Trabajo Fin de Máster
Máster en Organización Industrial y Gestión de Empresas

Algoritmos de Optimización para el servicio de urgencias: Caso de estudio en el Hospital Universitario Virgen del Rocío

Autor: David Gómez Medina

Tutor: Víctor Fernández-Viagas Escudero

Tutor externo: José Manuel Molina Pariente

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

Sevilla, 2021

Trabajo Fin de Máster
Máster en Organización Industrial y Gestión de Empresas

Algoritmos de Optimización para el servicio de urgencias: Caso de estudio en el Hospital Universitario Virgen del Rocío

Autor:

David Gómez Medina

Tutores:

Víctor Fernández-Viagas Escudero y José Manuel Molina Pariente

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

Sevilla, 2021

Trabajo Fin de Máster: Algoritmos de Optimización para el servicio de urgencias: Caso de estudio en el Hospital Universitario Virgen del Rocío

Autor: David Gómez Medina

Tutores: Víctor Fernández-Viagas Escudero y José Manuel Molina Pariente

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

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2021

El Secretario del Tribunal

ÍNDICE

Índice de Tablas

Índice de Ilustraciones

Capítulo 1 Introducción

La búsqueda de la eficiencia está cada vez más presente en los diferentes sectores y entre ellos, el sector sanitario. La gestión sanitaria, se encuentra en una situación compleja donde, con recursos limitados y costes de personal, maquinarias y tratamientos muy elevados han de hacer frente a una demanda creciente de pacientes [1]. Es por ello que cada vez más, se destinen recursos y esfuerzos para aumentar y optimizar los recursos hospitalarios a través de decisiones tácticas y organización de los recursos [4].

Esta situación se extiende al servicio de urgencias (SU), el más frecuentado de un hospital [2, 10]. El objetivo principal del SU es poder responder a la necesidad de una atención de emergencia a los pacientes que llegan durante los 365 días del año, 24 horas al día y por ello, se le denomina en muchas ocasiones como la red de seguridad del sistema de atención médica [6]. Este servicio es uno de los componentes clave de la sanidad y más aún cuando se considera uno de los factores por los que se juzga a un hospital [4, 5].

Durante la pandemia declarada el día 11 de marzo por la Organización Mundial de la Salud a causa del SARS-CoV-2 (severe acute respiratory syndrome coronavirus 2) se puso en evidencia la importancia que tiene este servicio. A raíz de la nueva enfermedad, aunque en los primeros meses incrementara la presión hospitalaria, esta presión no se reflejó en el SU. Fueron unos meses después de la declaración del estado de alarmacuando se multiplicó la llegada de pacientes y se tomaron decisiones operativas para hacer frente al estrés organizacional por la falta de recursos llegándose a rediseñar los circuitos, flujos de pacientes y las propias instalaciones de los hospitales para hacer frente a la saturación que se vivió en los primeros meses de pandemia. [21, 22]

La saturación del SU es una realidad y un problema que preocupa cada vez más en la gestión hospitalaria a nivel mundial ya que, al hacer frente a diario de situaciones de vida o muerte, esta saturación puede llegar a tener consecuencias fatídicas [4, 5, 7]. Hay numerosos problemas que se desencadenan a raíz de esta saturación del SU como son el deterioro del estado clínico, retrasos en los tratamientos que requiere un paciente, sobreexplotación de los facultativos que provoca un aumento en las tasas de fallos humanos entre otros, en definitiva, riesgos para el paciente [8, 9].

El origen de esta saturación se podría entender desde una perspectiva de desajuste entre la oferta (capacidad del SU) y la demanda (llegada de pacientes) [2]. De esta manera, una demanda creciente y con mucha variabilidad combinada con una oferta insuficiente o no efectiva provoca que el sistema se sature [3, 11- 13]. Contradictoriamente, de los pacientes que acuden al SU, la mayoría no requieren servicios de emergencia [2, 27]. Las consecuencias de esta saturación del SU pueden provocar el descontento de los pacientes al tener que esperar un tiempo mayor al deseado haciendo que, en algunas ocasiones haya pacientes que abandonen el SU sin llegar a ser atendidos. Ese descontento se extiende también al personal sanitario conllevando un estrés laboral al tener que trabajar por encima de su capacidad provocando posibles errores humanos [15, 17]. En definitiva, la saturación de un sistema tan crítico donde un retraso de minutos en atender a un paciente puede significar un cambio radical en su estado de salud debe reducirse al máximo.

Con el objetivo de poner fin a esta saturación se suelen plantear tres vías de mejora [16]. La primera sería un aumento de los recursos disponibles en el SU, donde la dirección del servicio normalmente se centra en agregar más personal como solución principal a este problema [15]. Esta medida no es siempre posible debido a las limitaciones presupuestarias existentes [18]. La segunda se centra en la gestión de la demanda donde se plantea tanto una mejora en el ajuste de los recursos disponibles, como una redistribución de los pacientes menos urgentes. La tercera es la investigación operativa que actúa normalmente sobre decisiones relacionadas con flujos de pacientes y la búsqueda de un mejor aprovechamiento de los recursos existentes a través de decisiones operativas. Esta vía incluye la toma de decisiones sobre las camas del SU, horarios del personal, programación y asignación de pacientes entre otros [por ejemplo 16,18,19]. Las implantaciones de estas mejoras propuestas a nivel operativo no son triviales, pero sí resultan ser mejoras más consistentes y que perduran en el tiempo [3]. Teniendo en cuenta que la programación y asignación de pacientes tiene un impacto importante sobre el rendimiento del departamento [19] y aprovechando que la programación dinámica de los pacientes en el SU no está muy estudiada en la literatura [24], nos centraremos en ésta en el presente trabajo. A este respecto, hay que constatar que la programación de pacientes a lo largo de su visita al SU no es sencilla, debido a la estocasticidad o variabilidad que presentan factores determinantes como la llegada de los pacientes o el tiempo y tratamientos que requieren que se entienden a través de distribuciones aleatorias y en muchos casos dependientes del tiempo [14,16,23]. Por otro lado, no es trivial realizar un enfoque dinámico de esta programación en la que se tomen decisiones sobre la

ordenación de los pacientes con tasas de refresco de segundos o pocos minutos lo que provoca que enfoques como los modelos de programación lineal pierdan su efectividad para casos en los que haya un número considerable de pacientes en el SU ya que el tiempo de computación sería demasiado elevado [22].

El presente trabajo se centra entonces en la optimización del SU, en particular, en la programación en tiempo real de los pacientes a lo largo de las diferentes etapas por las que pasan cuando llegan al SU. Para ello, se partirán de los modelos de programación lineal entera del SU para la programación de los pacientes presentado por [2]) para diseñar dos algoritmos basados en metaheurísticas conocidas como el algoritmo voraz iterativo (*Iterated Greedy*) y el algoritmo genético. Para la validación de las metaheurísticas, se erealizará un análisis computacional basado en datos reales provenientes del Hospital Universitario Virgen del Rocío en Sevilla, por donde pasan más de 200.000 pacientes al año [25]. Cabe mencionar que actualmente hay tres proyectos de innovación en el HUVR cuya finalidad es la optimización del SU del hospital.

1.1 Objetivos

El objetivo principal de este trabajo es desarrollar diferentes enfoques para la optimización del SU a través de la programación dinámica de los pacientes en tiempo real a través de metaheurísticas.

Los objetivos específicos son los siguientes:

- Diseñar y adaptar metaheurísticas basadas en un algoritmo genético y en un algoritmo voraz iterativo al caso de estudio.
- Validación y realizar de las soluciones propuestas por las metaheurísticas a través de la resolución de problemas basados en datos provenientes del Hospital Virgen del Rocío.
- Determinar el rendimiento de las metaheurísticas propuestas con respecto a metaheurísticas propuestas para el problema en la literatura a través de un extenso análisis computacional.

1.1 Sumario

Este documento se estructura en un total de seis capítulos y un anexo con esta composición:

Capítulo 1: Incluye la presentación del trabajo de fin de máster con una introducción al problema que se tratará a lo largo del mismo, objetivos y composición del documento.

Capítulo 2: Incluye la revisión bibliográfica sobre los artículos que tratan problemas similares al expuesto en el capítulo 1 y que se explicará en profundidad en el capítulo 3.

Capítulo 3: Se describe el problema específico que será objeto de estudio de este proyecto. Se hará tanto una descripción conceptual como un modelado matemático en el que se comenten las restricciones que rigen el problema.

Capítulo 4: Incluye una descripción de la estructura de la solución y posteriormente la descripción de los algoritmos de optimización de los que se partirá para presentar los nuevos algoritmos propuestos.

Capítulo 5: Se describe la generación de datos para la calibración de los algoritmos propuesto y tras su análisis, se procede a la comparación del comportamiento de los diferentes algoritmos de cara a la resolución del problema objeto de estudio.

Capítulo 6: Incluye las conclusiones posteriores al análisis de los resultados, así como propuestas de futuras líneas de investigación.

Anexo: Incluye las líneas de código de programación en C# de los algoritmos originales y propuestos, desarrollados en el capítulo 4.

Capítulo 2 Estado del Arte

Numerosos estudios existentes en la literatura han analizado esta problemática desde diferentes enfoques y simplificaciones, con el fin de analizar, modelar y optimizar la atención hospitalaria, más aún cuando la programación en los servicios de atención médica ha cobrado gran atención en estos últimos años [26]. En la literatura, la investigación a la hora de mejorar el SU se divide en dos grandes bloques: el táctico y el operativo.

El nivel táctico engloba aquellos problemas centrados en un diseño a medio- largo plazo de los recursos disponibles y configuraciones relativas al SU con el fin de satisfacer la demanda que tiene este servicio. Dentro de estas decisiones tácticas se encuentra la programación de los turnos del personal donde [13] emplea un algoritmo genético con búsqueda de soluciones no dominadas debido a una programación multiobjetivo (tratando de minimizar el tiempo de espera de los pacientes y presupuesto del SU). Otro ejemplo de la programación de los turnos del personal se encuentra en [23] en el que se emplean diferentes metamodelos (como redes neuronales, función de base radial, etc.) para la simulación del SU. Con estas simulaciones concluyeron que podía ser utilizada tanto a nivel táctico para una programación más a largo plazo como para decisiones más operativas para hacer frente a situaciones de ausencias de personal. Como estos autores, son muchos más los que hacen frente al problema de programación de turnos del personal desde la perspectiva de la simulación, no sin antes proponer el modelo de programación lineal y su resolución estocástica haciendo uso del método *Sample Average Approximation* [1, 3, 43].

Por otro lado, en el nivel operativo, el principal problema que se aborda y el que se trata en el presente trabajo es la secuenciación de pacientes. Este problema va cobrando importancia ya que se evidenció que este aspecto tiene un efecto significativo en el desempeño del SU de todo hospital [22]. Teniendo en cuenta que son muchos los factores que intervienen en el SU con gran interacción entre ellos, tonto tomar decisiones para mejorar la eficiencia y la calidad de atención médica como establecer estos medidores de desempeño no es algo trivial. En (34), se establecen varios indicadores que sirven al SU para medir la calidad y rendimiento de esta área de atención clínica. De ellos, el más extendido en la literatura es el tiempo de estancia en el SU (conocido como *Length of Stay* LOS) que hace referencia al tiempo total que pasa un paciente en el SU [22]. Otro indicador presente en artículos relacionados con el SU es el *Door to Doctor* [35] donde se mide el tiempo que pasa entre la

llegada de un paciente y su primera consulta. Por último, otros dos indicadores presentes en la literatura son: *left without being seen* (LWBS) que hace referencia al número de pacientes que abandonaron el SU sin recibir ni una consulta [36] y el rendimiento de cada uno de los recursos de urgencias.

Dada la complejidad de este problema, en la literatura hay numerosos enfoques para su resolución entre los que destacan, entre otro: modelos de programación lineal [19] programación dinámica [44], heurísticas constructivas [45] o el uso de metaheurísticas [46, 24]. En la literatura, uno de los artículos que aborda la secuenciación de pacientes es a [45], donde se presenta una heurística constructiva en la que se realiza una asignación de pacientes a facultativos con el objetivo de minimizar los tiempos de aquellos pacientes que presenten una prioridad asistencial más alta junto al objetivo de equilibrar la carga de trabajo entre los facultativos. Esta heurística parte de los tiempos de llegada de los pacientes, así como el código de prioridad asignado y el tiempo de flujo que se estima para elaborar las agendas de cada facultativo.

Por otro lado, [26] afrontaron el problema de secuenciación operativa de pacientes a lo largo del SU (etapas de triaje, consulta, tratamiento y observación) atendiendo tanto a los recursos humanos (facultativos, enfermería de triaje, etc.) como a los recursos materiales disponibles (camas de observación). Para esta resolución propusieron un modelo de programación lineal entera con el fin de reducir el tiempo de espera total de los pacientes en el SU. Estos autores concluyeron que, para su problema, el número de pacientes hasta el que consideraban admisible buscar el óptimo era 24, para un número mayor, el tiempo de computación superaría las dos horas.

Otra perspectiva de la optimización operativa de la secuenciación de pacientes es propuesta por [44], donde emplean una programación dinámica para resolver la programación de consultas de pacientes basándose en la asignación a slots de tiempo. A través de su metodología, programan a los pacientes de nuevo ingreso junto a los que ya han llegado anteriormente en aquellos slots libres y consumen tiempos desconocidos a priori de facultativos con diferentes habilidades que pueden atender o no, las patologías que presentan los pacientes. En [46], los autores se centran en la secuenciación de las pruebas de laboratorio del SU a través de un algoritmo genético priorizándolos a través de una prioridad asistencial o coeficiente de triaje. Concluyeron tras haber presentado el modelado a través de programación lineal entera del problema al que hacen frente, que, debido al número de pacientes y restricciones, no era posible una resolución exacta en un tiempo considerable. Hay

que destacar que, en el artículo, los autores calibran los diversos parámetros del algoritmo haciendo uso de la metodología RSM (*Response Surface Methodology).* Con este algoritmo, los autores consiguieron reducir el tiempo total de espera ponderado de los pacientes del SU.

Más recientemente, se puede encontrar que en [22] emplean una metaheurística basada en vecindad para la secuenciación de pacientes en el SU a través de todas las etapas más comunes. La metaheurística empleada es la conocida como ILS (*Iterated Local Search*) donde emplean de manera iterativa una exploración de vecindades de una solución inicial hasta que se cumpla una condición de parada. De manera adicional, esta metaheurística emplea para explorar las vecindades una VND *(Variable Neighborhood Descent)* donde se desarrollan cuatro generaciones diferentes de vecindades a una solución inicial en la que no se pasa de una a otra mientras que no haya habido una mejora en el valor de la función objetivo. Estas cuatro formas de generar vecindades son: intercambio aleatorio de dos pacientes con la misma prioridad que estén asignados a la misma cama, intercambio aleatorio de dos pacientes adyacentes con la misma prioridad que estén asignados a la misma cama, atrasar una posición todos los pacientes de la misma prioridad dentro de una cama, pasando el primero de esa categoría a la última posición y, por último, cambiar dos pacientes entre las camas asignadas. De esta manera, los autores proponen una solución al caso de disponer de más de 24 pacientes en el SU y querer optimizar el servicio en un tiempo razonable.

Por último, [24] aborda la programación de las consultas de los pacientes en el SU atendiendo a la prioridad asistencial y a los tiempos de los tratamientos que estos requieren. Para ello, presentan un algoritmo que se basa en la metaheurística conocida como GVNS *(General Variable Neighborhood Search)* en la que se hace uso de nuevo de diferentes formas de generar soluciones vecinas a una existente. Esta metaheurística se podría resumir de la siguiente manera: tras generar de manera aleatoria una solución vecina a una inicial dada usando una de las estructuras de vecindad, aplicar la VND. En el caso en el que la solución propuesta no mejore a la solución actual generar otra solución aleatoria empleando una nueva estructura de vecindad y volver a aplicar la VND. En el caso en que mejore, se acepta dicha solución y se vuelve a comenzar hasta que se satisfaga la condición de parada. En este caso, se emplean seis estructuras diferentes de vecindad: 1-swap, 1- insertion, 2-swap, 2-insertion, insertar los pacientes existentes entre dos seleccionados justo delante de un tercer elegido dentro de la secuencia del mismo facultativo e insertar los pacientes existentes entre dos seleccionados justo delante de un tercer elegido dentro de la secuencia de otro facultativo.

Capítulo 3 Descripción del problema

3.1 Introducción

En este apartado se procederá a explicar el funcionamiento del SU a modo de contextualización del problema que se quiere abordar.

Servicio de Urgencias

El SU se puede definir como el servicio de salud que presta atención en tiempos críticos a pacientes que llegan de manera aleatoria donde se requiere una coordinación entre los recursos humanos y materiales para ejecutar los tratamientos o actividades que estos requieren a lo largo de su etapa de urgencias [28, 10, 26]. Con la Ilustración 3-1 propuesta por [28], se explica el flujo tanto de la información como del paciente a través del servicio general de urgencias.

Las llegadas a los SUs se producen por pacientes que vienen por su propio pie, por ambulancia o siendo derivados de otros centros de atención médica que los han derivado a urgencias. Numerosos autores tratan y definen estas tasas de llegadas como una demanda aleatoria o mejor dicho estocástica ya que no se puede determinar de manera exacta el número de pacientes que llegarán en una determinada franja de tiempo, pero sí se pueden determinar las distribuciones estadísticas de esas llegadas donde por ejemplo [30, 31] las modelan según distribuciones de Poisson o Weibull respectivamente.

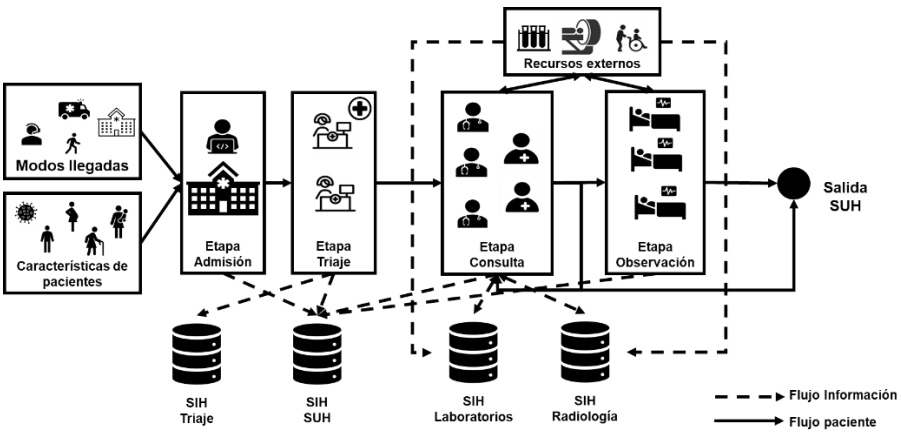


Ilustración 3-1 Flujo de Información en el SU. Fuente: [28]

A continuación, los pacientes pasan a la etapa de admisión donde se le toman los datos y se les da de alta en el SU esperando a ser atendidos por en la siguiente etapa.

Tras esto llegan a la etapa de triaje donde, personal cualificado normalmente del servicio de enfermería, realizan una primera observación y análisis del paciente con el fin de asignarles una prioridad asistencial. Esta clasificación se hace en función de muchos parámetros, pero entre ellos están el motivo de la consulta, estado del paciente o el método de llegada. Existen varios sistemas normalizados para la gestión de la prioridad asistencial y, particularmente, en los servicios de urgencias españoles se emplea el SET (Sistema Español de Triaje) [32]. En él se definen cinco niveles de prioridad en la que el uno se corresponde con una prioridad absoluta y el cinco a un nivel sin urgencia alguna. En este sistema se exponen también los tiempos máximos dependiente de la prioridad que deberían pasar antes de que ese paciente sea reconocido en la primera consulta con un personal médico tal y como se puede ver en la Tabla 3-1.

Nivel de gravedad	Nivel de Urgencia	Tiempo de Atención	Color
Nivel 1	Emergencia	Inmediata	Rojo
Nivel 2	Muy Urgente	15 minutos	Naranja
Nivel 3	Urgente	30 minutos	Amarillo
Nivel 4	Menor Urgente	60 minutos	Verde
Nivel 5	No Urgente	120 minutos	Azul

Tabla 3-1 Niveles de Gravedad según SET. Fuente: Elaboración propia

Posteriormente, el paciente es evaluado en esa primera consulta con un facultativo que solicitará, si así lo requiere, algún tipo de prueba de diagnóstico (como pruebas radiológicas, analíticas, ecografías, etc.) o algún tratamiento específico. Una vez realizadas las pruebas y obtenido los resultados, el paciente visita de nuevo al facultativo que se le ha sido asignado, es decir, al que le evaluó en la primera consulta donde se le podrán volver a pedir nuevas pruebas, consultas con otros especialistas o tratamientos. La duración de las consultas, pruebas y tratamientos dependen de tantos factores (como puede ser la experiencia de los facultativos [4, 33]) que no es posible determinarlos a priori con exactitud por lo que se suelen modelar conforme a distribuciones estadísticas donde, por ejemplo, se establecen que estos tiempos

siguen distribuciones uniformes o triangular [37]. El SU tiene salas de espera habilitadas para que los pacientes esperen a ser llamados y obtener los resultados de las pruebas de diagnóstico y/o a que hagan efecto algún tratamiento. Un paciente saldrá del SU en el momento en el que el facultativo le dé el alta, lo derive a hospitalización o al área de observación.

Como es sabido, durante la pandemia, las urgencias se convirtieron en la primera línea de atención de los pacientes infectados donde tuvieron que adaptarse y redefinirse para hacer frente a la escasez tanto de recursos materiales como humanos [20, 38] con el consecuente estrés laboral. Para dar respuesta a la necesidad asistencial, los servicios de urgencias han tenido que reorganizar los recursos, creando circuitos asistenciales específicos para casos con sospecha de infección o casos confirmados [20, 39]. Ello implica que recursos provenientes de otros departamentos de atención hospitalaria tuvieran que dejar su función original para la atención de pacientes COVID con las consecuentes cancelaciones por ejemplo de intervenciones quirúrgicas [41]. Del mismo modo, para aquellos pacientes que presentasen síntomas leves de la infección, se destinaron recursos humanos destinados a hacerles un seguimiento y recomendaciones durante su aislamiento domiciliario [39].

3.1 Modelado del problema específico

Son muchos los autores que han propuesto una semejanza entre el SU hospitalarias con los diferentes entornos productivos definidos en [29]. La visión que más se repite en la literatura es la expuesta por [47] donde define el servicio pediátrico de urgencias como un entorno *Flexible Job-shop.*

El entorno *Jobshop* se define en [29] como un conjunto de m máquinas por las que deben pasar diferentes trabajos, cada uno, siguiendo una determinada ruta para hacer sus respectivas actividades. El concepto de flexible dota la opción de que estas máquinas son capaces de realizar más de un tipo de actividad. Los problemas de este entorno tienen una complejidad NP-hard. [48]

Esta semejanza se entiende al contemplar a los diferentes recursos del SU como las diferentes máquinas y a los pacientes como los trabajos. Cada paciente llega en un instante determinado al SU y, dependiendo de su patología y prioridad deberá pasar por diferentes

etapas en el SU, donde se le practicarán diferentes pruebas y/o se le aplicarán diferentes tratamientos.

Las principales actividades que tienen lugar en el SU son:

- Primera consulta de evaluación del paciente.
- Extracción de muestras para pruebas diagnósticas.
- Pruebas diagnósticas y aplicación de tratamientos en consulta.
- Analíticas.
- Traslado de pacientes en caso de que así lo requiera.
- Pruebas radiodiagnósticas como TACs, rayos x, ecografías.
- Reevaluación del paciente.
- Ingreso en área de observación.
- Alta por ingreso hospitalario.
- Alta.

Por otro lado, los principales recursos existentes en el SU son:

- Consultas
- Servicio de enfermería.
- Laboratorio.
- Camas de observación.
- Salas de radiodiagnóstico.
- Celadores.

La interacción entre los pacientes y los diferentes recursos no se establece de manera trivial ya que hay un gran número de restricciones y consideraciones generales que se deben tener en cuenta a la hora de modelar el problema específico.

Para el desarrollo del presente trabajo, se partirá del modelo de programación lineal entera propuesto por [28] en el que se contemplan todas las restricciones y particularidades del SU del HUVR.

La función objetivo que se define en [28] se compone de tres indicadores clave a tener en cuenta en el SU. En primer lugar, se penaliza el incumplimiento del Tiempo de Espera de Primera Consulta Facultativa (TEPCOF), definido por el Plan Andaluz de Urgencias y Emergencias como el tiempo máximo de espera que debe pasar entre que un paciente llega y es atendido por primera vez por un facultativo. El TEPCOF depende de la prioridad asistencial

determinada en la etapa de triaje tal y como se ha expuesto en capítulo 2. En segundo lugar, la función objetivo tiene en cuenta el tiempo entre que un paciente llega y sale del SU normalizado por la suma de sus tiempos de procesos. Por último, contempla el equilibrio de las cargas de trabajo entre los facultativos.

3.2 Formulación MILP del problema. Fuente: [28]

En este apartado se procederá a la explicación de la formulación matemática a través de programación lineal entero del problema de secuenciación de pacientes en el SU.

3.2.1 Índices

- Pacientes i=1…I.
- Actividades k=1…K.
- Recursos del SU j=1…J.
- Tipos de recursos w=1…W.
- Turnos de trabajo s=1…S.
- Prioridades asistenciales v=1…V
- Tipología del paciente o=1…O

Sea un paciente que llega en un momento determinado dentro de un turno de trabajo s. A dicho paciente i, tras su etapa de triaje se le asigna una prioridad asistencial $p_i = v$, tipología o y una ruta determinada L_i compuestas por un conjunto de actividades k. Cada actividad tiene asociado una serie de recursos j que son necesarios para su ejecución. Del mismo modo, en cada turno no todos los recursos estarán disponibles.

3.2.2 Parámetros de entrada y variables de decisión

En la Tabla 3-2, se presentan parámetros auxiliares que intervienen en las restricciones modeladas por [28], así como las variables de decisión que permitirán construir la programación y evaluar la función objetivo.

Parámetros	
$EPCOF_v$	estándar de calidad que se debe cumplir para la prioridad asistencial *v*.
TP_v	Número de pacientes de prioridad asistencial *v*.
r_i	instante de tiempo de llegada del paciente *i*.
$type_i$	tipología del paciente *i*.
p_i	prioridad asistencial del paciente *i*.
tp_i	Tiempo total del proceso de urgencia del paciente *i*.
$TEPCOF_i$	tiempo máximo de espera para la 1ª consulta establecido en el PAUE para la prioridad asistencial del paciente *i*.
itr_i	instante de tiempo que marca el inicio para el descanso del recurso humano representado por el paciente ficticio *i*.
ftr_i	instante de tiempo que marca el fin para el descanso del recurso humano representado por el paciente ficticio *i*.
first	primera actividad de un proceso de urgencia.
last	última actividad de un proceso de urgencia.
α_{jw}	1 si el recurso *j* es de tipo *w*; 0 en caso contrario.
a_{js}	1 si el recurso *j* está disponible en el turno *s*, 0 en caso contrario
$\mathbf{e}_{vj}$	1 si el paciente de prioridad *v* se puede asignar al recurso *j*; 0, en caso contrario.
TPR_{oj}	1 si la tipología de paciente *o* se puede asignar al recurso *j*, 0 en caso contrario.
g_k	paciente al que pertenece la tarea *k*.
t_{kj}	consumo de recurso *j* para la actividad *k*.
$wt_{Li[l]}$	tiempo de espera tras la ejecución de la actividad L_i [*l*] para que comience la actividad sucesora L_i [*l+1*].
γ_{wk}	1 si el tipo recurso *w* es necesario para realizar la actividad *k*; 0 en caso contrario.
δ_{kj}	1 si la actividad *k* se puede realizar en el recurso *j*; 0 en caso contrario.
ρ_{il}	1 si la actividad *l* del paciente *i* tiene que empezar inmediatamente después de la tarea *l*-1, 0 en caso contrario.
LS	disponibilidad (en minutos) de un turno
U	cota superior ($U = LS \cdot \|S\| \cdot \|H\|$).
Variables de decisión	
X_{kj}	1 si la actividad *k* es asignada al recurso *j*; 0 en caso contrario.
B_{kjs}	1 si la actividad *k* está asignada al recurso *j* y termina en el turno *s*; 0 en caso contrario.
S_{kjs}	1 si la actividad *k* está asignada al recurso *j* y empieza en el turno *s*; 0 en caso contrario.
$Y_{kk'j}$	1 si la actividad *k* va delante de la actividad *k'* en un recurso *j*; 0 en caso contrario.
C_{kj}	tiempo de finalización de la actividad *k* en el recurso *j* (en minutos).
Z_i	1 si se cumple con el paciente el estándar del tiempo máximo de espera en 1ª consulta; 0 en caso contrario.
A_{vjs}	número de pacientes de prioridad *v* asignados al recurso *j* de tipo facultativoen el turno *s*.
$Q_{jj's}$	diferencia en términos absolutos de volumen entre dos recursos, *j* y *j'*, de tipo facultativo en el turno *s*.
E_v	Número de pacientes que no cumplen el *TEPCOF*.

Tabla 3-2 Parámetros y variables del modelo. Fuente: [28]

3.2.3 Función Objetivo y Restricciones

Función Objetivo

$$Minimizar \sum_{v\in V|p_v!=dummy} \sum_{s\in S} \sum_{j\in J|\alpha_{jconsulta}=1} \sum_{j'\in J|\alpha_{j'consulta}=1} Q_{vjj's} + \sum_{i\in I|p_i!=dummy} \frac{1}{tp_i} \cdot \left(\sum_{j\in J} C_{Li[last]j} - r_i\right) + \sum_{v\in V|p_v!=dummy} E_v \quad (1)$$

La función objetivo de este problema es realmente una función multiobjetivo en la que se desean minimizar tres funciones objetivo:

- Diferencia de carga entre los facultativos que se encuentran en consulta.
- Tiempo de estancia de los pacientes.
- Pacientes que incumplan el estándar de calidad TEPCOF.

Restricciones

A continuación, se presentarán y explicarán las restricciones del problema.

$$\sum_{j\in J|\alpha_{jw},\delta_{kj},e_{p[g_k]j},TPR_{type[g_k]j}=1} X_{kj} = 1 \quad \forall k \in K, \forall w \in W \mid \gamma_{wk} = 1 \quad (2.1)$$

$$X_{Li[l]j} = X_{Li[first]j}$$
$$\forall i \in I, \forall l \in L_i, \forall j \in J|\alpha_{jconsulta}, \gamma_{consultaLi[l]}, e_{p_ij}, TPR_{type_ij}, \delta_{Li[l]j} = 1 \quad (2.2)$$

$$C_{kj} + U \cdot \left(2 - X_{kj} - X_{kj'}\right) \geq C_{kj'}$$
$$\forall k \in K, \forall w \in W, \forall w' \in W, \forall j \in J, \forall j' \in J \mid w < w', j < j', \gamma_{wk}, \gamma_{w'k} = 1$$
$$\alpha_{jw}, \alpha_{j'w'}, \delta_{kj}, \delta_{kj'}, TPR_{type[g_k]j}, TPR_{type[g_k]j'}, e_{p[g_k]j}, e_{p[g_k]j'} = 1 \quad (3.1)$$

$$C_{kj'} + U \cdot \left(2 - X_{kj} - X_{kj'}\right) \geq C_{kj}$$
$$\forall k \in K, \forall w \in W, \forall w' \in W, \forall j \in J, \forall j' \in J \mid w < w', j < j', \gamma_{wk}, \gamma_{w'k} = 1$$
$$\alpha_{jw}, \alpha_{j'w'}, \delta_{kj}, \delta_{kj'}, TPR_{type[g_k]j}, TPR_{type[g_k]j'}, e_{p[g_k]j}, e_{p[g_k]j'} = 1 \quad (3.2)$$

Con los conjuntos de restricciones (2.1-3.2) se modela la asignación de las actividades a los recursos teniendo en cuenta la factibilidad de disponibilidad y tipología. Como se comentó en el

apartado 3.2.1, no todos los recursos pueden ejecutar todas las actividades y esa restricción se modela con el conjunto (2.1). Posteriormente, el conjunto (2.2) modela la restricción de que el recurso en el que un paciente haya realizado su primera consulta, deberá ser siempre el mismo para todas las actividades que requieran un recurso consulta (altas o reevaluaciones, por ejemplo). Con los conjuntos 3.1 y 3.2 el modelo garantiza la disponibilidad de al menos un recurso necesario para ejecutar una determinada actividad de un paciente.

$$C_{k'j} \geqslant C_{kj} + t_{k'j} \cdot X_{kj} - U \cdot (1 - Y_{kk'j})$$
$$\forall k \in K, \forall k' \in K, \forall w \in W, \forall j \in J \mid w! = Lab, k < k', \alpha_{jw}, \gamma_{wk}, \gamma_{wk'}, = 1$$
$$\delta_{kj}, \delta_{k'j}, TPR_{type[g_k]j}, TPR_{type\left[g_{k'}\right]j}, e_{p[g_k]j}, e_{p\left[g_{k'}\right]j} = 1 \quad (4.1)$$

$$C_{kj} \geqslant C_{k'j} + t_{kj} \cdot X_{kj} - U \cdot Y_{kk'j}$$
$$\forall k \in K, \forall k' \in K, \forall w \in W, \forall j \in J \mid w! = Lab, k < k', \alpha_{jw}, \gamma_{wk}, \gamma_{wk'} = 1$$
$$\delta_{kj}, \delta_{k'j} TPR_{type[g_k]j}, TPR_{type\left[g_{k'}\right]j}, e_{p[g_k]j}, e_{p[g_{k'}]j} = 1 \quad (4.2)$$

$$C_{Li[l]j} \geqslant C_{Li[l-1]j'} + t_{Li[l]j} + wt_{Li[l-1]} + U\ (X_{Li[l]j} + X_{Li[l-1]j'} - 2)$$
$$\forall i \in I, \forall j \in J, \forall j' \in J, \forall w \in W, \forall w' \in W, \forall l \in L_i \mid l \geq 2, \gamma_{wLi[l]}, \gamma_{w'Li[l-1]} = 1$$
$$\alpha_{jw}, \delta_{Li[l]j}, \delta_{Li[l-1]j'}, e_{p_i j}, TPR_{type_i j}, e_{p_i j'}, TPR_{type_i j'} = 1 \quad (5.1)$$

$$C_{Li[l]j} \leq C_{Li[l-1]j'} + t_{Li[l]j} + wt_{Li[l-1]} + U\ (2 - X_{Li[l]j} - X_{Li[l-1]j'})$$
$$\forall i \in I, \forall j \in J, \forall j' \in J, \forall w \in W, \forall w' \in W, \forall l \in L_i \mid l \geq 2, \rho_{iLi[l]}, \alpha_{jw} = 1$$
$$\gamma_{wLi[l]}, \gamma_{w'Li[l-1]}, \delta_{Li[l]j}, \delta_{Li[l-1]j'}, e_{p_i j}, TPR_{type_i j}, e_{p_i j'}, TPR_{type_i j'} = 1 \quad (5.2)$$

$$\sum_{\forall j \in J \mid \alpha_{jw}, \delta_{Li[first]j}, e_{p_i j}, TPR_{type_i j} = 1} \left(C_{Li[first]j} - (t_{Li[first]j} + r_i) \cdot X_{Li[first]j}\right) \geqslant 0$$

$$\forall i \in I, \forall w \in W, \mid \gamma_{wLi[first]} = 1 \quad (5.3.1)$$

$$\sum_{\forall j \in J \mid \alpha_{jw}, \delta_{Li[first]j}, e_{p_i j}, TPR_{type_i j} = 1} \left(C_{Li[first]j} - U \cdot (1 - X_{Li[first]j})\right) \leq ftr_i$$

$$\forall i \in I, \forall w \in W, \mid \gamma_{wLi[first]} = 1, p_i = dummy \quad (5.3.2)$$

$$C_{kj} \leq U \cdot X_{kj}$$

$$\forall k \in K, \forall j \in J | p_{g_k}! = dummy, \delta_{kj}, TPR_{type[g_k]j}, e_{p\left[g_{k'}\right]j} = 1 \quad (5.3.3)$$

Dado que en todos los recursos salvo en el laboratorio, las actividades han de preceder unas a otras, con los conjuntos 4.1 y 4.2 se garantiza la secuenciación de las actividades en el mismo recurso, impidiendo la ejecución de dos actividades en el mismo recurso al mismo tiempo. Centrándose en la ruta de un paciente, es necesario garantizar que se ejecuta una actividad tras otra sin solapamientos. Por otro lado, tras algunas actividades como puede ser la aplicación de un tratamiento, no se puede comenzar la siguiente actividad hasta que no haya pasado un determinado tiempo prefijado. Estas dos restricciones se modelan con el conjunto 5.1. En contraposición se encuentra el conjunto 5.2 que garantiza la restricción no-wait que puedan existir entre dos actividades. La finalidad de los conjuntos (5.3.1-5.3.3) no es otro que el modelado para calcular los *completion time* o tiempo de finalización de cada actividad en el recurso correspondiente.

$$\sum_{k \in K | \delta_{jk}, TPR_{type[g_k]j}, e_{p[g_k]j}=1} B_{kjs} \leq |K| \cdot a_{js}$$
$$\forall j \in J, \forall s \in S \qquad (6.1.1)$$

$$\sum_{k \in K | \delta_{jk}, TPR_{type[g_k]j}, e_{p[g_k]j}=1} S_{kjs} \leq |K| \cdot a_{js}$$
$$\forall j \in J, \forall s \in S \qquad (6.1.2)$$

$$\sum_{s \in S} S_{kjs} = X_{kj}$$
$$\forall k \in K, \forall j \in J | \delta_{jk}, TPR_{type[g_k]j}, e_{p[g_k]j} = 1 \qquad (6.2.1)$$

$$\sum_{s \in S} B_{kjs} = X_{kj}$$
$$\forall k \in K, \forall j \in J | \delta_{jk}, TPR_{type[g_k]j}, e_{p[g_k]j} = 1 \qquad (6.2.2)$$

$$C_{kj} \geq \left(t_{kj} + LS \cdot (s-1)\right) \cdot S_{kjs}$$
$$\forall s \in S, \forall k \in K, \forall j \in J | \delta_{jk}, TPR_{type[g_k]j}, e_{p[g_k]j} = 1 \qquad (6.3.1)$$

$$C_{kj} \leq LS \cdot s \cdot B_{kjs} + U \cdot \left(1 - B_{kjs}\right)$$
$$\forall s \in S, \forall k \in K, \forall j \in J | \delta_{jk}, TPR_{type[g_k]j}, e_{p[g_k]j} = 1 \qquad (6.3.2)$$

Dado que no todos los recursos están disponibles durante todo el día, los conjuntos de restricciones (6.1) modelan la disponibilidad de cada recurso respecto al turno en que están disponible, no permitiendo que ejecuten ninguna actividad si se encuentra fuera de su turno disponible. Por otro

lado, con los conjuntos (6.2) se modelan las variables que indican el en qué turno comienza y finaliza una actividad en un recurso determinado. Por último, con los conjuntos (6.3) se limitan las duraciones de las actividades, impidiendo que excedan al turno en el que se encuentran planificadas.

$$\sum_{j \in J | \alpha_{jconsulta}=1} \left(C_{Li[first]j} - \left(r_i + t_{Li[first]j}\right) \cdot X_{Li[first]j}\right) \leq TEPCOF_i \cdot Z_i + U \cdot (1 - Z_i)$$

$$\forall i \in I \qquad (7.1)$$

$$\sum_{i \in I | p_i = v} Z_i + E_v \geq EPCOF_v \cdot TP_v$$

$$v \in V | p_v! = dummy, TP_v > 0 \qquad (7.2)$$

$$A_{vjs} = \sum_{i \in I | p_i = v} \sum_{k \in K} B_{kjs} \cdot \delta_{jk}$$

$$\forall s \in S, \forall v \in V, \forall j \in J \mid \alpha_{jconsulta} = 1 \qquad (8.1)$$

$$Q_{vjj's} \geq A_{vjs} - A_{vj's}$$
$$\forall s \in S, \forall v \in V, \forall j \in J, \forall j' \in J \mid j < j'; \alpha_{jconsulta}, \alpha_{j'consulta}, a_{js}, a_{j's} = 1 (8.2.1)$$

$$Q_{vjj's} \geq A_{vj's} - A_{vjs}$$
$$\forall s \in S, \forall v \in V, \forall j \in J, \forall j' \in J \mid j < j'; \alpha_{jconsulta}, \alpha_{j'consulta}, a_{js}, a_{j's} = 1 (8.2.2)$$

Los conjuntos de restricciones 7 y 8 se emplean para en primer lugar calcular los pacientes que no han cumplido el TEPCOF según su prioridad asistencial. Específicamente esto lo hacen los conjuntos 7. Por otro lado, los conjuntos de restricciones 8 calculan el número de consultas que asume cada facultativo dependiente de la prioridad asistencial para posteriormente poder linealizar con los conjuntos 8.2 el valor absoluto de la función objetivo.

Por último, se presentan las restricciones pertenecientes a la definición de las variables del modelo.

$$X_{kj}\{0,1\} \quad \forall k \in K, \forall j \in J \mid \delta_{jk} = 1 \qquad (9)$$

$$C_{kj} \geq 0 \quad \forall k \in K, \forall j \in J | \delta_{jk} = 1 \tag{10}$$

$$Y_{kk'j}\{0,1\} \quad \forall k \in K, \forall k' \in K, \forall j \in J \mid k < k', \delta_{jk}, \delta_{jk'} = 1 \tag{11}$$

$$Z_i\{0,1\} \quad \forall i \in I \tag{12}$$

$$E_v \geq \forall v \in V \tag{13}$$

$$A_{vjs} \geq 0 \quad \forall v \in V, \forall j \in J, \forall s \in S \mid \alpha_{jconsulta} = 1 \tag{14}$$

$$Q_{vjj's} \geq 0$$
$$\forall v \in V, \forall j \in J, \forall j' \in J, \forall s \in S \mid j < j';\ \alpha_{jconsulta}, \alpha_{j'consulta} = 1 \tag{15}$$

$$B_{kjs}\{0,1\} \quad \forall k \in K, \forall j \in J, \forall s \in S \mid \delta_{jk} = 1 \tag{16}$$

$$S_{kjs}\{0,1\} \quad \forall k \in K, \forall j \in J, \forall s \in S \mid \delta_{jk} = 1 \tag{17}$$

Capítulo 4 Metodología de Resolución

4.1 Introducción

Siendo el SU el motivo de estudio de este proyecto, cabe mencionar que a este servicio acceden diariamente unas 500 personas, llegando a más de 650 en momento de alta presión [25], se desea desarrollar un algoritmo de optimización que arroje una solución a la programación de pacientes en un periodo de tiempo razonable dentro de la operativa del SU. Para ello se partirán de dos metaheurísticas para proponer dos variantes que posteriormente se calibrarán y compararán con las ya existentes.

En primer lugar, se partirá del algoritmo genético propuesto por [46] para la secuenciación de las pruebas laboratorios de urgencia de pacientes. Se ha tomado como referencia este algoritmo por ser uno de los problemas de secuenciación de pacientes abordado desde un enfoque de optimización a través de metaheurísticas. De manera adicional, hay que destacar que el algoritmo genético es el método metaheurístico más popular para resolver los problemas del entorno *flexible job shop* [50].

En segundo lugar, se partirá del algoritmo voraz iterativo conocido como *Iterated Greedy* propuesto por [49]. Se ha tomado este algoritmo como referencia por ser uno de las metaheurísticas más empleadas en los problemas más simples de entorno *job-shop* [48].

4.2 Estructura y construcción de la Solución

La solución a la secuenciación de los pacientes en un SU consiste en la definición de los tiempos de comienzo y finalización de cada una de las actividades en los recursos en los que se ejecuten. Con estos tiempos, se pueden sencillamente calcular el valor de la función objetivo verificando el cumplimiento o no del TEPCOF de cada paciente, calculando y normalizando el tiempo de permanencia en el SU y calculando el equilibrio de los facultativos.

Para construir esta solución y obtener los tiempos de inicio y fin de cada actividad de cada paciente, la solución con la que se trabajará en los algoritmos será una secuencia de pacientes con la que, haciendo uso de una función constructiva que garantice la factibilidad

de las restricciones. Esta función constructiva irá fijando paciente a paciente de la secuencia, las actividades atendiendo a su momento de llegada, a los tiempos de sus actividades y a la ruta que tienen prefijada por el código de diagnóstico propuesto en la etapa de triaje. Nótese que tal y como se propone en [28], habrá pacientes ficticios que simulen el descanso del personal.

A continuación, se presenta un breve ejemplo de la función constructiva en la que se simplifican los cálculos asumiendo que la duración de cada actividad es de 10 minutos.

Paciente	Llegada	Prioridad	Ruta
1	0	4	CON-EXTR-LAB-REV-ALT
2	15	3	CON-EXTR-LAB-REV-OBS-ALT
3	20	3	CON-RAY-REV-OBS-ALT
4	30	Ficticio	CON

Tabla 4-1 Datos Ejemplo Función Constructiva. Fuente: Elaboración Propia

En la Tabla 4-1 se presentan los datos de entrada de optimización, se suponen la llegada de 4 pacientes durante la jornada de trabajo. Por ejemplo, el paciente 2, llega en el instante de tiempo 15, con una prioridad asistencial 3 y con la siguiente ruta:

- Consulta.
- Extracción.
- Laboratorio.
- Reevaluación.
- Ingreso en Observación.
- Alta.

Nótese la presencia de un paciente ficticio a partir del instante de tiempo 30 que tiene una única actividad que simula el descanso el facultativo de la consulta.

Para una solución dada por el siguiente orden de pacientes:

Ilustración 4-1 Secuencia Solución Ejemplo Función Constructiva. Fuente: Elaboración Propia

La secuenciación de las actividades en los diferentes recursos del servicio de urgencias resultaría de la siguiente manera:

CONSULTA	1	1	3	4	3	1	2	2		2	
ENFERMERIA		1						2			
LABORATORIO			1						2		
OBSERVACIÓN						3					2
RAYOS				3							
	10	20	30	40	50	60	70	80	90	100	110

Ilustración 4-2 Secuenciación Actividades Función Constructiva. Fuente: Elaboración Propia

Nótese que las actividades se asignan a los recursos tan pronto como estén disponibles atendiendo a restricciones como la *no-wait* que existe entre la actividad de consulta y extracción y la necesidad de los recursos compartidos que necesita la extracción al realizarse dentro de la consulta por una persona del servicio de enfermería.

Con la construcción de la solución, lo que se consigue es obtener los *starting time* y *completion time* de las diferentes actividades de los pacientes, garantizándose la factibilidad cumpliendo las restricciones expuestas en el modelado del problema de optimización.

De este modo, se podría obtener un diagrama de Gantt en el que se vea la evolución de cada paciente a lo largo del SU.

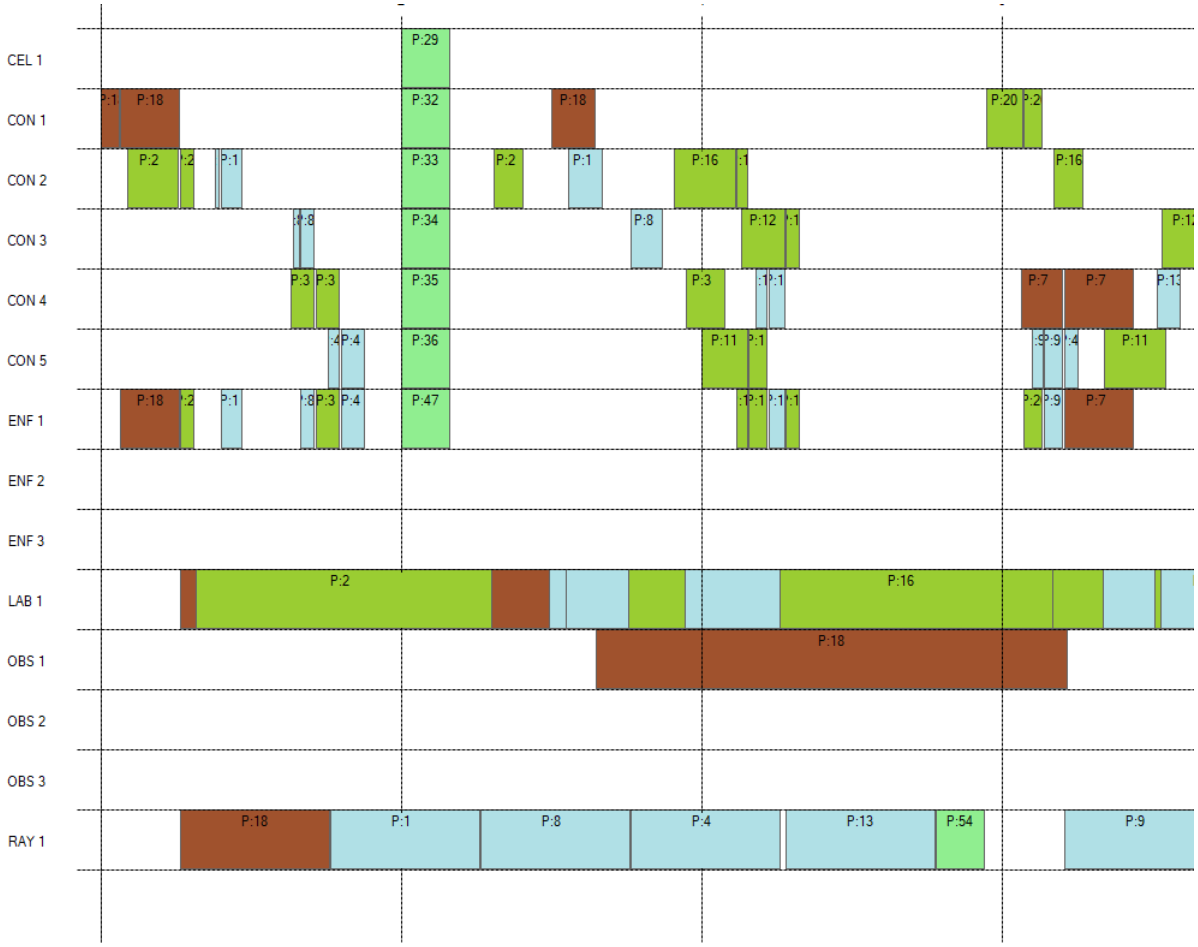


Ilustración 4-3 Diagrama de Gantt Construcción Solución. Fuente: Elaboración Propia

De la Ilustración 4-3 se puede apreciar la restricción de no solapamiento de actividades en recursos, la restricción *no-wait* entre actividades como primera consulta y extracción y la posibilidad de ejecutar varias pruebas en el laboratorio al mismo tiempo. Se puede destacar también como aparecen pacientes con una única actividad que hace referencia a los pacientes ficticios que se generan para reservar ese un hueco de descanso del personal. Ese descanso se da únicamente en los recursos donde hay presencia de recursos humanos siempre y cuando ese recurso esté disponible en el periodo de optimización.

A continuación, en la Tabla 4-2 se presentan los valores que tomarían variables asociadas a los dos primeros pacientes generados en esta instancia:

Actividad	Paciente	Prioridad Paciente	Instante de llegada	TEPCOF	Duración	Starting Time	Completion time	TEPCOF
1	1	4	46	90	2	46	48	1
2	1	4	46	90	9	48	57	1
3	1	4	46	90	60	92	152	1
4	1	4	46	90	130	57	187	1
5	1	4	46	90	14	187	201	1
6	1	4	46	90	0	201	201	1
7	2	2	0	10	21	11	32	0
8	2	2	0	10	6	32	38	0
9	2	2	0	10	119	38	157	0
10	2	2	0	10	12	157	169	0
11	2	2	0	10	0	169	169	0

Tabla 4-2 Valor de las variables Construcción Solución. Fuente: Elaboración propia

En la Tabla 4-2 se aprecia que el paciente 1 llega en el instante de tiempo 46 y tiene un total de 6 actividades en su ruta o proceso asistencial. Debido a que tiene una prioridad asistencial de valor 4 (menor urgente según SET), el TEPCOF que establece el hospital es de 90 minutos. Dado que su primera actividad (primera consulta) se cumple nada más llega, ese paciente cumple con el TEPCOF.

Por contraposición, el paciente 2, al presentar una prioridad asistencial más alta que el paciente 1, presenta un TEPCOF más ajustado, provocando que se incumpla el estándar de calidad por un minuto. Esta paciente presenta otra ruta o proceso asistencial que se ajustará al motivo de consulta por el que llega al SU.

4.3 Algoritmos de Optimización

En este apartado se presentarán dos algoritmos existentes en la literatura para que, partiendo de la estructura que tienen, se propongan variaciones al fin de diseñar un algoritmo genético y un algoritmo voraz que resuelva el problema de estudio de la manera más eficiente posible. Para la selección de los que serán los algoritmos propuestos, se hará un análisis computacional para su posterior comparación a través del ARPD *(average relative percentaje deviation).*

4.3.1 Iterated Greedy Original

El algoritmo voraz iterativo o *Iterated Greedy* (IG) es una metaheurística extendida por la literatura para abordar diferentes problemas de optimización en los diferentes entornos de producción [52]. Se tomará como referencia el propuesto [49], desarrollado para un entorno *flow shop*. Este entorno productivo consiste en un conjunto de máquinas por las que han de pasar una serie de trabajos en el que todos los trabajos siguen el mismo orden a través de las mismas.

Este algoritmo consiste en realizar de manera iterativa una intensificación y diversificación sobre una solución inicial.

El algoritmo propuesto por [49] parte de una solución inicial que generan a través de la heurística conocida como NEH. Este procedimiento consiste en, tras haber realizado una ordenación de los trabajos en orden descendiente del tiempo total de proceso (o suma de tiempos de procesos), se insertan los trabajos de uno en uno evaluando en qué posición la solución parcial tiene mejor función objetivo hasta completar la secuencia con todos los trabajos.

Intensificación

La intensificación consiste en la aplicación de una búsqueda local a la solución inicial. La búsqueda local es una metaheurística por la que, realizando pequeñas variaciones sobre la solución inicial dada, se exploran diferentes soluciones que se consideran próximas a la solución inicial dentro del espacio de soluciones. Dentro del problema de secuenciación hay numerosas vías de generar nuevas vecindades, algunas de las más utilizadas son:

- *1-Insertion*: Donde se extrae un componente de la secuencia de soluciones y se coloca en todas las posiciones posibles.
- *1-Swap*: Se toma como pivote un componente de la secuencia y se va intercambiando con el resto de componentes de la secuencia.
- *K-swap* o *k-intertion*: Siguen la estructura swap e insertion pero en lugar de extraer o pivotar con un único componente, se hace con k componentes adyacentes.

En el caso del algoritmo propuesto en [49], la intensificación elegida se basa en una búsqueda local exhaustiva en la que se emplea la vecindad *1-Insertion* y donde de cada vecindad elige

a la mejor solución. Esta exploración se realiza de manera iterativa mientras que en alguna de las exploraciones se haya encontrado una mejor solución.

A continuación, se resume el pseudocódigo de la búsqueda local planteada por [46]

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

Ilustración 4-4 Pseudocódigo Búsqueda local del IG Original. Fuente [46]

Diversificación

Por otro lado, la etapa de diversificación se compone de una fase de destrucción parcial de la solución, extrayendo de manera aleatoria una serie de elementos de la secuencia y de una etapa de construcción en la que dichos elementos extraídos se insertan y evalúan en cada una de las posiciones de la subsecuencia de los elementos que no han sido extraídos.

Tras esta diversificación se aplica de nuevo la etapa de intensificación.

Uno de los problemas que presenta la búsqueda local es el posible estancamiento alrededor de un óptimo local para ello, los autores emplean un criterio de aceptación que se basa en el recocido simulado. Esta técnica se basa en el proceso físico del recocido simulado de los metales donde las propiedades físicas de estos cambian conforme se van enfriando progresivamente [8]. De esta manera, en el caso en el que una solución no mejore a la mejor solución encontrada hasta el momento, esta tiene cierta probabilidad (dependiendo de la diferencia entre el valor de la solución actual y la mejor solución y de la temperatura) de que esa solución no sea rechazada y sea la solución de partida para la siguiente iteración.

En la Figura se expone el pseudocódigo del algoritmo presentado por [49].

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

Ilustración 4-5 Pseudocódigo IG Original. Fuente: [46]

4.3.2 Iterated Greedy Propuesto.

Partiendo del algoritmo presentado en el apartado 4.3.1, se proponen dos variaciones.

Nueva intensificación.

La nueva búsqueda local propuesta en el presente documento se puede designar como una búsqueda local 1-insertion acotada no exhaustiva. En este sentido la vecindad se genera de la siguiente manera:

1º Seleccionar una posición de la secuencia que hará como pivote.

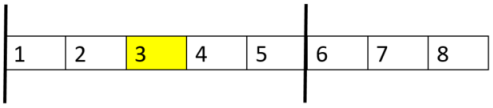
2º Calcular los límites de las posiciones con las que se intercambiará.

3º Realizar el intercambio del pivote con todas las posiciones dentro del rango establecido y evaluar las soluciones.

4º Elegir la mejor de las soluciones vecinas.

Este proceso se repite para todas las posiciones de la secuencia sin volverse a inicializar en el caso en el que encuentre una mejora de la función objetivo, como ocurre con el algoritmo desarrollado en el apartado 4.3.1

A continuación, se expone en la Ilustración 4-6, un ejemplo de la generación de una vecindad en el caso en el que cada posición como máximo pueda intercambiarse con pacientes que disten dos posiciones.



3	2	1	4	5	6	7	8
1	3	2	4	5	6	7	8
1	2	4	3	5	6	7	8
1	2	5	4	3	6	7	8

Ilustración 4-6 Ejemplo vecindad de la nueva búsqueda local propuesta. Fuente: Elaboración Propia

Nueva diversificación

La evolución que sufre la fase de diversificación erradica en la adición del parámetro reductor de la temperatura *alpha*. Al tener un parámetro que reduce la temperatura iteración tras iteración se consigue que la probabilidad de admitir como solución de partida una que tenga un peor valor de la función objetivo se vaya reduciendo con el paso de las iteraciones.

4.3.3 Otras variantes propuestas del algoritmo voraz

Numerosos son los artículos en los que diversos autores emplean evoluciones y variaciones sobre alguna de las etapas del algoritmo voraz iterativo. Las diferentes variaciones propuestas se han basado en cambios en la etapa de intensificación, es decir, otras propuestas de búsqueda local con la finalidad de reducir el espacio a explorar en cada vecindad y lograr así un mayor número de iteraciones en el algoritmo y otro en una modificación sobre la fase de diversificación. Para las diferentes variantes que se comentan a lo largo de este apartado se considerará tonto la opción como tomar como solución de partida la que resulta de aplicar la heurística NEH como tomar una solución aleatoria como solución inicial. El motivo de esta

consideración no es más que lograr disponer de más tiempo de computación para que el algoritmo haga un mayor número de iteraciones.

Algoritmo Voraz V1

En esta variante, en un intento de reducir como se ha comentado el tamaño de la vecindad a explorar en la búsqueda local, se propone una búsqueda local parcialmente exhaustiva. Esta búsqueda local aplica el mismo funcionamiento que tiene la propuesta por [49] con la salvedad de que la inserción en todas las posiciones no se les aplica a todos los trabajos, sino que solamente se seleccionan aleatoriamente un 25% que se probarán en todas las posiciones.

Del mismo modo, la búsqueda local no se reiniciará mientras se encuentre una mejoría en la función objetivo.

Algoritmo Voraz V2

Esta variante tiene un objetivo totalmente opuesto al algoritmo voraz V1. En esta variante se modifica sobre el funcionamiento del propuesto por [49] el procedimiento de inserción o de construcción tras la destrucción parcial. El nuevo procedimiento de inserción propuesto toma los elementos que han sido extraídos de la secuencia inicial y los inserta de manera aleatoria dentro de la subsecuencia restante, sin hacer comprobaciones de en qué posición se obtiene el mejor valor de la función objetivo.

De esta manera se pretende centrar el mayor tiempo computacional posible en la búsqueda local por inserción exhaustiva para comprobar la influencia que tiene esta dentro del algoritmo voraz original.

4.3.4 Algoritmo Genético Original.

El algoritmo genético (AG) pertenece al grupo de algoritmos evolutivos basados en población donde se impone la idea de la genética poblacional, presentada por John Holland en la década de 1970 [51]. Este algoritmo parte de la idea de que, tras tener una serie de individuos dentro de una población, solo sobrevivirán aquellos que sean más aptos.

En el AG, la población la constituye un conjunto determinado de posibles soluciones a la que se le llaman cromosomas y cada cromosoma llevará asociada una evaluación de su aptitud que representa el valor de la función objetivo de dicha solución. Sobre dicha población se le aplican una serie de acciones con el fin de recombinar los cromosomas y conseguir una evolución de la población. La aptitud, también conocida como *fitness*, toma el valor de la función objetivo en aquellos casos en el que el problema sea de maximización y la inversa de la función objetivo en aquellos casos en los que se desee minimizar. El caso que se presenta en este documento es de minimización tal y como se refleja en el modelo matemático [28].

Los operadores fundamentales de un algoritmo genético son: codificación de los cromosomas, cruce, mutación y selección.

A continuación, se presenta el diagrama de flujo del algoritmo genético propuesto por [46]

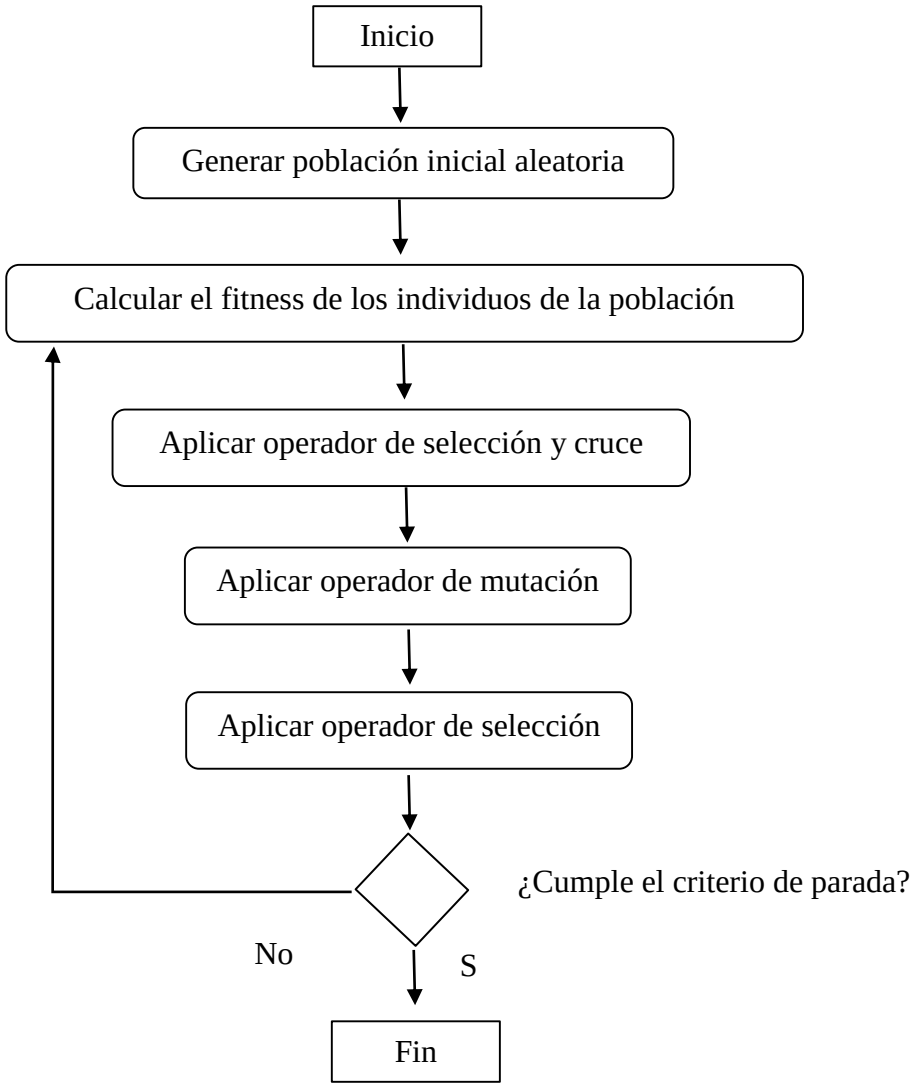


Ilustración 4-7 Diagrama de flujo AG Original. Fuente [46]

Codificación de los cromosomas

Tal y como se ha comentado en el apartado 4.2, el cromosoma estará formado por una secuencia que muestra un orden de pacientes. Su decodificación de cara a la evaluación del fitness de la solución vendrá dada tras la construcción de la solución expuesta en el mismo apartado 4.2.

Cruce

La finalidad de este operador es generar nuevos cromosomas a partir de la combinación de cromosomas ya existentes en la población. Para ello en el AG propuesto por [46], se define el cruce como un cruce de un solo punto con su posterior reparación para evitar que se queden pacientes sin programar o pacientes que se programan dos veces. El cruce a través de un solo punto parte de la selección aleatoria de una posición dentro de la longitud del cromosoma y el descendiente tomará los genes del primer padre hasta dicha posición y los genes del segundo padre desde esa posición hasta el final de la secuencia. Como esta metodología de cruce puede dar lugar a infactibilidades, es necesario hacer posteriormente una revisión de genes repetidos o ausentes.

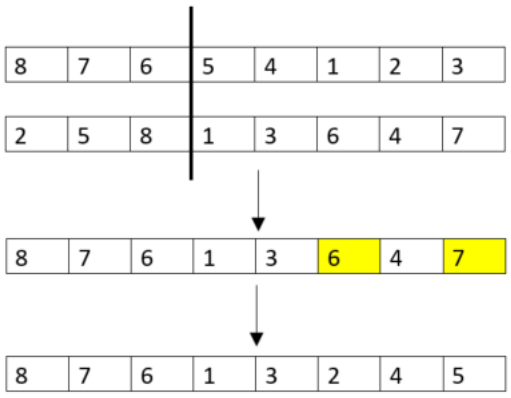


Ilustración 4-8 Ejemplo operador de cruce AG original. Fuente: Elaboración Propia

Mutación

Dado que uno de los principales problemas que suele aparecer en los algoritmos genéticos es el estancamiento en un óptimo local, el operador de mutación garantiza en cierta manera que se preserva la diversidad en la población. En este caso, la mutación definida por [46] es conocida como *swap mutation* en la que, para un cromosoma dado, se seleccionan dos posiciones de la secuencia y se intercambian los genes que se encuentran en dichas posiciones.

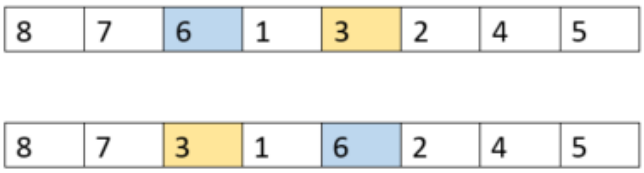


Ilustración 4-9 Operador de Mutación AG Original. Fuente: Elaboración Propia

Selección

El mecanismo de selección que se emplea es el conocido como la ruleta. De esta forma, se asigna una prioridad a cada uno de los cromosomas en función a su fitness y posteriormente de manera aleatoria se selecciona un individuo. Aquellos que tengan mayor fitness, tendrán, por ende, mayor probabilidad de ser elegido. Este mecanismo estocástico se aplica para seleccionar a los individuos que ejercerán de padres para la generación de nuevas soluciones y para que, tras haber generado todos los nuevos descendientes, elegir entre las dos poblaciones, qué cromosomas sobreviven y cuales se descartan.

4.3.5 Algoritmo Genético Propuesto

Como se hace evidente en la literatura, el algoritmo genético está formado por una serie de operadores que provocan una recombinación iterativa entre las soluciones iniciales buscando una evolución en el valor de la función objetivo. En la literatura se pueden encontrar diversos operadores tanto para la mutación como para el cruce y selección. Para seleccionar qué algoritmo genético se propondrá en el presente documento, se analizarán y ejecutarán variaciones del algoritmo genético.

Sobre el algoritmo presentado en el apartado 4.3.4, se propone la modificación del operador de cruce y se añaden dos nuevos operadores: elitismo y control de diversidad.

En la Ilustración 4-10, se puede apreciar el diagrama de flujo del nuevo algoritmo propuesto.

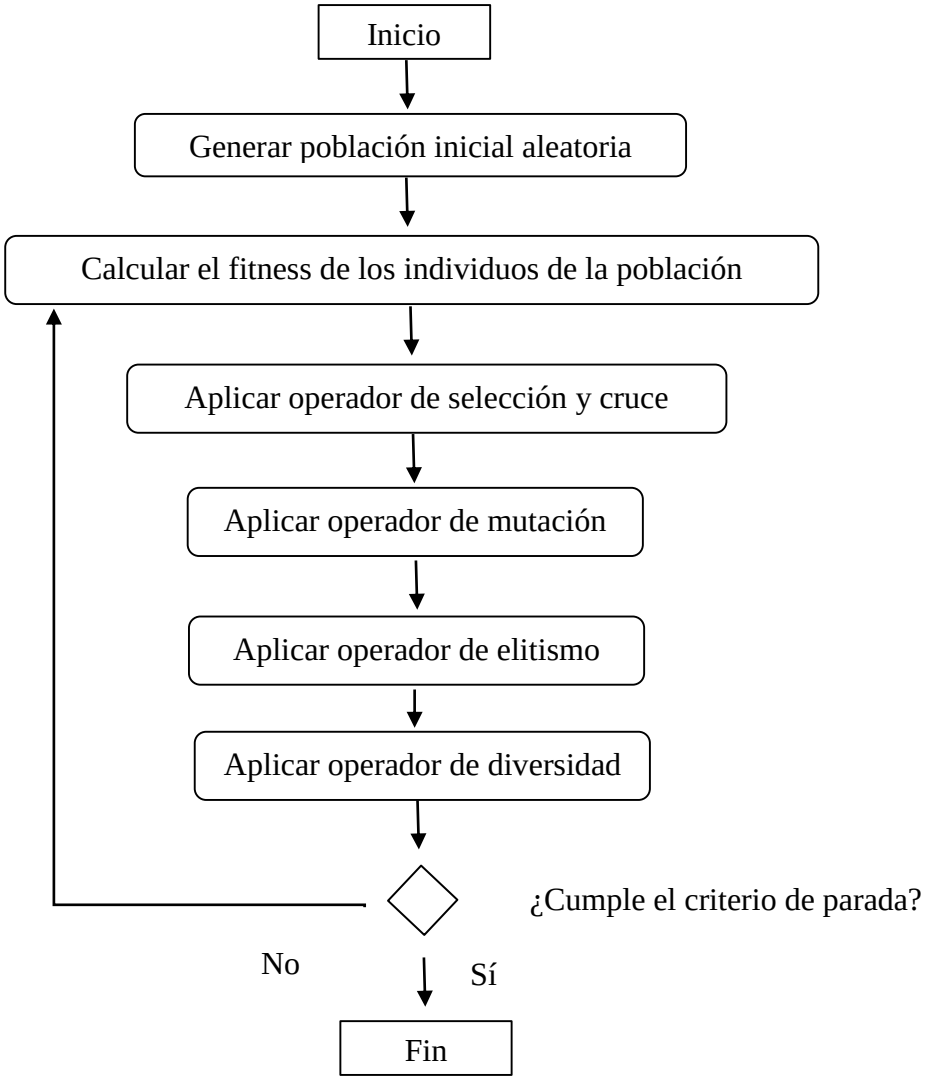


Ilustración 4-10 Diagrama de flujo AG propuesto. Fuente: Elaboración Propia

Cruce

El nuevo operador de cruce propuesto conoce como *random crossover* o cruce aleatorio. Dados dos progenitores seleccionados a través de la ruleta, el descendiente tomará de manera aleatoria algunos genes pertenecientes al primer progenitor y otros del segundo

progenitor. Al igual que en el operador de cruce propuesto originalmente por [46] se hace necesario la presencia de una revisión de la factibilidad de la solución.

El procedimiento se podría ejemplificar de la siguiente manera:

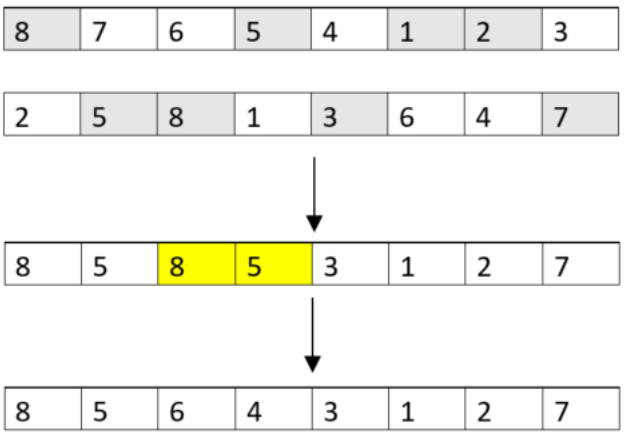


Ilustración 4-11 Operador de Cruce AG propuesto. Fuente: Elaboración Propia

Elitismo

Este operador sustituye al método de selección de cromosomas para sobrevivir y formar parte de la nueva población. En su lugar, se propone que, tras haber generado el número deseado de descendientes, solamente sobrevivan aquellos cromosomas de las dos poblaciones que tengan mayor fitness. El objetivo es eliminar la probabilidad de descartar a posibles candidatos con fitness altos.

Diversidad

Teniendo en cuenta que el eliminar un factor estocástico del algoritmo provocando que solamente sobrevivan los mejores individuos, la propuesta es que, tras generar la población con la "élite" de las soluciones, compararlas entre sí para que, en el momento en el que dos individuos compartan más de un porcentaje determinado de genes iguales, descartar el que menor fitness tenga e introducir un nuevo cromosoma aleatorio en su lugar.

4.3.6 Otras variantes propuestas del algoritmo genético

Algoritmo Genético V1

La primera propuesta de cambio o posible mejora del algoritmo propuesto por [46] es añadir una búsqueda local no intensiva, es decir, que no sea iterativa mientras encuentre una mejora. Esta búsqueda local se añadiría tras el operador de cruce, de esta manera, a cada descendiente se le ejecutaría una búsqueda local tras la mutación en caso de que se activase.

La búsqueda local elegida para el algoritmo genético V1 es la desarrollada en el apartado 4.3.2, la denominada como búsqueda local 1-swap acotada. Lo que se pretende analizar con esta variante es reducir el número de iteraciones del algoritmo genético promoviendo una mejora a través de la mejora de cada individuo generado.

Algoritmo Genético V2

La segunda posible variación es, partiendo del algoritmo genético presentado por [46], añadirle los operadores propuestos de elitismo y diversidad. De esta forma lo que se pretende es lograr una curva de mejora de la función objetivo con más pendiente al no permitir que buenas soluciones con fitness elevado se queden fuera de la nueva población por no haber sido elegidos. Se hace fundamental de igual forma, controlar la diversidad para evitar estancamientos y dotar a la población de individuos lo suficientemente alejados en el espacio de soluciones.

Algoritmo Genético V3

En esta versión del algoritmo genético, se toma como operador de cruce el denominado como reverse 1X definido por [53] tal y como se puede ver en Ilustración 4-12, los genes que se copian del progenitor 1 al descendiente 2, se colocan al final y de manera inversa a como aparecían en el progenitor. El resto de genes se toman del otro descendiente contando siempre con la corrección de la factibilidad.

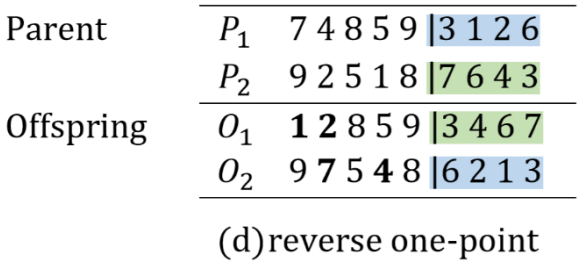


Ilustración 4-12 Cruce 1X inverso. Fuente: [53]

Algoritmo Genético V4

La última comparación, se ha realizado con los operadores del algoritmo genético híbrido con recocido simulado presentado en [53], en el que los operadores de cruce y mutación no están fijos, sino que en cada iteración se selecciona uno al azar. Este algoritmo originalmente está diseñado para el entorno *flow shop* con reentradas.

[53] presenta un total de seis operadores de cruce que se resume en la Ilustración 4-13.

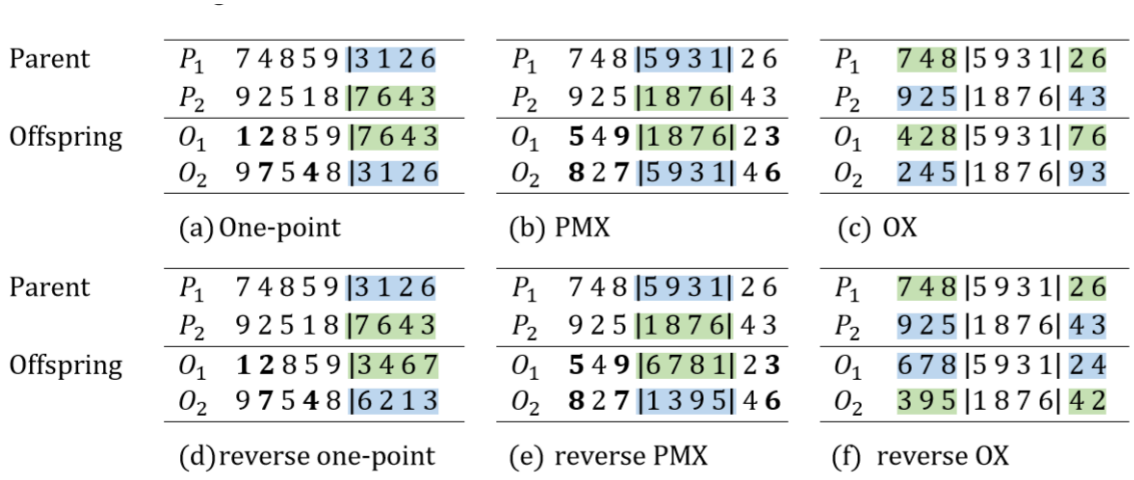


Ilustración 4-13 Operadores Cruce Algoritmo Genético V4. Fuente: [53]

Los cuatro nuevos operadores no comentados hasta el momento en el presente documento son el b), c), e) y f). Estos parten de la selección de dos posiciones al azar dentro de la secuencia e intercambiar o bien los genes que se encuentran dentro de esa selección (en

el caso del PMX), o fuera de esa selección con los del otro progenitor (en el caso del OX). La versión reverse de estos operadores provocan que el material genético que se trasfiere de un progenitor al otro, en ligar de aparecer en el orden secuencial original, aparezcan a la inversa.

Para la mutación, [53] también presenta tres posibles operadores que se resumen en la Ilustración 4-14.

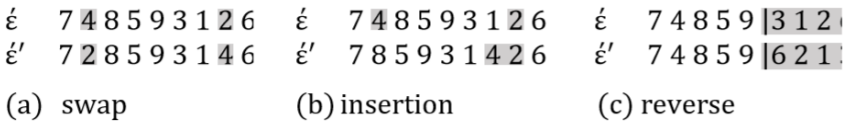


Ilustración 4-14 Operadores Mutación Algoritmo Genético V4. Fuente: [53]

Además de la mutación a través de un intercambio aleatorio, [53] presenta una mutación por inserción en la que se selecciona el gen de una posición y se insertará de manera adyacente a otro gen seleccionado de manera aleatoria. Además, presenta un tercer operador de cruce en el que se selecciona una posición de la secuencia y desde ese punto, se invierte el orden de los genes.

Para esta versión, se mantienen los operadores de elitismo y diversidad comentados en los apartados anteriores.

4.3.7 Pre-selección de los mejores algoritmos para su calibración

En este apartado se pretenden comparar los diferentes algoritmos propuestos resolviendo 120 instancias y comparando su desviación respecto a la mejor solución encontrada por alguno de los algoritmos en cada instancia. Con este análisis se pretende fundamentar la selección de los denominados algoritmos propuestos para su posterior calibración y comparación con los existentes en la literatura.

Las instancias serán generadas de manera aleatoria siguiendo la descripción del entorno y de las variables de entrada del problema comentadas en el apartado 5.2. Como condición de parada, se establece el tiempo máximo de 60 segundos.

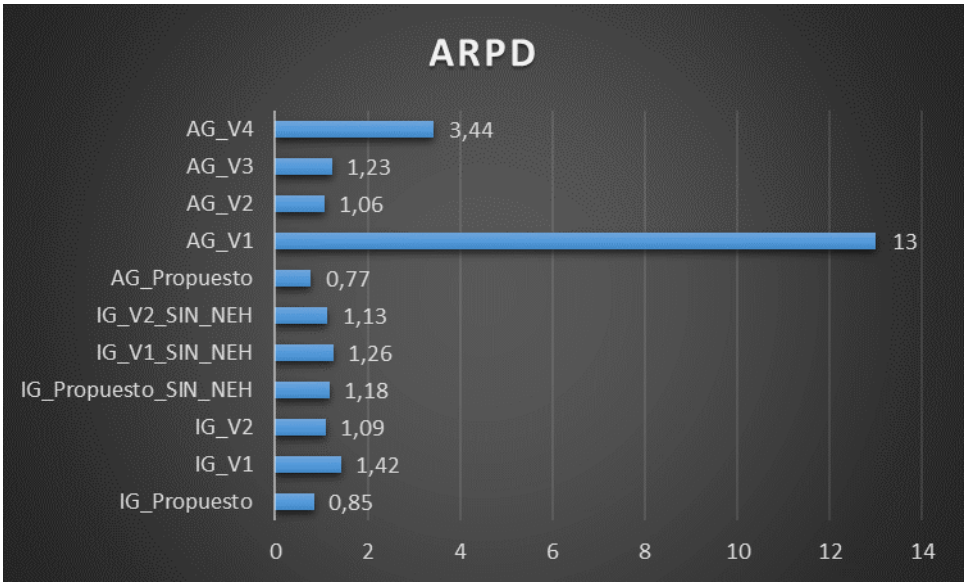


Ilustración 4-15 ARPD Preselección de Algoritmos. Fuente: Elaboración Propia

Con los datos resultantes presentados en la Ilustración 4-15, se evidencia la selección de los algoritmos propuestos frente a otras variaciones ya que estos son los que presentan una menor desviación en media porcentual de las mejores soluciones encontradas.

Capítulo 5 Análisis Computacional

5.1 Introducción

En este apartado se procederá a la adaptación de los algoritmos y funciones constructivas a la gestión del SU del HUVR. Posteriormente se definirán y calibrarán los diferentes parámetros necesarios para cada uno de los algoritmos para que finalmente, ejecutando una batería de instancias generadas siguiendo las distribuciones estadísticas con datos reales del HUVR y datos existentes en la literatura, comparar el comportamiento de los algoritmos existentes junto con los propuestos en el presente trabajo.

En cuanto a los experimentos computacionales que se han efectuado, todos los algoritmos se programaron en el lenguaje de programación C# y se ejecutaron en un ordenador con procesador AMD Ryzen 5 4500U de 2.38 Ghz. y 16 Gb de memoria RAM y con sistema operativo Windows 10 Home de 64 bits.

5.2 Datos de entrada a la optimización.

El presente proyecto tiene como objetivo el desarrollo de algoritmos que sean capaes en tiempo real de optimizar la secuencia de los pacientes por los diferentes recursos del servicio de urgencias (SU). Por ello, se generarán instancias que se asemejen en su mayor medida a la realidad del hospital. Siendo capaces a priori de generar casos hipotéticos en el que se dispongan de pacientes con diversas rutas y tiempos a través de SU se podría llegar a la optimización de los pacientes en tiempo real.

Generación de pacientes

En la literatura, la estimación del número de pacientes que van a frecuentar el SU de un hospital ha sido bastante estudiado, concluyéndose que, teniendo en cuenta la estocasticidad presente en este aspecto, hay variables que son significativas a la hora de estimar el número de pacientes que demandarán los SU.

Para la estimación del número de pacientes que llegan al SU del HUVR, se tomará la distribución de Poisson tal y como se expone en [15]. Para su simplificación solamente se tendrá en cuenta la franja horaria en la que se quiera optimizar. Es importante mencionar que para la estimación y cálculo de los valores esperado de llegada de pacientes en cada franja se toman como referencia dos años retrospectivos de datos de SU del HUVR.

Dado que el tiempo de permanencia medio en el HUVR se sitúa alrededor de las 4 horas se estimará el número de pacientes que llegan a lo largo de 4 horas dentro de los diferentes turnos, mañana, tarde y noche. De esta manera, se puede simular el número de pacientes que, en tiempo real se podrían encontrar en un momento determinado en el SU. Cada paciente de manera aleatoria se le asignará una ruta y prioridad asistencial siguiendo una distribución por medias obtenidos de los datos facilitados por el hospital. De los datos extraídos se puede afirmar que de media, se pueden encontrar alrededor de 35 pacientes en el SU del HUVR.

Se debe tener en cuenta que, a la hora de generar los pacientes, se generarán los respectivos pacientes ficticios explicados en el apartado 4.2.

Generación de tiempos de procesos.

Los tiempos de duración de las actividades de los pacientes también se encuentra dentro de un entorno estocástico en el que dos pacientes con el mismo motivo de consulta y prioridad asistencial pueden no tardar lo mismo en una determinada actividad del proceso de urgencias.

Es por ello por lo que se toman distribuciones para la estimación y predicción de los tiempos de proceso. De cara a la generación de los datos de entrada a las instancias que se resolverán simulando el comportamiento del SU del HUVR, dado que no se dispone de datos de duraciones en los datos que han sido facilitados, se tomarán como referencias las distribuciones propuestas para las principales actividades del SU en [15].

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

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

P: physician; PA: physician assistant.

Tabla 5-1 Distribución de tiempos de procesos del SU. Fuente: [8]

Recursos del Servicio de Urgencias

Para poder simular lo que ocurre realmente lo que sucede en un día cualquiera en el SU del HUVR se hace necesario dotar al sistema de los recursos tanto humanos como materiales de los que dispone a priori el hospital.

Dado que la demanda es variante respecto a las horas del día, como ya se ha comentado en el sub-apartado de generación de pacientes de este apartado consecuentemente, los recursos del SU también son diferentes para cada turno de trabajo: mañana, tarde y noche.

	Mañana	Tarde	Noche
Consultas	8	7	7
Enfermería	5	5	4
Camas de Observación	20	20	20
Celadores	3	3	3
Salas de Rayos	3	3	3

Tabla 5-2 Distribución de Recursos por turnos en HUVR. Fuente: Elaboración Propia

5.3 Calibración de parámetros.

Los parámetros propios de los algoritmos estructuran el funcionamiento de los mismos por lo que afectan significativamente a la calidad de la solución que proponen [42]. Para el análisis de los parámetros de cada algoritmo en primer lugar se definirán los parámetros y cómo influyen en el algoritmo, y tras proponer unos rangos de variación se ejecutarán 30 instancias diferentes para cada combinación de parámetros.

5.3.1 Parámetros Algoritmo Genético

El primer parámetro que se debe decidir en todo algoritmo genético es el tamaño de la población *P*, es decir, el número de diferentes soluciones que generaremos en cada iteración. Otro parámetro importante a calibrar es la probabilidad de cruce *Pc* y mutación *Pm*. Estas dos probabilidades provocan al ser menores que la unidad que pueda darse que una pareja de dos progenitores no genere un descendiente y que en caso de que lo generen, ese descendiente puede que no sufra una mutación.

De manera adicional, el algoritmo genético propuesto incluye la función de control de la diversidad que se activa en el momento en el que dos cromosomas tengan una similitud mayor a *S*, siendo *S* el número de pacientes que ocupan la misma posición en las dos secuencias y siendo *N* el número total de pacientes a optimizar.

Los rangos propuestos para estos parámetros son los siguientes:

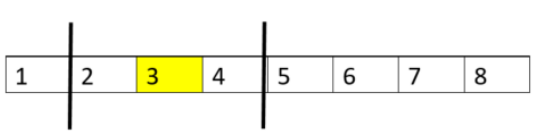
- Tamaño de Población *P*: $\{\frac{N}{2}, N, 2N\}$
- Probabilidad de cruce *Pc*: {0.9, 0.85, 0.8}
- Probabilidad de mutación *Pm*: {0.1, 0.2, 0.3}
- Similitud *S*: {N, N*0.9, N*0.8}

Como innovación frente al algoritmo planteado en [46], se plantea la dependencia de algunos de los parámetros con el tamaño del problema (número de pacientes). Los valores que tomaron para el algoritmo en (46) fueron *P* = 51, *Pc* = 0.8, *Pm* = 0.29.

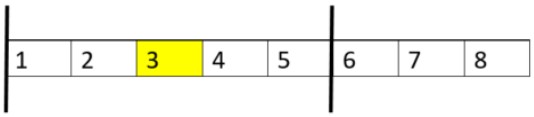
5.3.2 Parámetros Iterated Greedy

Dentro de la fase de intensificación del nuevo algoritmo (IG) propuesto, se encuentra el primer parámetro que calibrar. En primer lugar, dentro de la búsqueda local denominada como 1-swap acotada en el apartado 4.3.4, habría que calibrar el rango o la amplitud de posiciones en el que se puede hacer efectivo el intercambio. Dicho de otra forma, dado una determinada posición como pivote, decidir como máximo cuantas posiciones puede adelantar

o atrasar a dicho paciente *LS_ab*. A continuación, se presentan dos ejemplos de vecindades, uno para *LS_ab*=1 y otro para *LS_ab*=2



3	2	1	4	5	6	7	8
1	3	2	4	5	6	7	8
1	2	4	3	5	6	7	8
1	2	5	4	3	6	7	8



1	3	2	4	5	6	7	8
1	2	4	3	5	6	7	8

Ilustración 5-1 Ejemplo variación vecindad respecto al parámetro LS_ab. Fuente: Elaboración Propia

Del mismo modo, una vez finalizada la búsqueda, para aceptación o no de la nueva solución propuesta, tal y como se comentó en el apartado 4.3.4 sobre el recocido simulado, se debe decidir el valor de la temperatura inicial *To*, así como el factor reductor de la temperatura *alpha*.

En la literatura se encuentra que la temperatura inicial se calcula como:

$$T_o = \frac{T * \sum_{i=0}^{M} \sum_{j=0}^{N} p_{ij}}{N * M * 10}$$

Siendo *M* el número de máquinas que análogamente se corresponde con el número total de recursos disponibles en el SU, *N* el número de pacientes a programar y p_{ij} el tiempo de proceso estimado de cada actividad de la ruta de cada paciente en cada máquina. De esta manera se escala la temperatura respecto a las dimensiones del problema solo quedando por calibrar el factor *T*.

Por otro lado, dentro de la etapa de diversificación, en la fase de deconstrucción el último parámetro que se debe calibrar es el número de pacientes que se extraen de la solución propuesta *d.*

Los rangos propuestos para estos parámetros son los siguientes:

- Componentes que se extraen *d*: $\{\frac{N}{4}, \frac{N}{8}, \frac{N}{12}\}$
- Posiciones que puede adelantar o atrasar el intercambio en la búsqueda local *LS_ab*: $\{\frac{N}{4}, \frac{N}{8}, \frac{N}{12}\}$
- Factor de Temperatura *T*: {0.2, 0.4, 0.6, 0.8}
- *Alpha*: {0.91, 0.95, 0.99}

En el algoritmo (IG) original propuesto en [49] tras la calibración concluyeron que los mejores valores para los parámetros *d* =4 y *T* = 0.4.

5.3.3 Resultados de la calibración de los algoritmos

Para la calibración se han generado de manera aleatoria 30 instancias empleando las distribuciones de llegadas de pacientes y estimaciones de tiempos descritos en el apartado 5.2 del presente documento. Cada instancia se ha resuelto con cada una de las combinaciones de los parámetros de cada algoritmo (81 combinaciones de parámetros del AG y 108 combinaciones del IG). Como condición de parada, se ha establecido la única condición de parada sea el alcanzar el tiempo límite calculado como: $T_{lim} = \frac{N*N*M}{2000}$ (s).

A los resultados que proporcionen las resoluciones de las instancias con las diferentes combinaciones se les calculará el ARPD *(Average Relative Percentaje Deviation*). Puesto que en este punto de la experimentación no se están comparando los dos algoritmos entre ellos, sino que se desea saber la influencia de los parámetros sobre cada algoritmo, se tratarán por separados ambos análisis, aunque hayan resulto el mismo problema. El ARPD indica en tanto por ciento, cuanto se ha desviado cada combinación de parámetros de la mejor solución encontrada para cada instancia.

Para extraer conclusiones acerca de la significancia o no significancia de los diferentes parámetros sobre el algoritmo, se efectuará el Test de Kruskal Wallis. Este test no paramétrico se emplea para contrastar el análisis de la varianza de un factor o parámetro contrastando la hipótesis nula de que los efectos de los diferentes valores de ese parámetro son los mismos o, dicho de otro modo, que las muestras aleatorias provienen de distribuciones idénticas con un

nivel de significación del 5%. En el caso de rechazarse la hipótesis nula se podría afirmar que ese parámetro tiene una influencia significativa sobre el algoritmo estudiado.

Este análisis estadístico se ejecutará a través del lenguaje de programación R en su versión 4.1.0.

Parámetros del AG. Kruskal Wallis

En la Tabla 5-3 presentada a continuación se encuentran los resultados del test de Kruskal Wallis realizado sobre los diferentes parámetros.

	Parámetro	**Chi-squared**	**df**	**p-value**
Tamaño de población	***P***	55,309	2	**9,77E-10**
Probabilidad de cruce	***Pc***	2,219	2	0,3293
Probabilidad de mutar	***Pm***	31,046	2	**1,81E-04**
Similitud	***S***	5,774	2	**0,05573**

Tabla 5-3 Resultados Test Kruskal Wallis Calibración AG. Fuente: Elaboración Propia

En este caso, sale significativa la influencia que tiene el tamaño poblacional por lo que se tomará aquel valor de la población que presente un menor ARPD. Destaca también la significancia de la mutación frente a la de la probabilidad de cruce. El hecho de que el parámetro probabilidad de cruce no salga con diferencia significativa no implica que no tenga un efecto importante en el algoritmo, sino que, para los rangos de variación dados, el efecto que tiene este parámetro en el algoritmo es similar.

Por último, destaca el valor del p-valor del nuevo parámetro introducido, muy cercano a que sea significativo por lo que, quizás en el caso de haber ejecutado más instancias, la diferencia entre sus posibles valores si hubieran presentado diferencia significativa.

Para elegir qué valores de los diferentes parámetros deben tomar, se procederá a analizar los ARPD y se tomarán aquellos valores que presenten un menor ARPD. En la Tabla 5-4 se presentan los ARPD según los niveles que toma cada parámetro.

Tamaño de la población P	Niveles	N/2	N	2N
	ARPD	**1,550**	1,756	2,109
Probabilidad de cruce Pc	Niveles	0,90	0,85	0,8
	ARPD	**1,770**	1,826	1,858
Probabilidad de mutación Pm	Niveles	0,1	0,2	0,3
	ARPD	**1,593**	1,866	1,994
Similitud S	Niveles	N	0,9N	0,8N
	ARPD	1,957	**1,699**	1,797

Tabla 5-4 Valores ARPD para cada nivel de los parámetros del AG propuesto. Fuente: Elaboración propia

De este análisis se concluye que los valores que tomará el algoritmo genético propuesto para los parámetros son:

- Tamaño de Población *P*: $\{\frac{N}{2}\}$
- Probabilidad de cruce *Pc*: {0,9}
- Probabilidad de mutación *Pm*: {0,1}
- Similitud *S*: {N*0,9}

De esta manera, la población estará formada por un número de individuos equivalentes a la mitad del número de pacientes que estén en el servicio de urgencias. Por otro lado, de cada pareja que se obtenga de progenitores dentro de esa población, el 90% generarán descendientes de los cuales, solo un 10% sufrirán una mutación. Por último, se analizará que en la misma población no haya dos o más individuos que compartan más de un 90% de los genes.

Parámetros del IG. Kruskal Wallis

Del mismo modo, se presenta en primer lugar, en la Tabla 5-5, los resultados del test de Kruskal Wallis para los parámetros del algoritmo voraz iterativo (IG).

	Parámetro	**Chi-squared**	**df**	**p-value**
Componentes que se extraen	***d***	6,5012	2	**0,038**
Valores LS 1-swap acotada	***LS_ab***	7,789	2	**0,020**
Temperatura	***T***	0,293	2	0,96
Alpha	***alpha***	0,146	2	0,9295

Tabla 5-5 Resultados Test Kruskal Wallis Calibración IG. Fuente: Elaboración Propia

De estos resultados se puede apreciar la significancia de los parámetros que se podrían entender como principales dentro de las etapas principales de intensificación y diversificación. Por otro lado, hay que destacar la semejanza en el comportamiento del algoritmo para los valores de temperatura y el factor de enfriamiento dentro de los rangos establecidos.

Componentes que se extraen d	Niveles	N/4	N/8	N/12	
	ARPD	3,997	4,527	4,347	
Valores LS 1-swap acotada LS_ab	Niveles	N/4	N/8	N/12	
	ARPD	4,660	4,356	3,848	
Alpha	Niveles	0,91	0,95	0,99	
	ARPD	4,361	4,278	4,233	
Temperatura	Niveles	0,2	0,4	0,6	0,8
	ARPD	4,2723	4,453	4,394	4,043

Tabla 5-6 Valores ARPD para cada nivel de los parámetros del IG propuesto. Fuente: Elaboración propia

Una vez analizados los diferentes valores que toman los ARPD respecto a los diferentes parámetros, se pueden elegir los valores que tomarán los parámetros encontrando el mínimo de ARPD de cada factor.

De este análisis se concluye que los valores que tomará el algoritmo voraz iterativo (IG) propuesto para los parámetros son:

- Componentes que se extraen *d*: $\{\frac{N}{4}\}$
- Posiciones que puede adelantar o atrasar el intercambio en la búsqueda local *LS_ab*: $\{\frac{N}{12}\}$
- Factor de Temperatura *T*: {0,8}
- *Alpha*: {0,99}

Hay que destacar que el algoritmo resultante difiere en gran medida del propuesto por (49) en el número de componentes a extraer pasando de 4 a $\frac{N}{4}$ donde en situaciones en la que se encuentren 60 pacientes en el SU, en la fase de diversificación, el algoritmo extraerá 15 pacientes de la secuencia inicial para después probarlos en las diferentes posiciones de la subsecuencia.

También es destacable la restricción del intercambio de posiciones en la búsqueda local de la fase de intensificación permitiendo solamente intercambiar una posición con aquellas que se encuentren a $\frac{N}{12}$ posiciones por delante o por detrás.

5.4 Resultados computacionales y análisis

Una vez calibrados los dos algoritmos propuestos, se realizará un análisis computacional para comparar el funcionamiento de los cuatro algoritmos comentados a lo largo del presente documento. Para analizar el funcionamiento de cada uno de ellos, se generarán de manera aleatoria siguiendo los datos de distribuciones estadísticas del apartado 5.2 un total de 120 instancias que se resolverán con los cuatro algoritmos. El tiempo límite se establece como el único criterio de parada $T_{lim} = \frac{N*N*M}{2000}$ (s). El motivo de que este sea el criterio de parada es el deseo y la motivación de obtener un algoritmo que funcione de manera de manera iterativa en la red del HUVR y que se actualice y optimice la secuenciación de pacientes en tiempo real.

En primer lugar, se procede a analizar el número de instancias en la que cada uno de los algoritmos ha alcanzado la mejor solución de los cuatro. Nótese que puede darse el caso en el que dos algoritmos encuentren una solución que tenga el mismo valor de la función objetivo. En la Ilustración 5-2 se puede apreciar como el algoritmo que más veces alcanza una mejor solución es el algoritmo genético propuesto en este documento (48 veces), seguido de manera muy similar por el algoritmo voraz iterativo propuesto (46 veces). Hay que destacar también el mejor comportamiento del IG original pese a estar diseñado para otro entorno productivo frente al algoritmo genético del que se ha partido en este documento. El IG original ha alcanzado en 23 instancias la mejor solución frente a 5 del AG original.

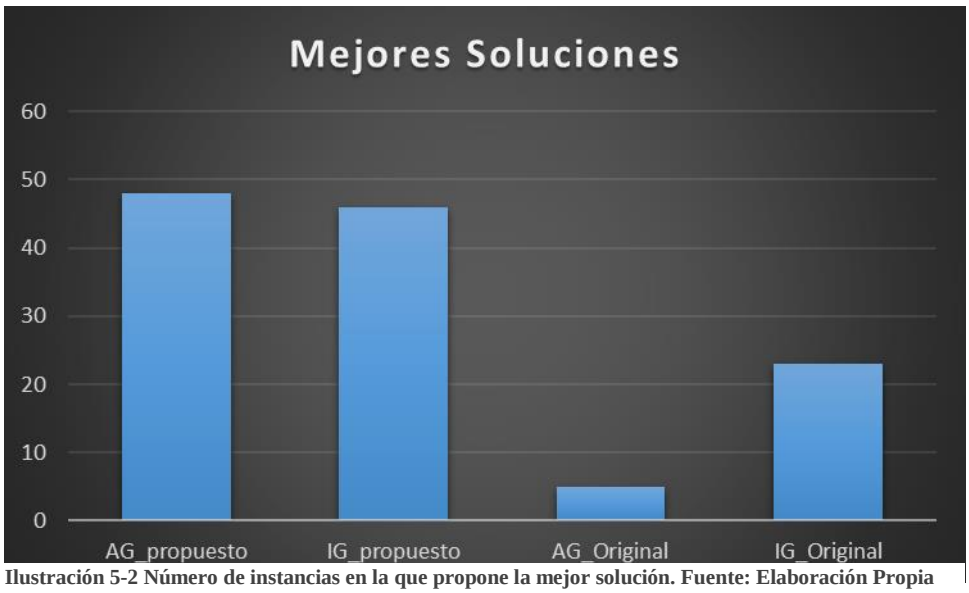


Ilustración 5-2 Número de instancias en la que propone la mejor solución. Fuente: Elaboración Propia

Por otro lado, el segundo análisis que se puede hacer es sobre el ARPD. De esta manera se puede observar cuán cerca están en promedio los diferentes algoritmos de la mejor solución encontrada dentro del tiempo límite establecido.

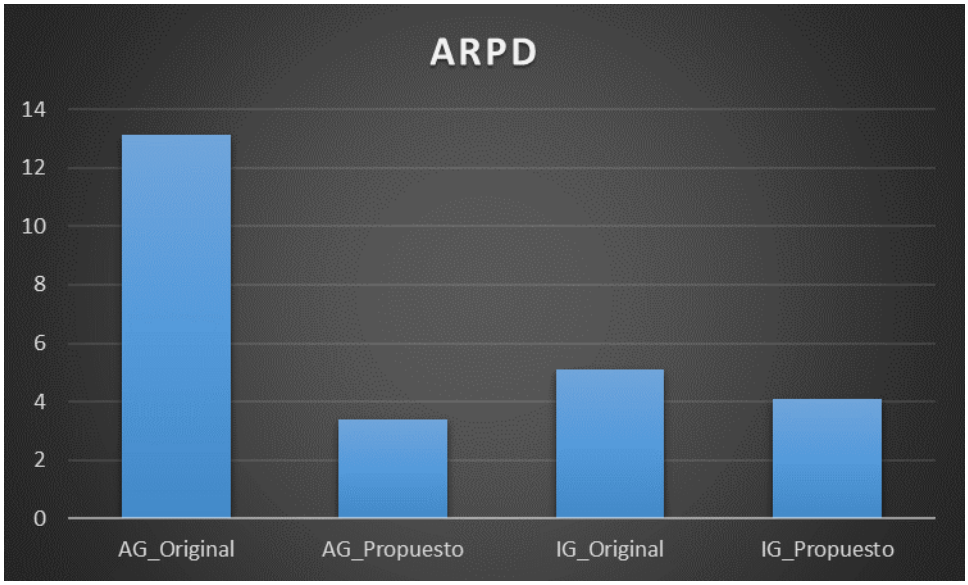


Ilustración 5-3 ARPD evaluado para 120 instancias. Fuente: Elaboración Propia

En este caso, en la Ilustración 5-3 se puede apreciar la comparación de los diferentes ARPD de los algoritmos originales y propuestos. Destaca en este gráfico lo lejos que en promedio se queda el algoritmo genético original frente a los otros tres algoritmos con un ARPD de un 13,21%. Esto indica que la curva de convergencia de este algoritmo es, a priori, más lenta que la de los otros algoritmos. El algoritmo que presenta un menor ARPD coincide con el que ha encontrado más veces la mejor solución de la instancia, el AG propuesto con un 3,41% de ARPD.

Por otro lado, los dos algoritmos iterativos poseen un ARPD bastante similar con una leve ventaja para el IG propuesto con un 4,08% frente al 5,09% del IG original.

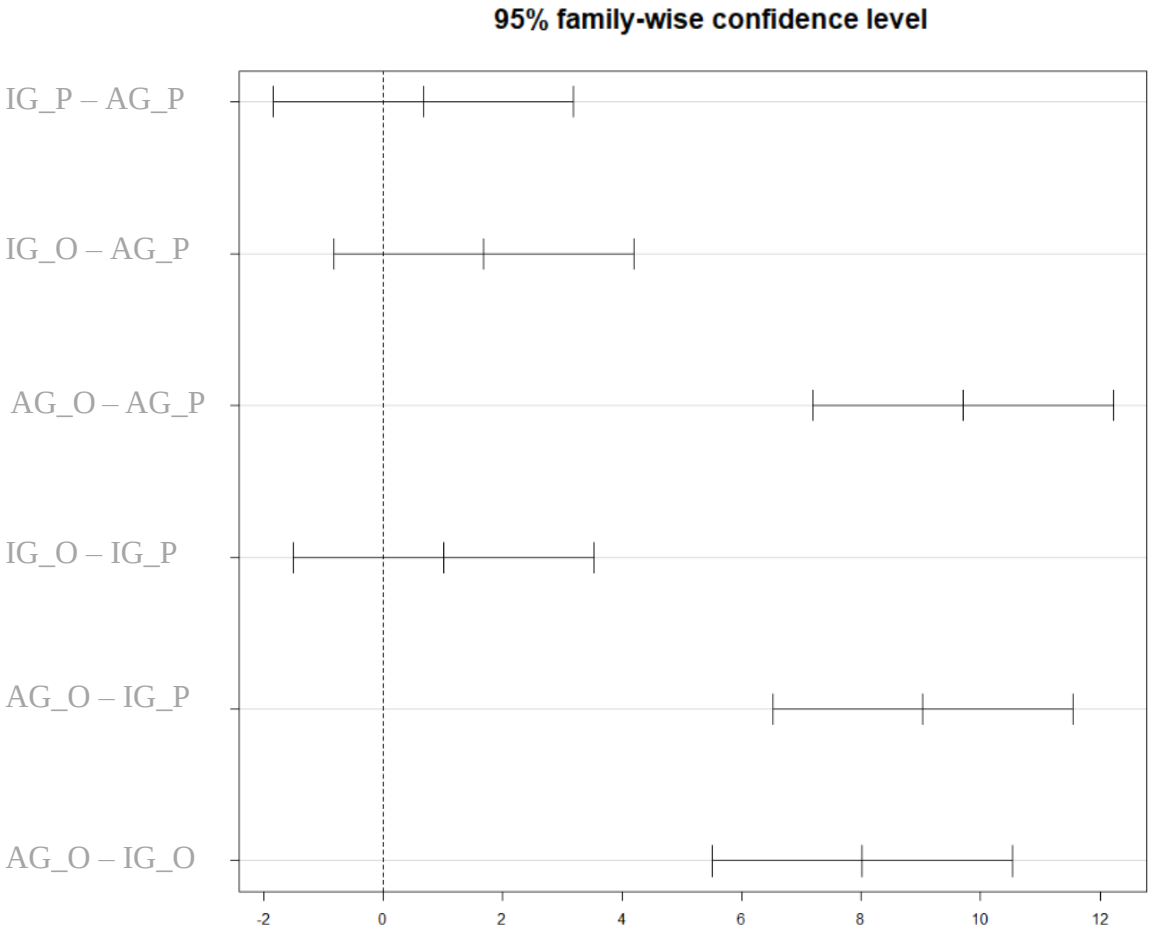
La comparación de número de veces que un algoritmo encuentra la mejor solución y su ARPD aporta mucha información ya que, por ejemplo, destaca el comportamiento del IG original que, aun sin ser el de los dos algoritmos que ha encontrado un mayor número de veces la mejor solución, su ARPD es bastante similar a los algoritmos propuestos.

Realizando por último el test de Kruskal Wallis para determinar si hay diferencia significativa entre los algoritmos, éste arroja los siguientes resultados:

	Chi-squared	df	p-value
Método de optimización	104,22	3	**2,2e-16**

Tabla 5-7 Resultado Test Kruskal Wallis Evaluación Algoritmos. Fuente: Elaboración Propia

Pese a obtener una diferencia significativa en el comportamiento de los algoritmos de optimización, se procede a analizar la diferencia de medias de ARPD de los algoritmos con la finalidad de concluir cuáles tienen un comportamiento similar o sin diferencia significativa.



Diferencias de medias para los diferentes Métodos de Optimización

IG_P: IG_Propuesto, | AG_P: AG_Propuesto | IG_O: IG_Original, | AG_O: AG_Original

Ilustración 5-4 Diferencia de ARPD entre algoritmos. Fuente: Elaboración Propia

En la Ilustración 5-4, se puede apreciar como aquellos casos en los aparece el AG_O (algoritmo genético original), desplaza la diferencia de medias en su perjuicio. Aquellos que tienen valores de ARPD similares, son aquellos que podrían plantearse como que tienen un comportamiento similar o sin diferencia significativa.

Para ello, descartando el Algoritmo Genético Original, se realiza de nuevo el test de Kruskal Wallis.

	Chi-squared	df	p-value
Método de optimización	10,741	2	**0,00471**

Tabla 5-8 Resultados Nuevo Test Kruskal Wallis. Fuente: Elaboración Propia

Se concluye de esta manera que hay diferencia significativa entre estos tres algoritmos (IG_O, IG_P y AG_P) siendo el Algoritmo Genético Propuesto el que presenta un menor ARPD.

Capítulo 6 Conclusiones

A lo largo de este proyecto se ha realizado el análisis y diseño de dos algoritmos de optimización basados en dos metaheurísticas muy extendidas en la literatura como son el algoritmo genético y el algoritmo voraz iterativo. La situación de partida es el servicio de urgencias, entorno que en la literatura se suele definir como un entorno *jobshop* flexible en la que cada paciente tiene que pasar por una serie de recursos para que se le practiquen ciertas actividades conforme a su ruta.

El objetivo principal de este proyecto es el diseño y desarrollo de un algoritmo eficiente que optimice la secuenciación de pacientes en el SU con el limitante de disponer de tiempos límites de computación muy bajos (del orden de segundos), para poder implementarlo de cara a una optimización recursiva en tiempo real.

Para ello, partiendo del modelado matemático de las restricciones que rigen el funcionamiento del servicio de urgencias, se ha realizado una función constructiva para que, dado una secuencia de pacientes, construya la solución admisible cumpliendo todas las restricciones.

De cara a la resolución, se han tomado como perspectivas iniciales, dos metaheurísticas que en la literatura se han aplicado al entorno *jobshop* y en especial una de ellas (el algoritmo genético) que era de gran interés por haber sido empleada en el entorno de urgencias hospitalarias. A partir de ellas, se han propuesto dos nuevos algoritmos:

- Algoritmo genético con cruce por selección de genes aleatorios entre los descendientes, elitismo para preservar dentro de la nueva generación a los mejores progenitores y descendientes y un control de la diversidad que evita una convergencia teniendo cromosomas idénticos o muy similares.
- Algoritmo voraz iterativo con una nueva búsqueda local acotada y con un factor reductor de la temperatura con el fin de ir reduciendo la probabilidad de aceptar un empeoramiento en la función objetivo.

Tras el diseño, se han generado 30 instancias con datos generados de manera aleatoria a través de distribuciones estadísticas basadas en datos extraídos del HUVR y de la literatura para su posterior ejecución con los algoritmos haciendo uso de los diferentes valores que pueden tomar parámetros determinantes de estas metaheurísticas.

Tras la elección de los mejores valores de los parámetros para cada algoritmo propuesto se procedió a la resolución de 120 instancias para comparar el comportamiento de los algoritmos con un tiempo de parada dependiente de las dimensiones del problema y que en la mayoría de los casos no superaría el minuto.

Los resultados presentados en el párrafo 5.3.4 son bastante reveladores en cuando al comportamiento de los algoritmos. Los algoritmos propuestos han resultado encontrar en el 78% de las instancias una mejor solución que los existentes en la literatura. De manera adicional, en el análisis de la desviación porcentual media medida a través del ARPD se observa como son de nuevo los algoritmos propuestos lo que presentan una menor desviación respecto a las mejores soluciones encontradas. Hay que destacar que el algoritmo que presenta mejores resultados tanto en número de veces que ha encontrado la mejor solución como en menor ARPD es el AG Propuesto. De manera adicional, analizando el comportamiento de los algoritmos existentes para el problema propuesto, el algoritmo IG presenta mejores resultados que el algoritmo genético en ambos indicadores.

El hecho de que en este proyecto el algoritmo que mejores resultados presente sea el Algoritmo Genético Propuesto, esto implica que, hasta el momento, este algoritmo sea el más eficiente a la hora de proponer una buena solución para el problema de estudio. Como posible futura línea de estudio se podría plantear nuevas evoluciones y variaciones a estos algoritmos por ejemplo añadiendo una búsqueda local tras la generación de un nuevo descendiente en el AG o probando una inserción de componentes por pares en lugar de la voraz existente para la etapa constructiva del IG.

Otra posible vía de investigación sería ver cómo afectan las heurísticas constructivas a estos algoritmos tomando quizás como referencia alguna constructiva que conlleve menor tiempo computacional que la NEH para dar al IG mayor tiempo para realizar iteraciones.

Por último, teniendo en cuenta el impacto que tienen las metaheurísticas en cuanto la capacidad de ofrecer una muy buena solución a un problema complejo en tiempos computacionales reducidos, sería interesante la implantación de algoritmos de gestión en tiempo real en diferentes entornos. Un ejemplo de esta implantación es el proyecto en el que estoy trabajando actualmente para el HUVR denominado “REACT – COVID19, Gestión de Recursos urgEntes ante cAmbios en la atenCión de pacienTes COVID”. El presente proyecto pretende dotar a los profesionales sanitarios de un Servicio de Urgencias Hospitalario (SUH) de un prototipo de un sistema de soporte a la toma de decisiones orientada a la mejora de

procesos para garantizar la calidad asistencial al paciente a través del cumplimiento de los estándares establecidos por los sistemas de salud (en términos de tiempo de espera), y adecuar la actividad asistencial de los profesionales clínicos (optimizar la eficiencia y la idoneidad del desempeño de la actividad). Para ello, me encuentro analizando metaheurísticas que resuelvan el problema de optimización e integrando los datos de seguimiento de los pacientes en tiempo real para dar una respuesta de secuenciación en tiempos que no superan el minuto. Una optimización en tiempo real con secuenciaciones de actividades que se adapten a la estocasticidad presente en el SU puede desembocar en una mejora de los servicios prestados y, por ende, en la calidad y rapidez de atención de los pacientes.

Bibliografía

[1] Omar, E. R., Garaix, T., Augusto, V., & Xie, X. (2015). A stochastic optimization model for shift scheduling in emergency departments. Health care management science, 18(3), 289-302. https://doi.org/10.1007/s10729-014-9300-4

[2] Xu, M., Wong, T. C., & Chin, K. S. (2013). Modeling daily patient arrivals at Emergency Department and quantifying the relative importance of contributing variables using artificial neural network. Decision Support Systems, 54(3), 1488-1498. https://doi.org/10.1016/j.dss.2012.12.019

[3] Marchesi, J. F., Hamacher, S., & Fleck, J. L. (2020). A stochastic programming approach to the physician staffing and scheduling problem. Computers & Industrial Engineering, 142, 106281. https://doi.org/10.1016/j.cie.2020.106281

[4] Cabrera, E., Taboada, M., Iglesias, M. L., Epelde, F., & Luque, E. (2012). Simulation optimization for healthcare emergency departments. Procedia Computer Science, 9, 1464-1473. https://doi.org/10.1016/j.procs.2012.04.161

[5] Zeltyn, S., Marmor, Y. N., Mandelbaum, A., Carmeli, B., Greenshpan, O., Mesika, Y., ... & Basis, F. (2011). Simulation-based models of emergency departments: Operational, tactical, and strategic staffing. ACM Transactions on Modeling and Computer Simulation (TOMACS), 21(4), 1-25. https://doi.org/10.1145/2000494.2000497

[6] Hoot, N. R., & Aronsky, D. (2008). Systematic review of emergency department crowding: causes, effects, and solutions. Annals of emergency medicine, 52(2), 126-136. https://doi.org/doi:10.1016/j.annemergmed.2008.03.014

[7] Green, L. V. (2008). Using Operations Research to reduce delays for healthcare. Using Operations Research to Reduce Delays for Healthcare. INFORMS TutORials in Operations Research () 1-16 https://doi.org/10.1287/educ.1080.0049

[8] Salama, M. (2021). Adaptive neighborhood simulated annealing for sustainability-oriented single machine scheduling with deterioration effect. Applied Soft Computing, 110. https://doi.org/10.1016/j.asoc.2021.107632

[9] Duma, D., & Aringhieri, R. (2020). An ad hoc process mining approach to discover patient paths of an Emergency Department. Flexible Services and Manufacturing Journal, 32(1), 6-34. https://doi.org/10.1007/s10696-018-9330-1

[10] Daldoul, D., Nouaouri, I., Bouchriha, H., & Allaoui, H. (2018). A stochastic model to minimize patient waiting time in an emergency department. Operations Research for Health Care, 18, 16-25. https://doi.org/10.1016/j.orhc.2018.01.008

[11] Jones, S. S., Evans, R. S., Allen, T. L., Thomas, A., Haug, P. J., Welch, S. J., & Snow, G. L. (2009). A multivariate time series approach to modeling and forecasting demand in the emergency department. Journal of biomedical informatics, 42(1), 123-139. https://doi.org/10.1016/j.jbi.2008.05.003

[12] Van Bockstal, E., & Maenhout, B. (2019). A study on the impact of prioritising emergency department arrivals on the patient waiting time. Health care management science, 22(4), 589-614. https://doi.org/10.1007/s10729-018-9447-5

[13] Feng, Y. Y., Wu, I. C., & Chen, T. L. (2017). Stochastic resource allocation in emergency departments with a multi-objective simulation optimization algorithm. Health care management science, 20(1), 55-75. https://doi.org/10.1007/s10729-015-9335-1

[14] Diefenbach, M., & Kozan, E. (2011). Effects of bed configurations at a hospital emergency department. Journal of Simulation, 5(1), 44-57. https://doi.org/10.1057/jos.2010.1

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

[16] Aringhieri, R., Bruni, M. E., Khodaparasti, S., & van Essen, J. T. (2017). Emergency medical services and beyond: Addressing new challenges through a wide literature review. Computers & Operations Research, 78, 349-368. https://doi.org/10.1016/j.cor.2016.09.016

[17] Wang, T., Guinet, A., Belaidi, A., & Besombes, B. (2009). Modelling and simulation of emergency services with ARIS and Arena. Case study: the emergency department of Saint Joseph and Saint Luc Hospital. Production Planning and Control, 20(6), 484-495. https://doi.org/10.1080/09537280902938605

[18] Allihaibi, W. G., Cholette, M. E., Masoud, M., Burke, J., & Karim, A. (2020). A heuristic approach for scheduling patient treatment in an emergency department based on bed blocking. International Journal of Industrial Engineering Computations, 11(4), 565-584. https://doi.org/10.5267/j.ijiec.2020.4.005

[19] Harzi, M., Condotta, J. -., Nouaouri, I., & Krichen, S. (2017). Scheduling patients in emergency department by considering material resources. Procedia Computer Science, 112, 713-722. https://doi.org/10.1016/j.procs.2017.08.153

[20] Alquézar-Arbé, A., Piñera, P., Jacob, J., Martín, A., Jiménez, S., Llorens, P., . . . red de investigacion SIESTA. (2020). Impact of the COVID-19 pandemic on hospital emergency departments: Results of a survey of departments in 2020 — the spanish encovur study. [Impacto organizativo de la pandemia COVID-19 de 2020 en los servicios de urgencias hospitalarios Españoles: Resultados del estudio encovur] Emergencias, 32(5), 320-331.

[21] Molina Gutiérrez, M. Á., Ruiz Domínguez, J. A., Bueno Barriocanal, M., de Miguel Lavisier, B., López López, R., Martín Sánchez, J., & de Ceano-Vivas la Calle, M. (2020). Impact of the COVID-19 pandemic on emergency department: Early findings from a hospital in Madrid. [Impacto de la pandemia COVID-19 en urgencias: primeros hallazgos en un hospital de Madrid] Anales De Pediatría, 93(5), 313-322. https://doi.org/10.1016/j.anpedi.2020.06.021

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

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

[24] Alves de Queiroz, T., Iori, M., Kramer, A., & Kuo, Y. -. (2021). Scheduling of patients in emergency departments with a variable neighborhood search. Lecture Notes in Computer Science (LNCS), 12559, 138-151. https://doi.org/10.1007/978-3-030-69625-2_11

[25] Urgencias. (2017). Memoria Hospital Universitario Virgen del Rocío 2017. https://www.hospitaluvrocio.es/memoria17/urgencias

[26] Harzi, M., Condotta, J. -., Nouaouri, I., & Krichen, S. (2017). Scheduling patients in emergency department by considering material resources. Procedia Computer Science, 112 713-722. https://doi.org/10.1016/j.procs.2017.08.153

[27] Van Bockstal, E., & Maenhout, B. (2019). A study on the impact of prioritising emergency department arrivals on the patient waiting time. Health Care Management Science, 22(4), 589-614. https://doi.org/10.1007/s10729-018-9447-5

[28] Molina-Pariente, J.M, Núñez Jaldon, A. M., Avilés Gómez, M.D., Bueno Monreal, C. (2021). Soporte a la toma de decisiones en la gestión de pacientes COVID en un servicio de urgencia hospitalaria. En B. Puebla-Martínez (Ed.). Ecosistema de una pandemia. COVID-19, la transformación mundial (pp. 27-51). Dykinson, S.L.

[29] Pinedo, M. L. (2016). Scheduling: Theory, algorithms, and systems, fifth edition (pp. 1-670) doi:10.1007/978-3-319-26580-3

[30] Zhu, W., Patterson, B. W., Smith, M., Rifleman, A. C., Carayon, P., & Li, J. (2020). A markov chain model for transient analysis of handoff process in emergency departments. IEEE Robotics and Automation Letters, 5(3), 4360-4367. https://doi.org/10.1109/LRA.2020.2996066

[31] Karboub, K., Mohamed, T., Moutaouakkil, F., Sofiene, D., & Dandache, A. (2020). Emergency patient's arrivals management based on IoT and Discrete Simulation using ARENA. https://doi.org/10.1007/978-3-030-58008-7_19

[32] Soler, W., Gómez Muñoz, M., Bragulat, E., & Álvarez, A. (2010). El triaje: herramienta fundamental en urgencias y emergencias. Anales del Sistema Sanitario de Navarra, 33(Supl. 1), 55-68. Scielo España: http://scielo.isciii.es/scielo.php?script=sci_arttext&pid=S1137-62720100002000008&lng=es&tlng=es.

[33] Rosocha, L., Vernerová, S., & Verner, R. (2015). Medical staff scheduling using simulated annealing. Quality Innovation Prosperity, 19(1), 1-11. https://doi.org/10.12776/QIP.V19I1.405

[34] Mscbs.gob.es. (2021). Retrieved 24 May 2021, from https://www.mscbs.gob.es/organizacion/sns/planCalidadSNS/docs/UUH.pdf.

[35] Yousefi, M., & Yousefi, M. (2020). Human resource allocation in an emergency department: A metamodel-based simulation optimization. Kybernetes, 49(3), 779-796. doi:10.1108/K-12-2018-0675

[36] Easter, B., Houshiarian, N., Pati, D., & Wiler, J. L. (2019). Designing efficient emergency departments: Discrete event simulation of internal-waiting areas and split flow sorting. American Journal of Emergency Medicine, 37(12), 2186-2193. https://doi.org/10.1016/j.ajem.2019.03.017

[37] Ala, A., & Chen, F. (2020). Alternative mathematical formulation and hybrid meta-heuristics for patient scheduling problem in health care clinics. Neural Computing and Applications, 32(13), 8993-9008. https://doi.org/10.1007/s00521-019-04405-4

[38] Chico-Sánchez, P., Gras-Valentí, P., Mora-Muriel, J. G., Algado-Sellés, N., Sánchez-Payá, J., Llorens, P., . . . Grupo de Trabajo COVID-19 de la Comision de Infecciones. (2020). Impact of the covid-19 pandemic on health care workers in a tertiary care hospital emergency department. [Impacto de la pandemia de covid-19 en los trabajadores sanitarios del servicio de urgencias de un hospital terciario] Emergencias, 32(4), 227-232.

[39] H. (2020, 27 marzo). El Virgen del Rocío crea zonas de aislamiento del COVID-19 en urgencias, UCI, plantas de hospitalización y laboratorios - Hospital Universitario Virgen del Rocío. Hospital Universitario Virgen del Rocío. Recuperado en 24 de mayo de 2021 de, https://www.hospitaluvrocio.es/historico-noticias/el-virgen-del-rocio-crea-zonas-de-aislamiento-del-covid-19-en-urgencias-uci-plantas-de-hospitalizacion-y-laboratorios/

[40] H. (2021). Recuperado 25 de mayo de 2021, de https://www.hospitaluvrocio.es/wp-content/uploads/2020/04/PROTOCOLO-CIRUGIA-DE-URGENCIAS-COVID-19.pdf

[41] García-Rojo, E., Manfredi, C., Santos-Pérez-de-la-Blanca, R., Tejido-Sánchez, Á., García-Gómez, B., Aliaga-Benítez, M., . . . Rodriguez-Antolín, A. (2021). Impact of COVID-19 outbreak on urology surgical waiting lists and waiting lists prioritization strategies in the Post-COVID-19 era | Impacto del brote de COVID-19 en las listas de espera de cirugía urológica y estrategias de priorización en la era post-COVID-19. Actas Urologicas Españolas, 45(3), 207–214. https://doi.org/10.1016/j.acuro.2020.11.001

[42] Villalba Matamoros, M. E., & Kumral, M. (2019). Calibration of genetic algorithm parameters for mining-related optimization problems. Natural Resources Research, 28(2), 443-456. doi:10.1007/s11053-018-9395-2

[43] Yeh, J. -., & Lin, W. -. (2007). Using simulation technique and genetic algorithm to improve the quality care of a hospital emergency department. Expert Systems with Applications, 32(4), 1073-1083. https://doi.org/10.1016/j.eswa.2006.02.017

[44] Ajmi, F., Othman, S. B., Zgaya, H., Hammadi, S., & Renard, J.-M. (2017). An innovative approach to jointly scheduling and assigning a consultation time to patients arriving in the emergency department. Studies in Health Technology and Informatics, 245, 989-993. https://doi.org/10.3233/978-1-61499-830-3-989

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

[46] Azadeh, A., Hosseinabadi Farahani, M., Torabzadeh, S., & Baghersad, M. (2014). Scheduling prioritized patients in emergency department laboratories. Computer Methods and Programs in Biomedicine, 117(2), 61-70. https://doi.org/10.1016/j.cmpb.2014.08.006

[47] Ben Othman, S., Hammadi, S., & Quilliot, A. (2015). Multi-objective evolutionary for multi-skill health care tasks scheduling. IFAC-PapersOnLine, 28(3) 704-709. https://doi.org/10.1016/j.ifacol.2015.06.165

[48] Aqel, G. A., Li, X., Gao, L., Gong, W., Wang, R., Ren, T., & Wu, G. (2019). Using iterated greedy with a new population approach for the flexible jobshop scheduling problem. IEEE International Conference on Industrial Engineering and Engineering Management, 1235-1239. https://doi.org/10.1109/IEEM.2018.8607708

[49] Ruiz R, Stützle T (2007) A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. European Journal of Operational Research, 177(3):2033–2049. https://doi.org/10.1016/j.ejor.2005.12.009

[50] Gao, Z. Cao, L. Zhang, Z. Chen, Y. Han, Q. Pan (2019). A review on swarm intelligence and evolutionary algorithms for solving flexible job shop scheduling problems IEEE/CAA J. Autom. Sin., 6 (4), 904-916, https://doi.org/10.1109/JAS.2019.1911540

[51] Modrak, V., Pandian, R. S., & Semanco, P. (2021). Calibration of GA parameters for layout design optimization problems using design of experiments. Applied Sciences., 11(15). https://doi.org/10.3390/app11156940

[52] Zeng, C., Tang, J., Fan, Z., & Yan, C. (2019). Auction-based approach for a flexible job-shop scheduling problem with multiple process plans. Engineering Optimization, 51(11), 1902-1919. https://doi.org/10.1080/0305215X.2018.1561884

[53] Rifai, A. P., Kusumastuti, P. A., Mara, S. T. W., Norcahy, R., & Dawal, S. Z. (2021). Multi-operator hybrid genetic algorithm-simulated annealing for reentrant permutation flow-shop scheduling. ASEAN Engineering Journal, 11(3), 109-126. doi:10.11113/AEJ.V11.16875

Anexo

Declaración de funciones Algoritmo Genético

```
        public class Cromosoma
        {
            public double Fitness { get; set; }
            public List<int> Secuencia { get; set; }
            public override string ToString()
            {
                return "Fitness: " + Fitness + " Seq " + string.Join(" ,",
Secuencia.ToArray());
            }
        }

static public List<Cromosoma> Poblacion_Inicial(ClaseDatosOpt datos, int N, int
P)
        {
            int i;
            List<Cromosoma> Poblacion = new List<Cromosoma>();
            //Añado el cromosoma NEH a la población inicial
            int seed = Aleatorio();
            var random = new Random(seed);
            //Relleno con el resto de cromosomas aleatorios
            double fit;
            int pos;

            while (Poblacion.Count < P)
            {
                List<int> New_seq = new List<int>();
                for (i = 1; i <= N; i++)
                {

                    pos = random.Next(0, i);
                    New_seq.Insert(pos, i);
                }
                fit = Evaluacion(New_seq, datos);
                Poblacion.Add(new Cromosoma() { Fitness = fit, Secuencia =
New_seq });
            }

            return Poblacion;
        }



        static public List<double> Wheel(List<Cromosoma> Poblacion, int P)
        {
            int i;
            List<double> Ruleta = new List<double>();
            //Relleno la ruleta
            double suma_fitness = 0;
            double p_acumulado = 0;
```

```
            for (i = 0; i < Poblacion.Count; i++)
            {
                suma_fitness += Poblacion[i].Fitness;
            }
            for (i = 0; i < Poblacion.Count; i++)
            {
                if (i == (Poblacion.Count - 1))
                {
                    p_acumulado = 1;
                }
                else
                {
                    p_acumulado += ((Poblacion[i].Fitness) / suma_fitness);
                }
                Ruleta.Add(p_acumulado);
            }
            return (Ruleta);
        }



        static public List<Cromosoma> Elitismo(List<Cromosoma> Poblacion,
List<Cromosoma> Poblacion_nueva, int N)
        {
            int i;
            Poblacion_nueva = Poblacion_nueva.OrderBy(x => x.Fitness).ToList();
            Poblacion = Poblacion.OrderByDescending(x => x.Fitness).ToList();
            for (i = 0; i < Poblacion.Count; i++)
            {
                if (Poblacion_nueva[i].Fitness <= Poblacion[i].Fitness)
                {
                    Poblacion_nueva[i] = Poblacion[i];
                }
            }
            return Poblacion_nueva;
        }



static public List<Cromosoma> Diversidad(List<Cromosoma> Poblacion_nueva, int N,
ClaseDatosOpt datos)
        {
            int i, j, h, k, cont, pos;

            int similitud = N*9/10;
            int seed = Aleatorio();
            var random = new Random(seed);
            for (i = 0; i < Poblacion_nueva.Count; i++)
            {
                for (j = 0; j < Poblacion_nueva.Count & i > j; j++)
                {
                    cont = 0;
                    double fit;
                    for (h = 0; h < N; h++)
                    {
                        if (Poblacion_nueva[i].Secuencia[h] ==
Poblacion_nueva[j].Secuencia[h])
                        {
                            cont++;
```

```
                        }
                    }
                    if (cont >= similitud)
                    {
                        List<int> New_seq = new List<int>();
                        for (k = 1; k <= N; k++)
                        {

                            pos = random.Next(0, k);
                            New_seq.Insert(pos, k);
                        }
                        fit = Evaluacion(New_seq, datos);

                        if(Poblacion_nueva[j].Fitness<=
Poblacion_nueva[i].Fitness)
                        {
                            Poblacion_nueva[j].Secuencia = New_seq;
                            Poblacion_nueva[j].Fitness = fit;
                        }
                        else
                        {
                            Poblacion_nueva[i].Secuencia = New_seq;
                            Poblacion_nueva[i].Fitness = fit;
                        }
                    }
                }
            }
            return Poblacion_nueva;
        }



static public double Evaluacion(List<int> secuencia, ClaseDatosOpt datos)
        {
            double FO, fitness;
            int[] planned = new int[datos.NI];
            int[] check_planned = new int[datos.NI];
            ClaseSolucionOpt solucion_heuristica;
            (solucion_heuristica, FO) = construir_solucion(secuencia.ToArray(),
datos, planned, check_planned);
            fitness = 1 / (FO);

            return (fitness);
        }




static public List<int> Cruce1X(List<Cromosoma> Poblacion, int pos_padre1, int
pos_padre2, int N, Random random)
        {

            List<int> Descendiente1 = new List<int>();

            List<int> Posiciones_repetidas1 = new List<int>();

            int i, h;
```

```
            //Cruce
            int cruce = random.Next(0, N);
            for ( i = 0; i < cruce; i++)
            {
                Descendiente1.Add(Poblacion[pos_padre1].Secuencia[i]);

            }
            for ( i = cruce; i < N; i++)
            {
                Descendiente1.Add(Poblacion[pos_padre2].Secuencia[i]);

            }


            //Corrección de factibilidad
            for (i = 1; i <= N; i++)
            {
                int contador1 = 0;

                for (h = 0; h < N; h++)
                {
                    if (Descendiente1[h] == i)
                    {
                        if (contador1 == 1)
                        {
                            Posiciones_repetidas1.Add(h);
                        }
                        contador1++;
                    }

                }
            }
            Posiciones_repetidas1 = Posiciones_repetidas1.OrderBy(x =>
x).ToList();

            for (i = 0; i < N; i++)
            {
                if (Posiciones_repetidas1.Count > 0)
                {
                    if (Descendiente1.Any(x => x ==
Poblacion[pos_padre2].Secuencia[i]))
                    { }
                    else
                    {
                        Descendiente1[Posiciones_repetidas1[0]] =
Poblacion[pos_padre2].Secuencia[i];
                        Posiciones_repetidas1.RemoveAt(0);
                    }
                }

            }


            return (Descendiente1);
        }
```

```
static public List<int> CruceRandom(List<Cromosoma> Poblacion, int pos_padre1,
int pos_padre2, int N, Random random)
        {

            List<int> Descendiente1 = new List<int>();

            List<int> Posiciones_repetidas1 = new List<int>();

            int i, h, aux;
            //Cruce
            for(i=0;i<N;i++)
            {
                aux = random.Next(0, 2);
                if(aux==0)
                {
                    Descendiente1.Add(Poblacion[pos_padre1].Secuencia[i]);

                }
                else
                {
                    Descendiente1.Add(Poblacion[pos_padre2].Secuencia[i]);

                }


            }
            //Corrección de factibilidad
            for (i = 1; i <= N; i++)
            {
                int contador1 = 0;

                for (h = 0; h < N; h++)
                {
                    if (Descendiente1[h] == i)
                    {
                        if (contador1 == 1)
                        {
                            Posiciones_repetidas1.Add(h);
                        }
                        contador1++;
                    }

                    //Salen dos vectores con las posiciones en las que hay
pacientes repetidos

                }
            }
            Posiciones_repetidas1 = Posiciones_repetidas1.OrderBy(x =>
x).ToList();

            for (i = 0; i < N; i++)
            {
                if (Posiciones_repetidas1.Count > 0)
                {
                    if (Descendiente1.Any(x => x ==
Poblacion[pos_padre2].Secuencia[i]))
                    { }
                    else
                    {
```

```
                        Descendiente1[Posiciones_repetidas1[0]] =
Poblacion[pos_padre2].Secuencia[i];
                        Posiciones_repetidas1.RemoveAt(0);
                    }
                }

            }

            return (Descendiente1);
        }



static public List<int> Mutacion(List<int> Descendiente)
        {
            int pos_mutacion_a;
            int pos_mutacion_b;
            int N = Descendiente.Count;
            int temp;
            int numero_mutaciones = 1;
            for (int mut = 0; mut < numero_mutaciones; mut++)
            {
                int seed = Aleatorio();
                var random = new Random(seed);
                pos_mutacion_a = random.Next(0, N);
                seed = Aleatorio();
                random = new Random(seed);
                pos_mutacion_b = random.Next(0, N);

                while (pos_mutacion_a == pos_mutacion_b)
                {
                    seed = Aleatorio();
                    random = new Random(seed);
                    pos_mutacion_b = random.Next(0, N);
                }

                temp = Descendiente[pos_mutacion_a];
                Descendiente[pos_mutacion_a] = Descendiente[pos_mutacion_b];
                Descendiente[pos_mutacion_b] = temp;
            }

            return (Descendiente);
        }
```

Funciones Principales Algoritmos Genéticos

```
static public (List<int>, double) Optimizador_AG_Azadeh(ClaseDatosOpt datos,
double tiempo_limite_ejecucion)
        {

            //Declaración de los parámetros auxiliares
            int i, N = datos.NI, P = 51;
            Stopwatch sw = new Stopwatch();

            //-----------------------------------------------------------------
-
            //BUCLE
            List<Cromosoma> Poblacion_nueva = new List<Cromosoma>();
            List<Cromosoma> Poblacion_descendientes = new List<Cromosoma>();
            List<Cromosoma> Poblacion = new List<Cromosoma>();

            sw.Start();
            //Población Inicial
            Poblacion_nueva = Poblacion_Inicial_Aleatoria(datos, N, P);
            //BUCLE
            double tiempo_maximo = tiempo_limite_ejecucion;

            int iteraciones = 0;
            double fit;

            int seed;
            var random = new Random();


            while (sw.Elapsed.TotalSeconds < tiempo_maximo )
            {

                seed = Aleatorio();
                random = new Random(seed);

                foreach (Cromosoma cromosoma in Poblacion_nueva)
                {
                    Poblacion.Add(cromosoma);
                }

                //Bucle de selección-cruce-mutación en la que cada pareja de
padres tendrá 1 hijo
                int pos_padre1 = 0, pos_padre2 = 0;
                double eleccion = new double();
                List<double> Ruleta = Wheel(Poblacion, P);


                while (Poblacion_descendientes.Count < P)
                {
                    List<int> Descendiente1 = new List<int>();

                    //Selección de individuos para que tengan descendientes
probabilidad de cruce =0.8
                    double pc = 0.8, probabilidad;
                    int activador = 0;
                    while(activador==0)
                    {
                        eleccion = random.NextDouble();
                        for (i = 0; i < P; i++)
                        {
```

```
                            if (Ruleta[i] >= eleccion)
                            {
                                pos_padre1 = i;
                                i = P;
                            }
                        }

                        eleccion = random.NextDouble();
                        for (i = 0; i < P; i++)
                        {
                            if (Ruleta[i] >= eleccion)
                            {
                                pos_padre2 = i;
                                i = P;
                            }
                        }

                        probabilidad = random.NextDouble();
                        if(probabilidad<=pc)
                        {
                            activador = 1;
                        }

                    }
                    //Cruce
                    (Descendiente1) = Cruce1X(Poblacion, pos_padre1, pos_padre2,
N, random);



                    //Mutación con probabilidad de mutar 0.29
                    double pm = 0.29;
                    probabilidad = random.NextDouble();
                    if(probabilidad<=pm)
                    {
                        Descendiente1 = Mutacion(Descendiente1);
                    }


                    fit = Evaluacion(Descendiente1, datos);
                    Poblacion_descendientes.Add(new Cromosoma() { Fitness = fit,
Secuencia = Descendiente1 });

                    //Descendientes incluidos

                }

                //Teniendo la Poblacion Original y Poblacion --> Se genera
Población Conjunta y con ruleta generamos la nueva
                for (i=0;i<Poblacion.Count();i++)
                {
                    Poblacion_descendientes.Add(Poblacion[i]);
                }

                Poblacion.Clear();
                Poblacion_nueva.Clear();
                Ruleta.Clear();
                Ruleta = Wheel(Poblacion_descendientes, P);
```

```
                while (Poblacion_nueva.Count<P)
                {

                    eleccion = random.NextDouble();
                    int pos_individuo_seleccionado = 0
;
                    for (i = 0; i < Poblacion_descendientes.Count(); i++)
                    {

                        if (Ruleta[i] >= eleccion)
                        {
                            pos_individuo_seleccionado = i;
                            i = Poblacion_descendientes.Count();

                        }
                    }


Poblacion_nueva.Add(Poblacion_descendientes[pos_individuo_seleccionado]);
                }

                Poblacion_descendientes.Clear();
                iteraciones++;
                //----------------------------- FIN WHILE
            }

            Poblacion_nueva=Poblacion_nueva.OrderByDescending(x =>
x.Fitness).ToList();
            sw.Stop();

            //-------------------------------------------------------------------
---




            double solucion = Evaluacion_IG(Poblacion_nueva[0].Secuencia, datos);


            return (Poblacion_nueva[0].Secuencia, solucion);
        }



static public (List<int>, double) Optimizador_AG_Propuesto(ClaseDatosOpt datos,
double tiempo_limite_ejecucion)
        {

            //Declaración de los parámetros auxiliares
            int i, N = datos.NI, P = datos.NI/2;
            Stopwatch sw = new Stopwatch();

            //-------------------------------------------------------------------
-
            //BUCLE
            List<Cromosoma> Poblacion_nueva = new List<Cromosoma>();
            List<Cromosoma> Poblacion_descendientes = new List<Cromosoma>();
            List<Cromosoma> Poblacion = new List<Cromosoma>();
```

```
            sw.Start();
            //Población Inicial
            Poblacion_nueva = Poblacion_Inicial_Aleatoria(datos, N, P);
            //BUCLE
            double tiempo_maximo = tiempo_limite_ejecucion;
            int iteraciones = 0;
            double fit;

            int seed;
            var random = new Random();


            while (sw.Elapsed.TotalSeconds < tiempo_maximo)
            {

                seed = Aleatorio();
                random = new Random(seed);

                foreach (Cromosoma cromosoma in Poblacion_nueva)
                {
                    Poblacion.Add(cromosoma);
                }

                //Bucle de selección-cruce-mutación en la que cada pareja de
padres tendrá 1 hijo
                int pos_padre1 = 0, pos_padre2 = 0;
                double eleccion = new double();
                List<double> Ruleta = Wheel(Poblacion, P);


                while (Poblacion_descendientes.Count < P)
                {
                    List<int> Descendiente1 = new List<int>();


                    //Selección de individuos para que tengan descendientes
probabilidad de cruce =0.8
                    double pc = 0.9, probabilidad;
                    int activador = 0;
                    while (activador == 0)
                    {
                        eleccion = random.NextDouble();
                        for (i = 0; i < P; i++)
                        {
                            if (Ruleta[i] >= eleccion)
                            {
                                pos_padre1 = i;
                                i = P;
                            }
                        }

                        eleccion = random.NextDouble();
                        for (i = 0; i < P; i++)
                        {
                            if (Ruleta[i] >= eleccion)
                            {
                                pos_padre2 = i;
                                i = P;
```

```
                        }
                    }

                    probabilidad = random.NextDouble();
                    if (probabilidad <= pc)
                    {
                        activador = 1;
                    }

                }
                //Cruce
                (Descendiente1) = CruceRandom(Poblacion, pos_padre1,
pos_padre2, N, random);



                //Mutación con probabilidad
                double pm = 0.1;
                probabilidad = random.NextDouble();
                if (probabilidad <= pm)
                {
                    Descendiente1 = Mutacion(Descendiente1);
                }


                fit = Evaluacion(Descendiente1, datos);
                Poblacion_descendientes.Add(new Cromosoma() { Fitness = fit,
Secuencia = Descendiente1 });

                //Descendientes incluidos

            }

            //Teniendo la Poblacion Original y Poblacion --> Se genera
Población Conjunta y elegimos a los mejores
            Poblacion_nueva = Elitismo(Poblacion, Poblacion_descendientes,
N);

            Poblacion.Clear();
            Poblacion_nueva = Diversidad(Poblacion_nueva, N, datos);
            Poblacion_descendientes.Clear();

            iteraciones++;
            //----------------------------- FIN WHILE
        }

        Poblacion_nueva=Poblacion_nueva.OrderByDescending(x =>
x.Fitness).ToList();
        sw.Stop();

        //-------------------------------------------------------------------
---
        //Sacar por pantalla la mejor solución



        double solucion = Evaluacion_IG(Poblacion_nueva[0].Secuencia, datos);
```

```
            return (Poblacion_nueva[0].Secuencia, solucion);
        }
```

Declaración de Funciones Iterated Greedy

```
static public List<int> NEH2(ClaseDatosOpt datos)
        {
            int i, h;
            List<int> Auxiliar = new List<int>(); //Lista auxiliar para
introducir los trabajos que voy extrayendo
            List<int> pi = new List<int>();
            List<double> FO_NEH = new List<double>();
            int[] orden = new int[datos.NI];
            //FASE CONSTRUCTIVA


            for (i = 0; i < datos.NI; i++)
            {
                orden[i] = i + 1;

            }

            LibreriaClases_SDD_Urg.funciones_auxiliares.ordenar_creciente(orden,
datos.tp_i, datos.NI);

            for (i = 0; i < datos.NI; i++)
            {
                if (datos.p_i[orden[i] - 1] < 6)
                {
                    Auxiliar.Add(orden[i]);
                }
            }
            for (i = 0; i < datos.NI; i++)
            {
                if (datos.p_i[orden[i] - 1] == 6)
                {
                    Auxiliar.Add(orden[i]);
                }
            }


            ////
            //NEH

            for (i = 0; i < 1; i++)
            {
                pi.Insert(i, Auxiliar[0]);
                Auxiliar.RemoveAt(0);
            }
            while (Auxiliar.Count > 0)
            {

                FO_NEH.Clear();
```

```
                for (i = 0; i <= pi.Count; i++)
                {
                    pi.Insert(i, Auxiliar[0]);
                    double FO = Evaluacion_IG(pi, datos);
                    FO_NEH.Insert(i, FO);
                    pi.RemoveAt(i);
                }
                h = FO_NEH.IndexOf(FO_NEH.Min());
                pi.Insert(h, Auxiliar[0]);

                Auxiliar.RemoveAt(0);
            }

            return (pi);
        }



static public List<int> LS_Exh_intensiva(List<int> pi_prima, ClaseDatosOpt datos,
Stopwatch sw, double tiempo_maximo)
        {
            int i, j, k, mejora=1;

            int seed = Aleatorio();
            var random = new Random(seed);
            List<int> Auxiliar = new List<int>();
            List<int> Seleccion = new List<int>();
            List<double> FO_insertion = new List<double>();
            while (mejora == 1 & sw.Elapsed.TotalSeconds < tiempo_maximo)
            {
                mejora = 0;
                //Copio pi prima en copia y en el vector que me asegura que no
extraigo dos veces el mismo paciente
                for (i = 0; i < pi_prima.Count; i++)
                {
                    Seleccion.Add(pi_prima[i]);
                }
                //Comienzo la insercion de todos los trabajos en todas las
posiciones
                for (i = 0; i < pi_prima.Count; i++)
                {
                    if(sw.Elapsed.TotalSeconds < tiempo_maximo)
                    {
                        Auxiliar.Clear();
                        for (j = 0; j < pi_prima.Count; j++)
                        {
                            Auxiliar.Insert(j, pi_prima[j]);
                        }
                        //Evaluamos FO inicial
                        double FO_pi_prima = Evaluacion_IG(pi_prima, datos);


                        //Elijo el trabajo que voy a insertar
                        int pivote = random.Next(0, Seleccion.Count);
                        //Elimino el paciente de la lista selección y del
auxiliar donde voy a probar el insertion
                        Auxiliar.Remove(Seleccion[pivote]);
```

```
                        //Limpio el vector donde pondré los valores de FO
                        FO_insertion.Clear();

                        //Comenzamos la insercion
                        for (k = 0; k < pi_prima.Count; k++)
                        {
                            Auxiliar.Insert(k, Seleccion[pivote]);

                            double FO_vecino = Evaluacion_IG(Auxiliar, datos);
                            FO_insertion.Insert(k, FO_vecino);
                            Auxiliar.Remove(Seleccion[pivote]);

                        }
                        int h = FO_insertion.IndexOf(FO_insertion.Min());
                        Auxiliar.Insert(h, Seleccion[pivote]);
                        Seleccion.Remove(Seleccion[pivote]);

                        if (FO_insertion[h] < FO_pi_prima)
                        {
                            mejora = 1;
                            pi_prima.Clear();
                            for (int w = 0; w < datos.NI; w++)
                            {
                                pi_prima.Add(Auxiliar[w]);
                            }
                        }
                    }


                }
            }

            return pi_prima;
        }



static public double Evaluacion_IG(List<int> secuencia, ClaseDatosOpt datos)
        {
            double FO;
            int[] planned = new int[datos.NI];
            int[] check_planned = new int[datos.NI];

            ClaseSolucionOpt solucion_heuristica;

            (solucion_heuristica, FO) = construir_solucion(secuencia.ToArray(),
datos, planned, check_planned);
            return (FO);
        }



static public List<int> LS_acotado_swap(List<int> pi_prima, ClaseDatosOpt datos,
Stopwatch sw, double tiempo_maximo)
        {
            int i, j, a = N/12, b = N/12, max_cambio, min_cambio, h, swap_def1=0,
swap_def2=0 ;
            double FO_pi_prima, FO_vecino, mejor_FO;
```

```
            int seed = Aleatorio();
            var random = new Random(seed);
            List<int> Auxiliar = new List<int>();

            //Hacer todos los vecinos con swap entre a y b elegidos de un punto
aleatorio
            h= random.Next(0, pi_prima.Count);
            if (h - a >= 0)
            {
                min_cambio = h - a;
            }
            else
            {
                min_cambio = 0;
            }

            if (h + b <= pi_prima.Count)
            {
                max_cambio = h + b;
            }
            else
            {
                max_cambio = pi_prima.Count;
            }

            // Tras decidir el rango de pacientes, hacer el swap con todos los de
dentro del rango
            Auxiliar.Clear();
            for (j = 0; j < pi_prima.Count; j++)  //Copio la secuencia sobre la
que voy a hacer el swap
            {
                Auxiliar.Insert(j, pi_prima[j]);
            }
            FO_pi_prima = Evaluacion_IG(pi_prima, datos);
            mejor_FO = FO_pi_prima;

            for ( i=min_cambio;i<max_cambio-1;i++)
            {
                for(int k=min_cambio+1;k<max_cambio;k++)
                {
                    if(sw.Elapsed.TotalSeconds<tiempo_maximo)
                    {
                        //Hago el swap
                        Auxiliar[i] = pi_prima[k];
                        Auxiliar[k] = pi_prima[i];

                        FO_vecino = Evaluacion_IG(Auxiliar, datos);

                        //Deshago el swap
                        Auxiliar[i] = pi_prima[i];
                        Auxiliar[k] = pi_prima[k];

                        if (mejor_FO > FO_vecino)
                        {
                            swap_def1 = i;
                            swap_def2 = k;
                            mejor_FO = FO_vecino;
                        }
                    }
```

```
            }
        }
        pi_prima[swap_def2] = Auxiliar[swap_def1];
        pi_prima[swap_def1] = Auxiliar[swap_def2];

        return pi_prima;
    }
```

Funciones Principales Iterated Greedy

```
static public (List<int>, double) IG_LS_SA_Original(Stopwatch sw, ClaseDatosOpt
datos, double tiempo_limite_ejecucion)
    {
        //Parámetros del Algoritmo
        double tiempo_maximo = tiempo_limite_ejecucion;

        int suma_tiempos_proceso=0;
        for (int l=0;l<datos.NI;l++)
        {
            suma_tiempos_proceso = suma_tiempos_proceso + datos.tp_i[l];
        }
        double T = 0.4 * suma_tiempos_proceso/(datos.NI *10*20);
        double alpha = 1;
        //Otras inicializaciones
        int contador_no_mejora = 0, i;
        double FO_pi_prima_prima;
        double FO_pi;
        double FO_pi_optima;
        List<int> pi_prima = new List<int>();
        List<int> pi_prima_prima = new List<int>();
        List<int> pi_optima = new List<int>();

        int seed = Aleatorio();
        var random = new Random(seed);


        //Inicio IG_puro

        List<int> pi = NEH2(datos);
        pi = LS_Exh_intensiva(pi, datos, sw, tiempo_limite_ejecucion);
        FO_pi = Evaluacion_IG(pi, datos);


        for (i=0;i<datos.NI;i++)
        {
            pi_optima.Add(pi[i]);
        }

        FO_pi_optima=FO_pi;
        //Bucle

        while (sw.Elapsed.TotalSeconds < tiempo_maximo & contador_no_mejora <
maximo_sin_mejora)
        {
```

```
                T = alpha * T;
                pi_prima.Clear();
                for (i = 0; i < pi.Count; i++)
                {
                    pi_prima.Add(pi[i]);
                }

                pi_prima = IG(pi_prima, datos);

                if (pi_prima.Count > 1)
                {
                    pi_prima_prima = LS_Exh_intensiva(pi_prima, datos, sw,
tiempo_limite_ejecucion);
                }
                FO_pi_prima_prima = Evaluacion_IG(pi_prima_prima, datos);

                //Comparación y SA
                if (FO_pi_prima_prima < FO_pi)
                {
                    contador_no_mejora = 0;
                    for (i = 0; i < pi_prima_prima.Count; i++)
                    {
                        pi[i] = pi_prima_prima[i];
                    }

                    if (FO_pi_prima_prima < FO_pi_optima)
                    {
                        for (i = 0; i < pi_prima_prima.Count; i++)
                        {
                            pi_optima[i] = pi_prima_prima[i];
                        }
                        FO_pi_optima = FO_pi_prima_prima;
                    }
                    FO_pi = FO_pi_prima_prima;
                }
                else
                {
                    //SA
                    contador_no_mejora++;
                    if (random.Next(seed) < Math.Exp(-(FO_pi_prima_prima - FO_pi)
/ T))
                    {
                        for (i = 0; i < pi_prima_prima.Count; i++)
                        {
                            pi[i] = pi_prima_prima[i];
                        }
                        FO_pi = FO_pi_prima_prima;
                    }
                }
            }

            return (pi_optima, FO_pi_optima);
        }
```

```
static public (List<int>, double) IG_LS_SA_Propuesto (Stopwatch sw, ClaseDatosOpt
datos, double tiempo_limite_ejecucion)
        {
            //Parámetros del Algoritmo
            double tiempo_maximo = tiempo_limite_ejecucion;
            int maximo_sin_mejora = 99999999;
            int suma_tiempos_proceso = 0;
            for (int l = 0; l < datos.NI; l++)
            {
                suma_tiempos_proceso = suma_tiempos_proceso + datos.tp_i[l];
            }
            double T = 0.8 * suma_tiempos_proceso / (datos.NI * 10 * 20);
            double alpha = 0.99;
            //Otras inicializaciones
            int contador_no_mejora = 0, i;
            double FO_pi_prima_prima;
            double FO_pi;
            double FO_pi_optima;
            List<int> pi_prima = new List<int>();
            List<int> pi_prima_prima = new List<int>();
            List<int> pi_optima = new List<int>();

            int seed = Aleatorio();
            var random = new Random(seed);


            //Inicio IG_puro

            List<int> pi = NEH2(datos);
            pi = LS_acotado_swap(pi, datos, sw, tiempo_limite_ejecucion);
            FO_pi = Evaluacion_IG(pi, datos);


            for (i = 0; i < datos.NI; i++)
            {
                pi_optima.Add(pi[i]);
            }

            FO_pi_optima = FO_pi;
            //Bucle

            while (sw.Elapsed.TotalSeconds < tiempo_maximo & contador_no_mejora <
maximo_sin_mejora)
            {
                T = alpha * T;
                pi_prima.Clear();
                for (i = 0; i < pi.Count; i++)
                {
                    pi_prima.Add(pi[i]);
                }

                pi_prima = IG2(pi_prima, datos);

                if (pi_prima.Count > 1)
                {
```

```
                    pi_prima_prima = LS_acotado_swap(pi_prima, datos, sw,
tiempo_limite_ejecucion);
                }
                FO_pi_prima_prima = Evaluacion_IG(pi_prima_prima, datos);

                //Comparación y SA
                if (FO_pi_prima_prima < FO_pi)
                {
                    contador_no_mejora = 0;
                    for (i = 0; i < pi_prima_prima.Count; i++)
                    {
                        pi[i] = pi_prima_prima[i];
                    }

                    if (FO_pi_prima_prima < FO_pi_optima)
                    {
                        for (i = 0; i < pi_prima_prima.Count; i++)
                        {
                            pi_optima[i] = pi_prima_prima[i];
                        }
                        FO_pi_optima = FO_pi_prima_prima;
                    }
                    FO_pi = FO_pi_prima_prima;
                }
                else
                {
                    //SA
                    contador_no_mejora++;
                    if (random.Next(seed) < Math.Exp(-(FO_pi_prima_prima - FO_pi)
/ T))
                    {
                        for (i = 0; i < pi_prima_prima.Count; i++)
                        {
                            pi[i] = pi_prima_prima[i];
                        }
                        FO_pi = FO_pi_prima_prima;
                    }
                }
            }

            return (pi_optima, FO_pi_optima);
        }
```

Código de la llamada principal

```
            for (int replicas=0;replicas<120;replicas++)
            {
                int activator = 1;

                if(activator==1)
                {
                    datos =
ModuloOrquestador.PreparacionDatosOpt.ConstruccionDatos(datos);
```

```
                    double[] tiempo_lim = new double[] {datos.NI* datos.NI
*35/2000};
                    for(int a=0;a<tiempo_lim.Length;a++)
                    {
                        //Inicializaciones
                        datos =
ModuloOrquestador.PreparacionDatosOpt.ConstruccionDatos(datos);
                        int[] planned = new int[datos.NI];
                        int[] check_planned = new int[datos.NI];
                        int N = datos.NI;
                        int metodo = 99;
                        double tiempoMilisegundos, tiempo, FO_solucion;


//*********************************************************************

//*********************************************************************


                        //ITERATED GREEDY
                        metodo = 0;
                        if (metodo == 0)
                        {
                            creacionFicherosCSV_ARPD_sinN(out FicheroCSV1);
                            List<int> solucion = new List<int>();
                            reloj.Restart();
                            (solucion, FO_solucion) = IG_LS_SA_Original(reloj,
datos, tiempo_lim[a]);

                            tiempoMilisegundos = reloj.ElapsedMilliseconds;
                            tiempo = tiempoMilisegundos / 1000;

                            FicheroCSV1.Write(replicas + ";" + "IG_Original" +
";" + datos.NI_real + ";" + datos.NI + ";" + FO_solucion + ";" + tiempo + ";");
                            FicheroCSV1.Write("\n");
                            FicheroCSV1.Close();
                        }

                        metodo = 1;
                        if (metodo == 1)
                        {
                            creacionFicherosCSV_ARPD_sinN(out FicheroCSV1);
                            List<int> solucion = new List<int>();
                            reloj.Restart();
                            (solucion, FO_solucion) = IG_LS_SA_Propuesto(reloj,
datos, tiempo_lim[a]);

                            tiempoMilisegundos = reloj.ElapsedMilliseconds;
                            tiempo = tiempoMilisegundos / 1000;

                            FicheroCSV1.Write(replicas + ";" + "IG_Propuesto" +
";" + datos.NI_real + ";" + solucion.Count + ";" + FO_solucion + ";" + tiempo +
";" );
                            FicheroCSV1.Write("\n");
                            FicheroCSV1.Close();
                        }
```

```
//*********************************************************************

//*********************************************************************
                        //ALGORITMO GENÉTICO

                        metodo = 2;
                        if (metodo == 2)
                        {

                            creacionFicherosCSV_ARPD_sinN(out FicheroCSV1);
                            List<int> solucion = new List<int>();
                            reloj.Restart();
                            (solucion, FO_solucion) =
Optimizador_AG_Azadeh(datos, tiempo_lim[a]);

                            tiempoMilisegundos = reloj.ElapsedMilliseconds;
                            tiempo = tiempoMilisegundos / 1000;

                            FicheroCSV1.Write(replicas + ";" + "AG_Original" +
";" + datos.NI_real + ";" + solucion.Count + ";" + FO_solucion + ";" + tiempo +
";" );
                            FicheroCSV1.Write("\n");
                            FicheroCSV1.Close();
                        }



                        metodo = 3;
                        if (metodo == 3)
                        {

                            creacionFicherosCSV_ARPD_sinN(out FicheroCSV1);
                            List<int> solucion = new List<int>();
                            reloj.Restart();
                            (solucion, FO_solucion) =
Optimizador_AG_Propuesto(datos, tiempo_lim[a]);

                            tiempoMilisegundos = reloj.ElapsedMilliseconds;
                            tiempo = tiempoMilisegundos / 1000;

                            FicheroCSV1.Write(replicas + ";" + "AG_Propuesto" +
";" + datos.NI_real + ";" + solucion.Count + ";" + FO_solucion + ";" + tiempo +
";");
                            FicheroCSV1.Write("\n");
                            FicheroCSV1.Close();
                        }

                    }
                }


//*********************************************************************

//*********************************************************************
            }
```