencia diferida, electivas y ambulatorias (Molina-Pariente, Hans, et al., 2015). Cada día, hay 3 quirófanos libres para hacer cirugías de urgencia diferida y electivas de 8.30 a 15.00 h, y 1 quirófano dedicado a hacer cirugías ambulatorias de 15.00 a 20.00 h.

Las cirugías de urgencia diferida tienen asignados dos días-quirófano (es decir, una combinación de un quirófano y un día) cada semana, debido a que llegan de forma imprevisible y tienen mucha prioridad (deben ser operadas lo antes posible), y a que solo unos pocos cirujanos pueden realizarlas. La mayoría de las cirugías plásticas se pueden hacer en cualquier quirófano disponible por cualquier cirujano disponible, salvo las microcirugías, que tienen asignados dos días-quirófano cada semana, por la complejidad, el equipamiento quirúrgico especial que requieren, y la duración estimada de la cirugía de 10 horas (Molina-Pariente, Hans, et al., 2015)

Para aplicarlo a nuestro problema de programación de operaciones, entenderemos los quirófanos (Operation Rooms: OR) como las máquinas del proceso a las cuales serán asignado los pacientes para ser operados, y los cirujanos que realizan la operación. De esta manera, se puede asimilar a un entorno de máquinas paralelas (Parallel machines: *Pm*).

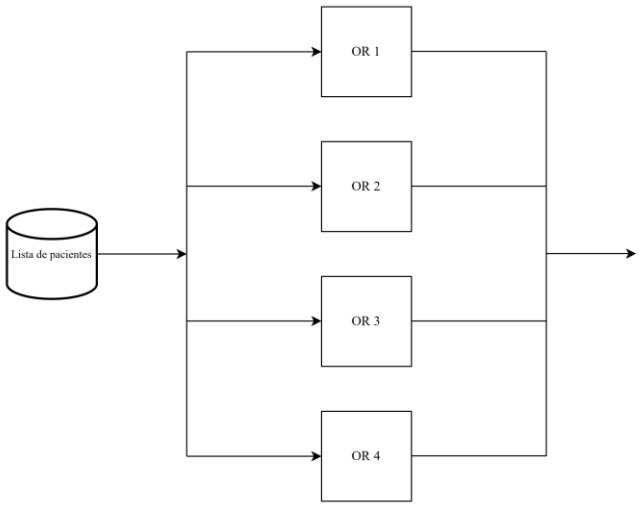

Figura 2-1: *Entorno de máquinas paralelas en optimización quirúrgica.*

Cada paciente en la lista de espera tiene un cirujano asignado, el cual está encargado de realizar tanto las consultas previas como la operación quirúrgica al paciente. A la pareja cirujano-paciente, se le asigna un OR durante la duración de la operación en el horizonte de planificación.

Esta asignación se realizará en función de distintos parámetros: la especialidad del cirujano, disponibilidad de la OR en función del tipo de operación requerida, la capacidad de trabajo tanto del cirujano como de la OR, el número de desplazamientos máximos que puede realizar un cirujano, las fechas en las que cada paciente se encuentra disponible… etc.

La duración esperada de cada cirugía es fundamental para una correcta programación de las operaciones quirúrgicas y se aproxima mediante la generación de dos parámetros aleatorios: la duración media esperada de la cirugía y la desviación estándar. Estos parámetros se utilizan para modelar una distribución normal logarítmica, que refleja la variabilidad en los tiempos de cirugía. De esta manera, cada vez que se simula una cirugía, se genera una duración específica basada en esta distribución, considerando tanto el tiempo necesario para realizar la operación como las actividades de preparación previa y posterior. Este enfoque permite capturar la naturaleza estocástica y la complejidad de los procedimientos quirúrgicos.

2.2 Función objetivo

La función objetivo busca maximizar el número de pacientes que han sido operados durante el horizonte de planificación:

$$Maximizar \sum_{i \in I} \sum_{j \in J} \sum_{h \in H} X_{ijh}$$

2.3 Restricciones y limitación de recursos

La unidad del hospital de referencia lleva a cabo su trabajo bajo restricciones, las cuales son necesarias identificar para poder modelar el problema. Se tomarán las siguientes consideraciones:

- Los pacientes que se encuentran en lista de espera de la unidad serán aquellos en los que se tenga que realizar cirugías de urgencia diferida, electivas o ambulatorias. Este problema se centra en los pacientes que requieran de intervención médica programable en función de los recursos disponibles, y, por lo tanto, las cirugías de emergencia no son objeto de estudio ya que estas cirugías se realizan de carácter urgente y con recursos adicionales (denominados recursos quirúrgicos urgentes) (Molina-Pariente, Hans, et al., 2015).

  La principal limitación de los pacientes, son sus fechas de disponibilidad y duración máxima de estancia. Cada paciente estará disponible en una fecha para la cual ya haya pasado todos los requisitos previos para someterse a una operación (pruebas preoperatorias y preparación previa). Una vez disponibles, no se puede mantener al paciente indefinidamente hasta que sea conveniente programar la operación, si no que existe una cantidad máxima de días establecida desde su fecha de disponibilidad hasta que se realiza la cirugía, definidas por el Servicio Nacional de Salud en 45, 180 y 365 días. A cada paciente se le será establecido un nivel de prioridad en función de la urgencia necesaria para su operación y el tiempo que lleva esperando desde su disponibilidad para la cirugía. A pesar de la diferencia en la prioridad de cada paciente en la lista de espera, no existen restricciones de sucesión o precedencia, de manera que un paciente pueda ser atendido antes o después en función de la secuencia que mejor resulte en la función objetivo, la cual únicamente busca maximizar el número de cirugías.

- Los cirujanos disponen de un número máximo de días semanales en los que pueden realizar cirugía. También se establecerá una restricción en los posibles ORs a los que un cirujano puede ser asignado para operar cada día, buscando reducir el tiempo de inactividad entre operaciones, la movilidad de los cirujanos (ya que físicamente los OR pueden estar a distancias considerables) y la superposición de cirugías consecutivas por el mismo cirujano (Molina-Pariente, Fernández-Viagas, et al., 2015).

- Los quirófanos multifuncionales (*OR*) están disponibles 8 horas diarias. Es necesario tener en cuenta la función de cada quirófano para la programación, ya que ciertas operaciones pueden ser realizadas únicamente en quirófanos equipados y preparados para ellas. Existen también reservas preestablecidas de manera que ciertos quirófanos estén únicamente disponibles para un tipo de operación durante un par de días a la semana, específicamente para las operaciones de quemaduras (por su necesidad de ser tratada lo antes posible) y microcirugía (por su larga duración). Cada quirófano estará funcionando a lo largo de la jornada laboral con un descanso entre mañana y tarde.
- Los recursos humanos, instrumentales perioperatorios y las instalaciones de recuperación están disponibles siempre que se necesiten, de manera que no podrán representar cuellos de botella.

3 METODOLOGÍA

En este apartado, se profundizará en varios aspectos esenciales para comprender el enfoque metodológico empleado en este trabajo. Primero, se detallará la notación empleada a lo largo del estudio, con el fin de establecer un lenguaje común que facilite la comprensión y comunicación de las ideas presentadas. Seguidamente, se procederá a presentar el modelo matemático que constituye el cimiento teórico de la investigación, compuesto por la función objetivo y sus restricciones asociadas. Además, se explorará el planteamiento de distintos algoritmos de decodificación de la solución, cada uno basado en una propuesta distinta de los cuales se usará el que tenga mejor rendimiento junto al algoritmo de Evolución Diferencial. Los algoritmos de decodificación desempeñan un papel crucial al transformar la secuencia de pacientes y datos del problema en un valor de la función objetivo, tarea que se repite múltiples veces en cada iteración del algoritmo, destacando así su rol fundamental en el proceso. Asimismo, se describirá detalladamente el algoritmo de Evolución Diferencial que se usará para la resolución del problema, incluyendo su desarrollo y adaptación específica al contexto de estudio. Este análisis permitirá una comprensión integral del enfoque metodológico empleado y sentará las bases para el posterior análisis y discusión de los resultados obtenidos.

3.1 Notación

La notación que se utilizará a lo largo del trabajo será la de las ***Figuras 3-1,2,3*** (Molina-Pariente,et al., 2015) :

Tabla 3–1. Notación de índices y conjuntos

Índices y conjuntos	
$h \in H$	Conjunto de períodos del horizonte de planificación
$i \in I$	Conjunto de pacientes en la lista de espera
$j \in J$	Conjunto de quirófanos (ORs)
$s \in S$	Conjunto de cirujanos

Tabla 3–2. Notación de parámetros

Parámetros	
r_{jh}	Capacidad de OR_jen el día h (en minutos)
a_{sh}	Capacidad de cirujanos s en el día h (en minutos)
u_s	Número OR_j en los que cirujanos s pueden realizar operaciones en cualquier día h (en minutos)
γ_i	Cirujanos s encargado del paciente i
rd_i	Fecha inicial de disponibilidad del paciente i
dd_i	Fecha final de disponibilidad del paciente i
δ_{ijh}	Parámetro binario que indica 1 si la operación del paciente i se puede realizar en OR_j en el día h; 0 en caso contrario
t_i	Tiempo esperado de cirugía del paciente i
w_i	Peso quirúrgico del paciente i

Tabla 3–3. Notación de las variables

Variables	
X_{ijh}	Parámetro binario que indica 1 si el paciente i se opera en OR_j en el día h; 0 en caso contrario
Z_{sjh}	Parámetro binario que indica 1 si el cirujano s opera en OR_j en el día h; 0 en caso contrario

3.2 Modelo

En esta subsección, se presenta el modelo matemático que, aunque no se utiliza directamente para resolver el problema, es esencial para establecer el marco teórico. Su inclusión se justifica para establecer una base sólida y coherente para el estudio, proporcionando relaciones entre variables y fundamentando decisiones metodológicas. Además, permite abordar las restricciones del problema y clarificar la notación y el funcionamiento de las variables, aspectos clave para interpretar los resultados y comunicar los hallazgos.

$$Maximizar \sum_{i\in I}\sum_{j\in J}\sum_{h\in H} X_{ijh} \tag{1}$$

Sujeto a:

$$\sum_{j\in J}\sum_{h\in H} X_{ijh} \leq 1 \quad (\forall i \in I) \tag{2}$$

$$\sum_{j\in J}\sum_{h\in H}^{rd_i} X_{ijh} = 0 \quad (\forall i \in I) \tag{3.1}$$

$$\sum_{j\in J}\sum_{\substack{h\in H \\ h\leq d_i}} X_{ijh} = 1 \quad (\forall i \in I | d_i \leq |H|) \tag{3.2}$$

$$\sum_{i\in I} t_i\, X_{ijh} \leq r_{jh} \quad (\forall j \in J, \forall h \in H) \tag{4}$$

$$\sum_{j\in J}\sum_{\substack{i\in I \\ \gamma_i = s}} t_i X_{ijh} \leq a_{sh} \quad (\forall s \in S, \forall h \in H\,) \tag{5}$$

$$\sum_{j\in J} Z_{sjh} \leq u_s \quad (\forall s \in S, \forall h \in H) \tag{6}$$

$$\sum_{\substack{i \in I \\ \gamma_i = s}} t_i \, X_{ijh} \leq r_{jh} \, Z_{sjh} \quad (\forall s \in S, \forall j \in J, \forall h \in H) \tag{7.1}$$

$$\sum_{\substack{i \in I \\ \gamma_i = s}} t_i \, X_{ijh} \leq Z_{sjh} \quad (\forall s \in S, \forall j \in J, \forall h \in H) \tag{7.2}$$

$$X_{ijh} = 0 \ \left(\forall s \in S, \forall j \in J, \forall h \in H \middle| \delta_{ijh} = 0\right) \tag{8}$$

$$X_{ijh} \in \{0,1\} \, (\forall s \in S, \forall j \in J, \forall h \in H) \tag{9}$$

$$Z_{sjh} \in \{0,1\} \, (\forall s \in S, \forall j \in J, \forall h \in H) \tag{10}$$

La restricción (2) garantiza que cada cirugía se programe como máximo una vez durante el horizonte de planificación. El conjunto de restricciones (3.1) y (3.2) determinan la fecha más temprana y la más tardía en que un paciente puede ser programado. La restricción (3.1) impide que el paciente sea programado antes de la fecha de inicio, mientras que la restricción (3.2) asegura que la cirugía de un paciente con una fecha límite dentro del horizonte de planificación se realice antes de su fecha más tardía (esta restricción actúa como blanda). La restricción (4) impide que la cantidad total de tiempo de quirófano asignado a los cirujanos en un día de quirófano sea superior a su capacidad regular. La restricción (5) prohíbe que la cantidad total de tiempo asignada a un cirujano sea superior a su capacidad en cualquier día. La restricción (6) limita el número de días de quirófano que se pueden asignar a un cirujano en un día. El conjunto de restricciones (7.1) y (7.2) definen si un cirujano está asignado a un día de quirófano. La restricción (8) garantiza que cada cirugía se realice en un día de quirófano adecuado. Por último, las restricciones (9) y (10) son restricciones binarias para las variables de decisión. (Molina-Pariente, Hans, et al., 2015).

3.3 Decodificación de la solución

El desarrollo de un algoritmo de decodificación es el primer paso para abordar este problema. El algoritmo de decodificación obtiene el valor correspondiente de la función objetivo a partir de una secuencia de pacientes y los datos del problema, siendo así, el puente entre los datos y la evaluación cuantitativa de la solución propuesta. El proceso de decodificación de soluciones implica la manipulación de datos para obtener una evaluación precisa del rendimiento de la solución. Este proceso es esencial ya que proporciona un valor que refleja fielmente cómo la solución se ajusta a los objetivos establecidos. Además, esta decodificación sienta las bases para el funcionamiento de los algoritmos de Evolución Diferencial, que necesitan constantemente decodificar secuencias durante todo su funcionamiento en los procesos de mutación, recombinación, búsqueda local y selección.

A continuación, se desarrollarán tres algoritmos de decodificación: DECOD1, el cual asigna pacientes al OR antes con mayor disponibilidad factible. DECOD2, que tiene un carácter teórico, ya que realiza la asignación igual que la decodificación DECOD1, pero con la libertad de también poder seleccionar el cirujano para cada paciente siempre que sea factible (el hospital de referencia no sigue este modelo de gestión). DECOD3, el cual implementa un sistema de holguras para minimizar los tiempos ociosos entre operaciones de los OR.

3.3.1 Algoritmo de decodificación DECOD1

En el algoritmo de decodificación, **DECOD1**, mostrado en la ***Figura 3-1***, se llevará a cabo la asignación de pacientes a las diferentes salas de operaciones disponibles, junto con sus respectivos cirujanos encargados de realizar las operaciones. Esta asignación se basa en una secuencia inicial que indica el orden en el cual los pacientes serán ubicados en las salas de operaciones.

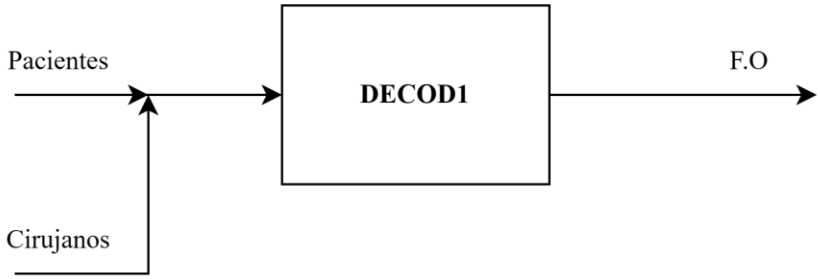

Figura 3-1: *Algoritmo de decodificación DECOD1*

El objetivo primordial consiste en colocar a los pacientes en la sala de operaciones ***con mayor disponibilidad,*** donde sea posible realizar la operación específica del paciente en el día en que esté disponible, y donde el cirujano tenga la capacidad de llevar a cabo la operación, teniendo en cuenta su propia carga de trabajo y su disponibilidad para trasladarse entre salas de operaciones. Es decir, se busca hacer la asignación que conlleve menor tiempo de terminación (Earliest Completion Time) del orden dado por la secuencia.

El algoritmo descrito mediante el pseudocódigo mostrado en la ***Figura 3-2*** itera sobre una secuencia de pacientes y, para cada uno, evalúa la posibilidad de programar la operación en un horizonte de días determinado. Luego, prioriza asignar al paciente a la sala de operaciones con la menor carga de trabajo, siempre que tanto el cirujano como la sala tengan la capacidad necesaria para realizar la operación. Este enfoque de asignación dinámica busca optimizar la utilización de recursos y garantizar que las operaciones se realicen dentro de los plazos establecidos.

```
def DECOD1(datos,secuencia):
    desempaquetar los datos
    for paciente in secuencia:
        if paciente no ha sido operado:
            for dia in horizonte de planificacion:
                if release date(paciente) <= dia:
                    for OR in OR ordenados disponibilidad: #Asigna el OR que antes está disponible, si no pasa al siguiente
                        tiempo de inicio de la operacion = max(tiempo de finalizacion OR, tiempo de finalizacon cirujano)
                        if duracion de operacion < capacidad cirujano and cirujano puede moverse a OR:
                            if duracion de operacion < capacidad OR and paciente puede operarse en OR:
                                Xijh,Zsjh = 1
                                tiempo de finalizacion de la operacion = duracion de la operacion + tiempo de inicio de la operacion
                                tiempo de finalizacion de OR = tiempo de finalizacion de la operacion
                                tiempo de finalizacion de cirujano = tiempo de finalizacion de la operacion
                                capacidad OR = duracion dia - tiempo de finalizacion de OR
                                capacidad cirujano -= duracion de la operacion
                                paciente = operado
                                FO = suma(pacientes operados)
```

Figura 3-2: *Pseudocódigo del algoritmo de decodificación DECOD1*

Para comprobar el funcionamiento del algoritmo de decodificación DECOD1, se realiza como ejemplo una instancia aleatoria de poco tamaño. Se utilizarán los datos de entrada mostrados en la ***Tabla 3-4***:

Tabla 3–4. Datos de entrada para ejemplo del DECOD1

$h \in H$	3 $días$	Períodos del horizonte de planificación
$i \in I$	20 $pacientes$	Número de pacientes en la lista de espera
$j \in J$	3 OR	Conjunto de quirófanos (ORs)
$s \in S$	5 $cirujanos$	Conjunto de cirujanos
r_{jh}, a_{sh}	480 $\frac{minutos}{día}$	Capacidad máxima de OR y cirujanos
u_s	2 OR	Número de quirófanos máximo por cirujano
rd_i	[0,1]	Release dates de los pacientes [día]
t_i	[30,240]	Tiempo de operación del paciente [minutos]
γ_i	[1,5]	Cirujano encargado del paciente
δ_{ijh}	$\begin{cases} Probabilidad\ 10\% = 0 \\ Probabilidad\ 90\% = 1 \end{cases}$	Parámetro binario que indica 1 si la operación del paciente i se puede realizar en OR_j en el día h; 0 en caso contrario
[19,18,17,16,15,14,13,12,11,10,9,8,7,6,5,4,3,2,1,0]		Secuencia inicial

Una vez establecidos los parámetros para el ejemplo, se generarán los valores que se usarán como datos de la instancia, siendo los obtenidos los mostrados en la ***Tabla 3-5***. Las due dates dd_i también se generan aleatoriamente, pero no tienen uso, ya que no la tomamos como restricción y no medimos ningún parámetro relacionado con el mismo como lateness o tardiness, posteriormente si harán falta para las heurísticas constructivas que inicializan la metaheurística. Mediante la decodificación, se obtendrá un valor de la función objetivo a la vez que se guardan los valores de los tiempos de las operaciones, para poder crear y representar un diagrama de Gantt que nos permita visualizar con claridad el funcionamiento de la decodificación, en este caso, el color de las operaciones viene dado por el cirujano que la haya realizado.

Tabla 3–5. Datos generados para el ejemplo del DECOD1

rd_i		[1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0] [día]								
t_i		[57, 188, 209, 150, 57, 193, 171, 233, 164, 111, 115, 111, 181,70, 200,45, 186, 125, 149, 186][$minutos$]								
δ_{ijh}	*0*	1	1	1	1	1	1	1	0	0
	1	1	1	1	1	1	1	1	1	1
	2	0	1	1	1	1	0	1	1	1
	3	1	1	1	1	1	1	1	1	1
	4	1	1	1	1	1	1	1	1	1
	5	1	1	1	1	0	1	1	1	1
	6	1	1	1	1	1	0	1	1	1
	7	1	1	1	1	1	1	1	1	1
	8	1	1	1	0	1	0	1	1	1
	9	0	1	1	1	1	1	1	1	0
	10	1	1	1	0	1	1	1	1	0
	11	1	1	1	1	1	1	1	1	1
	12	1	1	1	1	1	1	1	1	0
	13	1	1	1	1	1	1	1	1	1
	14	1	1	1	1	1	1	1	1	0
	15	1	1	1	1	1	1	1	1	1
	16	1	1	1	1	1	1	1	1	1
	17	1	1	0	1	1	1	1	1	1
	18	1	0	0	0	1	1	0	1	1
	19	1	1	1	1	1	1	1	1	1
	Pacientes i	δ_{ij1}			δ_{ij2}			δ_{ij3}		

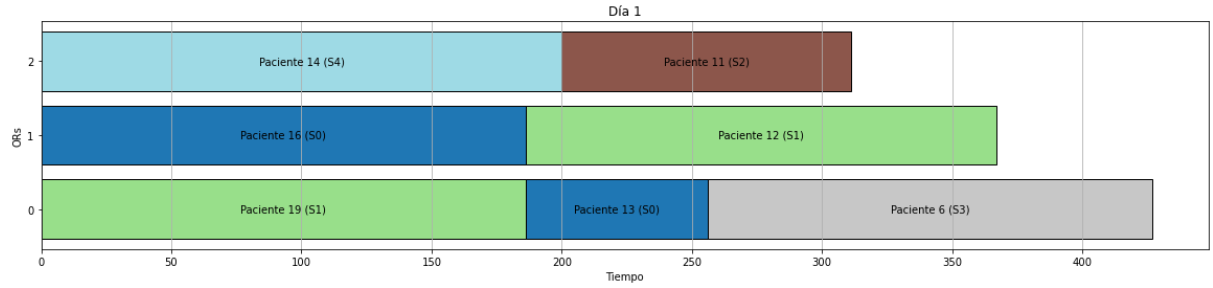

Figura 3-3: *Diagrama de Gantt del período 1 del ejemplo de DECOD1*

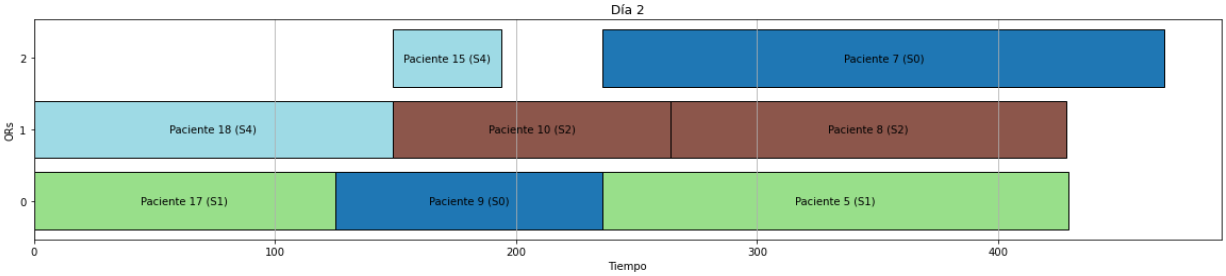

Figura 3-4: *Diagrama de Gantt del período 2 del ejemplo de DECOD1*

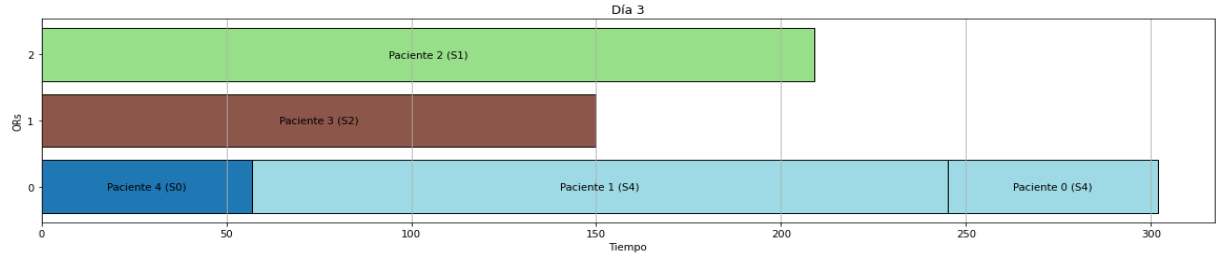

Figura 3-5: *Diagrama de Gantt del período 3 del ejemplo de DECOD1*

Recordando la secuencia utilizada [19,18,17,16,15,14,13,12,11,10,9,8,7,6,5,4,3,2,1,0]:

- En la ***Figura 3-3*** después de haber llenado inicialmente los OR con pacientes disponibles en el primer día, se puede comprobar el proceso de asignación por mayor disponibilidad previa: el siguiente paciente por colocar es el 13, el cual se coloca en la OR que antes está disponible (OR0). El mismo proceso se realiza con los siguientes pacientes.
- En la ***Figura 3-4*** se comprueba mediante los pacientes 17 y 18, que debido a tener su $rd_i > 0$, no están disponibles durante el primer día, y por lo tanto no son operados hasta el segundo día.
- En la ***Figura 3-4*** el paciente 15 experimenta uno de los problemas del algoritmo de decodificación, ya que su cirujano asignado (S4) es el mismo que el del paciente 18, el cual ya fue asignado previamente. En el momento de colocar al paciente 15, el algoritmo detecta que el OR con mayor disponibilidad es el OR2, pero debido a que el cirujano S4 está ocupado, la operación no puede comenzar hasta que termine el paciente 18. Esto deja un vacío que no podrá ser rellenado posteriormente, dejando un hueco de ineficiencia considerable.
- En la ***Figura 3-5*** se aprecia el efecto que tiene δ_{ijh}, que indica si un paciente i, se puede operar en el OR j, el día h. Para el paciente 0, δ_{0j2} solo puede operarse en j=0. Pese que en los otros OR la disponibilidad es más temprana, deber ser asignado a OR0 donde si puede realizarse la operación.

3.3.2 Algoritmo de decodificación DECOD2

En el segundo algoritmo de decodificación, **DECOD2**, se llevará a cabo la asignación de pacientes a las diferentes salas de operaciones disponibles, pero a diferencia del primer modelo, también se realizará la asignación de sus respectivos cirujanos encargados de realizar las operaciones tal y como se muestra en la ***Figura 3-6***.

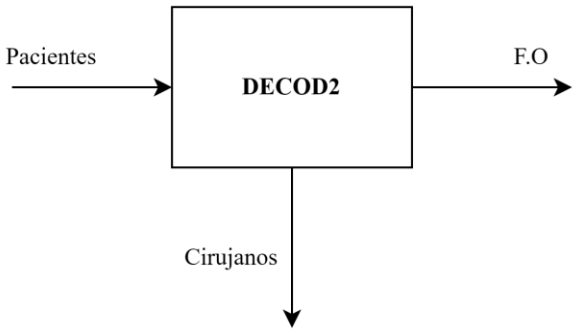

Figura 3-6: *Algoritmo de decodificación DECOD2*

Los datos y la instancia de prueba generada serán los mismos que los utilizados en el modelo DECOD1 (***Tablas 3-4,5)***, lo único que se añade es el dato γ_{si} cuyos valores aparecen en la ***Tabla 3-6*** y representa si un cirujano s es capaz de realizar la operación de un paciente i.

Para el ejemplo se ha generado el parámetro γ_{si} :

Tabla 3–6. Valores de γ_{si} en el ejemplo de DECOD2

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
S0	0	0	0	1	1	1	1	1	0	1	0	0	1	1	1	1	0	1	1	1
S1	1	0	0	1	1	1	1	1	1	1	1	1	0	1	1	1	0	0	1	0
S2	1	1	1	1	0	0	1	1	1	1	1	0	1	1	1	0	0	1	1	1
S3	0	1	1	1	1	1	0	1	0	0	1	1	0	1	0	1	0	0	1	0
S4	1	1	0	0	1	1	0	1	1	0	1	1	1	1	1	1	1	1	1	0

El algoritmo se modifica añadiendo una iteración que recorra el orden de cirujanos por orden de disponibilidad (de más temprana a más tardía), y en el momento que se cumplan las condiciones se le asigna el cirujano al paciente. La asignación de las OR se realiza previa a la de los cirujanos ya que, en este caso concreto, los recursos de las OR son más restrictivos que los cirujanos. El pseudocódigo se muestra en la ***Figura 3-7.***

```
def DECOD2(datos,secuencia):
    desempaquetar los datos
    for paciente in secuencia:
        if paciente no ha sido operado:
            for dia in horizonte de planificacion:
                if release date(paciente) <= dia:
                    for OR in OR ordenados por disponibilidad: #Asigna el OR antes disponible
                        for cirujano in cirujanos ordenados por disponibilidad #Asigna el Cirujano antes disponible
                            if cirujano puede realizar la operacion:
                                tiempo de inicio de la operacion = max(tiempo de finalizacion OR, tiempo de finalizacon cirujano)
                                if duracion de operacion < capacidad cirujano and cirujano puede moverse a OR:
                                    if duracion de operacion < capacidad OR and paciente puede operarse en OR:
                                        #Se puede hacer en la OR seleccionada
                                        Xijh,Zsjh = 1
                                        tiempo de finalizacion de la operacion = duracion de la operacion + tiempo de inicio de la operacion
                                        tiempo de finalizacion de OR = tiempo de finalizacion de la operacion
                                        tiempo de finalizacion de cirujano = tiempo de finalizacion de la operacion
                                        capacidad OR = duracion dia - tiempo de finalizacion de OR
                                        capacidad cirujano -= duracion de la operacion
                                        paciente = operado
                                        FO = suma(pacientes operados)
```

Figura 3-7: *Pseudocódigo del algoritmo de decodificación DECOD2*

Obteniendo los siguientes diagramas de Gantt a lo largo del horizonte de planificación:

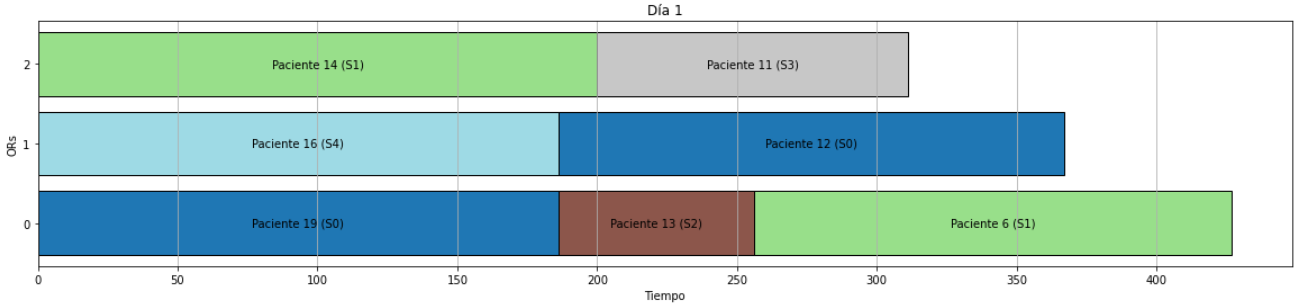

Figura 3-8: *Diagrama de Gantt del período 1 del ejemplo de DECOD2*

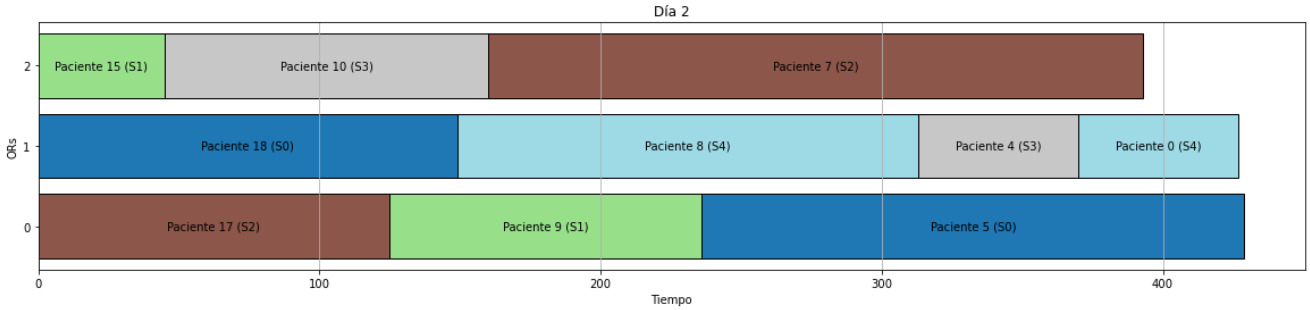

Figura 3-9: *Diagrama de Gantt del período 2 del ejemplo de DECOD2*

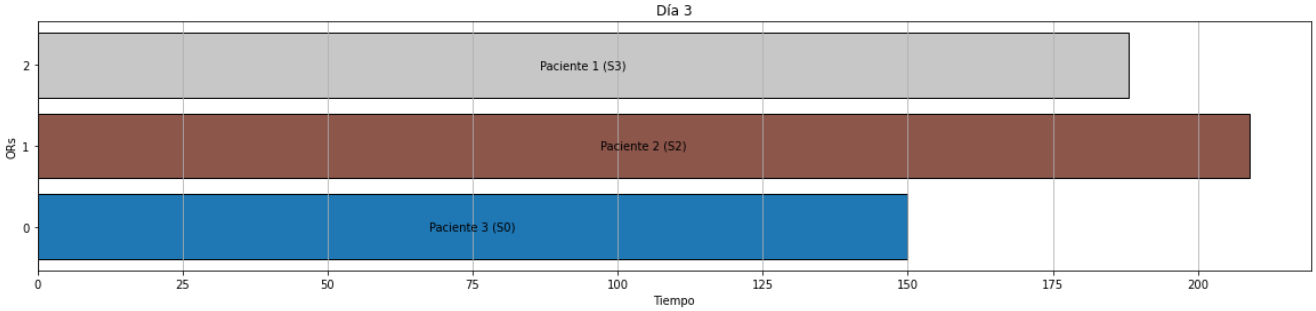

Figura 3-10: *Diagrama de Gantt del período 2 del ejemplo de DECOD2*

La principal ventaja de este algoritmo de decodificación que se muestra en las ***Figuras 3-6, 3-7 y 3-8*** es la capacidad de realizar una planificación con menos tiempos muertos entre operaciones, siendo la elección de cirujanos una gran herramienta de flexibilidad para el modelo. La decodificación DECOD2, conllevaría una modificación del problema original, lo cual no siempre es posible, y un tiempo de cómputo añadido, por lo tanto, se usa como un instrumento de comparación entre los modelos de decodificaciones.

Sería interesante considerar si la mejora en la eficiencia incrementaría lo suficiente al incluir la asignación del cirujano durante la decodificación para justificar el aumento en el tiempo de cómputo y modificaciones del problema que conllevaría.

3.3.3 Algoritmo de decodificación DECOD3

Este algoritmo de decodificación surge tras experimentar con los algoritmos de decodificación anteriores, y apreciar que el hecho de que asignar siempre en función del menor tiempo de terminación daba lugar a ineficiencias en la decodificación de las soluciones, que posteriormente impedía en el proceso de búsqueda a los algoritmos metaheurísticos encontrar soluciones de manera óptima. En este algoritmo, al igual que DECOD1, los cirujanos ya han sido previamente asignados. El algoritmo DECOD3 queda representado en la ***Figura 3-11***.

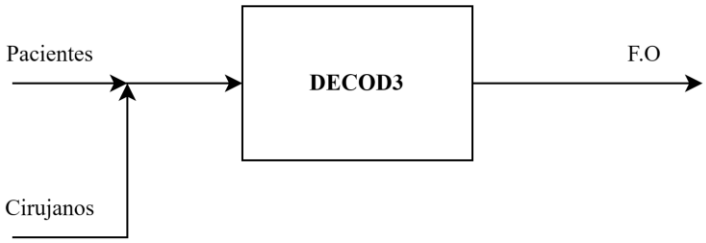

Figura 3-11: *Algoritmo de decodificación DECOD3*

Tal y como se mencionaba previamente sobre la **Figura 3-3***: Diagrama de Gantt del período 2 del ejemplo de DECOD1,* se puede comprobar que el paciente 15 es asignado en la OR2 debido a que es la que está disponible lo antes posible (Earliest Completion Time), pero esto deja un hueco el cual ya no se tendrá en cuenta durante el resto de la decodificación para poder colocar futuras operaciones, creando grandes vacíos de ineficiencia. El algoritmo de Evolución Diferencial no puede evitar estos vacíos en su proceso de búsqueda, ya que es intrínseco a la decodificación, responsable de haberlos generado. Ante este problema se plantea un nuevo criterio a la hora de asignar ORs: la holgura (11), donde se calcula la diferencia entre el menor tiempo de inicio posible de la operación (st_{hs}) (limitado por el cirujano encargado s en un día dado h) y los tiempos de finalización de las OR (ct_h) en un día dado h en el momento de la asignación. Se establece el valor absoluto para penalizar tanto las OR que estarían vacías durante un tiempo hasta que pudiese comenzar la operación y las OR que no estarán disponibles en el instante, retrasando el comienzo. En la ***Figura 3-12,*** se muestra el pseudocódigo de DECOD3.

$$holgura_i = |st_{hs} - ct_h| \qquad (11)$$

```
def DECOD3(datos,secuencia):
    desempaquetar los datos
    for paciente in secuencia:
        if paciente no ha sido operado:
            for dia in horizonte de planificacion:
                if release date(paciente) <= dia:
                    for OR in OrdenadosDeMenorAMayor(abs(tiempo de finalizacion de ORs - tiempo de finalizacion de cirujano)):
                        tiempo de inicio de la operacion = max(tiempo de finalizacion OR, tiempo de finalizacon cirujano)
                        if duracion de operacion < capacidad cirujano and cirujano puede moverse a OR:
                            if duracion de operacion < capacidad OR and paciente puede operarse en OR:
                                #Se puede hacer en la OR seleccionada
                                Xijh,Zsjh = 1
                                tiempo de finalizacion de la operacion = duracion de la operacion + tiempo de inicio de la operacion
                                tiempo de finalizacion de OR = tiempo de finalizacion de la operacion
                                tiempo de finalizacion de cirujano = tiempo de finalizacion de la operacion
                                capacidad OR = duracion dia - tiempo de finalizacion de OR
                                capacidad cirujano -= duracion de la operacion
                                paciente = operado
                                FO = suma(pacientes operados)
```

Figura 3-12: *Pseudocódigo del algoritmo de decodificación DECOD3*

Usando el conjunto de datos de las tablas 3-3 y 3-4 Datos Generados para el DECOD1, se realizará la misma simulación para comprobar el distinto funcionamiento, el cual se representará mediante su correspondiente diagrama de Gantt en las figuras 3-13,14,15:

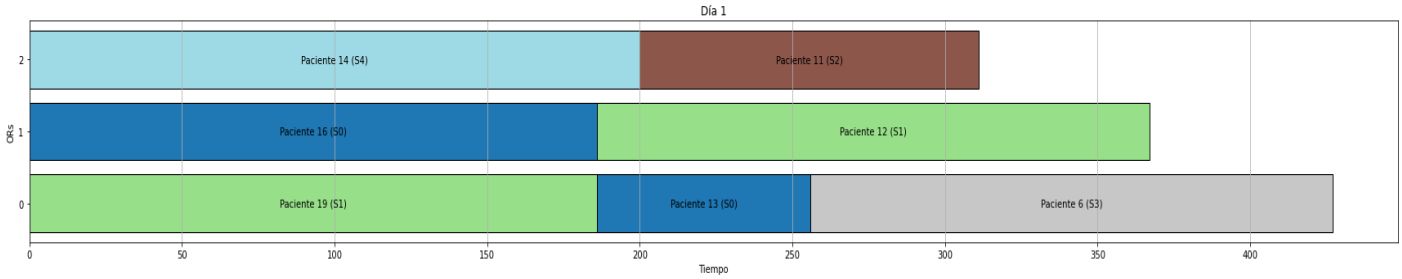

Figura 3-13: *Diagrama de Gantt del período 1 del ejemplo de DECOD3*

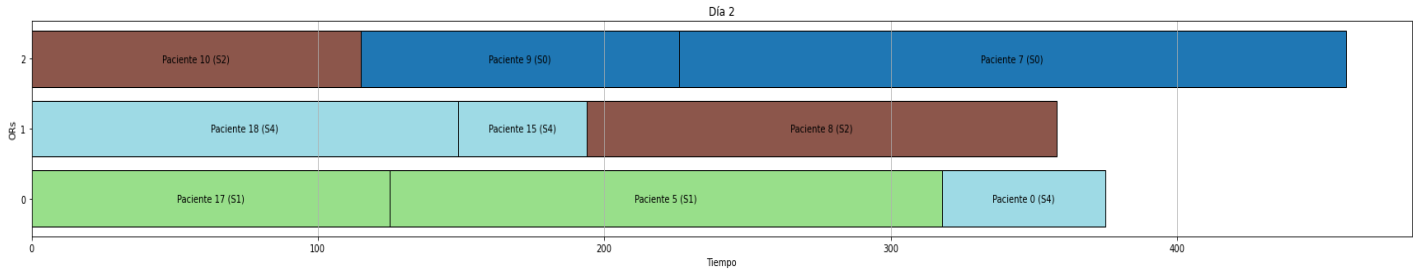

Figura 3-14: *Diagrama de Gantt del período 2 del ejemplo de DECOD3*

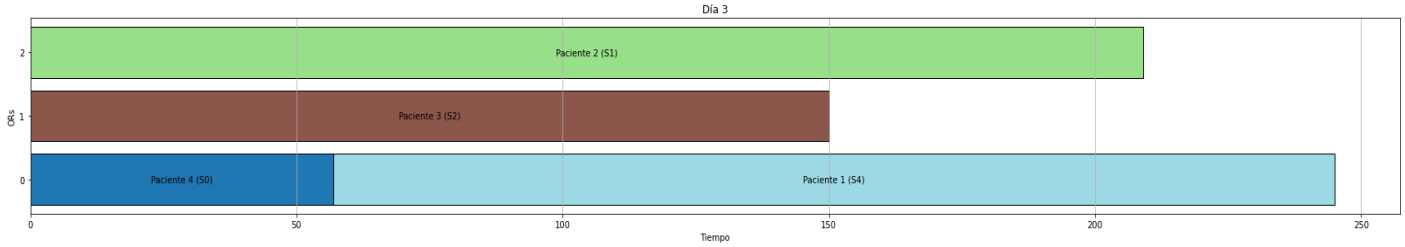

Figura 3-15: *Diagrama de Gantt del período 3 del ejemplo de DECOD3*

Recordando la secuencia utilizada [19,18,17,16,15,14,13,12,11,10,9,8,7,6,5,4,3,2,1,0]:

- Se puede comprobar mediante la ***Figura 3-14*** el funcionamiento de DECOD3, ya que al colocar al Paciente 15 en la OR2 (pese a ser el OR que tiene una disponibilidad más tardía) se logra eliminar los tiempos ociosos entre operaciones (sin necesidad de reorganizar la asignación de cirujanos, como es el caso del modelo DECOD2) y poder completar la OR1 con otras operaciones, lo cual facilitará al algoritmo metaheurístico el proceso de búsqueda para encontrar una solución cercana a la óptima. Pese a ello, será necesario comprobarlo en la simulación de la metaheurística para comprobar su eficacia.

Mediante el uso de la $holgura_j = |st_{hs} - ct_h|$ se consigue penalizar y por lo tanto reducir los tiempos ociosos en los quirófanos. Esto ayudará posteriormente a una mejor búsqueda del espacio de soluciones por parte del algoritmo de Evolución Diferencial, ya que se elimina una ineficiencia intrínseca a la hora de obtener el valor de la función objetivo para una secuencia dada.

3.4 Evolución Diferencial

El algoritmo por desarrollar será de Evolución Diferencial Discreta (DDE), diseñado y destacado para entornos en los que la optimización continua no es posible, como es el caso de los problemas de planificación y asignación de operaciones en la programación de la producción, y que surge a partir del algoritmo propuesto por Storn and Price (1995) de Evolución Diferencial (DE). Su funcionamiento se basa en la manipulación de una población de soluciones representadas como vectores de valores enteros (las operaciones). Estas soluciones experimentan modificaciones principalmente a través de operadores de mutación y recombinación (Pan et al., 2009).

El algoritmo de evolución diferencial inicia con una población inicial de soluciones candidatas, las cuales representan posibles soluciones al problema en cuestión. Los miembros de la población son sometidos a un proceso de mutación mediante el cual se generan nuevas soluciones a partir de combinaciones de soluciones existentes en la población. Luego, la recombinación entra en juego, combinando características de diferentes soluciones para crear nuevas soluciones. Las probabilidades de mutación y recombinación controlan la influencia de estos procesos en la población y ajustar estas probabilidades adecuadamente es fundamental, ya que influyen en la capacidad del algoritmo para explorar y explotar el espacio de búsqueda. La intensidad de la mutación determina la magnitud de los cambios que se aplican a las soluciones durante el proceso de mutación. La selección es el paso crucial donde se eligen las soluciones que seguirán adelante para la siguiente generación, esta elección se basa en la aptitud de las soluciones para resolver el problema. El tamaño de la población tiene un impacto significativo en la dinámica del algoritmo, ya que una población más grande puede aumentar la diversidad y la exploración del espacio de búsqueda, pero también conlleva un mayor costo computacional. El algoritmo de evolución que se desarrolla en este apartado será el algoritmo *4.1 Algoritmo principal.*

Para el desarrollo del algoritmo, se usan los siguientes parámetros y variables propuestos por Pan et al. (2009) mostrados en la ***Tabla 3-7***:

Tabla 3-7. Parámetros y variables del algoritmo de Evolución Diferencial Discreta

Parámetros	
POB	Tamaño de la población
t	Iteración t del algoritmo DDE
i	Operación i de una secuencia
$F, Z \in [0,1]$	Probabilidad de mutación e intensidad de mutación
$CR \in [0,1]$	Probabilidad de recomposición
Variables	
$X_i^t = [x_1^t, x_2^t, \dots, x_n^t]$	Solución objetivo
$V_i^t = [v_1^t, v_2^t, \dots, v_n^t]$	Solución mutada
$U_i^t = [u_1^t, u_2^t, \dots, u_n^t]$	Solución provisional

3.4.1 Población Inicial

En el contexto de los Algoritmos de Evolución Diferencial (DE), la población inicial es el punto de partida de todo algoritmo poblacional. Esta fase inicializa un conjunto de soluciones candidatas sobre las cuales se aplicarán los operadores genéticos para generar nuevas soluciones en cada iteración del algoritmo. La selección

adecuada de la población inicial puede incidir significativamente en la eficacia y eficiencia del algoritmo, ya que influye en la diversidad de las soluciones y en la capacidad del algoritmo para evitar óptimos locales y converger hacia soluciones de alta calidad en el espacio de búsqueda (Mokhtari et al., 2011). Puede ser generada aleatoriamente, mediante heurísticas constructivas, a partir de otras metaheurísticas, o una combinación de todas. Comenzar desde total aleatoriedad conlleva disminuir la probabilidad de llegar a una solución óptima y un mayor tiempo de computación (Mokhtari et al., 2011).

En este trabajo, el algoritmo de Evolución Diferencial partirá de una población inicial formada por individuos generados aleatoriamente, con la excepción de las variantes del algoritmo cuya modificación esté relacionada con la solución inicial (*4.2 Variantes de solución inicial)*. El tamaño de población *POB* también será constante, exceptuando las variantes que específicamente tengan como funcionamiento un tamaño de población dinámico (*4.3 Variantes de evolución de la población)*.

3.4.2 Mutación

La mutación implica añadir la modificación aleatoria ponderada de dos componentes aleatorios de la población a un tercer miembro para generar la solución mutada (Pan et al., 2008), lo que permite explorar diferentes configuraciones en el espacio de búsqueda.

$$V_i^t = X_a^t \oplus Z \otimes \left(X_b^{t-1} - X_c^{t-1}\right)$$

$$Z \otimes \left(X_b^{t-1} - X_c^{t-1}\right) = \Delta_i^t = \delta_{ij}^t \begin{cases} x_{bj}^{t-1} - x_{cj}^{t-1} & si\ \ rand < F \\ 0 & en\ otros\ casos \end{cases}$$

Donde a, b y c son 3 individuos aleatorios y distintos en la secuencia objetivo (X_i^t), F es la probabilidad de mutación y $Z \in [0,1]$ el factor de escala de la mutación. La solución mutada (V_i^t) no siempre producirá una solución factible para la planificación, ya que se pueden repetir y perder operaciones tras la mutación. Pese a ello, no es necesario garantizar la factibilidad de la solución mutada ya que se obtendrá una solución factible tras el proceso de recombinación. En la ***Figura 3-15,*** se representa el funcionamiento del proceso de mutación.

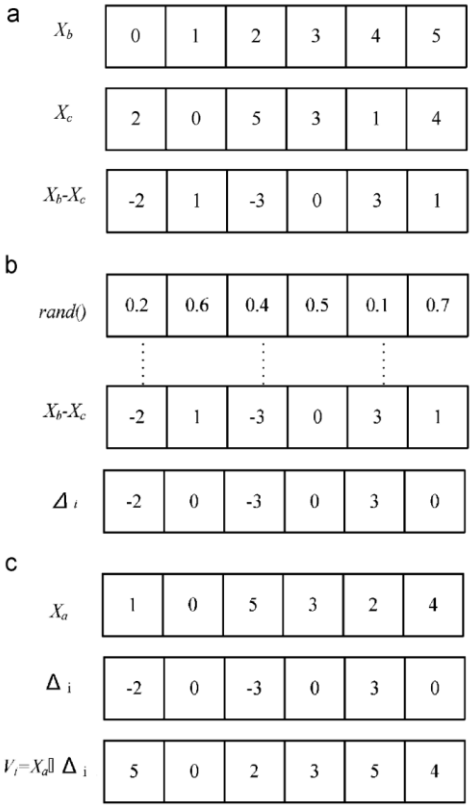

Figura 3-15: *Proceso de mutación* (Pan et al., 2009). **(a)** $\boldsymbol{X_b - X_c}$ **(b)** $\boldsymbol{Z \otimes \left(X_b^{t-1} - X_c^{t-1}\right) = \Delta_i^t}$ con F=0.5 **(c)** $\boldsymbol{V_i^t = X_a^t \oplus \Delta_i^t}$

3.4.3 Recombinación

La recombinación combina las características de la solución mutada (V_i^t) con las de una solución objetivo (X_i^{t-1}) para generar la solución provisional (U_i^t).

En un algoritmo tradicional (DE), el proceso de recombinación consiste en obtener aleatoriamente un punto de corte, rellenar hasta dicho punto con $i = 1,2 \dots n$, y terminar rellenando con X_i^{t-1} (eliminando las operaciones repetidas). Pan et al., (2009) demuestra que, tras varias iteraciones, esta metodología tiene una alta probabilidad de que la solución provisional (U_i^t) sea peor que la solución objetivo (X_i^{t-1}). Para evitar este problema, se siguen los siguientes pasos:

1) Generar aleatoriamente un punto de corte para dividir la solución objetivo X_i^{t-1} en dos partes.
2) Mover la primera parte de X_i^{t-1} a la parte izquierda de U_i^t
3) Establecer $j = 1$
4) Si v_{ij}^t $(de\ la\ solución\ mutada)$ no es repetitiva en U_i^t y rand() < CR, se coloca v_{ij}^t en la primera posición vacía de U_i^t
5) $j = j + 1$ y repetir 4) hasta que $j > n$ o U_i^t esté completa
6) Si U_i^t no ha sido completada, entonces se rellena con el resto de las operaciones de la segunda parte de X_i^{t-1} en el orden original

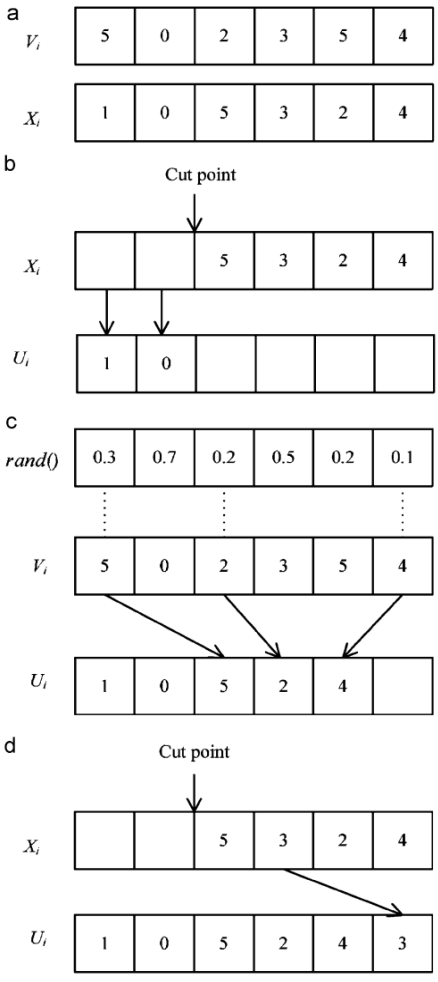

Figura 3-16: *Proceso de recombinación* (Pan et al., 2009). **(b)** Pasos 1) y 2) **(c)** Pasos 3), 4) con CR = 0.4 y 5) **(d)** Paso 6)

3.4.4 Local Search

El proceso de Local Search o Búsqueda Local es un proceso utilizado para explorar soluciones en los amplios espacios de búsqueda que afrontan las distintas técnicas de optimización. No forma parte del algoritmo de evolución diferencial, pero se usará en algunas de sus variantes (*4.4 Variantes de búsqueda local*).

Los enfoques de búsqueda local exploran vecindarios de soluciones mediante la definición de criterios, como la secuencia de soluciones a explorar, cuáles conservar para la siguiente iteración, y si se debe explorar todo el vecindario o solo una parte. Para ello, se determinan vecindarios intercambiando trabajos consecutivos o no consecutivos de una solución (Framiñan et al., 2014).

En este contexto, existen dos enfoques principales para aceptar soluciones del vecindario: uno donde solo se aceptan nuevas soluciones si son mejores o al menos no son peores que la mejor actual, y otro donde temporalmente se aceptan soluciones incluso si son ligeramente peores (Pinedo, 2016). Por último, los enfoques de búsqueda local suelen detenerse después de explorar un número determinado de soluciones, alcanzar un límite de tiempo predefinido, si no se ha generado ninguna mejora durante cierto tiempo o un número específico de soluciones exploradas.

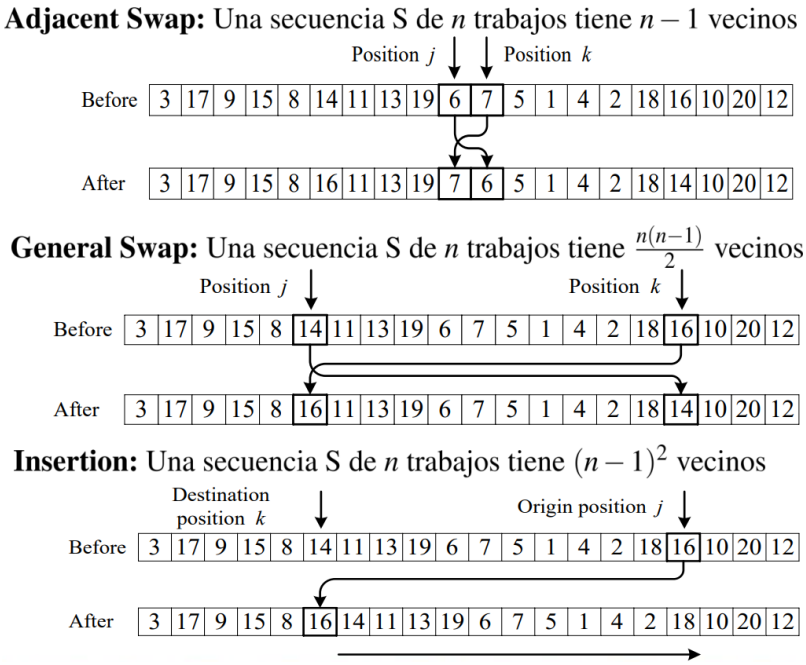

Figura 3-17: *Ejemplo de distintos tipos de búsqueda local* (Framiñan et al., 2014)

Para este problema, se aplicará el procedimiento de Local Search en algunas de las variantes del algoritmo principal. En función del criterio de parada, será: "First Improvement" , si el proceso de búsqueda local finaliza en el momento que se obtiene una mejor solución, o "Best Improvement", en el que se realizan todas las posibles iteraciones y se obtiene la vecindad con mejor valor del criterio de selección que se use.

Este proceso de búsqueda local suele conllevar un tiempo de computación considerable, es por ello que se incluirá en alguna de las distintas variantes del algoritmo para comprobar si sacrificar parte del tiempo de cómputo es rentable a cambio de una mayor exploración e intensificación en la búsqueda de vecindades.

En el apartado *5.2.3 Elección de Local Search*, se analizarán distintos tipos de búsqueda local y se elegirá el método a usar en función de su tiempo de cómputo. El criterio de para para todos los métodos es el de "First Improvement".

3.4.5 Selección

En todos los pasos del algoritmo metaheurístico, es necesario establecer un criterio de selección que permita identificar la calidad de las soluciones. La población es evaluada, y a todos los individuos se les asigna un valor de aptitud al que se denomina "Fitness". Este valor de aptitud está directamente relacionado con el valor de la función objetivo, de tal manera que se asignan un valor de Fitness más altos a mejores valores de la función objetivo, introduciendo así un operador de selección (Framiñan et al., 2014). En este trabajo, el "Fitness" será el número total de pacientes operados al finalizar el horizonte de planificación, aunque para casos donde fuese necesario un *fitness* auxiliar, se plantea usar el tiempo de funcionamiento respecto al tiempo disponible de los ORs.

La etapa de selección en cualquier algoritmo poblacional es esencial para la mejora continua de la calidad de la población y la búsqueda efectiva de soluciones. En este proceso, los individuos descendientes son sometidos a una evaluación, y mediante su Fitness son comparados con la población actual. Aquellos que sean superados por un descendiente serán sustituidos, lo que contribuye a maximizar el rendimiento del algoritmo al garantizar soluciones más cercanas al óptimo y una mayor probabilidad de engendrar descendientes de calidad superior. La selección no solo implica la eliminación de individuos menos aptos, sino también la preservación y promoción de aquellos con mayor adaptabilidad y eficacia en la resolución del problema.

En el algoritmo de Evolución Diferencial la selección se realiza siguiendo los siguientes pasos:

1. Se genera una población mutada (V^t) , y luego cruzada (U^t) , a partir de la población previa (X^{t-1})
2. Los descendientes (U^t) son evaluados y asignados un Fitness (el número de pacientes operados).
3. Cada progenitor es comparado con su descendiente
4. Aquellos descendientes que tengan mejor Fitness que su progenitor, los sustituirán en la nueva generación de la población (X^t), en caso contrario, el progenitor mantendrá su lugar.

4 ALGORITMO Y VARIANTES

4.1 Algoritmo principal

El algoritmo **DDE**, representado en la figura 4-1, consiste en el algoritmo de evolución diferencial principal, donde:

- La población inicial está formada por individuos/secuencias generadas aleatoriamente.
- En cada iteración, se tiene una población actual (X^{t-1}) , de la cual se genera la población mutada (V^t) , y a partir de ella, la población cruzada (U^t).
- Durante el proceso de selección se compara cada miembro de U^t con su progenitor en X^{t-1} para comprobar si el nuevo individuo supera a su progenitor, en cuyo caso lo sustituiría en la población.
- El criterio de selección será el valor de la F.O; el número de pacientes operados y el cálculo del valor de la F.O para cada secuencia se realiza mediante el algoritmo de decodificación DECOD3.
- Las iteraciones continúan hasta que se cumpla el criterio de parada, el cual será un intervalo de tiempo (que se mantendrá constante para todas las variaciones del algoritmo).

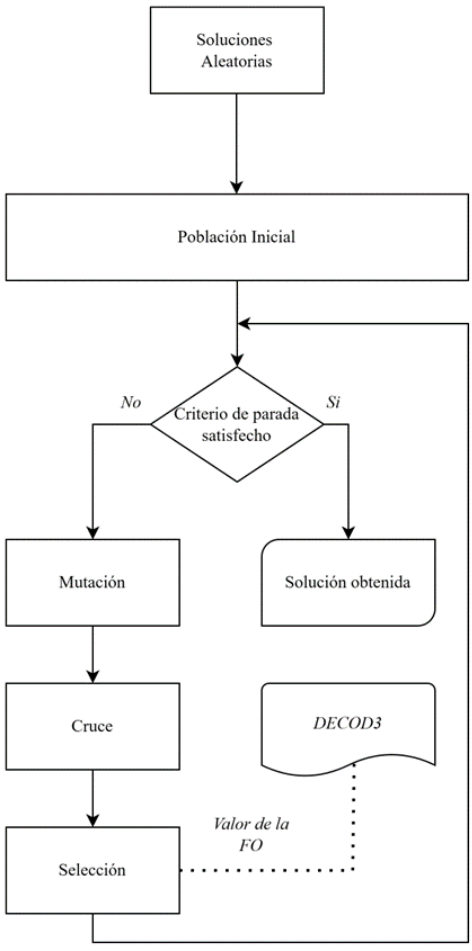

Figura 4-1: *Diagrama del algoritmo principal DDE*

4.2 Variantes de solución inicial

Mientras que el algoritmo principal se basa para su población inicial en soluciones generadas aleatoriamente, se investigará el efecto de combinar estas soluciones con heurísticas constructivas; para ello se desarrollará la variante **DDE-S1**. Estas heurísticas que se añaden a la población inicial, al contrastarse con las soluciones aleatorias, ofrecen un enfoque más estructurado y orientado a mejorar la calidad de las soluciones obtenidas.

Explorar esta variante proporcionará una visión más completa sobre cómo diferentes enfoques iniciales impactan en el rendimiento global del algoritmo. Las heurísticas generadas son:

- SPT (Shortest Processing Time): En el contexto de la programación de cirugías, esta heurística implica asignar las operaciones quirúrgicas con la duración más corta a las salas de operaciones disponibles en primer lugar. Al dar prioridad a las cirugías más cortas, se optimiza la utilización de los recursos disponibles y se mejora la eficiencia del sistema en su conjunto. Además, esta práctica asegura que las cirugías más cortas no se vean retrasadas o interrumpidas por procedimientos más largos, lo que contribuye a un flujo de trabajo más fluido y predecible en el entorno quirúrgico.
- OCC (Orden Creciente Cirujanos): Esta estrategia organiza las cirugías según el especialista quirúrgico responsable de cada una, coordinando la asignación de cirugías por parte del personal médico. El objetivo de esta heurística es reducir los tiempos de cambio entre cirugías, asignando las operaciones de cada cirujano consecutivamente. Esto ayuda a minimizar el impacto de las restricciones que limitan el número de días que un cirujano puede operar en una semana. Al programar todas sus operaciones de manera contigua, se evita la situación en la que una operación no pueda llevarse a cabo debido a que el cirujano ha alcanzado su límite de días disponibles o ORs en los que operar.
- DRO (Disponibilidad de Realizar Operación): Esta heurística se centra en la disponibilidad de realizar operaciones especializadas, otorgando prioridad en la secuencia a aquellos pacientes que requieran una atención quirúrgica en una OR especializada. La premisa fundamental es organizar las operaciones más complejas y desafiantes al principio del programa, con el objetivo de mitigar posibles problemas de asignación a medida que las opciones se reducen. Una vez priorizada las operaciones de OR especializada, se ordenan por SPT en caso de empate.

Para la variante **DDE-S1**, mostrada en la figura 4-2, se usarán las heurísticas SPT, OCC y DRO, rellenando el resto de los individuos de la población con secuencias generadas aleatoriamente.

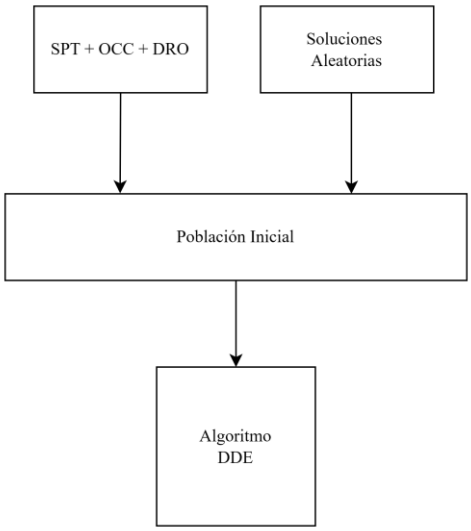

Figura 4-2: *Diagrama del algoritmo* ***DDE-S1***

4.3 Variantes de evolución de la población

A partir del algoritmo principal DDE, se crearán variantes que modifiquen la manera en la que la población evoluciona durante el algoritmo, buscando obtener variantes que puedan centrarse en explorar o intensificar el espacio de búsqueda con mayor fuerza.

Unas de las variantes del algoritmo consisten en implementar un mecanismo de reinicialización periódica de la población, para evitar el estancamiento en óptimos locales donde no realicen avances en la calidad de las soluciones, y su valor de la función objetivo no sea lo suficientemente bueno en comparación con el resto de la población. Después de un cierto número de iteraciones sin mejora, se reinician los individuos que no mejoren, lo que ayuda a revitalizar la búsqueda.

Este enfoque implica monitorear el progreso del algoritmo durante la búsqueda de soluciones y, tras detectar un período prolongado sin mejoras significativas por parte de un individuo de la población, activar la reinicialización del mismo. Este proceso de reinicialización permite actualizar la exploración del espacio de búsqueda, ofreciendo una oportunidad renovada para descubrir soluciones más prometedoras y escapar de los óptimos locales que pueden haber limitado el progreso anteriormente.

Se crearán distintos algoritmos en función de su criterio de reinicialización:

- **DDE-P1:** Los individuos de la población, excepto el mejor individuo de la población, que no mejoren durante X iteraciones se sustituirán por otro individuo generado aleatoriamente.
- **DDE-P2:** Los individuos de la población, excepto los individuos que tengan un valor de la F.O. mayor al promedio de la población o sean mejores que el X% de la población, que no mejoren durante X iteraciones se sustituirán por otro individuo generado aleatoriamente.
- **DDE-P3:** Los individuos de la población, excepto los individuos que tengan un valor de la F.O. mayor al promedio de la población o sean mejores que el X% de la población, que no mejoren durante X iteraciones serán eliminado de la población.

  Este último enfoque implica un proceso de depuración más agresivo, donde los individuos que no logran mostrar mejoras significativas y tienen un bajo fitness son excluidos de la población, reduciendo así la presencia de soluciones subóptimas, favoreciendo la exploración de soluciones más prometedoras mediante la liberación de capacidad computacional.

Las variantes DDE-P1 y DDE-P2 se muestran en la figura 4-3 y DDE-P3 en la figura 4-4, y sus parámetros se establecen durante la calibración *5.2.2 Parámetros de variantes.*

Otra estrategia posible para modificar la evolución del algoritmo es adaptar el tamaño de la población según la instancia. Esta adaptación se lleva a cabo mediante la siguiente variante del algoritmo:

- **DDE-P4:** Su objetivo es ajustar el número de individuos en el grupo en función del número de pacientes en cada instancia. De esta manera, a medida que aumenta el tamaño del espacio de búsqueda (en función de los pacientes en lista de espera), el algoritmo puede explorarlo más exhaustivamente mediante una población más grande. Esta variante permite una exploración más efectiva y precisa, ya que se enfoca en asignar recursos de manera proporcional a la complejidad de cada problema específico, maximizando así la capacidad del algoritmo para encontrar soluciones óptimas en diversos contextos.

  La relación de tamaño de población con número de pacientes se establece mediante calibración en el apartado *5.2.2 Parámetros de variantes.*

Todas las variantes parten del algoritmo principal DDE, y por lo tanto los individuos de la población inicial serán generados completamente aleatorios.

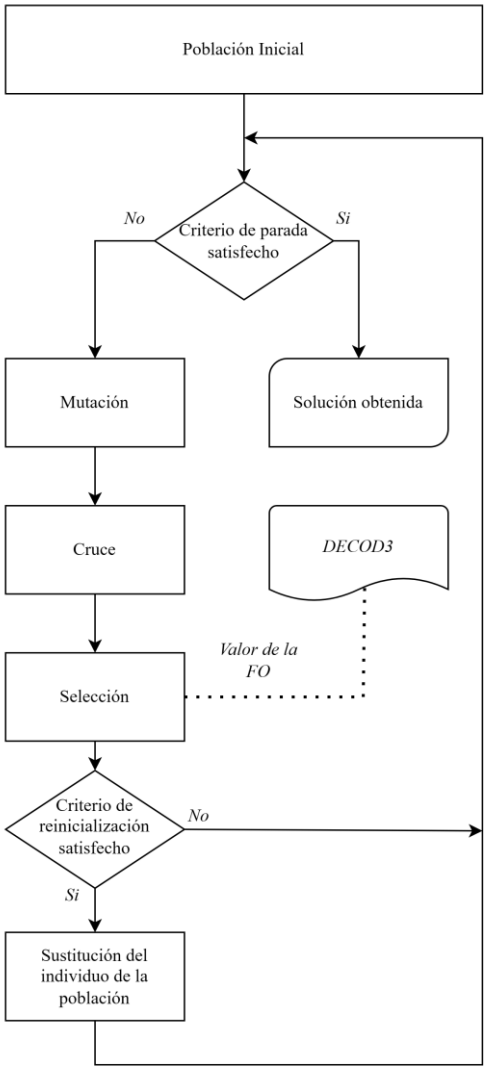

Figura 4-3: *Diagrama del algoritmo* ***DDE-P1 y DDE-P2***

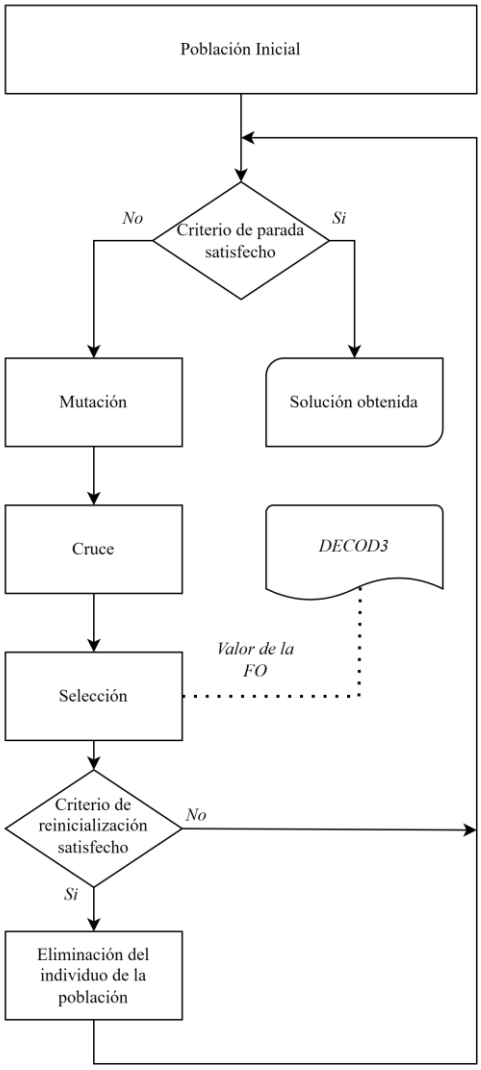

Figura 4-4: *Diagrama del algoritmo* ***DDE-P3***

4.4 Variantes de búsqueda local

La variante **DDE-LS**, ilustrada en la Figura 4-5, emplea el procedimiento de búsqueda local para identificar vecindades que optimicen las soluciones durante la evolución del algoritmo principal DDE. Su propósito radica en evaluar el balance entre el tiempo computacional invertido en la búsqueda local, que aumenta conforme crece el tamaño de las secuencias de pacientes, y la mejora resultante en las soluciones.

Dado el limitado tiempo de cómputo disponible para cada ejecución del algoritmo, se reduce el número de iteraciones realizadas. Por consiguiente, resulta imperativo que la Búsqueda Local garantice mejoras consistentes en cada iteración, para ello:

- A los individuos iniciales se les aplica una Local Search al formar la población inicial. La elección del método de búsqueda local se realiza mediante una calibración que examina los resultados de los distintos métodos en *5.2.3 Elección de búsqueda local.*
- En cada iteración, antes del proceso de selección, se aplica una Local Search a cada individuo de U^t

previo a la comparación su progenitor en X^{t-1}. De esta manera, se busca mejorar a los individuos mutados y recombinados antes de comprobar si pueden ser incluidos en la población.

- El tipo de búsqueda local tendrá un impacto significativo tanto en el tiempo de cómputo como en el grado de mejora. Será necesario calibrar para encontrar la búsqueda que ofrezca el mejor rendimiento, y esta calibración se llevará a cabo en la sección *5.2.2, "Elección de Local Search"*. La elección se realizará entre búsquedas locales de "First Improvement", ya que los tiempos de cómputo de "Best Improvement" no son factibles dado el limitado tiempo total disponible para el algoritmo (para instancias de gran tamaño, ni siquiera puede completar una iteración).

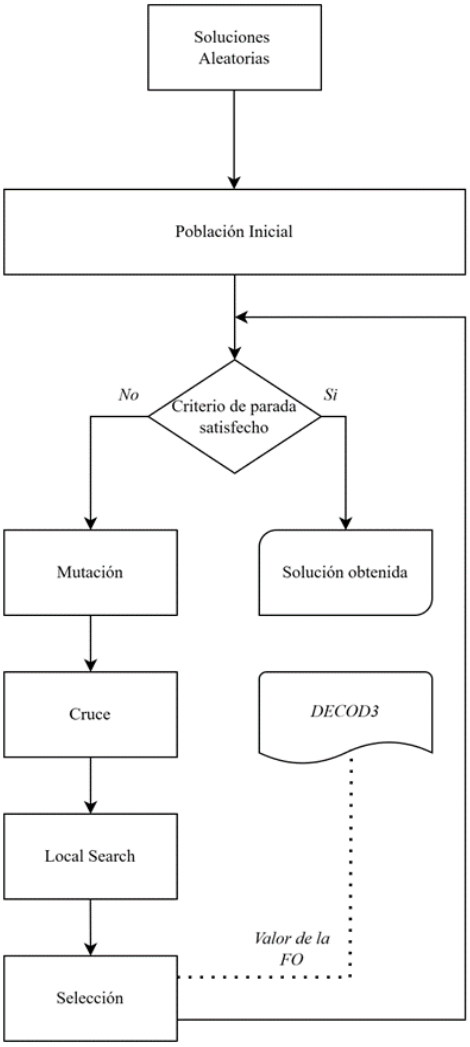

Figura 4-5: *Diagrama del algoritmo* ***DDE-LS***

En la ***Tabla 4-1*** se muestra un resumen de todos los tipos de algoritmo: El algoritmo principal y todas sus variantes:

Tabla 4-1. Tabla resúmen de los tipos de algoritmo

Algoritmos	
DDE	Es la versión principal del algoritmo de evolución diferencial. Parte de soluciones aleatorias. Mediante procesos de mutación y cruce va generando individuos que entrarán en la población (si mejoran a su progenitor).
$DDE - S1$	El algoritmo DDES1 añade una variación en la población inicial, usando heurísticas constructivas de SPT, OCC y DRO junto a soluciones aleatorias.
$DDE - P1$	El algoritmo DDEP1 reinicia a los individuos que no mejoren en un número de iteraciones por otro individuo aleatorio (solo la mejor solución está exenta de ser reinicializada).
$DDE - P2$	El algoritmo DDEP2 reinicia a los individuos que no mejoren en un número de iteraciones por otro individuo aleatorio siempre que su fitness se encuentre por debajo de un porcentaje establecido del fitness del resto de la población.
$DDE - P3$	El algoritmo DDEP3 elimina a los individuos que no mejoren en un número de iteraciones, disminuyendo el tamaño de población a lo largo de las iteraciones.
$DDE - P4$	El algoritmo DDEP4 usa un tamaño de población dinámico, que depende del número de pacientes en la lista de espera de la instancia.
$DDE - LS$	El algoritmo DDELS añade un proceso de búsqueda local en cada iteración a la población mutada antes de ser comparada con sus progenitores.

5 RESOLUCIÓN COMPUTACIONAL

5.1 Generación de instancias

Es necesario establecer los parámetros y valores mediante los cuales se generarán los datos de las instancias que serán usados para la experimentación en los algoritmos de optimización metaheurística. Las instancias están basadas en el estudio realizado por (Molina-Pariente, Hans, et al., 2015):

El horizonte de planificación se hará semanalmente, para programar las operaciones a lo largo de los 5 días laborales que la forman. Cada día tendrá una duración de 8h, de las cuales tanto los cirujanos como las OR estarán disponibles las 8h cada día. El número de OR serán los disponibles por parte de la unidad en el hospital de referencia, siendo las relevantes a este problema 3 OR. También se generarán instancias con 9 OR con objetivos teóricos para comparar resultados. Estos datos quedan representados en la Tabla 5-1.

Tabla 5–1. Datos del horizonte de planificación

Datos		
$\|H\|$	$5\ días$	Períodos del horizonte de planificación
$\|J\|$	$3{,}9\ OR$	Conjunto de quirófanos (ORs)
r_{jh}, a_{sh}	$480\ \frac{minutos}{día}$	Capacidad máxima de OR y cirujanos

El número de pacientes en lista de espera se irán generando de uno en uno hasta que el tiempo de cirugías totales exceda un porcentaje (β) respecto a la capacidad de las OR total disponible a lo largo del horizonte de planificación.

Los tiempos de cirugía (t) siguen una distribución normal logarítmica, cuyos parámetros de entrada serán:

- La duración media esperada de la cirugía (μ), la cual se genera aleatoriamente entre el conjunto {60, 120, 180, 240}.
- La desviación estándar (σ), determinada por el coeficiente de variación el cual se genera aleatoriamente en el intervalo $[0.1\mu \dots 0.5\mu]$.

Estos tiempos de cirugía tienen en cuenta tanto el tiempo necesario para realizar la operación, como el tiempo de preparación previa, limpieza y preparación para la siguiente cirugía.

La fecha máxima de estancia de cada paciente dd_i se genera a partir del número de días máximos antes del tratamiento $(MTBT)$ y el número de días que el paciente lleva en lista de espera (dwl), mediante la expresión:

$$dd_i = MTBT - dwl$$

Donde $MTBT$ se establece aleatoriamente del conjunto {45, 180, 360} fijado por parte de los servicios de atención sanitaria español y dwl se genera aleatoriamente a partir de una distribución discreta uniforme $[1, MTBT - 1]$.

El peso (w) que establece la prioridad de cada paciente se obtiene mediante combinación lineal de los parámetros normalizados de dwl y un valor llamado prioridad médica (mp) que se genera aleatoriamente a partir de una distribución discreta uniforme $[1, 5]$ (siendo 1 menor prioridad y 5 la mayor prioridad) también normalizado. Ambos indicadores tendrán el mismo peso y por lo tanto $a = 0.5$

$$w = a * mp + (1 - a) * dwl$$

El cirujano asociado al paciente (γ_i) se genera aleatoriamente entre el número total de cirujanos. Todos los parámetros relacionados con el paciente se muestran en la tabla 5-2.

Tabla 5–2. Datos de pacientes

Datos		
t_i	$t(x\|\mu,\sigma^2) = \frac{1}{\sigma x\sqrt{2\pi}} e^{-\frac{(\ln(x)-\mu)^2}{2\sigma^2}}$	Duración completa de la operación del paciente i
dd_i	$dd_i = MTBT - dwl$	Fecha máxima de salida del paciente i
w_i	$w = a * mp + (1-a) * dwl$	Peso quirúrgico del paciente i
γ_i	$U[1, \|S\|]$	Cirujano s asociado al paciente i
β	$100\%, 125\%$	Exceso de pacientes i respecto al tiempo total disponible

El número de cirujanos suele ser un valor adaptable y arbitrario, pero para las instancias se calcularán mediante la siguiente expresión:

$$|S| = \alpha \frac{\sum_{j\in J}\sum_{h\in H} r_{jh}}{l * a * mds}$$

Donde:

- α es un valor de control, el cual se puede variar en función del problema
- $\sum_{j\in J}\sum_{h\in H} r_{jh}$ es la capacidad máxima de todas las OR a lo largo del horizonte de planificación.
- l representa el número de semanas que forman el horizonte de planificación.
- a es la capacidad que tienen los cirujanos en un día cualquiera.
- mds el número máximo de días que un cirujano está disponible para realizar operaciones a lo largo de una semana.

Los cirujanos cuentan con un número máximo de OR en las cuales pueden realizar cirugías (u_s). Los parámetros de los cirujanos se muestran en la Tabla 5-3

Tabla 5–3. Datos de cirujanos

Datos		
$\|S\|$	$\|S\| = \alpha \frac{\sum_{j\in J}\sum_{h\in H} r_{jh}}{l * a * mds}$	Número de cirujanos
u_s	$U[1, \|J\|]$	Número de quirófanos en los que puede operar cada cirujano para cada día
mds	3,4	Días h que un cirujano está disponible para realizar operaciones a lo largo de una semana
α	1.5, 2	Valor de control para el calcular el exceso de cirujanos necesarios

La disponibilidad de la OR (δ_{ijh}) depende de si el tipo de cirugía puede realizarse en cualquiera de las OR multifuncionales o necesita de una OR especializada. Para ellos, se parametriza sabiendo que, en la lista de espera, el 90% de las cirugías pueden realizarse en cualquier OR y el 10% necesitan de una OR especializada

(la operación solo podrá hacerse en un ~30% de las OR totales, que son especializadas).

Se realizarán un conjunto de instancias para cada una de las 16 combinaciones entre los siguientes parámetros de la Tabla 5-4, las instancias resultantes se muestran en la Tabla 5-5:

Tabla 5–4. Variables de instancias

Variables de instancias		
$\|J\|$	$3, 9$	Número de OR
mds	$3, 4$	Días h que un cirujano está disponible para realizar operaciones a lo largo de una semana
α	$150\%, 200\%$	Valor de control para el calcular el exceso de cirujanos necesarios
β	$100\%, 125\%$	Exceso de pacientes i respecto al tiempo total disponible

Tabla 5–5. Datos principales de las instancias generadas

	$\boldsymbol{\alpha}$	**β**	**mds**	$\|\boldsymbol{S}\|$	$\|\boldsymbol{J}\|$
0	1.5	1	3	8	3
1	1.5	1.25	3	8	3
2	2	1	3	10	3
3	2	1.25	3	10	3
4	1.5	1	4	6	3
5	1.5	1.25	4	6	3
6	2	1	4	8	3
7	2	1.25	4	8	3
8	1.5	1	3	22	9
9	1.5	1.25	3	22	9
10	2	1	3	30	9
11	2	1.25	3	30	9
12	1.5	1	4	17	9
13	1.5	1.25	4	17	9
14	2	1	4	22	9
15	2	1.25	4	22	9

Con estas combinaciones de parámetros:

- Para la fase de calibración (5.2), se generarán 5 instancias por combinación, resultando en un total de 80 instancias, regenerando las instancias para cada calibración.
- Para la fase de evaluación computacional (5.3), se generarán 15 instancias por combinación, resultando en un total de 240 instancias.

5.2 Calibración

5.2.1 Parámetros del algoritmo

Calibrar los parámetros de un algoritmo de evolución diferencial es fundamental para garantizar su eficacia. Una forma de realizar esta calibración es mediante experimentos empíricos, donde se ejecuta el mismo algoritmo básico con diferentes configuraciones de parámetros en un conjunto diverso de instancias del problema. Al variar los parámetros y evaluar el desempeño del algoritmo en cada configuración, podemos identificar qué combinación de parámetros produce los mejores resultados en términos de calidad de la solución encontrada.

Una vez calibrados los parámetros, podemos comparar de manera justa y significativa diferentes versiones del algoritmo, asegurando que cualquier diferencia en los resultados se deba a las variaciones en el diseño del algoritmo y no a diferencias en la configuración de los parámetros.

Se usará el algoritmo DDE como benchmark, y los parámetros a modificar serán: el tamaño de la población, probabilidad y fuerza de la mutación, y probabilidad de cruce. Se probarán todas las combinaciones de los valores de la Tabla 5-6 para las *80 instancias* generadas con los valores del apartado *5.1 Generación de instancias.*

Todas las combinaciones contarán con el mismo tiempo de cómputo como criterio de parada, para la calibración será de 15 segundos.

Tabla 5–6. Parámetros del algoritmo

Datos		
POB	$5, 20, 50, 100$	Tamaño de la población inicial
F	$40\%, 80\%$	Probabilidad de mutación
Z	$50\%, 100\%$	Factor de escala de la mutación
CR	$40\%, 80\%$	Probabilidad de cruce

Para cada instancia de datos (j), se comparan los resultados de las 32 distintas combinaciones de parámetros (i) del algoritmo mediante ARPD (Average Relative Percentage Deviation), que proporciona una indicación de la desviación relativa respecto al mejor valor de la F.O. obtenido en cada instancia.

$$ARPD_{ij} = \left| \frac{FO_{ij} - \max\left(FO_j\right)}{\max\left(FO_j\right)} \right|$$

Obtenidos los $ARPD_{ij}$, se calcula la media de todos los $ARPD_{ij}$ de todas las instancias (j) para cada combinación de datos (i).

$$ARPD_i = \frac{1}{80} \sum_{j=1}^{80} ARPD_{ij}$$

La combinación de parámetros que mejor rendimiento relativa tenga, será la usada para la simulación de todos los algoritmos. A continuación, se muestran en la Tabla 5-7 los resultados de la calibración:

Tabla 5-7. Resultados de la calibración del algoritmo

POB	F	Z	CR	ARPD promedio
10	0.4	0.5	0.4	2.11%
10	0.4	0.5	0.8	2.62%
10	0.4	1	0.4	2.15%
10	0.4	1	0.8	2.01%
10	0.8	0.5	0.4	2.82%
10	0.8	0.5	0.8	1.24%
10	0.8	1	0.4	2.00%
10	0.8	1	0.8	2.35%
20	0.4	0.5	0.4	2.14%
20	0.4	0.5	0.8	2.91%
20	0.4	1	0.4	2.50%
20	0.4	1	0.8	3.29%
20	0.8	0.5	0.4	2.55%
20	0.8	0.5	0.8	2.27%
20	0.8	1	0.4	2.31%
20	0.8	1	0.8	2.71%
50	0.4	0.5	0.4	2.73%
50	0.4	0.5	0.8	2.36%
50	0.4	1	0.4	2.90%
50	0.4	1	0.8	2.36%
50	0.8	0.5	0.4	2.49%
50	0.8	0.5	0.8	2.78%
50	0.8	1	0.4	3.00%
50	0.8	1	0.8	2.91%
100	0.4	0.5	0.4	3.18%
100	0.4	0.5	0.8	2.83%
100	0.4	1	0.4	2.78%
100	0.4	1	0.8	3.04%
100	0.8	0.5	0.4	2.89%
100	0.8	0.5	0.8	3.44%
100	0.8	1	0.4	3.08%
100	0.8	1	0.8	3.32%
Mejor Combinación				**Mejor ARPD promedio**
10	0.8	0.5	0.8	1.24%
Índices de correlación del ARP promedio y parámetros				
POB = 0.664		F = 0.0188	Z = 0.094	CR = 0.055

El tamaño de población tiene una influencia importante sobre la calidad de las soluciones. Debido al limitado tiempo de cómputo (15s), las poblaciones grandes pese a su gran exploración del espacio de soluciones no son capaces de intensificar lo suficiente para encontrar buenas soluciones. Para explorar este problema y plantear una posible solución, se usará la versión ***DDE-P4,*** con un tamaño de población dinámico en función del número de pacientes en lista de espera.

5.2.2 Parámetros de variantes

Dentro de las distintas variantes del algoritmo, aparecen nuevos parámetros que calibrar, especialmente en las *4.3 Variantes de evolución de la población*, en las cuales los algoritmos ***DDE-P2 y DDE-P3*** llevan a cabo una sustitución o eliminación de aquellos individuos que no mejoren durante X iteraciones y se encuentren por debajo del percentil Y.

Se realizará por lo tanto una calibración para establecer:

1. El número de iteraciones que un individuo puede estar sin mejorar antes de ser sustituido o eliminado.
2. El porcentaje de individuos de la población a los que hay que superar, para que un individuo no sea eliminado pese a no mejorar.

Se usarán las combinaciones de los siguientes parámetros de la Tabla 5-8 para calibrar las variantes de evolución de la población:

Tabla 5-8. Parámetros del algoritmo

Datos	
$Nº\ Iteraciones\ \text{sin}\ mejora$	$10,20,50,100$
$Percentil\ de\ superioridad$	$50\%,70\%,90\%$

Aplicando todas las combinaciones a *80 instancias* generadas a partir de los datos de *5.1 Generación de instancias*, dando un tiempo de cómputo de 15s a cada una, se obtienen los siguientes resultados promedios, mostrados en la Tabla 5-9:

Tabla 5-9. Resultados de la calibración de parámetros de las variantes

Nº Iteraciones sin mejora	Percentil de superioridad	**ARPD promedio**
10	0.5	2.41%
10	0.7	0.21%
10	0.9	1.74%
20	0.5	1.44%
20	0.7	1.47%
20	0.9	1.67%
50	0.5	1.47%
50	0.7	1.46%
50	0.9	1.35%
100	0.5	2.22%
100	0.7	1.33%
100	0.9	1.26%
Mejor Combinación		**Mejor ARPD promedio**
10	0.7	0.21%

La mejor combinación resulta de usar **10 iteraciones sin mejora** como límite antes de sustituir o eliminar al individuo, siempre y cuando, sea inferior al **70%** de la población. Esta combinación de parámetros se usará para los algoritmos ***DDE-P2 y DDE-P3.***

Para el algoritmo ***DDE-P4,*** cuya característica principal es el tamaño de población adaptado a cada instancia en función del número de pacientes a analizar, también se realizará una calibración que permita analizar valores centrados en la exploración, valores centrados en la intensificación y valores intermedios, para ellos se realiza la calibración en función de los siguientes parámetros de la Tabla 5-10:

Tabla 5-10. Parámetros del algoritmo

Datos			
$Tamaño\ de\ población$	$\frac{Nº\ pacientes}{20}$,	$\frac{Nº\ pacientes}{10}$,	$\frac{Nº\ pacientes}{5}$
	$\frac{Nº\ pacientes}{2}$,	$Nº\ pacientes$,	$2 * Nºpacientes$

Aplicando todas las combinaciones a *80 instancias* distintas, dando un tiempo de cómputo de 15s a cada una, se obtienen los siguientes resultados promedios mostrados en la Tabla 5-11:

Tabla 5-11. Resultados de la calibración de parámetros de las variantes

Tamaño de población	**ARPD promedio**
$\frac{Nº\ pacientes}{20}$	1.7934%
$\frac{Nº\ pacientes}{10}$	1.6710%
$\frac{Nº\ pacientes}{5}$	1.3902%
$\frac{Nº\ pacientes}{2}$	2.2554%
$\frac{Nº\ pacientes}{1}$	2.5616%
$2 * Nº\ pacientes$	2.6076%
Mejor tamaño	**Mejor ARPD promedio**
$\frac{Nº\ pacientes}{5}$	1.3902%

El tamaño de la población dinámico que se usará para el algoritmo ***DDE-P4*** será de $POB = \frac{Nº\ pacientes}{5}$. Tras la calibración parece ser el tamaño de población que equilibra la exploración e intensificación de búsqueda del algoritmo de evolución diferencial.

En tamaños de población muy grandes ($2 * Nº\ pacientes$) el espacio es demasiado grande y no hay tiempo de cómputo suficiente para intensificar en cada una de las posibles soluciones. En tamaños pequeños ($\frac{Nº\ pacientes}{20}$) pese a que, si se puede intensificar en las soluciones, no son lo suficientemente variadas para encontrar consistentemente buenas soluciones.

5.2.3 Elección de Local Search

Seleccionar el método de búsqueda local más eficiente en términos de tiempo de cómputo es crucial, dado que se está utilizando tiempo en realizar búsquedas locales, sacrificando el número total de iteraciones que se podríoa realizar.

El criterio de parada establecido de First Improvement permite detener la búsqueda local tan pronto como se encuentre vecindad con mejor fitness. Esto evita explorar innecesariamente el espacio de soluciones y reduce significativamente el tiempo de ejecución del algoritmo de búsqueda local. Dado que el proceso de búsqueda

local se repite en cada iteración para cada individuo de la población, este enfoque es especialmente eficaz.

Las búsquedas locales que se analizarán son las siguientes:

1. *General Swap:* Se trata de un procedimiento de búsqueda local que se basa en intercambiar dos elementos de manera arbitraria dentro de una solución. Este intercambio tiene como objetivo evaluar si la modificación efectuada mejora la solución actual. La efectividad de esta estrategia está estrechamente ligada a la calidad de la vecindad que se genera a partir de estos intercambios.
2. *Adjacent Swap:* Esta técnica de búsqueda implica intercambiar dos elementos que se encuentran adyacentes dentro de una solución. Al realizar estos intercambios, se explora el espacio de soluciones vecinas. Este enfoque se caracteriza por la capacidad de encontrar soluciones óptimas locales, ya que mejora gradualmente la solución actual mediante intercambios simples.
3. *3-opt*: En esta estrategia, se lleva a cabo la reorganización de la secuencia al sustituir tres segmentos contiguos por otros tres segmentos. Esta técnica permite explorar diferentes combinaciones dentro de la solución, lo que puede conducir a mejoras significativas en la calidad de la solución.
4. *Insertion:* Consiste en insertar un elemento en una posición específica dentro de la secuencia. Esta inserción se realiza con el propósito de evaluar si mejora la calidad de la planificación general. Al considerar la inserción de elementos en puntos estratégicos, se busca optimizar la disposición de la secuencia para obtener una solución más efectiva.

Para comparar los métodos de búsqueda local, generaremos 80 instancias aleatorias con número de pacientes entre 40 y 300.

Aplicaremos todos los métodos de búsqueda local a las 80 instancias, registrando su tiempo de cómputo con el criterio de First Improvement para mejorar una misma secuencia aleatoria. Una vez obtenidas las medidas de tiempo para cada método, compararemos los resultados, los cuales aparecen en la Tabla 5-12. Seleccionaremos el método de búsqueda local con el menor tiempo promedio para todo tamaño de secuencia de pacientes en encontrar una mejor solución.

Tabla 5-12. Resultados de selección de LS

Método	Tiempo en encontrar mejora (segundos)			
	Promedio	P_{50}	P_{85}	P_{95}
General Swap	0.5054151	0.161478	0.63921365	1.33307
Adjacent Swap	0.62929228	0.181023	0.9815999	3.33055
3opt	0.90280567	0.1468095	0.9747449	2.30364
Insertion	3.07118652	0.2529665	2.107182	16.7037

Pese a que el método de búsqueda local ***3-opt,*** es capaz de obtener los tiempos más bajos, es inconsistente, ya que existen casos en los que se demora hasta 30s para secuencias de gran tamaño. El método ***General Swap*** es más estable, evitando instancias puntuales donde se atasque, y consistentemente obteniendo buenos tiempos de cómputo.

Por lo tanto, el método de búsqueda local que se usará será el de ***General Swap.***

6 RESULTADOS

6.1 Evaluación computacional de las metaheurísticas

Se generarán los datos a partir de los parámetros establecidos en *5.1 Generación de instancias*, creando 30 instancias de cada una de las 8 posibles combinaciones de parámetros, resultando en un total de 240 instancias, las cuales se guardan en una hoja de cálculos de Excel: "Datos.xlsx", la cual está dividida en 240 hojas de trabajo para cada instancia generada. Los algoritmos desarrollados en los apartados 4.1, 4.2, 4.3 y 4.4 se ejecutarán para cada instancia durante un tiempo de cómputo de 15s para cada algoritmo.

Tabla 6-1. Resultados ARPD de la evaluación computacional de los algoritmos

Instancia	Pacientes	DEE	DDE LS	DDE SOL	DDE P1	DDE P2	DDE P3	DDE P4	Instancia	Pacientes	DEE	DDE LS	DDE SOL	DDE P1	DDE P2	DDE P3	DDE P4
1	44	0.0%	4.8%	0.0%	0.0%	0.0%	4.8%	0.0%	121	151	3.1%	7.6%	0.0%	3.1%	2.3%	4.6%	4.6%
2	36	0.0%	6.1%	0.0%	0.0%	0.0%	0.0%	0.0%	122	142	6.9%	10.0%	0.0%	7.7%	7.7%	4.6%	7.7%
3	53	0.0%	10.0%	0.0%	2.0%	0.0%	6.0%	0.0%	123	146	3.8%	9.9%	1.5%	4.6%	4.6%	0.0%	4.6%
4	46	0.0%	7.3%	0.0%	0.0%	0.0%	4.9%	0.0%	124	144	2.4%	7.2%	0.0%	1.6%	4.0%	2.4%	4.0%
5	43	2.4%	9.5%	2.4%	0.0%	4.8%	7.1%	2.4%	125	140	0.0%	5.8%	0.0%	0.0%	1.7%	1.7%	1.7%
6	49	4.2%	8.3%	4.2%	4.2%	0.0%	4.2%	2.1%	126	140	2.4%	8.1%	0.0%	2.4%	3.3%	4.1%	4.1%
7	47	4.4%	8.9%	2.2%	2.2%	2.2%	0.0%	2.2%	127	143	6.3%	8.7%	0.0%	4.8%	7.1%	7.1%	4.0%
8	52	0.0%	10.2%	0.0%	0.0%	0.0%	4.1%	0.0%	128	133	1.7%	5.1%	0.0%	0.8%	5.9%	5.1%	4.2%
9	44	0.0%	4.9%	0.0%	0.0%	0.0%	4.9%	2.4%	129	139	5.6%	9.6%	0.0%	4.8%	4.0%	4.0%	4.0%
10	47	2.2%	8.9%	2.2%	0.0%	0.0%	6.7%	0.0%	130	149	1.5%	4.6%	0.0%	3.1%	3.8%	3.8%	4.6%
11	53	2.0%	10.2%	2.0%	4.1%	0.0%	8.2%	2.0%	131	145	4.7%	8.6%	0.0%	2.3%	3.9%	4.7%	3.9%
12	48	0.0%	8.9%	0.0%	0.0%	0.0%	4.4%	0.0%	132	156	0.8%	3.8%	0.8%	0.0%	1.5%	1.5%	2.3%
13	42	0.0%	7.3%	2.4%	2.4%	0.0%	7.3%	2.4%	133	138	2.5%	7.5%	0.0%	3.3%	3.3%	5.0%	4.2%
14	54	3.8%	11.5%	0.0%	1.9%	1.9%	9.6%	5.8%	134	138	2.5%	4.1%	0.0%	0.8%	1.6%	4.9%	2.5%
15	49	0.0%	10.9%	0.0%	2.2%	2.2%	4.3%	0.0%	135	152	5.1%	10.2%	0.0%	5.8%	6.6%	4.4%	6.6%
16	61	2.0%	8.0%	0.0%	2.0%	0.0%	6.0%	2.0%	136	180	5.6%	10.6%	0.0%	6.3%	8.5%	7.0%	7.7%
17	55	0.0%	6.5%	0.0%	2.2%	0.0%	4.3%	0.0%	137	184	4.9%	9.0%	0.0%	6.9%	8.3%	5.6%	7.6%
18	61	0.0%	9.8%	0.0%	3.9%	2.0%	5.9%	2.0%	138	185	7.6%	12.5%	0.0%	6.3%	6.3%	6.3%	4.9%
19	66	0.0%	17.5%	0.0%	5.3%	5.3%	14.0%	5.3%	139	174	9.2%	12.1%	0.0%	9.2%	7.1%	8.5%	9.2%
20	58	2.1%	6.3%	0.0%	0.0%	0.0%	2.1%	0.0%	140	165	7.4%	11.0%	0.0%	9.6%	7.4%	7.4%	5.9%
21	62	0.0%	9.3%	1.9%	3.7%	3.7%	9.3%	3.7%	141	168	8.0%	10.2%	0.0%	8.0%	6.6%	7.3%	9.5%
22	70	3.4%	15.5%	0.0%	3.4%	1.7%	3.4%	1.7%	142	177	4.3%	7.2%	0.0%	3.6%	5.0%	6.5%	3.6%
23	53	2.2%	10.9%	2.2%	0.0%	4.3%	10.9%	0.0%	143	172	8.5%	9.9%	0.0%	8.5%	10.6%	8.5%	8.5%
24	64	3.8%	13.2%	3.8%	1.9%	1.9%	7.5%	0.0%	144	168	5.9%	11.1%	0.0%	7.4%	1.5%	4.4%	6.7%
25	59	2.0%	13.7%	2.0%	5.9%	0.0%	5.9%	0.0%	145	173	3.8%	6.8%	0.0%	4.5%	4.5%	3.8%	1.5%
26	57	2.1%	6.4%	2.1%	0.0%	0.0%	4.3%	0.0%	146	181	10.3%	13.8%	0.0%	7.6%	8.3%	11.0%	7.6%
27	61	0.0%	7.8%	0.0%	0.0%	0.0%	3.9%	0.0%	147	183	2.8%	5.0%	0.0%	4.3%	5.0%	5.0%	4.3%
28	60	2.0%	8.0%	0.0%	2.0%	2.0%	6.0%	0.0%	148	182	4.3%	7.1%	0.0%	2.8%	5.7%	0.7%	2.1%
29	59	0.0%	10.2%	0.0%	0.0%	0.0%	2.0%	0.0%	149	179	6.3%	12.6%	0.0%	7.7%	7.0%	5.6%	9.8%
30	59	0.0%	8.2%	2.0%	0.0%	0.0%	6.1%	0.0%	150	170	4.4%	5.9%	0.0%	3.7%	5.1%	3.7%	5.1%
31	51	2.0%	10.2%	2.0%	2.0%	0.0%	2.0%	2.0%	151	139	0.8%	3.1%	0.8%	0.0%	2.3%	2.3%	1.6%
32	56	1.9%	7.4%	0.0%	3.7%	0.0%	1.9%	0.0%	152	146	1.5%	6.6%	0.0%	2.2%	2.2%	1.5%	1.5%
33	45	0.0%	4.5%	2.3%	2.3%	2.3%	2.3%	0.0%	153	141	0.0%	4.6%	1.5%	0.8%	0.0%	3.1%	2.3%
34	43	0.0%	7.3%	0.0%	0.0%	0.0%	2.4%	0.0%	154	149	2.9%	6.4%	0.0%	2.1%	2.9%	2.9%	2.9%
35	48	0.0%	6.5%	0.0%	0.0%	0.0%	4.3%	0.0%	155	132	1.6%	3.3%	1.6%	0.0%	1.6%	2.4%	0.8%
36	42	2.4%	4.9%	0.0%	2.4%	2.4%	2.4%	0.0%	156	135	0.0%	4.0%	0.0%	0.0%	0.8%	0.8%	0.8%
37	45	2.3%	4.5%	0.0%	0.0%	2.3%	2.3%	2.3%	157	133	2.4%	7.1%	0.0%	3.2%	1.6%	3.2%	3.2%
38	48	0.0%	8.7%	0.0%	0.0%	0.0%	4.3%	0.0%	158	148	3.0%	5.9%	0.0%	3.0%	0.7%	1.5%	0.7%
39	46	0.0%	4.5%	0.0%	0.0%	0.0%	2.3%	0.0%	159	147	0.0%	3.0%	0.0%	0.8%	1.5%	0.8%	2.3%
40	44	0.0%	9.5%	0.0%	2.4%	0.0%	4.8%	2.4%	160	134	4.0%	3.2%	0.0%	2.4%	4.0%	2.4%	4.8%
41	50	2.1%	6.3%	2.1%	2.1%	0.0%	2.1%	0.0%	161	144	2.9%	7.4%	0.0%	2.9%	2.9%	5.1%	2.9%
42	45	0.0%	6.8%	0.0%	2.3%	2.3%	4.5%	0.0%	162	150	5.6%	8.5%	0.0%	4.2%	2.1%	7.0%	5.6%
43	47	0.0%	11.1%	2.2%	0.0%	0.0%	2.2%	2.2%	163	159	2.1%	6.2%	0.0%	3.4%	2.8%	2.8%	3.4%
44	43	0.0%	7.1%	0.0%	2.4%	0.0%	2.4%	0.0%	164	149	2.9%	5.1%	2.2%	0.7%	0.0%	2.9%	1.4%
45	51	2.0%	6.1%	0.0%	2.0%	0.0%	4.1%	2.0%	165	138	2.3%	7.6%	0.0%	3.1%	2.3%	3.8%	2.3%
46	59	2.0%	5.9%	2.0%	2.0%	0.0%	5.9%	2.0%	166	185	2.0%	6.0%	0.0%	0.7%	1.3%	2.6%	2.0%
47	65	0.0%	3.6%	0.0%	0.0%	0.0%	1.8%	0.0%	167	175	3.5%	7.0%	0.0%	3.5%	2.8%	1.4%	2.8%
48	58	0.0%	12.0%	2.0%	2.0%	4.0%	6.0%	2.0%	168	183	0.0%	3.4%	0.7%	0.7%	1.4%	0.0%	0.0%
49	58	2.0%	8.0%	2.0%	2.0%	0.0%	4.0%	2.0%	169	177	4.1%	6.8%	0.0%	1.4%	2.7%	1.4%	2.1%
50	57	0.0%	8.3%	0.0%	0.0%	2.1%	4.2%	2.1%	170	176	2.8%	4.2%	0.0%	0.7%	2.1%	2.1%	0.7%

51	67	0.0%	5.4%	0.0%	1.8%	0.0%	3.6%	0.0%	171	189	2.6%	7.7%	0.0%	2.6%	3.9%	1.9%	1.9%
52	60	1.9%	3.8%	0.0%	0.0%	1.9%	3.8%	0.0%	172	173	3.5%	6.3%	0.0%	4.2%	2.8%	3.5%	3.5%
53	65	0.0%	5.5%	1.8%	0.0%	0.0%	1.8%	1.8%	173	174	0.0%	4.9%	0.0%	1.4%	1.4%	2.1%	2.8%
54	57	0.0%	6.3%	0.0%	0.0%	0.0%	0.0%	0.0%	174	176	3.4%	6.8%	0.0%	3.4%	2.7%	2.7%	2.7%
55	56	0.0%	8.2%	2.0%	0.0%	0.0%	4.1%	2.0%	175	191	6.3%	11.3%	0.0%	3.8%	5.0%	3.8%	5.0%
56	62	1.9%	11.3%	0.0%	0.0%	0.0%	3.8%	1.9%	176	173	4.2%	6.3%	0.0%	3.5%	4.2%	3.5%	4.2%
57	66	0.0%	7.3%	0.0%	0.0%	0.0%	1.8%	1.8%	177	182	3.3%	8.5%	0.0%	3.3%	2.0%	2.0%	4.6%
58	59	1.9%	7.7%	0.0%	1.9%	1.9%	5.8%	0.0%	178	185	2.6%	5.3%	0.0%	3.3%	4.6%	3.9%	3.9%
59	61	2.0%	5.9%	2.0%	0.0%	2.0%	5.9%	2.0%	179	183	4.0%	7.3%	0.0%	2.6%	4.6%	3.3%	3.3%
60	60	1.9%	9.6%	0.0%	1.9%	0.0%	1.9%	1.9%	180	190	0.6%	7.7%	0.0%	3.2%	3.9%	2.6%	2.6%
61	52	2.0%	5.9%	2.0%	0.0%	0.0%	2.0%	2.0%	181	147	0.7%	3.6%	0.7%	0.0%	0.0%	1.5%	0.7%
62	48	0.0%	4.3%	0.0%	0.0%	0.0%	2.1%	0.0%	182	133	1.6%	4.0%	0.8%	0.8%	0.0%	0.0%	0.8%
63	47	0.0%	4.4%	0.0%	0.0%	0.0%	2.2%	2.2%	183	133	0.8%	4.9%	0.0%	0.0%	0.0%	1.6%	0.8%
64	50	2.0%	6.1%	2.0%	0.0%	0.0%	2.0%	0.0%	184	141	0.0%	4.5%	2.3%	1.5%	1.5%	0.8%	1.5%
65	48	0.0%	6.4%	0.0%	0.0%	0.0%	2.1%	2.1%	185	140	0.0%	3.1%	2.3%	1.5%	0.8%	1.5%	0.8%
66	46	0.0%	4.4%	2.2%	2.2%	2.2%	4.4%	2.2%	186	146	0.0%	3.7%	0.0%	0.0%	1.5%	0.7%	2.2%
67	48	0.0%	2.2%	0.0%	0.0%	0.0%	2.2%	0.0%	187	144	0.0%	6.0%	0.0%	0.0%	1.5%	0.7%	1.5%
68	42	0.0%	4.9%	0.0%	0.0%	0.0%	2.4%	0.0%	188	136	0.0%	3.1%	0.0%	0.8%	0.8%	0.0%	0.0%
69	41	0.0%	7.5%	0.0%	0.0%	0.0%	2.5%	0.0%	189	134	0.0%	4.8%	0.0%	1.6%	0.8%	1.6%	1.6%
70	52	0.0%	3.9%	0.0%	2.0%	0.0%	2.0%	0.0%	190	139	3.1%	6.1%	0.8%	0.8%	0.0%	0.8%	2.3%
71	42	0.0%	4.9%	0.0%	0.0%	0.0%	2.4%	0.0%	191	135	1.6%	2.4%	0.0%	0.8%	0.0%	0.0%	0.0%
72	44	0.0%	2.3%	2.3%	0.0%	0.0%	2.3%	0.0%	192	138	0.0%	3.1%	2.3%	1.6%	0.8%	0.0%	1.6%
73	47	2.2%	4.3%	0.0%	2.2%	0.0%	4.3%	2.2%	193	140	0.8%	3.8%	0.8%	0.0%	1.5%	1.5%	0.0%
74	51	0.0%	6.1%	0.0%	0.0%	2.0%	4.1%	0.0%	194	131	0.8%	4.1%	0.8%	0.0%	0.0%	1.6%	0.0%
75	54	0.0%	7.7%	0.0%	1.9%	0.0%	1.9%	0.0%	195	134	2.3%	3.9%	3.1%	0.0%	3.9%	3.9%	2.3%
76	60	0.0%	5.8%	1.9%	0.0%	1.9%	1.9%	1.9%	196	183	1.3%	7.9%	0.0%	0.7%	2.0%	1.3%	4.0%
77	60	0.0%	5.8%	0.0%	0.0%	0.0%	1.9%	0.0%	197	186	1.3%	5.3%	1.3%	0.0%	2.0%	2.0%	0.7%
78	61	0.0%	5.8%	0.0%	0.0%	0.0%	3.8%	0.0%	198	162	0.0%	3.8%	0.0%	0.0%	0.0%	0.8%	1.5%
79	62	0.0%	3.8%	1.9%	0.0%	0.0%	3.8%	0.0%	199	174	2.8%	5.6%	0.0%	2.8%	2.8%	2.8%	2.1%
80	62	0.0%	5.7%	0.0%	1.9%	0.0%	3.8%	0.0%	200	190	3.1%	8.1%	0.0%	3.8%	3.8%	1.9%	3.1%
81	62	1.9%	7.5%	0.0%	0.0%	0.0%	3.8%	1.9%	201	174	2.1%	5.6%	0.0%	0.7%	0.7%	1.4%	1.4%
82	61	0.0%	7.7%	0.0%	0.0%	0.0%	0.0%	0.0%	202	200	0.0%	4.9%	0.0%	3.0%	1.2%	3.7%	3.0%
83	63	0.0%	7.4%	0.0%	0.0%	1.9%	3.7%	1.9%	203	183	0.6%	5.2%	0.6%	0.0%	0.0%	1.3%	0.0%
84	65	0.0%	3.5%	1.8%	1.8%	0.0%	3.5%	1.8%	204	186	3.3%	5.9%	1.3%	2.6%	2.0%	0.0%	2.0%
85	54	0.0%	8.5%	0.0%	0.0%	2.1%	2.1%	2.1%	205	170	2.1%	5.7%	2.1%	0.0%	0.0%	1.4%	2.1%
86	63	1.8%	5.5%	1.8%	0.0%	0.0%	3.6%	1.8%	206	182	3.9%	4.6%	0.0%	4.6%	5.3%	5.9%	3.9%
87	63	1.9%	7.4%	0.0%	0.0%	0.0%	1.9%	0.0%	207	180	2.0%	6.1%	0.0%	0.0%	2.0%	1.4%	0.0%
88	62	0.0%	3.7%	0.0%	1.9%	0.0%	3.7%	0.0%	208	186	1.3%	4.6%	0.0%	0.0%	1.3%	0.0%	2.0%
89	56	2.0%	6.1%	2.0%	2.0%	0.0%	6.1%	2.0%	209	178	2.7%	2.7%	0.0%	1.4%	2.0%	4.1%	3.4%
90	66	1.7%	8.6%	1.7%	0.0%	0.0%	5.2%	1.7%	210	172	0.7%	5.0%	0.0%	2.8%	1.4%	0.7%	1.4%
91	54	0.0%	1.9%	0.0%	0.0%	0.0%	0.0%	0.0%	211	138	0.0%	3.8%	0.8%	0.8%	0.8%	1.5%	0.8%
92	46	0.0%	4.4%	0.0%	0.0%	0.0%	0.0%	0.0%	212	131	0.0%	1.6%	0.8%	0.8%	0.0%	0.8%	0.8%
93	48	0.0%	2.1%	0.0%	0.0%	0.0%	0.0%	0.0%	213	130	0.8%	3.2%	0.8%	0.0%	0.0%	0.8%	0.0%
94	44	0.0%	4.7%	2.3%	0.0%	0.0%	2.3%	0.0%	214	149	0.0%	2.8%	0.0%	0.7%	0.7%	1.4%	0.7%
95	45	2.2%	6.7%	2.2%	0.0%	2.2%	2.2%	2.2%	215	155	0.7%	4.0%	0.0%	0.7%	0.7%	1.3%	0.7%
96	53	1.9%	5.8%	0.0%	1.9%	1.9%	1.9%	0.0%	216	144	0.0%	2.2%	0.0%	0.0%	0.0%	0.0%	0.7%
97	46	0.0%	2.2%	0.0%	0.0%	0.0%	0.0%	0.0%	217	141	0.0%	2.2%	0.7%	1.5%	0.7%	1.5%	0.7%
98	47	0.0%	0.0%	0.0%	0.0%	0.0%	2.2%	0.0%	218	147	0.0%	2.8%	0.0%	0.7%	1.4%	1.4%	0.7%
99	55	0.0%	3.7%	0.0%	0.0%	0.0%	1.9%	0.0%	219	137	0.8%	3.8%	0.8%	0.8%	0.8%	0.0%	1.5%
100	50	2.0%	4.0%	0.0%	2.0%	2.0%	2.0%	2.0%	220	146	1.4%	2.9%	1.4%	0.7%	0.0%	1.4%	0.7%
101	47	0.0%	4.3%	0.0%	0.0%	0.0%	2.2%	0.0%	221	139	0.8%	3.8%	0.0%	1.5%	1.5%	0.8%	0.8%
102	43	2.3%	2.3%	2.3%	2.3%	0.0%	2.3%	0.0%	222	142	0.7%	2.2%	1.5%	0.7%	1.5%	1.5%	0.0%
103	48	0.0%	2.1%	0.0%	0.0%	0.0%	0.0%	0.0%	223	147	0.0%	3.5%	0.7%	1.4%	0.0%	1.4%	0.7%
104	45	0.0%	4.5%	0.0%	0.0%	0.0%	4.5%	2.3%	224	131	0.8%	4.0%	0.8%	0.0%	0.0%	1.6%	0.8%
105	49	0.0%	6.3%	0.0%	0.0%	0.0%	2.1%	0.0%	225	146	0.7%	4.3%	0.7%	0.7%	0.0%	0.7%	1.4%
106	62	1.9%	5.6%	0.0%	0.0%	0.0%	1.9%	0.0%	226	179	2.0%	2.6%	1.3%	1.3%	1.3%	0.0%	2.6%
107	61	0.0%	5.5%	1.8%	1.8%	1.8%	3.6%	1.8%	227	176	0.7%	2.0%	0.0%	0.7%	1.4%	1.4%	1.4%
108	64	0.0%	5.4%	0.0%	0.0%	0.0%	1.8%	0.0%	228	179	0.0%	2.7%	0.7%	0.7%	0.7%	0.7%	1.3%
109	60	1.9%	3.8%	0.0%	1.9%	0.0%	3.8%	0.0%	229	189	0.6%	1.3%	0.6%	0.6%	0.0%	1.3%	0.0%
110	70	0.0%	8.2%	1.6%	1.6%	1.6%	3.3%	1.6%	230	172	1.4%	4.1%	0.0%	2.1%	0.0%	2.1%	2.1%
111	60	0.0%	3.9%	0.0%	0.0%	0.0%	2.0%	0.0%	231	172	0.7%	3.4%	1.4%	0.7%	0.0%	2.1%	0.7%
112	59	0.0%	5.9%	2.0%	2.0%	2.0%	3.9%	0.0%	232	181	1.9%	3.9%	0.0%	1.3%	0.6%	1.3%	1.3%
113	57	2.0%	5.9%	0.0%	0.0%	2.0%	2.0%	0.0%	233	182	0.7%	3.9%	0.0%	1.3%	2.0%	0.0%	0.7%
114	60	1.9%	5.7%	1.9%	1.9%	1.9%	3.8%	0.0%	234	165	0.0%	2.1%	0.7%	0.7%	1.4%	0.7%	0.7%
115	61	1.9%	7.4%	1.9%	1.9%	0.0%	5.6%	1.9%	235	181	1.9%	4.5%	0.0%	1.3%	1.3%	3.2%	2.6%
116	53	2.1%	6.3%	2.1%	0.0%	4.2%	4.2%	2.1%	236	181	3.8%	6.4%	0.0%	3.8%	2.6%	2.6%	4.5%
117	58	0.0%	4.0%	2.0%	0.0%	0.0%	4.0%	0.0%	237	182	2.0%	4.6%	0.0%	1.3%	1.3%	1.3%	2.6%
118	60	0.0%	3.8%	0.0%	0.0%	0.0%	1.9%	0.0%	238	161	0.8%	3.0%	0.0%	0.0%	0.8%	0.0%	0.8%
119	58	0.0%	4.0%	2.0%	2.0%	2.0%	2.0%	0.0%	239	188	1.9%	3.1%	0.0%	1.9%	1.2%	1.2%	1.2%
120	66	1.8%	5.3%	0.0%	1.8%	0.0%	1.8%	0.0%	240	185	0.6%	3.2%	0.0%	1.3%	1.3%	1.3%	2.5%

6.2 Análisis de los resultados

Finalizado el proceso de simulación, se analizarán los resultados obtenidos, profundizando en todos los elementos que puedan aportar conclusiones valiosas.

A partir de la tabla de valores obtenidas en la evaluación computacional (Tabla 6-1), donde se han obtenido los valores ARPD de cada algoritmo en cada instancia, se calcula en la Tabla 6-2 los resultados promedios, donde se exponen los valores promedios de la desviación porcentual promedio relativa (ARPD) de las 240 instancias para cada algoritmo, obtenidos al calcular el promedio de $ARPD_{ij} = \left|\frac{FO_{ij}-\max(FO_j)}{\max(FO_j)}\right|$.

Tabla 6-2. Resultados del ARPD promedio en la evaluación computacional de los algoritmos

ARPD						
DDE	DDE-S1	DDE-LS	DDE-P1	DDE-P2	DDE-P3	DDE-P4
1.6014%	0.5569%	6.1404%	1.6221%	1.5801%	3.0492%	1.7589%

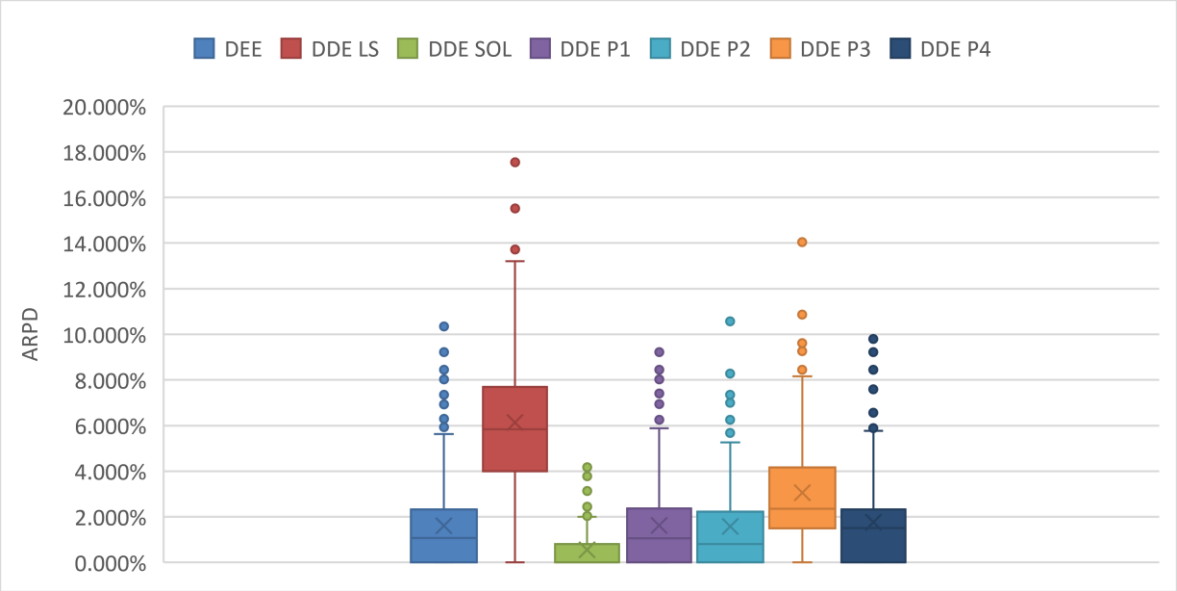

Figura 6-1: *Diagrama de cajas de los valores ARPD de los algoritmos*

Comparando los ARPD de los distintos algoritmos, se puede apreciar en la Figura 6-1 y Tabla 6-2, que solo las variaciones **DDE-S1** (Añadir heurísticas a la población inicial) y **DDE-P2** (Reinicializar los individuos de la población con mal fitness y que no mejoran) superan a la versión básica del algoritmo de evolución diferencial discreta **DDE.**

El algoritmo con Local Search, **DDE-LS**, ha sido el de peor rendimiento, probablemente debido a que el tiempo de cómputo que pierde durante la búsqueda local no mejora lo suficiente para que merezca la pena sacrificar la cantidad de iteraciones que el algoritmo podría haber realizado en ese tiempo.

En las variaciones donde se modifican las características de evolución de la población a lo largo de las iteraciones, destaca el mal rendimiento de **DDE-P3** (el cual eliminaba directamente de la población a todo miembro con mal fitness y que se quedase estancado en un valor de la función objetivo). El resto de los algoritmos parecen tener un rendimiento parecido al algoritmo principal. En la figura 6-2, se muestran la cantidad de veces que cada algoritmo ha obtenido respecto a los demás el mejor y peor valor de la F.O:

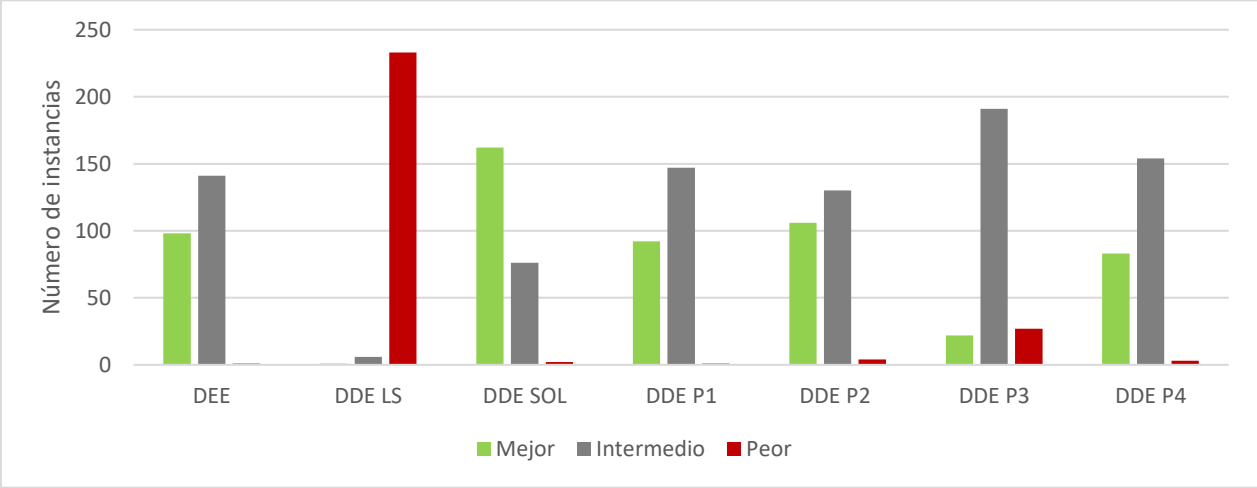

Figura 6-2: *Diagrama de mejores y peores rendimientos de los algoritmos*

Analizando el coeficiente de relación de Pearson en la Tabla 6-3 entre los valores del ARPD de cada algoritmo y el número de pacientes, se observa que los algoritmos DDE-P1, DDE-P2 y DDE-P4 en comparación a los otros, mejoran ligeramente con el crecimiento del número de pacientes, sobre todo el método P4 que está diseñado para tener un tamaño de población dinámico en función del número de pacientes.

Tabla 6-3. Coeficientes de correlación entre nº de pacientes y ARPD

Coeficiente de relación entre la cantidad de pacientes y el ARPD						
DDE	DDE-S1	DDE-LS	DDE-P1	DDE-P2	DDE-P3	DDE-P4
0.435821	-0.255862	-0.109995	0.377714	0.472081	-0.165244	0.686897

También resulta interesante analizar el porcentaje respecto al total de pacientes en lista de espera que se han conseguido operar en cada algoritmo, al cual llamaremos **Rendimiento**.

Tabla 6-4. Resultados rendimiento de la simulación de los algoritmos

Rendimiento promedio						
DDE	DDE-S1	DDE-LS	DDE-P1	DDE-P2	DDE-P3	DDE-P4
88.00%	88.88%	83.95%	87.98%	88.02%	86.69%	87.86%
Rendimiento máximo						
DDE	DDE-S1	DDE-LS	DDE-P1	DDE-P2	DDE-P3	DDE-P4
98.2%	100.0%	97.9%	100.0%	100.0%	98.1%	100.0%

En la Tabla 6-4, se muestra que el rendimiento promedio de cada algoritmo sigue un patrón similar a los resultados obtenidos del ARPD promedio.

En cambio, al analizar la capacidad que tiene cada algoritmo para operar a todos los pacientes de la lista de espera, solo los algoritmos **DDE-S1, DDE-P1, DDE-P2, DDE-P4**, son capaces de hacerlo. Pese a que en promedio el algoritmo genérico **DDE** obtenga mejores resultados, nunca consigue planificar las operaciones de

todos los pacientes en lista de espera. Los algoritmos **DDE-LS** y **DDE-P3** mantienen los malos resultados de rendimiento.

Para profundizar en el funcionamiento de cada algoritmo, se usará una de las instancias usada en la simulación con la cual se analizará la evolución de los valores de la F.O de los distintos algoritmos a lo largo de sus iteraciones, al igual que mostrar el efecto de las distintas decodificaciones en el valor de la F.O. También se mostrarán los diagramas de Gantt resultantes de las distintas planificaciones de los algoritmos. La instancia que se usará de ejemplo es la mostrada en la Tabla 6-5:

Tabla 6-5. Datos de la instancia de ejemplo

Instancia					
α	β	mds	Cirujanos	ORs	Pacientes
2	1	4	22	9	147

La decodificación usada tiene un rol fundamental en la efectividad de los algoritmos. Originalmente, se desarrolló la ***DECOD1*** en el apartado 3.3.1, pero tras experimentar con él y apreciar el hecho de que asignar siempre en función del menor tiempo de terminación (Earliest Completion Time) daba lugar a ineficiencias en la decodificación de las soluciones, que posteriormente impedía en el proceso de búsqueda a los algoritmos metaheurísticos encontrar soluciones de manera óptima.

A partir de esta ineficiencia surge el ***DECOD3*** el cual mediante el uso de *holguras* identificaba la mejor posición para insertar una operación. En la Figura 6-3, se muestra la diferencia de rendimientos del algoritmo DDE al usar el método de decodificación DECOD1 y DECOD3 en la instancia de la Tabla 6-5 durante 15s.

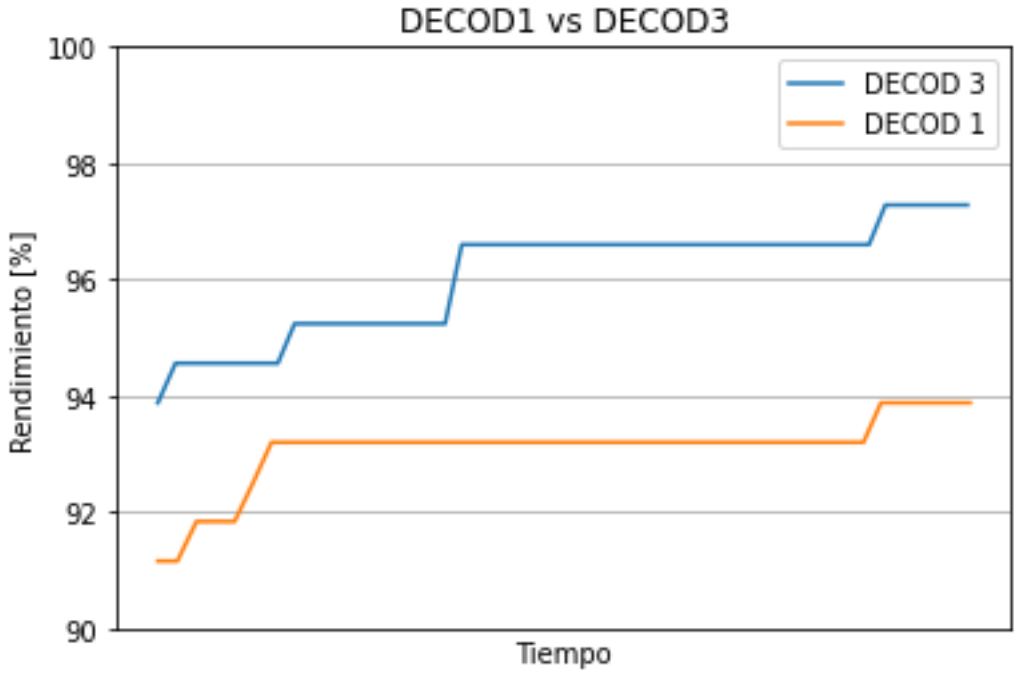

Figura 6-3: Diferencia de rendimiento del algoritmo DDE al usar dos decodificaciones distintas

En la Figura 6-3 se comprueba que desde el primer valor (mejor fitness de la población inicial aleatoria), el modelo DECOD3 obtiene un rendimiento superior que se mantiene a lo largo del algoritmo, ya que evita las ineficiencias que acarreaba el modelo DECOD1.

Se analizará la evolución del rendimiento (porcentaje de pacientes en lista de espera que se consiguen planificar en el calendario) para cada algoritmo (Figuras 6-4,5,6,7,8,9,10) ante la instancia de la Tabla 6-5, con un mismo tiempo de cómputo de 15s para todos los algoritmos.

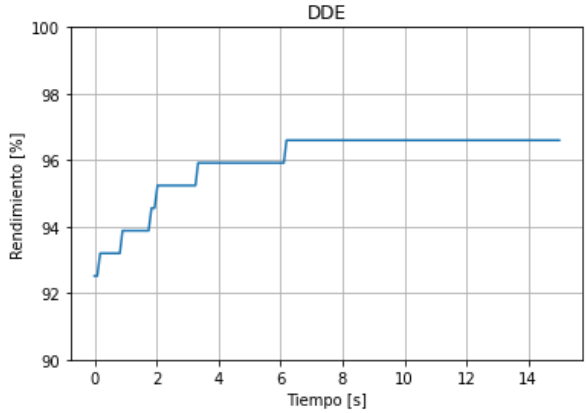

Figura 6-4: *Evolución del rendimiento en el algoritmo DDE*

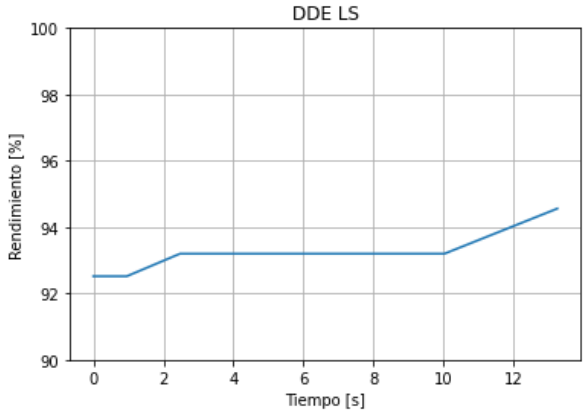

Figura 6-5: *Evolución del rendimiento en el algoritmo DDE-LS*

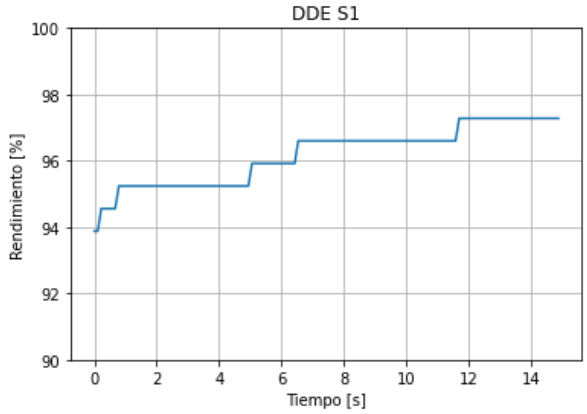

Figura 6-6: *Evolución del rendimiento en el algoritmo DDE-S1*

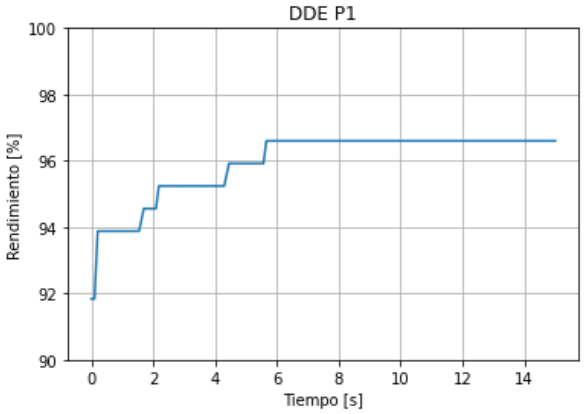

Figura 6-7: *Evolución del rendimiento en el algoritmo DDE-P1*

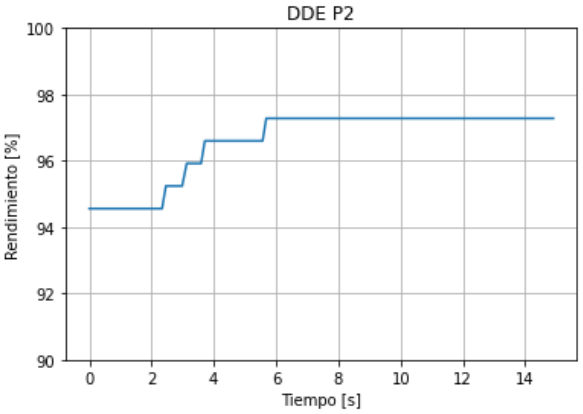

Figura 6-8: *Evolución del rendimiento en el algoritmo DDE-P2*

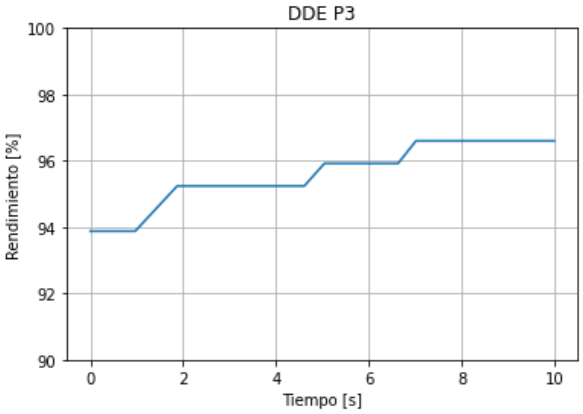

Figura 6-9: *Evolución del rendimiento en el algoritmo DDE-P3*

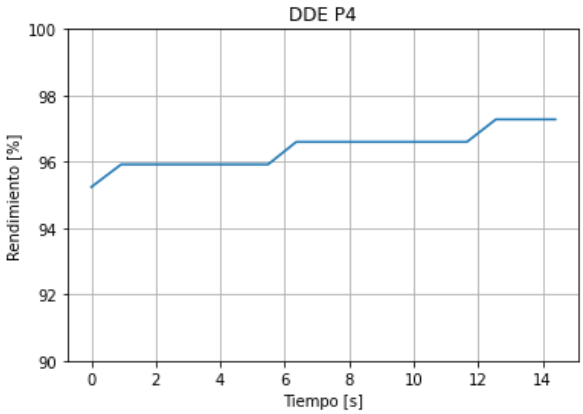

Figura 6-10: *Evolución del rendimiento en el algoritmo DDE-P4*

En la figura 6-11, se muestra tanto la evolución del algoritmo DDE para la instancia de ejemplo de la tabla 6-5, como la diferencia inicial al aplicar un algoritmo de decodificación u otro, ya que el tramo rojo representa el cambio en el rendimiento inicial del modelo DECOD1 y el modelo DECOD3

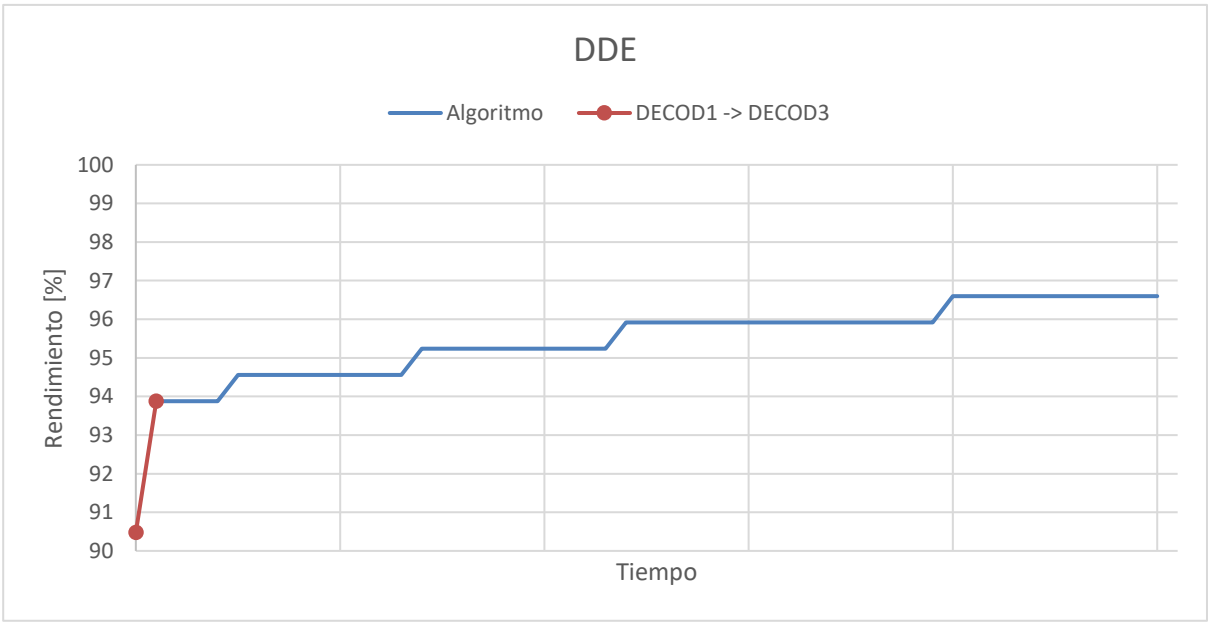

Figura 6-11: *Evolución del rendimiento en la decodificación y en el algoritmo DDE.*

A continuación, se mostrarán las distintas planificaciones que obtiene el algoritmo con mejor resultados, **DDES1 (Figura 6-12)**, y compararlo con una planificación dada por una secuencia aleatoria (Figura 6-13) para una misma instancia mediante su diagrama de Gantt correspondiente:

❖ **Diagrama de Gantt del algoritmo DDE-S1:** 144 de 147 pacientes operados. 97,96% rendimiento

Figura 6-12: *Diagrama de Gantt del algoritmo DDE.*

❖ **Diagrama de Gantt de una secuencia Aleatoria:** 130 de 147 pacientes operados. 88,01% rendimiento

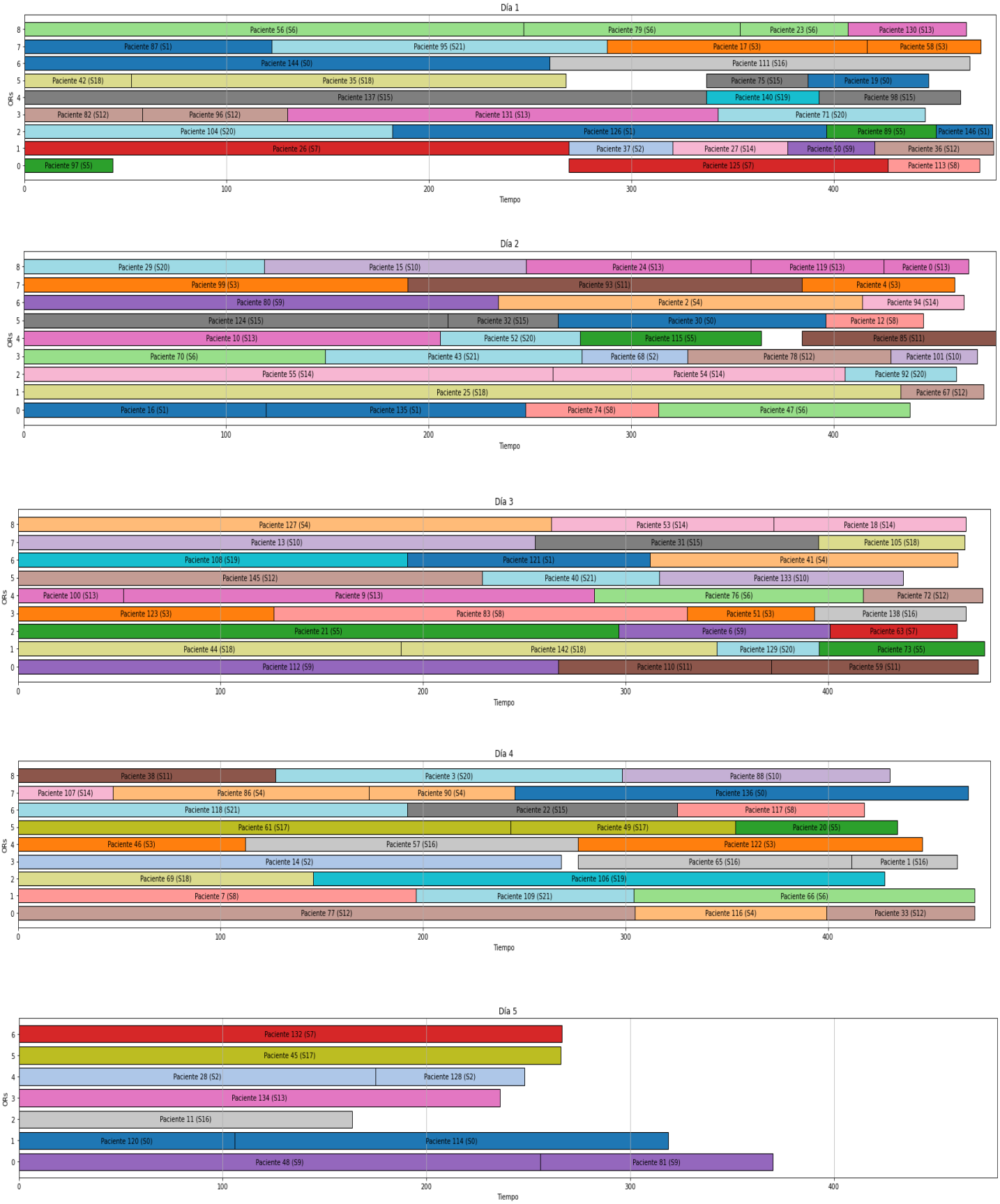

Figura 6-13: *Diagrama de Gantt para una secuencia aleatoria.*

A medida que avanza el horizonte de planificación, las secuencias menos efectivas no priorizan la asignación de operaciones complejas, como aquellas que requieren una OR específica debido al tipo de cirugía. Esta falta de priorización resulta en una disminución de la eficiencia durante el último día de planificación, especialmente cuando la flexibilidad es mínima, debido a las limitaciones del modelo, y resulta imposible asignar estas operaciones complejas. Estas limitaciones incluyen un máximo de días a la semana para operar, un límite en el número máximo de ORs disponibles por día, y restricciones sobre qué cirugías pueden realizarse en ciertas ORs.

Estos factores combinados pueden generar desafíos significativos en la planificación quirúrgica y destacan la importancia de no solo tener un buen algoritmo de decodificación, sino también un algoritmo de optimización que evite en poco tiempo estas ineficiencias.

7 CONCLUSIONES

En este Trabajo de Fin de Grado se aborda el problema de la planificación de cirugías en una unidad de especialidad de un hospital de referencia a nivel andaluz, en el cual a partir de los pacientes en lista de espera se busca obtener el programa semanal que maximice el número de pacientes operados.

Para ello, se han diseñado distintos algoritmos de decodificación, los cuales, a partir de una secuencia de permutación de los pacientes en la lista de espera y los datos del problema, genera el programa semanal equivalente de las operaciones quirúrgicas. Esto incluye la asignación de los pacientes y su cirujano de consulta a un quirófano admisible durante el horizonte de planificación, satisfaciendo todas las limitaciones quirúrgicas de personal, quirófano y pacientes involucradas. El entorno quirúrgico se modela durante el trabajo como un entorno productivo de *Parallel Machines.*

Se han desarrollado siete versiones del algoritmo de optimización metaheurística de Evolución Diferencial Discreta, con el objetivo de compararlas, analizar sus resultados, y obtener la configuración que consiga la mayor cantidad de pacientes operados durante el horizonte de planificación. Las distintas metaheurísticas desarrolladas son variaciones del algoritmo de Evolución Diferencial Discreta genérico, al que se le añaden distintas modificaciones como: una solución inicial basada en heurísticas constructivas, cambios en la dinámica de evolución de la población, o la inclusión de procesos de búsqueda local.

Cada algoritmo metaheurístico pasa por un proceso de calibración para establecer el valor de sus parámetros iniciales, el cual se realiza simulando todas las combinaciones de parámetros en una batería de 80 instancias distintas, obteniendo así la mejor combinación de parámetros para cada algoritmo. Este mismo proceso se realiza para elegir el mejor proceso de búsqueda local.

Establecidos los algoritmos y sus parámetros, se lleva a cabo la resolución computacional de todos ellos en una batería de 240 instancias distintas, a partir de sus resultados, se obtiene la mejor versión de todas: DDE-S1, la cual se basa en añadir una población inicial de heurísticas constructivas y consigue obtener un ARPD respecto al resto de algoritmos del 0.5569%. También se analizarán los resultados, el funcionamiento de cada algoritmo, sus evoluciones en el proceso de búsqueda de soluciones, y diagramas de Gantt que ejemplifiquen el eficiente resultado final de una secuencia al aplicar el algoritmo DDE-S1 respecto a una secuencia sin optimización metaheurística. A partir de los resultados, se identifica que configuración y parametrización de un algoritmo de evolución diferencial aporta mejores soluciones para el entorno y características de este trabajo.

7.1 Líneas de futuro

Una planificación óptima de las operaciones quirúrgicas es fundamental para poder reducir las crecientes listas de espera del sistema sanitario. En este trabajo, se ha conseguido un aumento notable en la eficiencia y rendimiento semanal de pacientes operados mediante el uso de optimización metaheurística, pero es esencial una continua investigación en la programación quirúrgica que asegure mucho más que la cantidad de pacientes que se operan.

Por ello, con vistas a futuros trabajos e investigaciones sería interesante analizar distintas políticas, modificaciones y ampliaciones que puedan mejorar aún más el proceso de programación quirúrgica para el hospital de referencia analizado:

- Este trabajo parte de una lista de pacientes en la cual la asignación de cirujanos ya ha sido realizada, por lo cual, es una restricción adicional que limita la flexibilidad a la hora de optimizar. En el apartado *3.3.2 Algoritmo de decodificación DECOD2* se plantea y propone un algoritmo de decodificación que

permita llevar a cabo la asignación de cirujanos a los pacientes de manera simultánea a la planificación del programa. Esta ampliación depende tanto de la política del hospital, que permita asignar un cirujano distinto para la operación del que hizo la consulta, como de realizar un análisis que compruebe que el tiempo de cómputo que añade la asignación de cirujanos obtenga resultados de calidad lo suficientemente mejorada.

- Otra posible ampliación sería buscar optimizar más de un parámetro. En este caso, se busca maximizar el número de pacientes operados, pero no se tiene en cuenta muchos otros parámetros de importancia sanitaria, como puede ser la carga de trabajo de los cirujanos, la prioridad y urgencia sanitaria de los pacientes, la calidad del personal auxiliar sanitario, etc. Esto sería posible mediante un análisis Pareto de las distintas funciones objetivos, y sería aplicable a los algoritmos de optimización metaheurística desarrollados en este trabajo tal y como expone (Pan et al., 2009).
- En el proceso de resolución computacional, y a lo largo de este trabajo, se ha usado un tiempo máximo de cómputo de 15 segundos para todos los algoritmos, principalmente por que el objetivo de uso de estos algoritmos metaheurísticos es la optimización rápida para obtener programas semanales en el menor tiempo posible. Pese a ello, sería interesante analizar las distintas versiones de los algoritmos bajo un tiempo de cómputo mayor, ya que muchos de ellos (principalmente el que incluye un proceso de búsqueda local) se ven perjudicados por el limitado tiempo de cómputo, ya sea por no poder intensificar o explorar lo suficiente el espacio de soluciones durante el proceso de búsqueda. Para el desarrollo de este trabajo, analizar siete algoritmos metaheurísticos para 240 instancias usando un tiempo de cómputo alto no ha sido factible, pero podría ser interesante como línea de futuro.

Sería interesante llevar a cabo todas las propuestas, para ver la posible evolución y mejora en entornos más complejos, siendo cualquier avance que mejora la calidad de vida y salud de todos los pacientes involucrados en el sistema sanitario fundamental, de ahí la importancia en los conocimientos de la programación de operaciones y las diferentes técnicas de optimización.

REFERENCIAS

Abdalkareem, Z. A., Amir, A., Al-Betar, M. A., Ekhan, P., & Hammouri, A. I. (2021). Healthcare scheduling in optimization context: a review. *Health and Technology*, *11*(3), 445-469. https://doi.org/10.1007/S12553-021-00547-5/TABLES/15

Acharya, B., Sucheta Panda, ·, Sivakumar, · Eashwar, Ganesan, G., Garg, L., Dhir, R., Kumarand, V., Sharma, M., Panda, S., & Sivakumar, E. (2022). An Analytical Study of Multiprocessor Scheduling Using Metaheuristic Approach. *SN Computer Science 2022 3:6*, *3*(6), 1-24. https://doi.org/10.1007/S42979-022-01398-1

Ahmad, M. F., Isa, N. A. M., Lim, W. H., & Ang, K. M. (2022). Differential evolution: A recent review based on state-of-the-art works. *Alexandria Engineering Journal*, *61*(5). https://doi.org/10.1016/j.aej.2021.09.013

Azhir, E., Jafari Navimipour, N., Hosseinzadeh, M., Sharifi, A., & Darwesh, A. (2019). Deterministic and non-deterministic query optimization techniques in the cloud computing. *Concurrency and Computation: Practice and Experience*, *31*(17). https://doi.org/10.1002/cpe.5240

Barajas-Gamboa, J. S. (2019). *Evolución de la cirugía: ¿estamos preparados para romper paradigmas?* https://doi.org/10.29375/01237047.3689

Blum, C., & Roli, A. (2003). Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. *ACM Computing Surveys*, *35*(3), 268-308. https://doi.org/10.1145/937503.937505

Cirugía ambulatoria en pacientes mayores de 65 años: nuestra experiencia | Revista Española de Geriatría y Gerontología. (s. f.). Recuperado 8 de febrero de 2024, de https://www.elsevier.es/es-revista-revista-espanola-geriatria-gerontologia-124-articulo-cirugia-ambulatoria-pacientes-mayores-65-13011667

Forrest, S. (1996). Genetic Algorithms. *ACM Computing Surveys*, *28*(1).

Framiñan, J. M., Leisten, R., & Ruiz García, R. (2014). *Manufacturing Scheduling Systems An Integrated View on Models, Methods and Tools*.

Gerges, N., Rak, J., & Jabado, N. (2010). New technologies for the detection of circulating tumour cells. *British Medical Bulletin*, *94*(1), 49-64. https://doi.org/10.1093/BMB/LDQ011

Gómez-Ríos, M. A., Abad-Gurumeta, A., Casans-Francés, R., & Calvo-Vecino, J. M. (2019). Claves para optimizar la eficiencia de un bloque quirúrgico. *Revista Española de Anestesiología y Reanimación*, *66*(2), 104-112. https://doi.org/10.1016/J.REDAR.2018.08.002

Ministerio de Sanidad. (2023). *INFORMACIÓN Y ESTADÍSTICAS SANITARIAS 2023*. www.sanidad.gob.es

Mokhtari, H., Abadi, I. N. K., & Cheraghalikhani, A. (2011). A multi-objective flow shop scheduling with resource-dependent processing times: Trade-off between makespan and cost of resources. *International Journal of Production Research*, *49*(19), 5851-5875. https://doi.org/10.1080/00207543.2010.523724

Molina-Pariente, J. M., Fernandez-Viagas, V., & Framinan, J. M. (2015). Integrated operating room planning and scheduling problem with assistant surgeon dependent surgery durations. *Computers &*

Industrial Engineering, *82*, 8-20. https://doi.org/10.1016/J.CIE.2015.01.006

Molina-Pariente, J. M., Hans, E. W., Framinan, J. M., & Gomez-Cia, T. (2015). New heuristics for planning operating rooms. *Computers & Industrial Engineering*, *90*, 429-443. https://doi.org/10.1016/J.CIE.2015.10.002

Pan, Q. K., Tasgetiren, M. F., & Liang, Y. C. (2008). A discrete differential evolution algorithm for the permutation flowshop scheduling problem. *Computers & Industrial Engineering*, *55*(4), 795-816. https://doi.org/10.1016/J.CIE.2008.03.003

Pan, Q. K., Wang, L., & Qian, B. (2009). A novel differential evolution algorithm for bi-criteria no-wait flow shop scheduling problems. *Computers & Operations Research*, *36*(8), 2498-2511. https://doi.org/10.1016/J.COR.2008.10.008

Pinedo, M. L. (2005). *Springer Series in Operations Research Planning and Scheduling in Manufacturing and Services*.

Pinedo, M. L. (2016). Scheduling: Theory, algorithms, and systems, fifth edition. *Scheduling: Theory, Algorithms, and Systems, Fifth Edition*, 1-670. https://doi.org/10.1007/978-3-319-26580-3/COVER

ANEXO: CÓDIGO PYTHON

FUNCIONES.PY

```
import numpy as np
import pandas as pd
import decod3,decod1
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
import random
import time


def tiempo(func):
    def wrapper(*args, **kwargs):
        start_time = time.time()
        result = func(*args, **kwargs)
        end_time = time.time()
        print(f"Tiempo total: {end_time - start_time} secs")
        return result
    return wrapper
def tiempo_valor(func):
    def wrapper(*args, **kwargs):
        start_time = time.time()
        result = func(*args, **kwargs)
        end_time = time.time()
        elapsed_time = end_time - start_time
        return elapsed_time
    return wrapper

"""
DECOD3 ------------------------------------------------------------------------
------------------------------------------------------------
"""
def PuedeMoverseCirujano(Xsjh,cirujano,OR,movimientomax_cirujanos,h):
    """
    COMPRUEBA SI UN CIRUJANO EN UN DIA DADO PUEDE USAR LA OR DADA
    """
    if sum(Xsjh[cirujano,:,h]) > movimientomax_cirujanos[cirujano]:
        return False
    if sum(Xsjh[cirujano,:,h]) == movimientomax_cirujanos[cirujano]:
        if Xsjh[cirujano,OR,h] == 1:
            return True
        else:
            return False
    if sum(Xsjh[cirujano,:,h]) < movimientomax_cirujanos[cirujano]:
        return True
```

```
def DiasMaxCirujano(dias_max_cirujano,mds,cirujano,dia):
    """
    COMPRUEBA SI PARA UN DIA Y CIRUJANO DADO, LE QUEDAN
    DIAS LIBRES PARA OPERAR
    """
    if sum(dias_max_cirujano[:,cirujano])>mds: return False
    if sum(dias_max_cirujano[:,cirujano])<mds: return True
    if sum(dias_max_cirujano[:,cirujano])==mds:
        if dias_max_cirujano[dia,cirujano] == 1: return True
        else: return False


def CargarDatos(file_path,dataset,version):
    # Leer excel
    df = pd.read_excel(file_path,sheet_name=f'DATASET {dataset}.{version}')

    num_dias = int(df['num_dias'].iloc[0])
    num_pacientes = int(df['num_pacientes'].iloc[0])
    num_OR = int(df['num_OR'].iloc[0])
    num_cirujanos = int(df['num_cirujanos'].iloc[0])
    mds = int(df['mds'].iloc[0])
    tiempo_trabajo = int(df['tiempo_trabajo'].iloc[0])

    movimientomax_cirujanos =
df["movimientomax_cirujanos"].iloc[:num_cirujanos].to_numpy()

    pacientes = df.iloc[:,2:9].to_numpy()

    secuencia = df.iloc[:,10].to_numpy()

    tiempo_dias = df.iloc[:num_dias,12].to_numpy()

    Dijh = np.zeros((num_pacientes, num_OR, num_dias))

    col = 21

    auxiliar = num_OR
    for i in range(num_dias):
        Dijh[:,:,i] = df.iloc[:,col:col+auxiliar].to_numpy()
        col = col+auxiliar+1


    return num_dias, tiempo_dias, num_OR, mds, num_pacientes, num_cirujanos,
tiempo_trabajo, movimientomax_cirujanos, pacientes, secuencia, Dijh
```

```
def indices_ordenados_menormayor(arr):
    # Crear un array de tuplas donde el primer elemento es el valor y el segundo
es el índice
    arr_indexado = [(valor, idx) for idx, valor in enumerate(arr)]

    # Ordenar el array por valor (de menor a mayor) y luego por índice (de menor
a mayor)
    arr_ordenado = sorted(arr_indexado, key=lambda x: (x[0], x[1]))

    # Extraer los índices del array ordenado
    indices_ordenados = [x[1] for x in arr_ordenado]

    return np.array(indices_ordenados)

def indices_ordenados_mayormenor(arr):
    # Crear un array de tuplas donde el primer elemento es el valor y el segundo
es el índice
    arr_indexado = [(valor, idx) for idx, valor in enumerate(arr)]

    # Ordenar el array por valor (de mayor a menor) y luego por índice (de menor
a mayor)
    arr_ordenado = sorted(arr_indexado, key=lambda x: (-x[0], x[1]))

    # Extraer los índices del array ordenado
    indices_ordenados = [x[1] for x in arr_ordenado]

    return np.array(indices_ordenados)


def Gantt(solucion):
    """
    CODIGO PARA GENERAR UN DIAGRAMA DE GANTT
    A PARTIR DE UNA MATRIZ SOLUCION EN LA QUE
    SE VAN ALMACENANDO LOS TIEMPOS DE INICIO Y FIN
    DE CADA OPERACION, EL PACIENTE, EL CIRUJANO, LA OR,
    Y EL DIA DE CADA OPERACION
    """
    num_plots = solucion.shape[0]

    cirujano_ids = np.unique(solucion["cirujano"])

    num_cirujanos = len(cirujano_ids)
    colors = plt.cm.tab20(np.linspace(0, 1, num_cirujanos))

    cirujano_color_map = {cirujano_id: colors[i] for i, cirujano_id in
enumerate(cirujano_ids)}

    for i in range(num_plots):
        starting_times = solucion[i]["startingtime"]
```

```
        finishing_times = solucion[i]["finishingtime"]
        places = solucion[i]["OR"]
        authors = solucion[i]["cirujano"]
        title = f"Día {i + 1}"

        valid_operations = [(start, finish, place, author, idx) for start,
finish, place, author, idx in
                            zip(starting_times, finishing_times, places,
authors, range(1, len(starting_times) + 1)) if
                            finish > 0]

        if not valid_operations:
            continue
        starting_times, finishing_times, places, authors, op_numbers =
zip(*valid_operations)

        operations = [f'Paciente {op_number - 1} (S{authors[i]})' for i,
op_number in enumerate(op_numbers)]
        fig, ax = plt.subplots(figsize=(30, 4))
        for j in range(len(starting_times)):
            place = places[j]
            cirujano_id = authors[j]
            color = cirujano_color_map[cirujano_id
            ax.barh(place, width=finishing_times[j] - starting_times[j],
left=starting_times[j], color=color,
                    edgecolor='black')
            ax.text(starting_times[j] + (finishing_times[j] - starting_times[j])
/ 2, place, operations[j], ha='center',
                    va='center')
        ax.set_xlabel('Tiempo')
        ax.set_ylabel('ORs')
        ax.set_title(title)
        ax.xaxis.grid(True)
        ax.set_yticks(range(int(min(places)), int(max(places)) + 1))
        ax.set_xlim([0, 480])
        plt.show()

def plot_evol(x_values,y_values, title):
    plt.plot(x_values, y_values)
    plt.title(title)
    plt.xlabel('Tiempo [s]')
    plt.ylabel('Rendimiento [%]')
    y_min = np.min(y_values)
    if y_min>=90:
        y_min = 90
    elif y_min >= 80:
        y_min = 80
    elif y_min >=70:
        y_min = 75
```

```
    plt.ylim(y_min, 100)
    plt.grid(True)
    plt.show()

def plot_evol2(x_values_1, y_values_1, x_values_2, y_values_2, title):
    plt.plot(x_values_1, y_values_1, label='DECOD 3')
    plt.plot(x_values_2, y_values_2, label='DECOD 1')
    plt.title(title)
    plt.xlabel('Tiempo')
    plt.ylabel('Rendimiento [%]')
    plt.ylim(90, 100)
    plt.xticks([])
    plt.legend()
    plt.grid(True)
    plt.show()

"""
GENERACION DE DATOS, MAIN ------------------------------------------------------
------------------------------------------------------------------------------
"""

def tiempo_cirugia():
    """
    SE GENERA UN TIEMPO DE CIRUJIA A PARTIR DE LA MEDIA Y CV
    EN UNA DIST LOGNOR
    """
    media = np.random.choice([60, 120, 180, 240])
    coef_var = np.random.uniform(0.1 * media, 0.5 * media)
    sigma = coef_var / media
    mu = np.log(media) - 0.5 * sigma**2
    t = round(np.random.lognormal(mean=mu, sigma=sigma),4)
    return t


def duedates_weights(num_pacientes):
    """
    SE USAN LAS FORMULAS PARA CALCULAR
    LAS DUE DATES Y EL PESO DE CADA PACIENTE

    SE PUEDE HACER SIN TENER EN CUENTA EL TIEMPO
    DE CIRUJIA DE CADA UNO
    """
    lista_dd = []
    lista_w = []
    for i in range(num_pacientes):
        #Calculo de las dd
        MTBT = np.random.choice([45,180,360])
        dwl = np.random.randint(1,MTBT-1)
```

```
        due_date = MTBT - dwl
        lista_dd.append(due_date)
        #Calculo del clinical weight
        a = 0.5
        mp = np.random.randint(1,5)
        mp_norm = mp/5
        dwl_norm = dwl/MTBT
        w = a*mp_norm + (1-a) * dwl_norm
        lista_w.append(round(w,4))
    return np.array(lista_dd), np.array(lista_w)

def Dijh_OR_especializada(num_pacientes, num_OR,num_dias):
    """
    SE GENERA LA MATRIZ Dijh
     EL 90% DE LOS CASOS, EL PACIENTE SE PUEDE OPERAR EN CUALQUIER OR
     EL 10% DE LOS CASOS, EL PACIENTE SOLO SE PUEDE OPERAR EN EL 30% DE LAS OR
    """
    if num_OR == 3: num_ones = 1
    if num_OR == 9: num_ones = 3
    array_3d = np.zeros((num_pacientes, num_OR, num_dias), dtype=int)


    for paciente_idx in range(num_pacientes):
        if random.random() < 0.9:
            for dia_idx in range(num_dias):
                    array_3d[paciente_idx, :, dia_idx] = 1

        else:
            for dia_idx in range(num_dias):
                ones_indices = random.sample(range(num_OR), num_ones)
                for OR_idx in range(num_OR):
                    if OR_idx in ones_indices:
                        array_3d[paciente_idx, OR_idx, dia_idx] = 1
    return array_3d

def numeropacientes_listatiempos(capacidad_OR,beta):
    """
    SE VAN GENERANDO PACIENTES HASTA IGUALAR
    EL TIEMPO DISPONIBLE TOTAL DE LAS OR
    """
    tiempo_total = beta * sum(sum(capacidad_OR))
    cont = 0
    lista_tiempos = []
    while True:
        tiempo = tiempo_cirugia()
        if tiempo > tiempo_total: break
        else:
            lista_tiempos.append(tiempo)
            tiempo_total = tiempo_total - tiempo
```

```
            cont = cont + 1
    return cont,lista_tiempos


"""
ALGORITMO DE EVOLUCION DIFERENCIAL ----------------------------------------
-------------------------------------------------------------------------------
-------
"""

def MUTACION(Xa,Xb,Xc,prob,fuerza):
    #a
    Xa,Xb,Xc = np.array(Xa),np.array(Xb),np.array(Xc)
    diff = Xb-Xc
    #b
    size = len(diff)
    rand = np.random.rand(size)
    delta = np.zeros(size)
    """
    for i in range(size):
        if rand[i] < prob:
            delta[i] = diff[i]
    """
    #c
    n = size-1
    V = []
    for i in range(size):
        if rand[i] < prob:#Lo pongo aqui para ahorrarme un for
            delta[i] = int(fuerza*diff[i])
            valor = Xa[i] + delta[i]
            if valor < 0:
                V.append(valor + 1 + n)
            elif valor > n:
                V.append(valor-n-1)
            else:
                V.append(valor)
        else:
            V.append(Xa[i])
    if len(V) != size: print("Error mutacion")
    #print(diff,delta)
    return V

def RECOMBINACION(X,V,CR):
    #a
    X,V = np.array(X),np.array(V)
    U = np.array([])
    #b
    size = len(X)
```

```
    corte = np.random.randint(0,size) #El valor del corte son los numeros que se
incluyen
    U = np.append(U,X[:corte])
    #c
    rand = np.random.rand(size)
    for i in range(size):
        if rand[i] < CR and V[i] not in U:
            U = np.append(U,V[i])
    #d
    if len(U) != size:
        for i in range(size-corte):
            if X[corte:][i] not in U:
                U = np.append(U,X[corte:][i])
    if len(U) != size: print("Error Recombinacion")
    return U


def FITNESS(DATOS,sec,):
    pacientes_total = decod3.DECOD3(DATOS,"pacientes",False,False, sec)

    return pacientes_total

"""
LOCAL SEARCHES----------------------------------------------------------------
----------------------------------------------------------------
"""
#Funciones para vecindades
def insert(secuencia, outerindex, innerindex):
    secuencia_loop = np.copy(secuencia)
    x = np.delete(secuencia_loop, outerindex)
    secuencia_loop = np.insert(x, innerindex, secuencia[outerindex])
    return secuencia_loop

def generalswap(secuencia, outerindex, innerindex):
    secuencia_loop = np.copy(secuencia)
    secuencia_loop[outerindex], secuencia_loop[innerindex] =
secuencia_loop[innerindex], secuencia_loop[outerindex]
    return secuencia_loop

def adjacentswap(secuencia, index):
    secuencia_loop = np.copy(secuencia)
    secuencia_loop[index], secuencia_loop[index + 1] = secuencia_loop[index +
1], secuencia_loop[index]
    return secuencia_loop

#1. General Swap
def FirstImprovement_generalswap(DATOS,secuencia):
    num_pacientes = DATOS[4]
    mejorobj = decod3.DECOD3(DATOS, "pacientes", False, False, secuencia)
```

```
    mejorsec = secuencia[:]
    mejora = False
    for j in range(num_pacientes):
        if mejora == True: break
        for i in range(num_pacientes):
            secuencia_loop = generalswap(secuencia, j, i)
            obj = decod3.DECOD3(DATOS, "pacientes", False, False,
secuencia_loop)
            if obj < mejorobj:
                mejora = True
                mejorobj = obj
                mejorsec = secuencia_loop[:]
            if mejora == True: break
    return mejorsec

#NO RETURNEAN MEJOR SEC, NECESITAN MODIFICACION SI SE QUIEREN USAR!!!!!!!!!!!!!!

#2. Adjacent Swap
def FirstImprovement_adjacentswap(DATOS,secuencia):
    num_pacientes = DATOS[4]
    mejorobj = decod3.DECOD3(DATOS, "pacientes", False, False, secuencia)
    mejora = False
    for j in range(num_pacientes-1):
        secuencia_loop = adjacentswap(secuencia, j)
        obj = decod3.DECOD3(DATOS, "pacientes", False, False, secuencia_loop)
        if obj < mejorobj:
            mejora = True
            mejorobj = obj
        if mejora == True: break

# 3. 3-opt Local Search
def LS_3_OPT(DATOS, secuencia):
    num_pacientes = DATOS[4]
    mejorobj = decod3.DECOD3(DATOS, "pacientes", False, False, secuencia)
    mejora = False
    for i in range(num_pacientes - 2):
        if mejora == True: break
        for j in range(i + 1, num_pacientes - 1):
            if mejora == True: break
            for k in range(j + 1, num_pacientes):
                if mejora == True: break
                # Three possible combinations for edge swapping
                for seq in [[i, j, k], [i, k, j], [j, i, k]]:
                    secuencia_loop = secuencia.copy()
                    secuencia_loop[seq[0]:seq[1]+1] =
secuencia_loop[seq[0]:seq[1]+1][::-1]
                    secuencia_loop[seq[1]+1:seq[2]+1] =
secuencia_loop[seq[1]+1:seq[2]+1][::-1]
```

```
                    obj = decod3.DECOD3(DATOS, "pacientes", False, False,
secuencia_loop)
                    if obj > mejorobj:
                        mejorobj = obj
                        mejora = True



#4. Insertion
def FirstImprovement_insert(DATOS, secuencia):
    num_pacientes = DATOS[4]
    mejorobj = decod3.DECOD3(DATOS, "pacientes", False, False, secuencia)
    mejora = False
    for j in range(num_pacientes):
        if mejora == True: break
        for i in range(num_pacientes):
            secuencia_loop = insert(secuencia,j,i)
            obj = decod3.DECOD3(DATOS, "pacientes", False, False,
secuencia_loop)
            if obj < mejorobj:
                mejora = True
                mejorobj = obj
            if mejora == True: break
```

MAIN.PY

```
import numpy as np
import pandas as pd
import funciones

"""
GENERACION DE DATOS
"""
def main(ORs, alpha, beta, mds):
    # Horizonte de días [h] = 5
    num_dias = 5
    num_sem = 1
    tiempo_dias = np.full(num_dias, 480, dtype=np.float64)
    tiempo_trabajo = 480

    # Nº de OR = 3,9
    num_OR = ORs
    capacidad_OR = np.full((num_dias, num_OR), tiempo_trabajo)

    # MDS -> número de días máximos donde el cirujano puede realizar cirugías
(3,4)
    mds = mds
    # Nº de cirujanos
    alpha = alpha
    disponibilidad_cirujanos = 390

    num_cirujanos = int(round(alpha * (sum(sum(capacidad_OR))) / (num_sem *
disponibilidad_cirujanos * mds), 0))

    # Nº de pacientes en lista --> se van generando pacientes hasta llegar a un
%(beta=100<--,125%) del tiempo total disponible en los 5 días
    beta = beta
    num_pacientes, lista_tiempos =
funciones.numeropacientes_listatiempos(capacidad_OR, beta)
    # Pacientes (release date[rdi], due date[ddi], peso[wi], tiempo de
cirugia[ti], relacion cirujano-paciente [ci], paciente operado, identificador
[ID])
    pacientes = np.zeros((num_pacientes, 7))
    pacientes[:, 0] = np.zeros(num_pacientes)  # np.random.randint(0, 2,
num_pacientes) #Release date
    pacientes[:, 1], pacientes[:, 2] =
funciones.duedates_weights(num_pacientes)  # DueDates y Clinical Weight
    pacientes[:, 3] = lista_tiempos  # Tiempo de cirugia
    pacientes[:, 4] = np.random.randint(0, num_cirujanos, num_pacientes)  #
Cirujano que opera
```

```
    pacientes[:, 6] = np.arange(num_pacientes)  # ID

    # Dijh (si se puede hacer una operacion o no)
    Dijh = funciones.Dijh_OR_especializada(num_pacientes, num_OR, num_dias)

    # Movimiento max  = 1,j
    movimientomax_cirujanos = np.random.randint(1, num_OR, size=num_cirujanos)

    # Secuencia inicial
    secuencia = np.arange(num_pacientes - 1, -1, -1)

    return num_dias, tiempo_dias, num_OR, mds, num_pacientes, num_cirujanos,
tiempo_trabajo, movimientomax_cirujanos, pacientes, secuencia, Dijh

"""
BUCLE PARA CREAR DISTINTOS DATASETS EN EXCEL
"""
dataset_repetition = 15 #<---------------------

if __name__ == "__main__":
    #Se crea el excel
    matriz_datos = np.zeros((16*dataset_repetition,6))
    with pd.ExcelWriter('DATOS.xlsx') as writer:
        i = 0
        it = 0
        for ORs in [3,9]:
            for mds in [3,4]:
                for alpha in [1.5,2]:
                    for beta in [1,1.25]:
                        for n in range(dataset_repetition):
                            #Se generan datos
                            (num_dias, tiempo_dias, num_OR, mds, num_pacientes,
num_cirujanos, tiempo_trabajo, movimientomax_cirujanos, pacientes, secuencia,
Dijh) = main(ORs,alpha,beta,mds)
                            #Se escriben los datos en la worksheet i
                            movimientomax_cirujanos_df =
pd.DataFrame(movimientomax_cirujanos)
                            movimientomax_cirujanos_df.to_excel(writer,
sheet_name=f'DATASET {i}.{n}', index=False, header=["movimientomax_cirujanos"] *
len(movimientomax_cirujanos_df.columns))

                            pacientes_df = pd.DataFrame(pacientes)
                            pacientes_df.to_excel(writer, sheet_name=f'DATASET
{i}.{n}', index=False, startcol=movimientomax_cirujanos_df.shape[1] + 1,
header=["pacientes"] * len(pacientes_df.columns))

                            secuencia_df = pd.DataFrame(secuencia)
                            secuencia_df.to_excel(writer, sheet_name=f'DATASET
{i}.{n}', index=False, startcol=movimientomax_cirujanos_df.shape[1] +
```

```
pacientes_df.shape[1] + 2, header=["secuencia auxiliar"] *
len(secuencia_df.columns))

                            tiempo_dias_df = pd.DataFrame(tiempo_dias)
                            tiempo_dias_df.to_excel(writer, sheet_name=f'DATASET
{i}.{n}', index=False, startcol=movimientomax_cirujanos_df.shape[1] +
pacientes_df.shape[1] + secuencia_df.shape[1] + 3, header=["tiempo_dias"] *
len(tiempo_dias_df.columns))

                            other_info_df = pd.DataFrame({'num_dias':
[num_dias], 'num_pacientes': [num_pacientes], 'num_OR': [num_OR],
'num_cirujanos': [num_cirujanos],'mds': [mds],'tiempo_trabajo':
[tiempo_trabajo]})
                            other_info_df.to_excel(writer,sheet_name=f'DATASET
{i}.{n}', index=False, startcol=movimientomax_cirujanos_df.shape[1] +
pacientes_df.shape[1] + secuencia_df.shape[1] + tiempo_dias_df.shape[1] + 4)

                            metadata_df = pd.DataFrame({'OR': [ORs], 'mds':
[mds], 'alpha': [alpha], 'beta': [beta]})
                            metadata_df.to_excel(writer, sheet_name=f'DATASET
{i}.{n}', index=False, startrow=3, startcol=14)


                            start_col_dijh = movimientomax_cirujanos_df.shape[1]
+ pacientes_df.shape[1] + secuencia_df.shape[1] + tiempo_dias_df.shape[1] + 11
                            for j in range(Dijh.shape[2]):
                                pd.DataFrame(Dijh[:,:,j]).to_excel(writer,
sheet_name=f'DATASET {i}.{n}', index=False, startcol=start_col_dijh,
header=["Dijh_" + str(j)] * Dijh[:,:,j].shape[1])
                                start_col_dijh += num_OR + 1  # Adjusting the
start column for the next iteration

                            #Guardamos datos clave para escribirlos en hoja
excel final de resultados de cada dataset
                            matriz_datos[it,0] = alpha
                            matriz_datos[it,1] = beta
                            matriz_datos[it,2] = mds
                            matriz_datos[it,3] = num_cirujanos
                            matriz_datos[it,4] = num_OR
                            matriz_datos[it,5] = num_pacientes
                            it = it + 1
                        i = i + 1

    with pd.ExcelWriter('RESULTADOS.xlsx') as writer:

        headers = ["Alpha", "Beta", "mds", "Cirujanos", "ORs", "Pacientes"]
        df = pd.DataFrame(matriz_datos, columns=headers)
        df.to_excel(writer,sheet_name="Combinaciones Instancias",index=False)
```

ALGORITMOS.PY

```
import funciones
import decod3
import numpy as np
import pandas as pd
import time

def HEURISTICAS(pacientes,Dijh,tipo_heuristica):
    if tipo_heuristica == "SPT":
        sorted_indices = np.argsort(pacientes[:, 3])
        return pacientes[:,6][sorted_indices]
    elif tipo_heuristica == "OCC":
        sorted_indices = np.argsort(pacientes[:, 4])
        return pacientes[:,6][sorted_indices]
    elif tipo_heuristica == "DRO":
        Dijh_ordenada = np.sum(Dijh[:, :, 0].astype(int), axis=1)
        sorted_indices = np.lexsort((pacientes[:, 3], Dijh_ordenada))
        return pacientes[:,6][sorted_indices]
    else:
        print("ERROR INPUT")

"""
DDE1 ------------------------------------------------------------------------
------------------------------------------------------------------------------
-------------------------------------------------------------
"""
def
DDE(POB_size,prob_mutacion,prob_recombinacion,fuerza_mutacion,max_time,excel,DAT
ASET_n,version):

    """
    CARGA INICIAL DE DATOS
    """

    #num_dias,tiempo_dias,num_OR,mds,num_pacientes,num_cirujanos,tiempo_trabajo,
movimientomax_cirujanos,pacientes,secuencia,Dijh =
funciones.CargarDatos(excel,DATASET_n,version)

    num_dias,tiempo_dias,num_OR,mds,num_pacientes,num_cirujanos,tiempo_trabajo,m
ovimientomax_cirujanos,pacientes,secuencia,Dijh = excel
    DATOS =
num_dias,tiempo_dias,num_OR,mds,num_pacientes,num_cirujanos,tiempo_trabajo,movim
ientomax_cirujanos,pacientes,secuencia,Dijh



    """
    CREACION DE LA POBLACION INICIAL. FULL ALEATORIA
```

```
    """
    X,V,U =
np.zeros((POB_size,num_pacientes)),np.zeros((POB_size,num_pacientes)),np.zeros((
POB_size,num_pacientes))
    X_fo,U_fo = np.zeros((POB_size,)),np.zeros((POB_size,))
    FO_best = 0
    FO_best_pos = 0
    X_best = np.zeros(num_pacientes)

    datos_evol_x,datos_evol_y = [],[]

    for i in range(POB_size):
        X[i,:] = np.random.permutation(num_pacientes)
        X_fo[i] = decod3.DECOD3(DATOS, "pacientes", False, False, X[i,:])

    FO_best_pos = int(np.argmax(X_fo))
    FO_best = int(X_fo[FO_best_pos])
    X_best = X[FO_best_pos,:]


    """
    ITERACIONES DEL ALGORITMO DDE
    """
    start_time = time.time()

    iteraciones = 0
    while True:
        if time.time() - start_time >= max_time:
            """
            GUARDAR MEJOR SOLUCION
            """
            FO_best_pos = int(np.argmax(X_fo))
            X_best = X[FO_best_pos,:]
            FO_best = decod3.DECOD3(DATOS, "pacientes", False, False, X_best)
            break

        datos_evol_x.append(time.time() - start_time)
        datos_evol_y.append((int(np.max(X_fo))/num_pacientes)*100 )

        for i in range(POB_size):

            """
            MUTACION
            """
            V[i,:] = funciones.MUTACION(X[i,:],
X[np.random.randint(0,POB_size),:], X[np.random.randint(0,POB_size),:],
prob_mutacion, fuerza_mutacion)

            """
```

```
            RECOMBINACION
            """
            U[i,:] = funciones.RECOMBINACION(X[i,:], V[i,:], prob_recombinacion)
            U_fo[i] = decod3.DECOD3(DATOS, "pacientes", False, False, U[i,:])


            """
            SELECCION
            """
            if U_fo[i] > X_fo[i]:
                X[i,:] = U[i,:]
                X_fo[i] = U_fo[i]
        iteraciones = iteraciones + 1

    """
    GENERAR GANTT Y CHARTS
    """
    return FO_best

"""
 ----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------
"""
"""
DDE-S1 ----------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------
"""
def
DDE_S1(POB_size,prob_mutacion,prob_recombinacion,fuerza_mutacion,max_time,excel,
DATASET_n,version):

    """
    CARGA INICIAL DE DATOS
    """
    num_dias,tiempo_dias,num_OR,mds,num_pacientes,num_cirujanos,tiempo_trabajo,m
ovimientomax_cirujanos,pacientes,secuencia,Dijh = excel
    DATOS =
num_dias,tiempo_dias,num_OR,mds,num_pacientes,num_cirujanos,tiempo_trabajo,movim
ientomax_cirujanos,pacientes,secuencia,Dijh


    """
    CREACION DE LA POBLACION INICIAL. FULL ALEATORIA
    """
    X,V,U =
np.zeros((POB_size,num_pacientes)),np.zeros((POB_size,num_pacientes)),np.zeros((
POB_size,num_pacientes))
```

```
X_fo,U_fo = np.zeros((POB_size,)),np.zeros((POB_size,))
FO_best = 0
FO_best_pos = 0
X_best = np.zeros(num_pacientes)

datos_evol_x,datos_evol_y = [],[]


for i in range(POB_size):
    if i == 0:
        X[i,:] = HEURISTICAS(pacientes, Dijh, "SPT")
        X_fo[i] = decod3.DECOD3(DATOS, "pacientes", False, False, X[i,:])
    elif i == 1:
        X[i,:] = HEURISTICAS(pacientes, Dijh, "OCC")
        X_fo[i] = decod3.DECOD3(DATOS, "pacientes", False, False, X[i,:])
    elif i == 2:
        X[i,:] = HEURISTICAS(pacientes, Dijh, "DRO")
        X_fo[i] = decod3.DECOD3(DATOS, "pacientes", False, False, X[i,:])
    else:
        X[i,:] = np.random.permutation(num_pacientes)
        X_fo[i] = decod3.DECOD3(DATOS, "pacientes", False, False, X[i,:])

FO_best_pos = int(np.argmax(X_fo))
FO_best = int(X_fo[FO_best_pos])
X_best = X[FO_best_pos,:]
"""
ITERACIONES DEL ALGORITMO DDE
"""
start_time = time.time()

iteraciones = 0
while True:
    if time.time() - start_time >= max_time:
        """
        GUARDAR MEJOR SOLUCION
        """
        FO_best_pos = int(np.argmax(X_fo))
        X_best = X[FO_best_pos,:]
        FO_best = decod3.DECOD3(DATOS, "pacientes", False, False, X_best)
        break

    datos_evol_x.append(time.time() - start_time)
    datos_evol_y.append((int(np.max(X_fo))/num_pacientes)*100 )


    for i in range(POB_size):

        """
        MUTACION
```

```
            """
            V[i,:] = funciones.MUTACION(X[i,:],
X[np.random.randint(0,POB_size),:], X[np.random.randint(0,POB_size),:],
prob_mutacion, fuerza_mutacion)

            """
            RECOMBINACION
            """
            U[i,:] = funciones.RECOMBINACION(X[i,:], V[i,:], prob_recombinacion)
            U_fo[i] = decod3.DECOD3(DATOS, "pacientes", False, False, U[i,:])

            """
            SELECCION
            """
            if U_fo[i] > X_fo[i]:
                X[i,:] = U[i,:]
                X_fo[i] = U_fo[i]
        iteraciones = iteraciones + 1
    """
    GENERAR GANTT
    """
    AANUM_FINAL_PAC, AARENDIMIENTO = decod3.DECOD3(DATOS,"todo", True,
False,X_best)
    print(AANUM_FINAL_PAC)

    AANUM_FINAL_PAC, AARENDIMIENTO = decod3.DECOD3(DATOS,"todo", True,
False,np.random.permutation(num_pacientes))
    print(AANUM_FINAL_PAC)

    return FO_best


"""
 ------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------
"""

"""
DDE-LS ----------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------
"""
def
DDE_LS(POB_size,prob_mutacion,prob_recombinacion,fuerza_mutacion,max_time,excel,
DATASET_n,version):

    """
    CARGA INICIAL DE DATOS
```

```
    """
    num_dias,tiempo_dias,num_OR,mds,num_pacientes,num_cirujanos,tiempo_trabajo,m
ovimientomax_cirujanos,pacientes,secuencia,Dijh = excel
    DATOS =
num_dias,tiempo_dias,num_OR,mds,num_pacientes,num_cirujanos,tiempo_trabajo,movim
ientomax_cirujanos,pacientes,secuencia,Dijh

    """
    CREACION DE LA POBLACION INICIAL. FULL ALEATORIA
    """
    X,V,U =
np.zeros((POB_size,num_pacientes)),np.zeros((POB_size,num_pacientes)),np.zeros((
POB_size,num_pacientes))
    X_fo,U_fo = np.zeros((POB_size,)),np.zeros((POB_size,))
    FO_best = 0
    FO_best_pos = 0
    X_best = np.zeros(num_pacientes)

    datos_evol_x,datos_evol_y = [],[]


    for i in range(POB_size):
        X[i,:] = np.random.permutation(num_pacientes)
        X_fo[i] = decod3.DECOD3(DATOS, "pacientes", False, False, X[i,:])

    FO_best_pos = int(np.argmax(X_fo))
    FO_best = int(X_fo[FO_best_pos])
    X_best = X[FO_best_pos,:]
    """
    ITERACIONES DEL ALGORITMO DDE
    """
    start_time = time.time()

    iteraciones = 0
    while True:
        if time.time() - start_time >= max_time:
            """
            GUARDAR MEJOR SOLUCION
            """
            FO_best_pos = int(np.argmax(X_fo))
            X_best = X[FO_best_pos,:]
            FO_best = decod3.DECOD3(DATOS, "pacientes", False, False, X_best)
            break

        datos_evol_x.append(time.time() - start_time)
        datos_evol_y.append((int(np.max(X_fo))/num_pacientes)*100 )


        for i in range(POB_size):
```

```
            """
            MUTACION
            """
            V[i,:] = funciones.MUTACION(X[i,:],
X[np.random.randint(0,POB_size),:], X[np.random.randint(0,POB_size),:],
prob_mutacion, fuerza_mutacion)
            """
            RECOMBINACION
            """
            U[i,:] = funciones.RECOMBINACION(X[i,:], V[i,:], prob_recombinacion)
            U_fo[i] = decod3.DECOD3(DATOS, "pacientes", False, False, U[i,:])

            """
            LOCAL SEARCH
            """
            U[i,:] = funciones.FirstImprovement_generalswap(DATOS, U[i,:])
            U_fo[i] = decod3.DECOD3(DATOS, "pacientes", False, False, U[i,:])

            """
            SELECCION
            """
            if U_fo[i] > X_fo[i]:
                X[i,:] = U[i,:]
                X_fo[i] = U_fo[i]
        iteraciones = iteraciones + 1



    """
    GENERAR GANTT
    """
    funciones.plot_evol(datos_evol_x,datos_evol_y,"DDE LS")

    return FO_best


"""
DDE-P1 ----------------------------------------------------------------------
------------------------------------------------------------------------------
-------------------------------------------------------------------
"""
def
DDE_P1(POB_size,prob_mutacion,prob_recombinacion,fuerza_mutacion,num_sustitucion
,max_time,excel,DATASET_n,version):

    """
    CARGA INICIAL DE DATOS
    """
```

```
    num_dias,tiempo_dias,num_OR,mds,num_pacientes,num_cirujanos,tiempo_trabajo,m
ovimientomax_cirujanos,pacientes,secuencia,Dijh = excel
    DATOS =
num_dias,tiempo_dias,num_OR,mds,num_pacientes,num_cirujanos,tiempo_trabajo,movim
ientomax_cirujanos,pacientes,secuencia,Dijh

    """
    CREACION DE LA POBLACION INICIAL. FULL ALEATORIA
    """

    repeticion = np.zeros((POB_size,2))
    X,V,U =
np.zeros((POB_size,num_pacientes)),np.zeros((POB_size,num_pacientes)),np.zeros((
POB_size,num_pacientes))
    X_fo,U_fo = np.zeros((POB_size,)),np.zeros((POB_size,))
    FO_best = 0
    FO_best_pos = 0
    X_best = np.zeros(num_pacientes)

    datos_evol_x,datos_evol_y = [],[]


    for i in range(POB_size):
        X[i,:] = np.random.permutation(num_pacientes)
        X_fo[i] = decod3.DECOD3(DATOS, "pacientes", False, False, X[i,:])

    repeticion[:,0] = X_fo[:]
    FO_best_pos = int(np.argmax(X_fo))
    FO_best = int(X_fo[FO_best_pos])
    X_best = X[FO_best_pos,:]

    """
    ITERACIONES DEL ALGORITMO DDE
    """

    start_time = time.time()

    iteraciones = 0
    while True:
        if time.time() - start_time >= max_time:
            FO_best_pos = int(np.argmax(X_fo))
            X_best = X[FO_best_pos,:]
            FO_best = decod3.DECOD3(DATOS, "pacientes", False, False, X_best)
            break


        datos_evol_x.append(time.time() - start_time)
        datos_evol_y.append((int(np.max(X_fo))/num_pacientes)*100 )
```

```
        for i in range(POB_size):

            """
            MUTACION
            """
            V[i,:] = funciones.MUTACION(X[i,:],
X[np.random.randint(0,POB_size),:], X[np.random.randint(0,POB_size),:],
prob_mutacion, fuerza_mutacion)

            """
            RECOMBINACION
            """
            U[i,:] = funciones.RECOMBINACION(X[i,:], V[i,:], prob_recombinacion)
            U_fo[i] = decod3.DECOD3(DATOS, "pacientes", False, False, U[i,:])

            """
            SELECCION
            """
            if U_fo[i] > X_fo[i]:
                X[i,:] = U[i,:]
                X_fo[i] = U_fo[i]

            """
            MONITORIZAR MEJORA
            """
            if repeticion[i,0] == X_fo[i]:
                repeticion[i,1] = repeticion[i,1] + 1
            else:
                repeticion[i,1] = 0
            repeticion[i,0] = X_fo[i]

            """
            REALIZAR SUSTITUCION
            """
            FO_best = np.max(X_fo[:])
            if repeticion[i,1] >= num_sustitucion and X_fo[i] < FO_best :
                X[i,:] = np.random.permutation(num_pacientes)
                X_fo[i] = decod3.DECOD3(DATOS, "pacientes", False, False,
X[i,:])
                repeticion[i,0] = X_fo[i]
                repeticion[i,1] = 0

        iteraciones = iteraciones + 1
_best, X_best)

    """
    GENERAR GANTT
    """
```

```
    funciones.plot_evol(datos_evol_x,datos_evol_y,"DDE P1")

    return FO_best

"""
 ------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------
"""
"""
DDE-P2 ------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------
"""
def
DDE_P2(POB_size,prob_mutacion,prob_recombinacion,fuerza_mutacion,num_sustitucion
,porcentaje,max_time,excel,DATASET_n,version):

    """
    CARGA INICIAL DE DATOS
    """

    num_dias,tiempo_dias,num_OR,mds,num_pacientes,num_cirujanos,tiempo_trabajo,m
ovimientomax_cirujanos,pacientes,secuencia,Dijh = excel
    DATOS =
num_dias,tiempo_dias,num_OR,mds,num_pacientes,num_cirujanos,tiempo_trabajo,movim
ientomax_cirujanos,pacientes,secuencia,Dijh

    """
    CREACION DE LA POBLACION INICIAL. FULL ALEATORIA
    """
    repeticion = np.zeros((POB_size,2))
    X,V,U =
np.zeros((POB_size,num_pacientes)),np.zeros((POB_size,num_pacientes)),np.zeros((
POB_size,num_pacientes))
    X_fo,U_fo = np.zeros((POB_size,)),np.zeros((POB_size,))
    FO_best = 0
    FO_best_pos = 0
    X_best = np.zeros(num_pacientes)

    datos_evol_x,datos_evol_y = [],[]


    for i in range(POB_size):
        X[i,:] = np.random.permutation(num_pacientes)
        X_fo[i] = decod3.DECOD3(DATOS, "pacientes", False, False, X[i,:])

    repeticion[:,0] = X_fo[:]
    FO_best_pos = int(np.argmax(X_fo))
```

```
    FO_best = int(X_fo[FO_best_pos])
    X_best = X[FO_best_pos,:]

    """
    ITERACIONES DEL ALGORITMO DDE
    """

    start_time = time.time()

    iteraciones = 0
    while True:
        if time.time() - start_time >= max_time:
            FO_best_pos = int(np.argmax(X_fo))
            X_best = X[FO_best_pos,:]
            FO_best = decod3.DECOD3(DATOS, "pacientes", False, False, X_best)
            break


        datos_evol_x.append(time.time() - start_time)
        datos_evol_y.append((int(np.max(X_fo))/num_pacientes)*100 )


        for i in range(POB_size):

            """
            MUTACION
            """
            V[i,:] = funciones.MUTACION(X[i,:],
X[np.random.randint(0,POB_size),:], X[np.random.randint(0,POB_size),:],
prob_mutacion, fuerza_mutacion)

            """
            RECOMBINACION
            """
            U[i,:] = funciones.RECOMBINACION(X[i,:], V[i,:], prob_recombinacion)
            U_fo[i] = decod3.DECOD3(DATOS, "pacientes", False, False, U[i,:])

            """
            SELECCION
            """
            if U_fo[i] > X_fo[i]:
                X[i,:] = U[i,:]
                X_fo[i] = U_fo[i]

            """
            MONITORIZAR MEJORA
            """
            if repeticion[i,0] == X_fo[i]:
                repeticion[i,1] = repeticion[i,1] + 1
```

```
            else:
                repeticion[i,1] = 0
            repeticion[i,0] = X_fo[i]

            """
            REALIZAR SUSTITUCION
            """
            if repeticion[i,1] >= num_sustitucion and X_fo[i] <=
np.percentile(X_fo[:],porcentaje) :
                X[i,:] = np.random.permutation(num_pacientes)
                X_fo[i] = decod3.DECOD3(DATOS, "pacientes", False, False,
X[i,:])
                repeticion[i,0] = X_fo[i]
                repeticion[i,1] = 0

        iteraciones = iteraciones + 1
    """
    GENERAR GANTT
    """
    funciones.plot_evol(datos_evol_x,datos_evol_y,"DDE P2")

    return FO_best

"""
 ----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------
"""

"""
DDE-P3 ----------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------
"""
def
DDE_P3(POB_size,prob_mutacion,prob_recombinacion,fuerza_mutacion,num_sustitucion
,porcentaje,max_time,excel,DATASET_n,version):

    """
    CARGA INICIAL DE DATOS
    """
    num_dias,tiempo_dias,num_OR,mds,num_pacientes,num_cirujanos,tiempo_trabajo,m
ovimientomax_cirujanos,pacientes,secuencia,Dijh = excel
    DATOS =
num_dias,tiempo_dias,num_OR,mds,num_pacientes,num_cirujanos,tiempo_trabajo,movim
ientomax_cirujanos,pacientes,secuencia,Dijh

    """
    CREACION DE LA POBLACION INICIAL. FULL ALEATORIA
```

```
    """
    repeticion = np.zeros((POB_size,2))
    X,V,U =
np.zeros((POB_size,num_pacientes)),np.zeros((POB_size,num_pacientes)),np.zeros((
POB_size,num_pacientes))
    X_fo,U_fo = np.zeros((POB_size,)),np.zeros((POB_size,))
    FO_best = 0
    FO_best_pos = 0
    X_best = np.zeros(num_pacientes)

    datos_evol_x,datos_evol_y = [],[]


    for i in range(POB_size):
        X[i,:] = np.random.permutation(num_pacientes)
        X_fo[i] = decod3.DECOD3(DATOS, "pacientes", False, False, X[i,:])

    repeticion[:,0] = X_fo[:]
    FO_best_pos = int(np.argmax(X_fo))
    FO_best = int(X_fo[FO_best_pos])
    X_best = X[FO_best_pos,:]

    """
    ITERACIONES DEL ALGORITMO DDE
    """

    start_time = time.time()

    iteraciones = 0
    while True:
        if time.time() - start_time >= max_time or POB_size <= 3:
            FO_best_pos = int(np.argmax(X_fo))
            X_best = X[FO_best_pos,:]
            FO_best = decod3.DECOD3(DATOS, "pacientes", False, False, X_best)
            break


        datos_evol_x.append(time.time() - start_time)
        datos_evol_y.append((int(np.max(X_fo))/num_pacientes)*100 )


        for i in range(POB_size):

            """
            MUTACION
            """
            V[i,:] = funciones.MUTACION(X[i,:],
X[np.random.randint(0,POB_size),:], X[np.random.randint(0,POB_size),:],
prob_mutacion, fuerza_mutacion)
```

```
            """
            RECOMBINACION
            """
            U[i,:] = funciones.RECOMBINACION(X[i,:], V[i,:], prob_recombinacion)
            U_fo[i] = decod3.DECOD3(DATOS, "pacientes", False, False, U[i,:])

            """
            SELECCION
            """
            if U_fo[i] > X_fo[i]:
                X[i,:] = U[i,:]
                X_fo[i] = U_fo[i]

            """
            MONITORIZAR MEJORA
            """
            if repeticion[i,0] == X_fo[i]:
                repeticion[i,1] = repeticion[i,1] + 1
            else:
                repeticion[i,1] = 0
            repeticion[i,0] = X_fo[i]

            """
            REALIZAR SUSTITUCION
            """
        for i in range(POB_size):
            if repeticion[i,1] >= num_sustitucion and X_fo[i]
<=np.percentile(X_fo[:],porcentaje) :
                X = np.delete(X,i,axis=0)
                X_fo = np.delete(X_fo,i)
                repeticion = np.delete(repeticion,i,axis=0)
                break
        POB_size = len(X)



        iteraciones = iteraciones + 1

    """
    GENERAR GANTT
    """
    funciones.plot_evol(datos_evol_x,datos_evol_y,"DDE P3")

    return FO_best

"""
```

```
 ------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------
"""
"""
 DDE-P4 ------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------
"""
def
DDE_P4(tamaño,prob_mutacion,prob_recombinacion,fuerza_mutacion,max_time,excel,DA
TASET_n,version):


    """
    CARGA INICIAL DE DATOS
    """
    num_dias,tiempo_dias,num_OR,mds,num_pacientes,num_cirujanos,tiempo_trabajo,m
ovimientomax_cirujanos,pacientes,secuencia,Dijh = excel
    DATOS =
num_dias,tiempo_dias,num_OR,mds,num_pacientes,num_cirujanos,tiempo_trabajo,movim
ientomax_cirujanos,pacientes,secuencia,Dijh


    """
    CREACION DE LA POBLACION INICIAL. FULL ALEATORIA
    """
    POB_size = int(num_pacientes/tamaño)
    X,V,U =
np.zeros((POB_size,num_pacientes)),np.zeros((POB_size,num_pacientes)),np.zeros((
POB_size,num_pacientes))
    X_fo,U_fo = np.zeros((POB_size,)),np.zeros((POB_size,))
    FO_best = 0
    FO_best_pos = 0
    X_best = np.zeros(num_pacientes)

    datos_evol_x,datos_evol_y = [],[]


    for i in range(POB_size):
        X[i,:] = np.random.permutation(num_pacientes)
        X_fo[i] = decod3.DECOD3(DATOS, "pacientes", False, False, X[i,:])

    FO_best_pos = int(np.argmax(X_fo))
    FO_best = int(X_fo[FO_best_pos])
    X_best = X[FO_best_pos,:]


    """
```

```
    ITERACIONES DEL ALGORITMO DDE
    """
    start_time = time.time()

    iteraciones = 0
    while True:
        if time.time() - start_time >= max_time:
            """
            GUARDAR MEJOR SOLUCION
            """
            FO_best_pos = int(np.argmax(X_fo))
            X_best = X[FO_best_pos,:]
            FO_best = decod3.DECOD3(DATOS, "pacientes", False, False, X_best)
            break


        datos_evol_x.append(time.time() - start_time)
        datos_evol_y.append((int(np.max(X_fo))/num_pacientes)*100 )


        for i in range(POB_size):

            """
            MUTACION
            """
            V[i,:] = funciones.MUTACION(X[i,:],
X[np.random.randint(0,POB_size),:], X[np.random.randint(0,POB_size),:],
prob_mutacion, fuerza_mutacion)

            """
            RECOMBINACION
            """
            U[i,:] = funciones.RECOMBINACION(X[i,:], V[i,:], prob_recombinacion)
            U_fo[i] = decod3.DECOD3(DATOS, "pacientes", False, False, U[i,:])

            """
            SELECCION
            """
            if U_fo[i] > X_fo[i]:
                X[i,:] = U[i,:]
                X_fo[i] = U_fo[i]
        iteraciones = iteraciones + 1



    """
    GENERAR GANTT
    """
```

```
    funciones.plot_evol(datos_evol_x,datos_evol_y,"DDE P4")



    return FO_best

if __name__ == "__main__":

    """
    SIMULACION FINAL.
    """
    resultados_FO = np.zeros((7,240))
    instancia = 0
    for i in range(16):
        for j in range(15):
            num_dias,tiempo_dias,num_OR,mds,num_pacientes,num_cirujanos,tiempo_t
rabajo,movimientomax_cirujanos,pacientes,secuencia,Dijh =
funciones.CargarDatos("DATOS.xlsx",i,j)
            DATOS =
num_dias,tiempo_dias,num_OR,mds,num_pacientes,num_cirujanos,tiempo_trabajo,movim
ientomax_cirujanos,pacientes,secuencia,Dijh

            #DDE(POB_size, prob_mutacion, prob_recombinacion, fuerza_mutacion,
max_time, excel, DATASET_n, version)
            resultados_FO[0,instancia] = DDE(10,0.8, 0.8, 0.5, 15, DATOS, i, j)
            #DDE_LS(POB_size, prob_mutacion, prob_recombinacion,
fuerza_mutacion, max_time, excel, DATASET_n, version)
            resultados_FO[1,instancia] = DDE_LS(10, 0.8, 0.8, 0.5, 15, DATOS, i,
j)
            #DDE_S1(POB_size, prob_mutacion, prob_recombinacion,
fuerza_mutacion, max_time, excel, DATASET_n, version)
            resultados_FO[2,instancia] = DDE_S1(10, 0.8, 0.8, 0.5, 15, DATOS, i,
j)
            #DDE_P1(POB_size, prob_mutacion, prob_recombinacion,
fuerza_mutacion, num_sustitucion, max_time, excel, DATASET_n, version)
            resultados_FO[3,instancia] = DDE_P1(10, 0.8, 0.8, 0.5, 10, 15,
DATOS, i, j)
            #DDE_P2(POB_size, prob_mutacion, prob_recombinacion,
fuerza_mutacion, num_sustitucion, porcentaje, max_time, excel, DATASET_n,
version)
            resultados_FO[4,instancia] = DDE_P2(10, 0.8, 0.8, 0.5, 10, 0.7, 15,
DATOS, i, j)
            #DDE_P3(POB_size, prob_mutacion, prob_recombinacion,
fuerza_mutacion, num_sustitucion, porcentaje, max_time, excel, DATASET_n,
version)
            resultados_FO[5,instancia] = DDE_P3(10, 0.8, 0.8, 0.5, 10, 0.7, 15,
DATOS, i, j)
            #DDE_P4(tamaño, prob_mutacion, prob_recombinacion, fuerza_mutacion,
max_time, excel, DATASET_n, version)
```

```
            resultados_FO[6,instancia] = DDE_P4(5, 0.8, 0.8, 0.5, 15, DATOS, i,
j)
            instancia = instancia + 1

    """
    ANALISIS DE PERFORMANCE DE ALGORITMOS
    """
    #Establecer el dataset a analizar
    i = 14
    j = 12

    #Carga de datos
    num_dias,tiempo_dias,num_OR,mds,num_pacientes,num_cirujanos,tiempo_trabajo,m
ovimientomax_cirujanos,pacientes,secuencia,Dijh =
funciones.CargarDatos("DATOS_FINALES_MEMORIA.xlsx",i,j)
    DATOS =
num_dias,tiempo_dias,num_OR,mds,num_pacientes,num_cirujanos,tiempo_trabajo,movim
ientomax_cirujanos,pacientes,secuencia,Dijh

    #Algoritmo
    for index in range(1):
        res1 = DDE(10,0.8, 0.8, 0.5, 15, DATOS, i, j)
        res2 = DDE_LS(10, 0.8, 0.8, 0.5, 15, DATOS, i, j)
        res3 = DDE_S1(10, 0.8, 0.8, 0.5, 30, DATOS, i, j)
        res4 = DDE_P1(10, 0.8, 0.8, 0.5, 10, 15, DATOS, i, j)
        res5 = DDE_P2(4, 0.8, 0.8, 0.5, 10, 0.7, 15, DATOS, i, j)
        res6 = DDE_P3(15, 0.8, 0.8, 0.5, 10, 0.7, 15, DATOS, i, j)
        res7 = DDE_P4(5, 0.8, 0.8, 0.5, 15, DATOS, i, j)

"""                 *

"""#GUARDAR CALIBRACION (PROVISIONAL)
"""
with pd.ExcelWriter('resultados.xlsx') as writer:
    PARAMETROS_DF = pd.DataFrame(RESULTADOS)
    PARAMETROS_DF.to_excel(writer, sheet_name="PARAMETROS", index=True)

    VALORES_DF = pd.DataFrame(PARAMETROS_RESULTADOS)
    VALORES_DF.to_excel(writer, sheet_name="VALORES", index=True)
"""
```

LOCALSEARCH.PY

```
import funciones
import decod3
import numpy as np

#Funciones para vecindades
def insert(secuencia, outerindex, innerindex):
    secuencia_loop = np.copy(secuencia)
    x = np.delete(secuencia_loop, outerindex)
    secuencia_loop = np.insert(x, innerindex, secuencia[outerindex])
    return secuencia_loop

def generalswap(secuencia, outerindex, innerindex):
    secuencia_loop = np.copy(secuencia)
    secuencia_loop[outerindex], secuencia_loop[innerindex] =
secuencia_loop[innerindex], secuencia_loop[outerindex]
    return secuencia_loop

def adjacentswap(secuencia, index):
    secuencia_loop = np.copy(secuencia)
    secuencia_loop[index], secuencia_loop[index + 1] = secuencia_loop[index +
1], secuencia_loop[index]
    return secuencia_loop

#1. General Swap
@funciones.tiempo_valor
def FirstImprovement_generalswap(DATOS,secuencia):
    num_pacientes = DATOS[4]
    mejorobj = decod3.DECOD3(DATOS, "pacientes", False, False, secuencia)
    mejora = False
    for j in range(num_pacientes):
        if mejora == True: break
        for i in range(num_pacientes):
            secuencia_loop = generalswap(secuencia, j, i)
            obj = decod3.DECOD3(DATOS, "pacientes", False, False,
secuencia_loop)
            if obj < mejorobj:
                mejora = True
                mejorobj = obj
            if mejora == True: break

#2. Adjacent Swap
@funciones.tiempo_valor
def FirstImprovement_adjacentswap(DATOS,secuencia):
    num_pacientes = DATOS[4]
    mejorobj = decod3.DECOD3(DATOS, "pacientes", False, False, secuencia)
    mejora = False
    for j in range(num_pacientes-1):
```

```
        secuencia_loop = adjacentswap(secuencia, j)
        obj = decod3.DECOD3(DATOS, "pacientes", False, False, secuencia_loop)
        if obj < mejorobj:
            mejora = True
            mejorobj = obj
        if mejora == True: break

# 3. 3-opt Local Search
@funciones.tiempo_valor
def LS_3_OPT(DATOS, secuencia):
    num_pacientes = DATOS[4]
    mejorobj = decod3.DECOD3(DATOS, "pacientes", False, False, secuencia)
    mejora = False
    for i in range(num_pacientes - 2):
        if mejora == True: break
        for j in range(i + 1, num_pacientes - 1):
            if mejora == True: break
            for k in range(j + 1, num_pacientes):
                if mejora == True: break
                # Three possible combinations for edge swapping
                for seq in [[i, j, k], [i, k, j], [j, i, k]]:
                    secuencia_loop = secuencia.copy()
                    secuencia_loop[seq[0]:seq[1]+1] =
secuencia_loop[seq[0]:seq[1]+1][::-1]
                    secuencia_loop[seq[1]+1:seq[2]+1] =
secuencia_loop[seq[1]+1:seq[2]+1][::-1]
                    obj = decod3.DECOD3(DATOS, "pacientes", False, False,
secuencia_loop)
                    if obj > mejorobj:
                        mejorobj = obj
                        mejora = True



#4. Insertion
@funciones.tiempo_valor
def FirstImprovement_insert(DATOS, secuencia):
    num_pacientes = DATOS[4]
    mejorobj = decod3.DECOD3(DATOS, "pacientes", False, False, secuencia)
    mejora = False
    for j in range(num_pacientes):
        if mejora == True: break
        for i in range(num_pacientes):
            secuencia_loop = insert(secuencia,j,i)
            obj = decod3.DECOD3(DATOS, "pacientes", False, False,
secuencia_loop)
            if obj < mejorobj:
                mejora = True
                mejorobj = obj
```

```
            if mejora == True: break


timing = np.zeros((4,36))
index = 0
for index_n in range(4):
    for version in range(9):
        num_dias,tiempo_dias,num_OR,mds,num_pacientes,num_cirujanos,tiempo_traba
jo,movimientomax_cirujanos,pacientes,secuencia,Dijh =
funciones.CargarDatos("DATOS_CALIBRACION_LS.xlsx",index_n,version)
        DATOS =
num_dias,tiempo_dias,num_OR,mds,num_pacientes,num_cirujanos,tiempo_trabajo,movim
ientomax_cirujanos,pacientes,secuencia,Dijh
        secuencia = np.random.permutation(num_pacientes)

        timing[0,index] = FirstImprovement_generalswap(DATOS, secuencia)
        timing[1,index] = FirstImprovement_adjacentswap(DATOS, secuencia)
        timing[2,index] = LS_3_OPT(DATOS, secuencia)
        timing[3,index] = FirstImprovement_insert(DATOS, secuencia)

        index = index + 1
```

DECOD1.PY

```
import numpy as np
import funciones
"""

DECODIFICACION DE LA SOLUCION -> OR con menos carga a partir de solucion inicial

"""
def DECOD1(DATOS,FO,GRAFICO,SOLUCION,lista_inicial=None):
    #Desempaquetado de datos
    num_dias,tiempo_dias,num_OR,mds,num_pacientes,num_cirujanos,tiempo_trabajo,m
ovimientomax_cirujanos,pacientes,secuencia,Dijh = DATOS
    #Poner capacidades en type correcto
    tiempo_dias = tiempo_dias.astype(float)
    pacientes[:,5] = pacientes[:,5].astype(int)
    #Comprobar si se da una secuencia, si no se da,usaremos la del excel
    if lista_inicial is not None:
        secuencia = lista_inicial.astype(int)
    #Resetear valores

    #Xijh
    Xijh = np.zeros((num_pacientes, num_OR, num_dias))

    #Zsjh
    Zsjh = np.zeros((num_cirujanos, num_OR, num_dias))

    capacidad_OR = np.full((num_dias, num_OR), tiempo_trabajo).astype(float)
    capacidad_cirujanos = np.full((num_dias, num_cirujanos),
tiempo_trabajo).astype(float)
    ct_OR=np.zeros((num_dias,num_OR))
    ct_cirujanos=np.zeros((num_dias,num_cirujanos))
    dias_max_cirujano = np.full((num_dias,num_cirujanos), 0)
    pacientes[:,5] = np.zeros(num_pacientes)
    dt = np.dtype([('startingtime', np.float64), ('finishingtime', np.float64),
('OR', np.int64), ('cirujano', int)])
    solucion = np.zeros((num_dias, num_pacientes), dtype=dt) #[día,paciente] ->
(startingtime, finishingtime, OR, cirujano)

    """
    DECODING 1 : SEC + MEDICO --> F.O
    """
    for i in secuencia:
        if pacientes[i,5] == 0: #Paciente no ha sido operado
            cirujano = int(pacientes[i,4])
            for h in range(num_dias):
                if pacientes[i,5] == 1: break
                if pacientes[i,0]<= h: #Condicion paciente
                    #print(h,pacientes[i,6],pacientes[i,3],"|", cirujano)
```

```
                    for j in funciones.indices_ordenados_menormayor(ct_OR[h]):
#Vamos de OR con disponibilidad mas temprana a mas tardia, el bucle para en
cuanto encontramos solucion
                        startingtime = max(ct_OR[h,j],ct_cirujanos[h,cirujano])
                        tiempo_restante = tiempo_dias[h] - startingtime
                        if pacientes[i,5] == 1: break
                        if funciones.PuedeMoverseCirujano(Zsjh, cirujano, j,
movimientomax_cirujanos, h) and funciones.DiasMaxCirujano(dias_max_cirujano,
mds, cirujano, h): #Condicion cirujano
                            if pacientes[i, 3] < tiempo_restante and
Dijh[int(pacientes[i,6]),j,h] == 1: #Condicion OR
                                #Se puede hacer en la OR seleccionada
                                Zsjh[cirujano, j, h] =  1
                                Xijh[int(pacientes[i, 6]), j, h] =  1
                                dias_max_cirujano[h,cirujano] = 1
                                pacientes[i,5] = 1
                                ct_OR[h,j] = pacientes[i,3] + startingtime
                                ct_cirujanos[h,cirujano] = pacientes[i,3] +
startingtime
                                capacidad_cirujanos[h,cirujano] -=
pacientes[i,3]
                                capacidad_OR[h, j] -= pacientes[i,3]
                                solucion[h,i] = (startingtime, pacientes[i,3] +
startingtime, j, cirujano)
                                #print(h,ct_OR[h,:],ct_cirujanos[h,:],"\n")
                                break

    SOL =
num_dias,tiempo_dias,num_OR,mds,num_pacientes,num_cirujanos,capacidad_OR,capacid
ad_cirujanos,movimientomax_cirujanos,pacientes,secuencia,Dijh, Xijh,
Zsjh,solucion
    pacientes_operados = np.sum(pacientes[:,5])
    rendimiento = np.mean((1 - capacidad_OR / 480))
    if GRAFICO == True:
        funciones.Gantt(solucion)
    if SOLUCION == False:
        if FO == "pacientes":
            return pacientes_operados
        if FO == "rendimiento":
            return rendimiento
        if FO == "todo":
            return pacientes_operados,rendimiento
    else:
        if FO == "pacientes":
            return pacientes_operados,SOL
        if FO == "rendimiento":
            return rendimiento,SOL
        if FO == "todo":
            return pacientes_operados,rendimiento,SOL
```

```
"""
EJECUCION DE DECOD1 PARA PRUEBAS
"""
if __name__ == "__main__":
        num_dias,tiempo_dias,num_OR,mds,num_pacientes,num_cirujanos,tiempo_traba
jo,movimientomax_cirujanos,pacientes,secuencia,Dijh =
funciones.CargarDatos("Datos.xlsx",0)
        DATOS =
num_dias,tiempo_dias,num_OR,mds,num_pacientes,num_cirujanos,tiempo_trabajo,movim
ientomax_cirujanos,pacientes,secuencia,Dijh

        secuencia = np.random.permutation(num_pacientes)
        AANUM_FINAL_PAC, AARENDIMIENTO, SOLUCIONCOMPLETA  = DECOD1(DATOS,"todo",
True,True,secuencia)
        print(f"Pacientes operados:{AANUM_FINAL_PAC}.
Rendimiento:{AARENDIMIENTO}. Dataset-{0}")
```

DECOD2.PY

```
import numpy as np
import funciones

"""

DECODIFICACION DE LA SOLUCION -> OR con menos carga a partir de solucion inicial

"""

num_dias,tiempo_dias,num_OR,num_pacientes,num_cirujanos,capacidad_OR,capacidad_c
irujanos,movimientomax_cirujanos,pacientes,secuencia,Dijh =
funciones.CargarDatos()

#Xijh
Xijh = np.zeros((num_pacientes, num_OR, num_dias))

#Zsjh
Zsjh = np.zeros((num_cirujanos, num_OR, num_dias))

#Dsi (matriz para ver que operaciones puede hacer cada medico)------------------
-------------------------------------
Dsi = np.random.choice([0,1],size=(num_cirujanos,num_pacientes),p=[0.3,0.7])


ct_OR=np.zeros((num_dias,num_OR))
ct_cirujanos=np.zeros((num_dias,num_cirujanos))
dt = np.dtype([('startingtime', np.float64), ('finishingtime', np.float64),
('OR', np.int64), ('cirujano', int)])
solucion = np.zeros((num_dias, num_pacientes), dtype=dt) #[día,paciente] ->
(startingtime, finishingtime, OR, cirujano)

"""
DECODING 2 : SEC --> MEDICO + F.O
"""

for i in secuencia:
    if pacientes[i,5] == 0: #Paciente no ha sido operado
        for h in range(num_dias):
            if pacientes[i,5] == 1: break
            if pacientes[i,0]<= h: #Condicion paciente
                #print(h,ct_OR[h,:],ct_cirujanos[h,:],max(ct_OR[h,OR],ct_cirujan
os[h,cirujano]))
                #print(h,pacientes[i,6],pacientes[i,3],"|",OR, cirujano)
                for j in
funciones.indices_ordenados_mayormenor(capacidad_OR[h,:]): #Vamos de OR con
mayor capacidad a menos, el bucle para en cuanto encontramos solucion
```

```
                    for cirujano in
funciones.indices_ordenados_menormayor(ct_cirujanos[h,:]):
                        if Dsi[cirujano,i] ==1:
                            if pacientes[i,5] == 1: break
                            if pacientes[i,3] <  capacidad_cirujanos[h,cirujano]
and funciones.PuedeMoverseCirujano(Zsjh, cirujano, j, movimientomax_cirujanos,
h): #Condicion cirujano
                                if pacientes[i, 3] < capacidad_OR[h, j] and
Dijh[int(pacientes[i,6]),j,h] == 1: #Condicion OR
                                    #Se puede hacer en la OR seleccionada
                                    Zsjh[cirujano, j, h] =  1
                                    Xijh[int(pacientes[i, 6]), j, h] =  1
                                    pacientes[i,5] = 1
                                    startingtime =
max(ct_OR[h,j],ct_cirujanos[h,cirujano])
                                    ct_OR[h,j] = pacientes[i,3] + startingtime
                                    ct_cirujanos[h,cirujano] = pacientes[i,3] +
startingtime
                                    capacidad_cirujanos[h,cirujano] -=
pacientes[i,3]
                                    capacidad_OR[h, j] = tiempo_dias[h] -
ct_OR[h,j]
                                    solucion[h,i] = (startingtime,
pacientes[i,3] + startingtime, j, cirujano)
                                    #print(h,ct_OR[h,:],ct_cirujanos[h,:],"\n")
                                    break


funciones.Gantt(solucion)
FO = np.sum(pacientes[:,5])
print(f"El valor de la FO es: {int(FO)}")
```

DECOD3.PY

```
import numpy as np
import funciones
"""

DECODIFICACION DE LA SOLUCION -> OR con menos carga a partir de solucion inicial

"""
def DECOD3(DATOS,FO,GRAFICO,SOLUCION,lista_inicial=None):
    #Desempaquetado de datos
    num_dias,tiempo_dias,num_OR,mds,num_pacientes,num_cirujanos,tiempo_trabajo,m
ovimientomax_cirujanos,pacientes,secuencia,Dijh = DATOS
    #Poner capacidades en type correcto
    tiempo_dias = tiempo_dias.astype(float)
    pacientes[:,5] = pacientes[:,5].astype(int)
    #Comprobar si se da una secuencia, si no se da,usaremos la del excel
    if lista_inicial is not None:
        secuencia = lista_inicial.astype(int)
    #Resetear valores

    #Xijh
    Xijh = np.zeros((num_pacientes, num_OR, num_dias))

    #Zsjh
    Zsjh = np.zeros((num_cirujanos, num_OR, num_dias))

    capacidad_OR = np.full((num_dias, num_OR), tiempo_trabajo).astype(float)
    capacidad_cirujanos = np.full((num_dias, num_cirujanos),
tiempo_trabajo).astype(float)
    ct_OR=np.zeros((num_dias,num_OR))
    ct_cirujanos=np.zeros((num_dias,num_cirujanos))
    dias_max_cirujano = np.full((num_dias,num_cirujanos), 0)
    pacientes[:,5] = np.zeros(num_pacientes)
    dt = np.dtype([('startingtime', np.float64), ('finishingtime', np.float64),
('OR', np.int64), ('cirujano', int)])
    solucion = np.zeros((num_dias, num_pacientes), dtype=dt) #[día,paciente] ->
(startingtime, finishingtime, OR, cirujano)

    """
    DECODING 1 : SEC + MEDICO --> F.O
    """
    for i in secuencia:
        if pacientes[i,5] == 0: #Paciente no ha sido operado
            cirujano = int(pacientes[i,4])
            for h in range(num_dias):
                if pacientes[i,5] == 1: break
                if pacientes[i,0]<= h: #Condicion paciente
```

```
                    #print(h,ct_OR[h,:],ct_cirujanos[h,:],max(ct_OR[h,OR],ct_cir
ujanos[h,cirujano]))
                    #print(h,pacientes[i,6],pacientes[i,3],"|", cirujano)
                    #for j in funciones.indices_ordenados_menormayor(ct_OR[h]):
#Vamos de OR con disponibilidad mas temprana a mas tardia, el bucle para en
cuanto encontramos solucion
                    for j in np.argsort(abs(ct_OR[h,:]-
ct_cirujanos[h,cirujano])): #Menor holgura
                        startingtime = max(ct_OR[h,j],ct_cirujanos[h,cirujano])
                        tiempo_restante = tiempo_dias[h] - startingtime
                        if pacientes[i,5] == 1: break
                        if funciones.PuedeMoverseCirujano(Zsjh, cirujano, j,
movimientomax_cirujanos, h) and funciones.DiasMaxCirujano(dias_max_cirujano,
mds, cirujano, h): #Condicion cirujano
                            if pacientes[i, 3] < tiempo_restante and
Dijh[int(pacientes[i,6]),j,h] == 1: #Condicion OR
                                #Se puede hacer en la OR seleccionada
                                Zsjh[cirujano, j, h] =  1
                                Xijh[int(pacientes[i, 6]), j, h] =  1
                                dias_max_cirujano[h,cirujano] = 1
                                pacientes[i,5] = 1
                                ct_OR[h,j] = pacientes[i,3] + startingtime
                                ct_cirujanos[h,cirujano] = pacientes[i,3] +
startingtime
                                capacidad_cirujanos[h,cirujano] -=
pacientes[i,3]
                                capacidad_OR[h, j] -= pacientes[i,3]
                                solucion[h,i] = (startingtime, pacientes[i,3] +
startingtime, j, cirujano)
                                #print(h,ct_OR[h,:],ct_cirujanos[h,:],"\n")
                                break

    SOL =
num_dias,tiempo_dias,num_OR,mds,num_pacientes,num_cirujanos,capacidad_OR,capacid
ad_cirujanos,movimientomax_cirujanos,pacientes,secuencia,Dijh, Xijh,
Zsjh,solucion
    pacientes_operados = np.sum(pacientes[:,5])
    rendimiento = np.mean((1 - capacidad_OR / 480))
    #print(f"El valor de la FO es: {int(FO)}")
    if GRAFICO == True:
        funciones.Gantt(solucion)
    if SOLUCION == False:
        if FO == "pacientes":
            return pacientes_operados
        if FO == "rendimiento":
            return rendimiento
        if FO == "todo":
            return pacientes_operados,rendimiento
    else:
```

```
        if FO == "pacientes":
            return pacientes_operados,SOL
        if FO == "rendimiento":
            return rendimiento,SOL
        if FO == "todo":
            return pacientes_operados,rendimiento,SOL

"""
EJECUCION DE DECOD1 PARA PRUEBAS
"""
if __name__ == "__main__":
        num_dias,tiempo_dias,num_OR,mds,num_pacientes,num_cirujanos,tiempo_traba
jo,movimientomax_cirujanos,pacientes,secuencia,Dijh =
funciones.CargarDatos("pruebabucle.xlsx",17)
        DATOS =
num_dias,tiempo_dias,num_OR,mds,num_pacientes,num_cirujanos,tiempo_trabajo,movim
ientomax_cirujanos,pacientes,secuencia,Dijh

        secuencia = np.random.permutation(num_pacientes)
        AANUM_FINAL_PAC, AARENDIMIENTO, SOLUCIONCOMPLETA  = DECOD3(DATOS,"todo",
True,True)
        print(f"Pacientes operados:{AANUM_FINAL_PAC}.
Rendimiento:{AARENDIMIENTO}. Dataset-{0}")
        #num_dias,tiempo_dias,num_OR,mds,num_pacientes,num_cirujanos,capacidad_O
R,capacidad_cirujanos,movimientomax_cirujanos,pacientes,secuencia,Dijh, Xijh,
Zsjh,solucion = sol



def DECOD1(datos,secuencia):
    desempaquetar los datos
    for paciente in secuencia:
        if paciente no ha sido operado:
            for dia in horizonte de planificacion:
                if release date(paciente) <= dia:
                    for OR in OR ordenados disponibilidad: #Asigna el OR que antes está disponible, si no pasa al siguiente
                        tiempo de inicio de la operacion = max(tiempo de finalizacion OR, tiempo de finalizacon cirujano)
                        if duracion de operacion < capacidad cirujano and cirujano puede moverse a OR:
                            if duracion de operacion < capacidad OR and paciente puede operarse en OR:
                                Xijh,Zsjh = 1
                                tiempo de finalizacion de la operacion = duracion de la operacion + tiempo de inicio de la operacion
                                tiempo de finalizacion de OR = tiempo de finalizacion de la operacion
                                tiempo de finalizacion de cirujano = tiempo de finalizacion de la operacion
                                capacidad OR = duracion dia - tiempo de finalizacion de OR
                                capacidad cirujano -= duracion de la operacion
                                paciente = operado
                                FO = suma(pacientes operados)
```

```
def DECOD2(datos,secuencia):
    desempaquetar los datos
    for paciente in secuencia:
        if paciente no ha sido operado:
            for dia in horizonte de planificacion:
                if release date(paciente) <= dia:
                    for OR in OR ordenados por disponibilidad: #Asigna el OR antes disponible
                        for cirujano in cirujanos ordenados por disponibilidad #Asigna el Cirujano antes disponible
                            if cirujano puede realizar la operacion:
                                tiempo de inicio de la operacion = max(tiempo de finalizacion OR, tiempo de finalizacon cirujano)
                                if duracion de operacion < capacidad cirujano and cirujano puede moverse a OR:
                                    if duracion de operacion < capacidad OR and paciente puede operarse en OR:
                                        #Se puede hacer en la OR seleccionada
                                        Xijh,Zsjh = 1
                                        tiempo de finalizacion de la operacion = duracion de la operacion + tiempo de inicio de la operacion
                                        tiempo de finalizacion de OR = tiempo de finalizacion de la operacion
                                        tiempo de finalizacion de cirujano = tiempo de finalizacion de la operacion
                                        capacidad OR = duracion dia - tiempo de finalizacion de OR
                                        capacidad cirujano -= duracion de la operacion
                                        paciente = operado
                                        FO = suma(pacientes operados)
def DECOD3(datos,secuencia):
    desempaquetar los datos
    for paciente in secuencia:
        if paciente no ha sido operado:
            for dia in horizonte de planificacion:
                if release date(paciente) <= dia:
                    for OR in OrdenadosDeMenorAMayor(abs(tiempo de finalizacion de ORs - tiempo de finalizacion de cirujano)):
                        tiempo de inicio de la operacion = max(tiempo de finalizacion OR, tiempo de finalizacon cirujano)
                        if duracion de operacion < capacidad cirujano and cirujano puede moverse a OR:
                            if duracion de operacion < capacidad OR and paciente puede operarse en OR:
                                #Se puede hacer en la OR seleccionada
                                Xijh,Zsjh = 1
                                tiempo de finalizacion de la operacion = duracion de la operacion + tiempo de inicio de la operacion
                                tiempo de finalizacion de OR = tiempo de finalizacion de la operacion
                                tiempo de finalizacion de cirujano = tiempo de finalizacion de la operacion
                                capacidad OR = duracion dia - tiempo de finalizacion de OR
                                capacidad cirujano -= duracion de la operacion
                                paciente = operado
                                FO = suma(pacientes operados)
```