Universidad de Valladolid

FACULTAD DE CIENCIAS

TRABAJO DE FIN DE GRADO

Grado en Estadística

Algoritmos de optimización para la gestión de horarios en el sector turístico

Autora: Silvia García Lara

Tutores: Juan Camilo Yepes Borrero y Mario Villaizán Vallelado

Año 2025

Agradecimientos

Quiero agradecer a mis tutores, Juan Camilo Yepes y Mario Villaizán, por su orientación y apoyo a lo largo de la realización de este trabajo.

A mi familia, por estar siempre. En especial, a mi madre, por su apoyo incondicional y su confianza en mí, y a mi padre, por el ejemplo y los valores que me ha transmitido.

A mi pareja, por su paciencia, su ánimo y apoyo constante en cada etapa de este camino.

Resumen

Este trabajo presenta un estudio sobre la planificación de turnos de personal en hoteles mediante el uso de algoritmos de optimización. Se plantean tres enfoques: un modelo exacto basado en Programación Lineal Entera Mixta (MILP), una heurística constructiva y un algoritmo genético, con el objetivo de asignar turnos de forma eficiente, teniendo en cuenta restricciones operativas y las coberturas mínimas.

Se simulan diferentes escenarios mensuales que varían en función de la ocupación del hotel y el número de habitaciones, permitiendo evaluar el comportamiento y la escalabilidad de los métodos utilizados en situaciones de distinta complejidad. El trabajo se ha desarrollado en Python, utilizando las bibliotecas específicas para la implementación de cada método.

Los resultados obtenidos permiten comparar los enfoques teniendo en cuenta la calidad de las soluciones, el tiempo de resolución y el cumplimiento de las restricciones. Este estudio quiere construir herramientas que permitan gestionar los recursos humanos del sector hotelero, donde una planificación adecuada de los turnos es clave para garantizar tanto la eficiencia operativa como el bienestar del personal.

Palabras clave: planificación de turnos, optimización, modelo exacto, heurística constructiva, metaheurística, algoritmo genético, gestión hotelera

Abstract

This research presents a study on staff shift scheduling in hotels using optimization algorithms. Three approaches are proposed: an exact model based on Mixed-Integer Linear Programming (MILP), a constructive heuristic, and a genetic algorithm, with the aim of assigning shifts efficiently while considering operational constraints and minimum coverage requirements.

Several monthly scenarios are simulated, varying in terms of hotel occupancy and the number of rooms, allowing for the evaluation of the behavior and scalability of the different methods under varying levels of complexity. The work has been developed in Python, using specific libraries for the implementation of each method.

The results obtained enable the comparison of the approaches in terms of solution quality, computation time, and constraint satisfaction. This study aims to develop tools for human resource management in the hotel sector, where proper shift planning is essential to ensure both operational efficiency and staff well-being.

Keywords: shift scheduling, optimization, exact model, constructive heuristic, metaheuristic, genetic algorithm, hotel management

Índice

Índice de figuras

Índice de tablas

1. Introducción

Según datos del Instituto Nacional de Estadística (INE), la actividad turística representó el 12,3 % del PIB español de 2023, demostrando un papel central del sector en la economía del país. Partiendo de esta base, es interesante analizar la encuesta de ocupación hotelera realizada mensualmente por el INE. A continuación, en la tabla 1.1 se presentan los resultados de la encuesta para los seis últimos meses de 2024. Los datos muestran, por un lado, el número de viajeros por cada tipo de alojamiento, y por otro, la cantidad de noches que los viajeros han pasado en el alojamiento, dicho mes (pernoctaciones).

Alojamiento / Mes	**2024M12**	**2024M11**	**2024M10**	**2024M09**	**2024M08**	**2024M07**	**2024M06**
Viajero							
Hoteles	6 553 489	7 171 221	10 774 309	11 821 379	13 586,095	12 703 213	11 905 082
Campings	240 130	257 872	588 775	1 079 320	2 191 697	1 893 246	1 270 158
Apartamentos turísticos	820 381	804 492	1 133 798	1 419 906	1 949 539	1 749 930	1 490 981
Alojamientos de turismo rural	321 628	318 213	424 160	482 574	687 907	559 284	456 854
Albergues	186 947	209 002	405 344	529 722	589 739	545 540	552 739
Pernoctaciones							
Hoteles	18 033 188	20 055 197	33 942 709	39 040 928	47 895 779	44 038 012	38 226 012
Campings	1 510 465	1 484 263	2 675 268	4 893 930	11 574 559	9 480 659	5 171 253
Apartamentos turísticos	4 134 534	4 178 623	5 693 939	7 036 788	11 091 467	9 297 058	6 934 829
Alojamientos de turismo rural	832 678	712 612	990 897	1 223 901	2 408 872	1 728 726	1 118 696
Albergues	389 002	441 555	729 963	843 988	1 124 287	1 197 186	953 882

Tabla 1.1: Viajeros, pernoctaciones por tipo de alojamiento (INE)

Para todos los meses mostrados, los hoteles han sido el alojamiento más demandado sin lugar a dudas. El mes de agosto es, entre estos, el mes en el que mayor afluencia han tenido los alojamientos. En dicho mes en concreto, hay más de 13 millones de personas que residen en un hotel frente a 5 millones que deciden alojarse en el resto de alojamientos. Además, la cantidad de pernoctaciones en hoteles supera en más de 30 millones al segundo alojamiento en la lista, los campings. Estos datos muestran la importancia de una gestión eficiente de los recursos humanos en los establecimientos hoteleros para garantizar un servicio de calidad y optimizar la experiencia de los visitantes.

Una planificación inadecuada de los turnos puede tener consecuencias muy negativas, tanto para la empresa como para sus clientes y empleados. En primer lugar, durante los picos de demanda, la falta de previsión suele llevar a una sobrecarga de trabajo y a la necesidad de realizar horas extra o contrataciones de última hora. Al mismo tiempo, los turnos largos y los cambios de turno injustos incrementan el riesgo de incumplir los descansos mínimos legales. El resultado, por lo tanto, es el mal ambiente de trabajo e insatisfacción de los trabajadores. En meses de baja ocupación, una plantilla con demasiados empleados

genera costes salariales innecesarios y muchas horas no productivas. Este desajuste no solo perjudica a la rentabilidad, sino que, además, la injusta asignación de turnos y fines de semana libres suele traducirse en un ambiente laboral negativo, elevando el estrés y aumentando la rotación de personal. Por otra parte, la calidad del servicio se ve directamente afectada por la satisfacción de los empleados; estos se fatigan o desmotivan, lo que les lleva a cometer más errores y ofrecer una mala experiencia a los clientes. En un mercado tan exigente como el turístico, esto se refleja rápidamente en plataformas de opiniones online, dañando la reputación del establecimiento y reduciendo las futuras reservas. Por eso, disponer de un modelo que ajuste de forma óptima la planificación de turnos no es solo una mejora operativa, también consigue equilibrar costes y calidad, para garantizar a la vez la satisfacción del cliente y el bienestar del empleado.

La optimización tiene como objetivo encontrar la mejor solución dentro de un conjunto de soluciones factibles, buscando maximizar o minimizar una función objetivo mientras se respetan las restricciones del problema. Esta técnica se aplica en numerosos ámbitos, como la planificación de recursos, la logística o la economía, entre otros. Los problemas de optimización se pueden clasificar en diferentes categorías, dependiendo de las características de la función objetivo, las variables de decisión y las restricciones. Si tanto la función objetivo como las restricciones son expresiones lineales de las variables de decisión, se trata de un problema de programación lineal. En aquellos problemas en los que algunas variables son enteras y otras pueden ser continuas, hablamos de MILP. Un caso particular de estos son los problemas en los que las variables de decisión solo toman valores enteros; estos se denominan Programación Lineal Entera (ILP). Por último, cuando la función objetivo o alguna de las restricciones no es lineal, el problema se clasifica como programación no lineal. La elección del tipo de optimización depende de la naturaleza del problema. En el caso de la gestión de horarios en el sector turístico, se han tratado todas las variables como variables discretas (asignación de turnos); por tanto, el modelo más adecuado es el de ILP.

Se aplicarán tres enfoques diferentes para resolver el problema de optimización. En primer lugar, se utilizará un método exacto, que garantice encontrar la solución óptima, aunque con un coste computacional elevado. En segundo lugar, se implementará un algoritmo heurístico, que busca soluciones aproximadas y eficientes, sin la necesidad de explorar todas las posibles combinaciones. Finalmente, se empleará un método metaheurístico, que permitirá explorar grandes espacios de soluciones que permitan escapar de los óptimos locales. Antes de aplicar estos algoritmos, se realizará una revisión de los trabajos previos relacionados con la gestión de horarios, con el objetivo de conocer las soluciones que ya se han propuesto. Posteriormente, se procederá con la simulación de datos que replicarán la realidad de un hotel, lo que permitirá validar la efectividad de los

modelos propuestos. Una vez definidos los datos de entrada y la estructura del problema, se describirá el procedimiento seguido para aplicar el modelo exacto. A continuación, se presentará el modelo heurístico, y por último, se aplicará un método metaheurístico para resolver el problema de programación de horarios. Para terminar, se realizará una evaluación comparativa de los algoritmos implementados, analizando su desempeño y extrayendo conclusiones sobre el rendimiento de cada enfoque en función de la eficiencia operativa y la satisfacción de las restricciones impuestas.

1.1. Objetivos

Este trabajo se enfoca en la optimización de la gestión de horarios de los empleados en el sector hotelero. Para ello, se propondrán diversos algoritmos de optimización que se aplicarán a datos generados, con el objetivo de replicar el funcionamiento real de un hotel. Se considerarán aspectos como la existencia de múltiples puestos dentro del hotel con diferentes turnos, así como las restricciones legales que buscan garantizar el bienestar del personal. La gestión de horarios puede tratarse como un problema de *scheduling* o programación de tareas. Este tipo de problema consiste en asignar recursos limitados a tareas específicas, cumpliendo con una serie de restricciones de manera óptima. En este contexto, se considerará como recursos a los empleados del hotel y como tareas a las diferentes labores que deben realizar, como la limpieza de habitaciones o la atención en recepción, entre otras.

El trabajo se centra en los siguientes objetivos:

(O1) Formular el problema de planificación de horarios aplicado a un hotel con un modelo de optimización con restricciones

(O2) Implementar algoritmos de optimización para resolver dicho modelo sobre los datos generados que simulan el funcionamiento de un hotel.

(O3) Comparar el rendimiento de los distintos algoritmos empleados en cuanto a la calidad de la solución como el tiempo de cómputo necesario.

El objetivo principal es, por tanto, encontrar una asignación óptima de recursos para minimizar el costo operativo, respetando siempre las restricciones laborales.

1.2. Asignaturas relacionadas

El proyecto se apoya en los conceptos estudiados en diversas asignaturas a lo largo del grado, especialmente en aquellas relacionadas con la investigación operativa. En particular, en el primer curso la asignatura *Introducción a la Investigación Operativa* ha aportado los primeros conocimientos de formulación y resolución de problemas de programación lineal orientada a la modelización de problemas reales y manejo de programas de optimización. En el segundo curso, la asignatura de *Programación Entera* amplía estos conocimientos con la presentación de los problemas de programación lineal entera y flujo de redes. Por último, la asignatura *Modelos de Investigación Operativa* aporta la introducción a la resolución heurística que permite dar soluciones aproximadas. Estas materias han proporcionado las bases teóricas necesarias para la formulación del problema, la selección de los modelos adecuados y la implementación de los algoritmos de optimización empleados en este estudio. Por otra parte, también ha sido necesario aplicar conocimientos de informática y estadística, tanto para la codificación del modelo como para la comparación y extracción de los resultados obtenidos. Estos conocimientos han sido adquiridos en asignaturas como *Computación Estadística*, *Análisis de datos* o *Programación Orientada a Objetos*, entre otras.

2. Revisión bibliográfica

La programación de horarios con múltiples empleados y tareas, como en los hoteles, implica gestionar diversas restricciones y objetivos. A lo largo de los años, se han desarrollado distintos enfoques para abordar este problema, desde los métodos exactos, los métodos heurísticos y los metaheurísticos.

Uno de los enfoques más utilizados para resolver problemas de asignación de personal es el uso de métodos exactos. En el trabajo *A flexible model and a hybrid exact method for integrated employee timetabling and production scheduling* [2] los autores proponen un modelo híbrido que combina MILP con otras técnicas más flexibles para organizar tanto los turnos de los empleados como las tareas a realizar. Gracias a esto, se consigue una asignación ajustada a las restricciones del problema. Este tipo de enfoque es útil cuando se necesita garantizar una solución óptima, es decir, la mejor solución posible según el objetivo planteado. Sin embargo, su uso se ve limitado en problemas de gran escala debido al alto costo computacional. En concreto, para este problema el modelo se ha aplicado a tan solo 15 empleados, lo que indica que está pensado para problemas de pequeño tamaño. De manera similar, *Alain Billionnet* [3] planteó un modelo MILP para asignar personal con demanda variable a lo largo de una semana. En este trabajo se considera que los trabajadores más cualificados pueden cubrir tareas de menor nivel y garantiza que los trabajadores tengan los días de descanso semanales requeridos. Este estudio concluye que, a través de una formulación sencilla, se resuelve el problema en pocos segundos en ordenadores no especializados. El modelo reduce el número de asignaciones por debajo del nivel de cualificación y maximiza los días libres consecutivos, lo que mejora la calidad de la solución para el personal. Esto demuestra que los modelos exactos también pueden ser aplicados obteniendo buenos resultados si se modelan correctamente. Además, trabajos como *Workload balancing for the nurse scheduling problem: A real-world case study from a french hospital* [1] han aplicado modelos similares en problemas Problema de programación de turnos en enfermería (NSP) en hospitales mediante MILP, con el objetivo de mejorar la equidad en la distribución de tareas y el bienestar del personal. Este tipo de estrategias es perfectamente aplicable a la gestión hotelera, donde también es crucial mantener un reparto justo y eficiente de las tareas.

Por otro lado, a diferencia de los métodos exactos, los métodos heurísticos han demostrado ser muy efectivos cuando el objetivo es obtener soluciones en un tiempo razonable y con recursos limitados. En *A two-stage heuristic approach for nurse scheduling problem: A case study in an emergency department* [10] se propone aplicar una heurística en dos etapas para resolver el problema NSP en un servicio de urgencias. En la primera fase, cada turno de 8 horas se asigna a un enfermero que tenga la menor penalización por violar

restricciones duras. En la segunda fase, implementan una búsqueda local que refina el calendario minimizando el incumplimiento de las restricciones blandas (por ejemplo, evitar turnos nocturnos consecutivos o asignar días libres aislados). De este modo, la heurística se posiciona como una buena alternativa a los métodos exactos que se puede escalar a problemas grandes en hospitales donde la rapidez y la calidad de las asignaciones de turnos son cruciales.

Por último, dentro de los métodos metaheurísticos, los Algoritmos genéticos (GA) han sido ampliamente utilizados en la optimización de horarios. Los autores *S. Rajakumar, V. Arunachalam, y V. Selladurai* [8] demostraron la eficacia de estos algoritmos para equilibrar la carga de trabajo en sistemas de máquinas paralelas, permitiendo explorar soluciones en espacios de búsqueda con gran amplitud. Este enfoque también se ha aplicado con éxito en la programación de personal, ya que permite adaptar dinámicamente las asignaciones cuando las condiciones y restricciones cambian. En concreto, [9] plantean un enfoque híbrido en dos fases: primero, construyen un calendario inicial factible mediante un modelo de programación matemática que garantiza el cumplimiento de todas las restricciones duras (cubertura mínima de turnos, descansos entre turnos y carga máxima legal de horas), y a continuación implementan un algoritmo genético para conseguir mejorar la solución minimizando las penalizaciones por violaciones de restricciones blandas (preferencias individuales, evitar turnos consecutivos). Este enfoque logró mejorar a los métodos exactos en términos de calidad y rapidez.

En el trabajo *Personnel scheduling: A literature review* [6] , se analizaron cerca de 300 artículos de programación de personal, mostrando los avances que había hasta el momento. Llama la atención que la mayoría de los artículos revisados tratan el problema NSP. Por el contrario, apenas existen referencias a la aplicación de estas técnicas en el sector hotelero, a pesar de que la planificación de turnos es necesaria para garantizar tanto el servicio al cliente como el bienestar de los empleados. El problema NSP tiene similitudes con la planificación de turnos en el sector hotelero. En ambos casos, es necesario cubrir los turnos disponibles, garantizar los descansos mínimos entre jornadas, entre otras restricciones legales y organizativas. Sin embargo, también existen diferencias importantes. En el sector hotelero, los picos de demanda son mucho más notables, especialmente los fines de semana y temporadas vacacionales. Por tanto, el modelo de planificación requiere más capacidad de adaptación en este sentido. Es por esta razón que este trabajo propone un modelo adaptado específicamente al entorno hotelero que pueda cubrir este vacío de estudio.

3. Metodología

La hipótesis principal de este trabajo es que, frente a un problema de planificación de turnos complejo como el del personal hotelero, los algoritmos heurísticos y metaheurísticos permiten obtener soluciones de buena calidad con menores tiempos de ejecución respecto a un modelo exacto, especialmente a medida que crece el número de empleados del hotel.

Para validar esta hipótesis se han desarrollado e implementado en Python tres enfoques distintos: el modelo exacto está basado en programación lineal entera y resuelto con la biblioteca PuLP, que busca soluciones óptimas bajo todas las restricciones del problema. Para el algoritmo heurístico se elige un algoritmo *greedy*, debido a su simplicidad y eficiencia computacional, ya que consigue soluciones factibles en tiempos muy reducidos. Con él se pretende resolver el modelo minimizando las violaciones de restricciones. Además, se aplicará su versión con *multi-start*, que genera varias soluciones iniciales y aplica mejoras locales para garantizar la factibilidad. Su ventaja es la rapidez, aunque no garantiza la solución óptima. Como metaheurística, se ha elegido un algoritmo genético. Esta técnica, a diferencia de otras metaheurísticas, permite trabajar con muchas soluciones a la vez, combinarlas e ir mejorando iterativamente; gracias a ello, el algoritmo explora un espacio de soluciones muy amplio, de forma que no se estanca en óptimos locales. Cada uno de los métodos se evalúa sobre distintos escenarios tomando capacidades de hoteles de hasta 5000 habitaciones, comparando su rendimiento en términos de calidad, eficiencia computacional y cumplimiento de las restricciones.

3.1. Conjunto de datos

Los datos que se han utilizado a lo largo del trabajo son simulados pero tomando como base parámetros de hoteles reales, con el fin de evaluar el rendimiento y escalabilidad de los métodos de asignación de turnos. Determinamos la ocupación estimada mensual de habitaciones basándonos en los datos que proporciona el INE de la media de habitaciones ocupadas en los hoteles cada mes de 2024 en España. Para cada mes se multiplica el número total de habitaciones elegido por el porcentaje de ocupación dividido por cien, de modo que calculamos las habitaciones ocupadas de media diariamente para ese mes. Para calcular el número de empleados necesarios por área (limpieza, recepción, restauración y seguridad), se estima primero cuántas horas de trabajo requiere cada una al día. Para ello, multiplicamos el número de habitaciones ocupadas por un tiempo medio de servicio por habitación (0,5 horas en limpieza, 0,25 horas en recepción, 0,4 horas en restauración y 0,1 horas en seguridad). Después, dividimos ese total de horas entre la jornada laboral diaria

(8 horas) para obtener cuántos empleados hacen falta cada día en cada área. Se elige un 25 % extra de empleados para tener en cuenta ausencias imprevistas (bajas, vacaciones, etc.). Además, como mínimo debe haber al menos, según el área, dos empleados más que el número de turnos por día (por ejemplo, si seguridad opera en 3 turnos: 3 + 2 = 5 empleados como mínimo). Las tablas 3.1 y 3.2 muestran dos ejemplos de conjunto de empleados, la primera con un hotel de 150 habitaciones y la segunda con el número mínimo de empleados (0 habitaciones).

Mes	**Limpieza**	**Recepción**	**Seguridad**	**Restauración**
Enero	7	4	5	6
Febrero	8	4	5	7
Marzo	9	5	5	7
Abril	10	5	5	8
Mayo	10	5	5	8
Junio	9	5	5	8
Julio	9	5	5	7
Agosto	9	5	5	7
Septiembre	8	4	5	7
Octubre	8	4	5	7
Noviembre	8	4	5	6
Diciembre	7	4	5	6

Tabla 3.1: Distribución mensual de empleados por área - capacidad 150

Mes	**Limpieza**	**Recepción**	**Seguridad**	**Restauración**
Enero	2	4	5	4
Febrero	2	4	5	4
Marzo	2	4	5	4
Abril	2	4	5	4
Mayo	2	4	5	4
Junio	2	4	5	4
Julio	2	4	5	4
Agosto	2	4	5	4
Septiembre	2	4	5	4
Octubre	2	4	5	4
Noviembre	2	4	5	4
Diciembre	2	4	5	4

Tabla 3.2: Distribución mensual mínimos empleados por área

Se consideran tres turnos: el turno de mañana tiene un horario de 7:00 a 15:00, el turno de tarde es de 15:00 a 23:00 y el turno de noche de 23:00 a 7:00. Cada área de empleados opera en diferentes turnos según las necesidades del hotel. En este caso se determina que el equipo de limpieza trabaja solo en los turnos de mañana, los equipos de recepción y restauración trabajan en los turnos de mañana y tarde; por último, los empleados de seguridad trabajan en los turnos de mañana, tarde y noche para garantizar la vigilancia las 24 horas del día. En el caso de no estar asignado a un turno de trabajo, se asigna a 'Libre', que representa el día de descanso.

Los valores mostrados en la tabla 3.3 representan el coste operativo asociado a cubrir un turno de cada una de las áreas del hotel. Estos costes incluyen los salarios, así como otros gastos asociados a cada tipo de turno y equipamiento necesario. Los equipos con costes más altos están ligados a su mayor responsabilidad y a las condiciones laborales de cada equipo, así como a la variedad de turnos que deben cubrir. A mayor número de turnos realizados por un equipo, mayor es el coste de cada turno. Además, el turno de noche supone un incremento en el sueldo que se refleja en el coste operativo. Para evitar asignaciones incorrectas a turnos (por ejemplo, un turno de noche en el área de limpieza), se ha asignado un coste lo suficientemente alto que asegura que estos turnos nunca se escojan (en la tabla siguiente indicado con '-').

Área	**Mañana**	**Tarde**	**Noche**	**Libre**
Limpieza	50	-	-	0
Recepción	60	60	-	0
Restauración	65	65	-	0
Seguridad	70	70	80	0

Tabla 3.3: Coste por turno y área

4. Modelo exacto

Para resolver el problema dando una solución exacta, se plantea una resolución con MILP dando una asignación mensual de turnos para los empleados de un hotel. El modelo considera diversas áreas de trabajo, concretamente Recepción, Limpieza, Restauración y Seguridad. La asignación de turnos se realiza teniendo en cuenta restricciones laborales. El objetivo principal del modelo es minimizar el coste operativo del hotel de manera que se cumplan una serie de restricciones que garanticen el bienestar del personal.

Asignamos a cada empleado a un área de trabajo; en función de esta asignación, se definen los turnos en los que pueden realizar su labor. Para los empleados de Limpieza se considera únicamente el turno de mañana. Las áreas Recepción y Restauración operan en los turnos de la mañana y de la tarde. El área de Seguridad cuenta con un tercer turno de noche. El número total de empleados necesarios para cada mes se determina en función de la ocupación estimada de las habitaciones del hotel. Además, se asigna una proporción de empleados a cada área según los puestos que hay para cada trabajo en cada una de ellas.

4.1. Definición del modelo

A continuación se definen los parámetros y conjuntos que se utilizarán para modelar el problema.

- $D = \{1, \ldots, D\}$: conjunto de días en el mes elegido.
- E: conjunto de empleados.
- A: conjunto de áreas o departamentos del hotel.
- T: conjunto de posibles asignaciones o turnos (Mañana, Tarde, Noche y Libre).
- $a(e) \in A$: área asignada al empleado e.
- E_a: empleados que pertenecen al área a.
- $D_{\text{sem}} \subseteq D$: días entre semana (lunes a viernes).
- $D_{\text{fs}} \subseteq D$: días de fin de semana (sábado y domingo).
- $m_{a,t}^{sem}$: número mínimo de trabajadores necesarios en el área $a \in A$ y turno $t \in T_a$ durante los días de entre semana (lunes a viernes).

- $m_{a,t}^{fs}$: número mínimo de trabajadores necesarios en el área $a \in A$ y turno $t \in T_a$ durante los días de fin de semana (sábado y domingo)
- $\{W_s\}_{s=1}^{S}$: partición de D en $s \in S$ semanas de siete días.
- $C_{a,t}$: coste operativo del turno $t \in T$ en área $a \in A$ (Se usa como $C_{a(e),t}$ para obtener el coste en el área asignada al empleado e).

El subconjunto de turnos T_a se define como:

$$T_a = \begin{cases} \{\text{Mañana, Libre}\}, & \text{si } a = \text{Limpieza}, \\ \{\text{Mañana, Tarde, Libre}\}, & \text{si } a = \text{Recepción}, \\ \{\text{Mañana, Tarde, Libre}\}, & \text{si } a = \text{Restauración}, \\ \{\text{Mañana, Tarde, Noche, Libre}\}, & \text{si } a = \text{Seguridad} \end{cases}$$

4.1.1. Variables de decisión:

$$y_{e,d,t} = \begin{cases} 1, & \text{si el empleado } e \text{ trabaja el turno } t \text{ en el día } d, \\ 0, & \text{en otro caso,} \end{cases} \qquad e \in E,\ d \in D,\ t \in T_{a(e)}.$$

4.2. Formulación del problema

4.2.1. Resumen del modelo

$$\text{mín} \sum_{e \in E} \sum_{d=1}^{D} \sum_{t \in T_{a(e)}} \left(C_{a(e),t}\right) y_{e,d,t}. \tag{1}$$

$$\sum_{t \in T_{a(e)}} y_{e,d,t} = 1 \quad \forall e \in E,\ d = 1, \ldots, D. \tag{2}$$

$$y_{e,d,\text{Noche}} + y_{e,d+1,\text{Mañana}} \leq 1, \tag{3}$$

$$y_{e,d,\text{Tarde}} + y_{e,d+1,\text{Mañana}} \leq 1, \tag{4}$$

$$y_{e,d,\text{Noche}} + y_{e,d+1,\text{Tarde}} \leq 1, \quad \forall e \in E,\ d = 1,\ldots,D-1. \tag{5}$$

$$y_{e,d,\text{Noche}} + y_{e,d+1,\text{Libre}} + y_{e,d+2,\text{Mañana}} \leq 2, \tag{6}$$

$$y_{e,d,\text{Tarde}} + y_{e,d+1,\text{Libre}} + y_{e,d+2,\text{Mañana}} \leq 2, \tag{7}$$

$$y_{e,d,\text{Noche}} + y_{e,d+1,\text{Libre}} + y_{e,d+2,\text{Tarde}} \leq 2, \quad \forall e \in E,\ d = 1,\ldots,D-2. \tag{8}$$

$$\sum_{d=1}^{D} y_{e,d,\text{Libre}} \ \leq\ 9 \quad \forall e \in E. \tag{9}$$

$$\sum_{d \in W_s} y_{e,d,\text{Libre}} \ \geq\ 1 \quad \forall e \in E,\ \forall s. \tag{10}$$

$$\sum_{e \in E_a} y_{e,d,t} \ \geq\ m^{sem}_{a,t}, \quad \forall a,\ t \in T_a,\ d \in D_{sem}. \tag{11}$$

$$\sum_{e \in E_a} y_{e,d,t} \ \geq\ m^{fs}_{a,t}, \quad \forall a,\ t \in T_a,\ d \in\ D_{fs}. \tag{12}$$

$$\sum_{e \in E_a} y_{e,d,t} \ \geq\ 1, \quad \forall a \in A,\ t \in T_a,\ d \in D. \tag{13}$$

4.2.2. Función objetivo

La ecuación (1) es la función objetivo que minimiza el coste total, teniendo en cuenta los costes asociados a cada turno para el hotel en las diferentes áreas.

4.2.3. Explicación detallada de las restricciones

Para la resolución de este problema, los horarios del personal deberán cumplir con una serie de normas.

Primeramente, se considera que no se puede asignar más de un turno por día a cada trabajador, por lo que la primera restricción impone que cada empleado debe tener exactamente un único turno asignado por día, ya sea laboral o libre (2). Desde el punto de vista legal, los empleados deben tener un mínimo de 12 horas de separación entre el fin de un turno y el comienzo del siguiente. Si se asigna a un empleado el turno de noche en el día d, su jornada acabará a las 7:00 de la mañana del día siguiente (d+1), por lo que no podremos asignar en el día d+1 el turno de mañana ya que comenzaría a las 7:00 de la misma mañana (3). De igual modo, si se asignara un turno de tarde, comenzaría a las 15:00, lo que solo dejaría 8 horas de descanso entre turnos (4). Por último, si asignamos un turno de tarde que acaba a las 23:00, no podremos asignar un turno de mañana el día siguiente, ya que comienza a las 7:00, dejando tan solo 8 horas de descanso entre turnos (5). Por ello, para cumplir con el descanso mínimo entre turnos que es de 12 horas, se detectan los 3 turnos que no pueden asignarse de un día para otro. Si forzamos a que la suma de cada una de las transiciones prohibidas sea menor que 1, se consigue evitar que estos turnos se asignen seguidos. En concreto, para esta configuración, al asignar turnos de 8 horas, se consigue un descanso de 16 horas o más. Similarmente, cada semana completa del mes, todos los trabajadores deben tener al menos 1 día y medio de descanso, lo cual corresponde a un total de 36 horas seguidas sin trabajar. Para poder cumplir con esto, cada vez que se asigna un turno libre, se prohiben tres secuencias de turnos que corresponden con las restricciones (6), (7) y (8), forzando a que la suma de cada una de las secuencias prohibidas sea menor que 2, conseguimos evitar que estos tres turnos se asignen seguidos.

En el modelo planteado, los contratos de todos los trabajadores son a jornada completa, 8 horas al día y 40 horas semanales. Se establece un máximo de 9 días libres en un mes que, al no tener coste, se asignarán casi al completo (9). La restricción (10) complementa a la anterior asignando a cada empleado al menos un descanso en cada semana completa de lunes a domingo, garantizando una distribución homogénea de los 9 días libres mensuales y el descanso semanal mínimo de un día y medio. Para ello, determinamos qué día de la semana es el día 1 del mes, se calcula un índice de semana completa que agrupa las fechas en bloques de siete días consecutivos (lunes a domingo), y luego se cuenta cuántos turnos se han asignado a 'Libre' en esas semanas completas. Asimismo, cada turno y cada día deberá haber un mínimo de empleados en plantilla, teniendo en cuenta que los fines de semana la ocupación del hotel será superior que entre semana, por lo que la cobertura mínima de empleados deberá ser mayor.

Se considera que como mínimo el 25 % de los empleados debe estar trabajando de lunes a viernes en cada turno para cada una de las áreas (11). Es decir, si por ejemplo, hay 12 empleados de limpieza, deberá haber al menos 3 trabajadores cada día de lunes a viernes, ya que solo trabajan en el turno de mañana. En cambio, si hay 12 empleados de seguridad, 12 empleados / 3 turnos = 4 empleados. El 25 % de 4 es 1, por lo que al menos 1 empleado deberá trabajar en cada uno de los 3 turnos, Mañana, Tarde y Noche cada día de la semana. Definimos el número mínimo de trabajadores necesarios de lunes a viernes como:

$$m_{a,t}^{sem} = \left\lceil 0{,}25 \frac{|E_a|}{|T_a|} \right\rceil.$$

Los fines de semana la ocupación es la más alta, por lo que se prevé que, del total de empleados disponibles para cada turno, al menos el 50 % de los empleados deberán tener asignado un turno de trabajo el sábado y el domingo (12). Se tendrá en cuenta que el número total de empleados se debe de dividir entre el número de turnos para cada área al igual que en la restricción (11). Definimos el número mínimo de trabajadores necesarios los sábados y domingos como:

$$m_{a,t}^{fs} = \left\lceil 0{,}50 \frac{|E_a|}{|T_a|} \right\rceil.$$

Finalmente, nunca debe haber un turno sin empleados (13); para cada área se deberá asignar como mínimo a una persona, incluso en los casos en los que la ocupación es muy baja, donde el 25 % o 50 % de los empleados pueda ser menor a uno.

5. Modelo heurístico

Los algoritmos heurísticos se han ido desarrollando a lo largo de los años para evitar la dificultad computacional de la resolución exacta de problemas de optimización más complejos. Estos algoritmos pretenden resolver el problema de forma aproximada, proporcionando soluciones factibles que se acerquen lo suficiente al valor óptimo, empleando un tiempo razonable.

Las heurísticas se pueden clasificar en dos grandes grupos: las heurísticas constructivas y las heurísticas de mejora. Las heurísticas constructivas se encargan de construir una solución desde cero, mientras que las heurísticas de mejora parten de una solución inicial y la van refinando progresivamente. En este trabajo se ha elegido aplicar una heurística constructiva, ya que estas son muy útiles en problemas de optimización que requieren construir una solución desde cero de forma rápida. Los métodos constructivos que más se usan son los algoritmos *greedy* debido a su elevada eficiencia computacional.

5.1. Funcionamiento del Algoritmo *Greedy*

El nombre *greedy* viene del inglés, y significa 'voraz' o 'glotón'. Esta heurística le debe su nombre a la forma en la que encuentra la solución factible al problema. El algoritmo en cada iteración incorpora un elemento a la solución en función de la información disponible en ese momento, y una vez elegido el método no se replantea sacarlo. El funcionamiento de un algoritmo *greedy* puede resumirse en los siguientes pasos:

1. **Inicialización:** Se parte de un estado inicial normalmente vacío y se comienza a construir un conjunto de candidatos que podrían formar parte de la solución final.
2. **Selección:** En cada iteración, se evalúan todas las opciones disponibles en función de un criterio de evaluación. Se selecciona aquella opción que optimice dicho criterio en el momento.
3. **Actualización:** Se incorpora la opción seleccionada a la solución parcial y se actualiza el conjunto de candidatos, eliminando las alternativas que ya no son viables.
4. **Terminación:** El proceso se repite hasta que se cumple una condición de parada.

5.2. Implementación del algoritmo

El algoritmo recorre el mes día a día, en cada asignación se deciden los empleados que ocuparán cada turno siguiendo el criterio de minimizar el coste. La asignación resultante debe minimizar el incumplimiento de las restricciones propuestas en el capítulo 2. Para ello, se asignan primero los puestos mínimos y luego el resto de los turnos teniendo en cuenta los descansos y días libres requeridos. El algoritmo toma una semilla aleatoria que determina el orden de asignación; para reducir el sesgo que esto provoca, se emplea un enfoque *greedy multi-start*, que repite la ejecución de la heurística varias veces con semillas distintas priorizando las asignaciones que tengan cero violaciones de restricciones o, en caso de empate, la de menor coste. El bucle principal de asignación recorre del primero al último día del mes; para cada día, se identifica si es un día de fin de semana o si es un día de entre semana. Si es sábado o domingo, se aplicarán los requerimientos de fin de semana, y si es un día de lunes a viernes, los requerimientos de entre semana. Si el día actual es lunes, se ponen a cero los contadores de días libres en la semana y se calcula cuántos días faltan hasta el domingo de esa misma semana, con el objetivo de decidir más adelante si un empleado necesita uno de esos últimos días libres para cumplir con el mínimo semanal. A continuación, para cada área se ejecutan dos fases de asignación: la primera fase se encarga de cubrir con los requisitos mínimos y la segunda asigna los turnos restantes o días libres a los empleados que aún no tienen turno.

En la primera fase se determinan cuántos empleados hacen falta en cada turno según si es fin de semana o entre semana. Primero se toma la lista de posibles turnos para cada área y se ordena de tal forma que el turno de menor coste quede el primero. Se elige para cada tipo de turno (Mañana, Tarde, Noche) un punto de partida aleatorio para repartir la carga; de este modo, si un día un turno de mañana se asigna al empleado 3, al día siguiente el turno de mañana empieza a buscar a partir del empleado 5, evitando que siempre los mismos empleados queden al inicio. Finalmente, para cada turno que se debe cubrir, se seleccionan los empleados viables que cumplen con las restricciones. Todos los empleados de esa área que todavía no tengan un turno asignado en el día actual forman parte de la lista de candidatos inicial. Luego, se filtra, eliminando los candidatos que violen la regla de descanso de 12 horas, observando que el turno asignado el día anterior sea compatible con el turno que se va a asignar el día actual. De la misma manera, mirando los dos días anteriores se descartan los turnos que incumplan la restricción de 36 horas mínimas si hay un día de descanso de por medio. Si no quedan candidatos tras estos filtros, se relajan las reglas y se acepta cualquiera que aún esté libre hoy para cubrir el mínimo y dar una solución inicial, aunque no se cumplan inicialmente todas las restricciones. Para terminar, se elige uno de los candidatos y se asigna el turno. El resultado de esta fase es que para cada día y cada área estarán cubiertos todos los puestos mínimos de cada

turno con los empleados que cumplen mejor las condiciones locales y tienen menor coste. A continuación, para facilitar la comprensión, se muestra el pseudocódigo de la primera fase del algoritmo.

Algorithm 1 Heurístcia *greedy* – Fase 1: Cobertura de turnos mínimos diarios

```
1:  procedure GREEDYFASE1(m, a)
2:      Ini: generar calendario (m,a), h ← ∅
3:      for d = 1 to D do
4:          reqs ← if es_fin_semana(d) then min_fds else min_sem
5:          if d es lunes then
6:              for all área A do
7:                  for all empleado e do
8:                      libresSemana[A][e] ← 0
9:                  end for
10:             end for
11:         end if
12:         for all área A do
13:             Ordenar T_A por coste creciente
14:             for t ∈ T_A do
15:                 for i = 1 to reqs[t] do
16:                     C ← candidatos no asignados aún en d para área A
17:                     C ← candidatos filtrados por restricciones diarias
18:                     if C = ∅ then
19:                         C ← restaurar candidatos iniciales
20:                     end if
21:                     e ← seleccionar C
22:                     h[d][A][e] ← t
23:                     coste+ = coste_turno[A][t]
24:                 end for
25:             end for
26:         end for
27:     end for
28:     return (h, coste)
29: end procedure
```

Donde:

- m, a : mes y a del calendario

- D : días en (m,a)
- T_A : turnos del área A
- min_fds : cobertura mínima los sábados y domingos
- min_sem : cobertura mínima de lunes a viernes
- $r[t]$: empleados requeridos en el turno t como mínimo

Tras de haber reservado las plazas mínimas, se asignará un turno o día libre al resto de trabajadores que todavía no tienen turno. Para la asignación se siguen las reglas que priorizan evitar el incumplimiento de las restricciones. Si el empleado aún no ha acumulado 9 días libres en el mes y además lleva al menos 6 días consecutivos trabajando seguidamente, se fuerza a tener un día libre. De igual manera, si aún no tiene un día libre esa semana y estamos en el último día posible de la semana para dárselo (sin haber alcanzado 9 libres mensuales) también se fuerza la asignación de un día libre. En caso de que el empleado ya tenga al menos 1 libre en la semana o ya haya tenido 9 días libres en total, se intenta asignar un turno de trabajo (mañana, tarde o noche, según corresponda al área) que no incumpla las restricciones de 12 horas y 36 horas. El turno se asignará al primer candidato; si no existe ningún candidato que satisfaga esas restricciones, se le asigna el turno más barato, aun sabiendo que rompe alguna regla. En este caso, será una solución no factible que se corregirá en el futuro. En cualquier otro caso, se asigna 'Libre' directamente. Una vez obtenida la planificación, se calcula el coste total sumando el de las dos fases y se comprueba el cumplimiento de las restricciones. El pseudocódigo descrito a continuación muestra el algoritmo de la segunda fase explicada.

Algorithm 2 Heurística *greedy* - Fase 2: Asignación del resto de turnos

```
1:  procedure GREEDYFASE2(m,a)
2:      h, coste_1 ← GREEDYFASE1(m, a)
3:      for d = 1 to D do
4:          for all área A do
5:              for all empleado e ∉ h[d][A] do
6:                  if e necesita día libre then          ▷ por descanso semanal o mensual
7:                      t ← 'Libre'
8:                  else
9:                      t ← if turno valido then turno de menor coste else t ←'Libre'
10:                 end if
11:                 h[d][A][e] ← t
12:                 coste_2+ = coste_turno[A][t]
13:             end for
14:         end for
15:     end for
16:     v ← VERIFICARESTRICCIONES(h, m, a)
17:     coste_total = coste_2 + coste_1
18:     if n_viol > 0 then s =Infactible else s =Óptimo
19:     return (h, s, v, coste_total)
20: end procedure
```

Donde:

- m, a : mes y a del calendario
- D : días en (m,a)

5.3. *Greedy* aleatorizado con Multi-Start

La solución que nos proporciona el enfoque *greedy* puede incumplir alguna restricción. Por ello, se implementa un procedimiento multi-start que ejecuta la heurística que elige la solución con menos violaciones y el mínimo coste. Este enfoque requiere fijar un número de arranques que determina las veces que se ejecuta el algoritmo, para cada valor de semilla. Comparamos primero el número de violaciones priorizando a la solución con menos infracciones; en caso de empate, se elige a la que tenga menor coste; si hay empate tanto en infracciones como en coste, se elige la primera solución obtenida. De esta mane-

ra, el algoritmo no se queda estancado en óptimos locales que incumplen las restricciones. La implementación de este enfoque se muestra en el siguiente pseudocódigo.

Algorithm 3 Heurística *greedy Multi-Start*

```
1:  procedure GREEDYMULTISTART(m, a, I)
2:      h* ← null
3:      f* ← +∞
4:      v* ← +∞
5:      for i = 1 to I do
6:          h_i ← GREEDY(m, a, seed = i)
7:          (f_i, v_i) ← VERIFICARESTRICCIONES(h_i, m, a)
8:          if v_i < v* or (v_i = v* and f_i < f*) then
9:              h* ← h_i
10:             f* ← f_i
11:             v* ← v_i
12:         end if
13:     end for
14:     return (h*, f*, v*)
15: end procedure
```

Donde:

- m, a : mes y a del calendario
- I : número de intentos o múltiples arranques
- h_i : horario generado en el intento i
- h^* : mejor individuo global tras los I intentos
- $f(h)$: función objetivo que evalúa el coste de un horario h
- $v(h)$: número de violaciones de restricciones de un horario h
- $f_i \equiv f(h_i)$: coste del horario h_i
- $v_i \equiv v(h_i)$: violaciones del horario h_i

6. Modelo metaheurístico

Con la intención de llegar a una mejor solución óptima, se crearon las metaheurísticas; estos son algoritmos heurísticos que pretenden ir más allá explorando más posibilidades, saliendo así de los óptimos locales en los que se quedan atrapadas las heurísticas. Dentro de estos algoritmos, hay dos grandes grupos: los métodos basados en trayectorias y los métodos basados en población. Las metaheurísticas basadas en trayectorias mejoran una sola solución a la vez, mientras que las basadas en población trabajan con un conjunto de soluciones que mejoran juntas. Algunos algoritmos metaheurísticos basados en trayectorias pueden ser los métodos GRASP (Greedy Randomised Adaptive Search) o la búsqueda Tabú. En cuanto a los métodos basados en población, podemos encontrarnos, por ejemplo, con los algoritmos genéticos o con los problemas de colonias de hormigas (Ant Colony). Para la resolución de este problema se ha elegido utilizar un algoritmo basado en población, ya que explora diversas asignaciones de turnos simultáneamente. En concreto, tal y como se expuso en el capítulo de metodología, se aplicará un algoritmo genético ya que con su estructura sencilla consigue explorar un gran espacio de soluciones.

6.1. Funcionamiento del Algoritmo Genético

El algoritmo genético es una metaheurística que se basa en la evolución natural. Funciona escogiendo a los mejores individuos de una población y combinándolos para formar las generaciones precedentes siguiendo los siguientes pasos:

1. **Inicialización:** se escoge una población con N individuos, cada uno de los individuos es una solución posible generada normalmente a partir de una heurística que ya es una solución factible.

2. **Función de evaluación (fitness):** cada individuo es evaluado mediante una función que determina la calidad de la solución .

3. **Selección de padres:** se eligen dos individuos de la población para generar descendencia, es decir, a partir de ellos sale un tercer individuo que combina ambas soluciones para crear una mejor solución.

4. **Cruce (crossover):** dados dos padres y una probabilidad de cruce, se combinan sus cromosomas para crear hijos.

5. **Mutación:** a cada hijo se le aplica una mutación con una probabilidad dada, para introducir variación y evitar estancarse en óptimos locales.

6. **Reemplazo:** se decide que individuos forman la población en el siguiente ciclo.

7. **Terminación:** se repiten los pasos hasta que se cumple un criterio de terminación.

6.2. Implementación del algoritmo

La implementación se basa en una solución inicial obtenida con la heurística greedy. A partir de esta solución, se genera un conjunto de candidatos que forman una población inicial. En cada una de las generaciones, se seleccionan dos individuos y se cruzan, intercambiando días entre ambos horarios, con una probabilidad determinada. Después, se realiza una mutación que intercambia turnos entre empleados en un día aleatorio, con el fin de introducir variabilidad y explorar nuevas soluciones. La función de fitness evalúa la calidad del horario penalizando a aquellas soluciones que incumplen alguna de las restricciones con un alto coste. El algoritmo itera durante un número de generaciones dado y devuelve el mejor horario encontrado. A continuación, se detalla el pseudocódigo del algoritmo descrito:

Algorithm 4 Algoritmo Genético

procedure GENETICALGORITHM(S, m, a, N, G, p_c, p_m)
  Ini: generar población P de N soluciones a partir de S
  for $g = 1$ **to** G **do**
    $N \leftarrow \emptyset$
    while $|N| < N$ **do**
      seleccionar padres $p_1, p_2 \in P$
      $(h_1, h_2) \leftarrow$ **if** rand $< p_c$ **then** `cruce`(p_1, p_2) **else** (p_1, p_2)
      aplicar `mutación` a h_1 y h_2 con probabilidad p_m
      $N \leftarrow N \cup \{h_1, h_2\}$
    end while
    $P \leftarrow$ mejores N individuos de N según $f(h)$
  end for
  return mejor $h \in P$ según $f(h) = \text{coste}(h)$
end procedure

Donde:

- S: solución inicial obtenida con una heurística `greedy`
- m, a: mes y año del calendario

- N: tamaño de la población
- G: número de generaciones
- p_c: probabilidad de cruce
- p_m: probabilidad de mutación
- $f(h)$: función objetivo que evalúa el coste del horario h

7. Resultados

En este apartado se presentarán los resultados obtenidos a partir de los modelos desarrollados en los capítulos anteriores. A modo ilustrativo, se toma para el cálculo de las asignaciones un mes representativo y un hotel de 150 habitaciones. Esto nos permitirá observar con mayor detalle el cumplimiento de las restricciones y el valor de la función objetivo entre los distintos modelos. Posteriormente, se comprobará la validez de los métodos elegidos de manera global. Finalmente, se realizará una comparativa de resultados entre los métodos validados, con el objetivo de identificar cuál de ellos resulta más eficaz en términos de coste y tiempo de ejecución, en función del número de empleados del hotel. Además, se realizarán estimaciones puntuales y tests de significancia para analizar las diferencias entre los resultados de los distintos modelos.

7.1. Análisis inicial de los métodos

7.1.1. Asignación Modelo exacto

En primer lugar se muestran los resultados obtenidos al aplicar el modelo exacto definido en el Capítulo 4, tomando como ejemplo el mes de febrero de 2025. Tal y como se observa en la tabla 7.1, el coste mínimo obtenido es de 27 735 € y el tiempo de ejecución del problema ha sido un total de 0,3 segundos.

Mes	**Año**	**Tiempo (s)**	**Estado**	**Coste (€)**
Febrero	2025	0,3	Optimal	27 735,00

Tabla 7.1: Resultados Modelo Exacto - Febrero 2025

Para este ejemplo, se proporciona una tabla que muestra el número total de turnos asignados para cada empleado en cada área. La tabla consta de seis columnas: la primera indica el identificador del empleado, la segunda corresponde al área a la que pertenece, las 4 siguientes indican el número de asignaciones a cada tipo de turno a lo largo del mes (mañana, tarde, noche o día libre).

Empleado	Área	Mañana	Tarde	Noche	Libre
LIMP01	Limpieza	19	0	0	9
LIMP02	Limpieza	19	0	0	9
LIMP03	Limpieza	19	0	0	9
LIMP04	Limpieza	19	0	0	9
LIMP05	Limpieza	19	0	0	9
LIMP06	Limpieza	19	0	0	9
LIMP07	Limpieza	19	0	0	9
LIMP08	Limpieza	19	0	0	9
RECE01	Recepción	7	12	0	9
RECE02	Recepción	18	1	0	9
RECE03	Recepción	6	13	0	9
RECE04	Recepción	11	8	0	9
REST01	Restauración	4	15	0	9
REST02	Restauración	9	10	0	9
REST03	Restauración	16	3	0	9
REST04	Restauración	18	1	0	9
REST05	Restauración	0	19	0	9
REST06	Restauración	4	15	0	9
REST07	Restauración	11	8	0	9
SEGU01	Seguridad	5	9	5	9
SEGU02	Seguridad	10	8	1	9
SEGU03	Seguridad	9	1	9	9
SEGU04	Seguridad	1	15	3	9
SEGU05	Seguridad	8	1	10	9

Tabla 7.2: Resumen de asignaciones Modelo Exacto - Febrero 2025

Al analizar la tabla 7.2, vemos que todos los empleados tienen un total de 28 turnos asignados, lo que corresponde a un turno por día del mes de febrero de 2025 (2). De la misma manera, se observa el cumplimiento de la restricción (9) definida en el Capítulo 4, ya que ningún empleado excede los 9 días libres al mes; en este caso, todos disponen de exactamente 9 días libres asignados. Por otra parte, vemos cómo a cada empleado le asignan únicamente los turnos pertenecientes a su área. En concreto, vemos que los empleados de limpieza sólo realizan turnos de mañana; los de recepción y restauración, turnos de mañana y de tarde; y los empleados de seguridad pueden cubrir los turnos de mañana, tarde y de noche.

Adicionalmente, se realiza un diagrama de Gantt para cada área con el objetivo de visualizar de forma más clara las asignaciones de cada empleado. En cada diagrama, las columnas representan un día del mes, y cada fila corresponde a un empleado. Las casillas se colorean según el turno asignado. El color naranja indica un turno de mañana, el azul claro un turno de tarde y el gris un turno de noche; si la casilla está vacía (blanca) se ha asignado un día libre.

Figura 7.1: Planificación de turnos Modelo Exacto - Febrero 2025

En los horarios representados por los diagramas de Gantt, se puede observar que los días libres de cada empleado están bien distribuidos a lo largo del mes, lo cual se ha conseguido con la restricción que exige un día libre cada semana completa (10), evitando así demasiados días de descanso (o de trabajo) seguidos. Además, se cumplen las restricciones de descanso mínimo obligatorio de 12 horas entre turnos (3), (4) y (5), ya que no se

ha detectado ninguna asignación que implique un cambio directo de un turno de noche a mañana, de tarde a mañana o de noche a tarde. Del mismo modo, cada día libre asignado garantiza al menos 36 horas de descanso continuo ((6), (7) y (8)). También, se aprecia que en cada día del mes y para cada área se cumplen los mínimos requeridos, existiendo al menos un empleado asignado en cada uno de ellos (11), (12) y (13). Asimismo, se observa que la mayoría de los turnos de trabajo son asignados en los fines de semana del mes elegido (días 1, 2, 8, 9, 15, 16, 22 y 23), mientras que de lunes a viernes, el número de empleados con días libres aumenta.

7.1.2. Asignación heurística Greedy

De la misma manera que en el apartado anterior, se proporcionan los resultados obtenidos con la mejor solución encontrada por el algoritmo basado en la heurística greedy, descrito en la primera parte del Capítulo 5. Para el mes de febrero, el algoritmo devuelve un coste óptimo de 28 395 € sin violar ninguna de las restricciones y consigue hacerlo en un tiempo de ejecución de 0,005 segundos, lo que representa una gran mejora de eficiencia en comparación con el modelo exacto.

Mes	**Año**	**Algoritmo**	**Tiempo (s)**	**Estado**	**Coste (€)**	**Violaciones**
Febrero	2025	Greedy	0,005	Optimal	28 395,00	0

Tabla 7.3: Resultados Greedy - Febrero 2025

En la tabla 7.4 se muestra un resumen de los resultados obtenidos con la heurística, incluyendo las asignaciones de turnos por empleado y por área. A diferencia del modelo exacto, podemos ver en el resumen de asignaciones que no todos los empleados disponen de exactamente 9 días libres; también hay empleados con 7 u 8 días libres en el mes. No obstante, el modelo consigue satisfacer todas las restricciones, manteniendo la factibilidad de la solución.

Empleado	Área	Mañana	Tarde	Noche	Libre
Lim1	Limpieza	19	0	0	9
Lim2	Limpieza	19	0	0	9
Lim3	Limpieza	19	0	0	9
Lim4	Limpieza	19	0	0	9
Lim5	Limpieza	19	0	0	9
Lim6	Limpieza	19	0	0	9
Lim7	Limpieza	19	0	0	9
Lim8	Limpieza	19	0	0	9
Rec1	Recepción	3	16	0	9
Rec2	Recepción	9	12	0	7
Rec3	Recepción	8	12	0	8
Rec4	Recepción	16	3	0	9
Res1	Restauración	12	7	0	9
Res2	Restauración	13	6	0	9
Res3	Restauración	10	10	0	8
Res4	Restauración	11	8	0	9
Res5	Restauración	6	13	0	9
Res6	Restauración	16	4	0	8
Res7	Restauración	4	15	0	9
Seg1	Seguridad	10	1	10	7
Seg2	Seguridad	3	4	13	8
Seg3	Seguridad	6	2	12	8
Seg4	Seguridad	5	14	0	9
Seg5	Seguridad	6	13	0	9

Tabla 7.4: Resumen de asignaciones Heurística Greedy - Febrero 2025

A continuación, se presentan los resultados de la aplicación del algoritmo añadiendo el enfoque multi-start descrito en la segunda parte del Capítulo 5. Con el objetivo de analizar el impacto del número de arranques en la calidad de la solución, se ejecuta el algoritmo varias veces variando este parámetro. Seguidamente, se muestran los resultados con 30, 50, 100 y 500 ejecuciones, lo que permite comparar el comportamiento del algoritmo en cada caso, en términos de coste y tiempo de ejecución.

Mes	Año	Algoritmo	N starts	Tiempo (s)	Estado	Coste	Violaciones
Febrero	2025	Greedy	-	0,005	Optimal	28 395,00	0
Febrero	2025	Multi-Start	30	0,115	Optimal	28 425,00	0
Febrero	2025	Multi-Start	50	0,163	Optimal	28 395,00	0
Febrero	2025	Multi-Start	100	0,302	Optimal	28 335,00	0
Febrero	2025	Multi-Start	500	1,679	Optimal	28 260,00	0

Tabla 7.5: Resultados heurística greedy con multi-start - Febrero 2025

En la tabla 7.5 se muestran los resultados obtenidos para el mes de febrero. En este caso, la heurística greedy sin arranque consigue obtener un buen resultado con un coste de 28 395 € y un tiempo de ejecución casi imperceptible. Por otra parte, el enfoque multi-start solo consigue superar el coste de la heurística con 100 arranques, lo que implica un aumento considerable en el tiempo de ejecución, igualando al modelo exacto.

En el siguiente paso, se ejecuta el modelo para el mes de abril de 2025, con el objetivo de comprobar si se observa un comportamiento similar al obtenido en febrero.

Mes	Año	Algoritmo	N starts	Tiempo (s)	Estado	Coste	Violaciones
Abril	2025	Greedy	–	0,003	Infactible	36 070,00	2
Abril	2025	Multi-Start	30	0,135	Optimal	35 570,00	0
Abril	2025	Multi-Start	50	0,210	Optimal	35 570,00	0
Abril	2025	Multi-Start	100	0,428	Optimal	35 570,00	0
Abril	2025	Multi-Start	500	2,294	Optimal	35 485,00	0

Tabla 7.6: Resultados heurística greedy con multi-start - Febrero 2025

En este mes, el algoritmo greedy produce una solución no factible, con dos restricciones incumplidas. No obstante, con tan solo 30 arranques del enfoque multi-start, se obtiene una solución factible, sin violaciones , en menos de 0,15 segundos y con un coste de 35 570 €. A partir de este punto, únicamente se logra mejorar la solución utilizando 500 ejecuciones, lo que implica un tiempo de 2,295 segundos. Por tanto, se considera que la opción con 30 arranques es la más adecuada al eliminar las violaciones, manteniendo un tiempo de ejecución adecuado.

En la Figura 7.2 se muestran los diagramas de Gantt correspondientes a cada área con las asignaciones de turnos obtenidos para el mes de febrero mediante el enfoque multi-start con 50 arranques. Esta asignación es, en este caso, la misma obtenida por la heurística greedy.

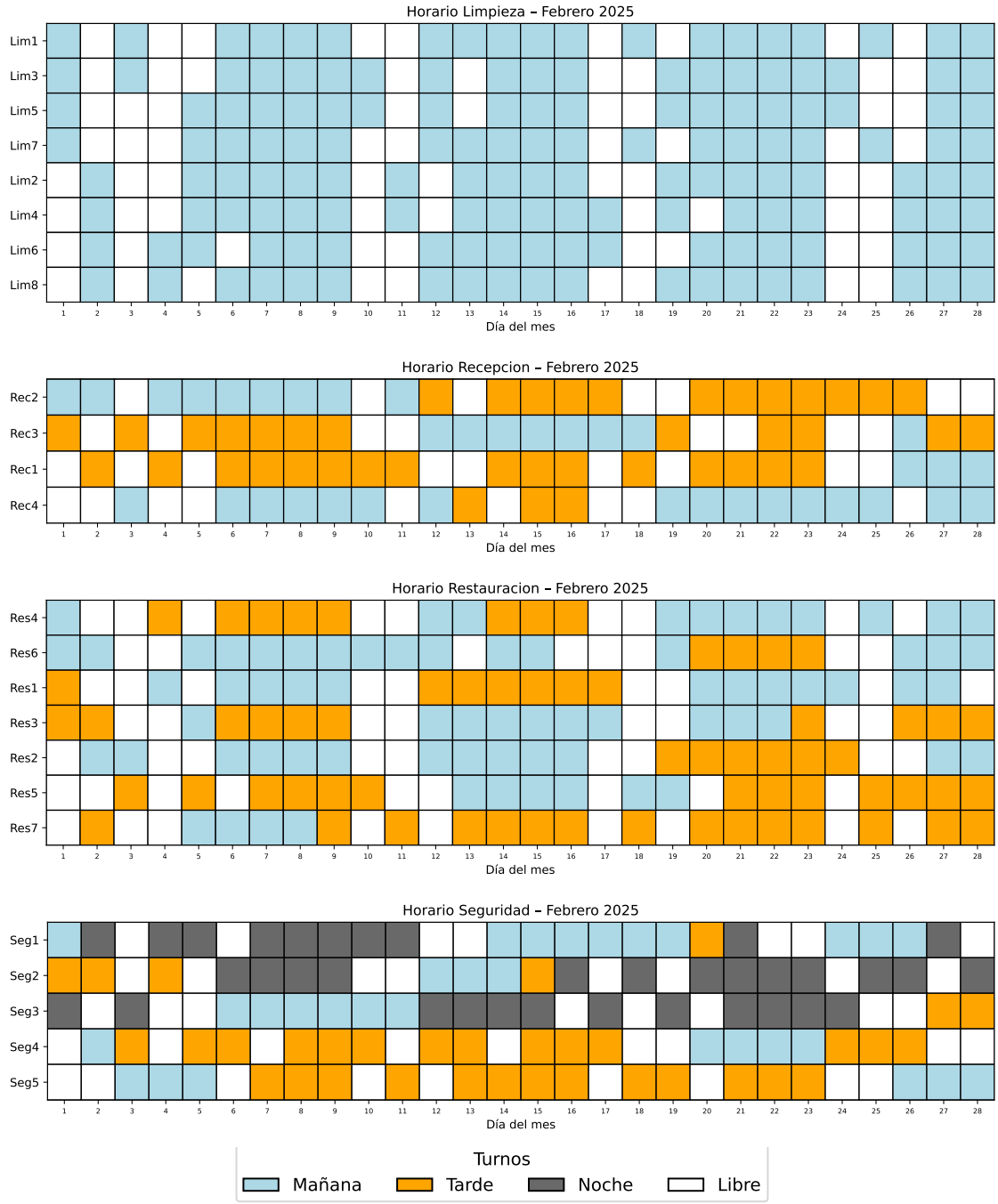

Figura 7.2: Planificación de turnos Greedy multi-start(50) - Febrero 2025

7.1.3. Asignación Algoritmo Genético

Continuando con el mes de abril, en el que se ha detectado la infactibilidad de la heurística greedy, se presentan los resultados obtenidos con el algoritmo genético, con

la intención de comprobar si consigue resolver dicha infactibilidad. Para ello, se toman los siguientes parámetros: una población de 50 individuos y 100 generaciones, con una probabilidad de cruce del 60 % y una probabilidad de mutación del 30 %. Tomando como solución inicial la heurística greedy, se muestran resultados en la siguiente tabla.

Parámetro	**Valor**
Mes	Abril
Año	2025
Algoritmo	Genético
Solución inicial	Greedy
Tamaño población	50
N generaciones	100
Prob. cruce	0,6
Prob. mutación	0,3
Tiempo (s)	31,8
Estado	Optimal
Coste (€)	35 815,00
Violaciones	0

Tabla 7.7: Resultados Algoritmo Genético (50, 100)

Aunque esta configuración obtiene una solución factible con un coste razonable, el tiempo de ejecución se dispara a 31,8 segundos. Por ello, se decide ejecutar el algoritmo con un número mucho menor tanto de individuos como de generaciones, para así poder comprobar si esto reduce el tiempo de ejecución sin perjudicar a la factibilidad.

En la tabla 7.8 podemos ver que con 5 individuos y 5 generaciones (manteniendo las mismas probabilidades de cruce y mutación), el algoritmo sigue aportando una solución factible, con un coste 5 € superior al obtenido con la configuración anterior. Además, el tiempo de ejecución mejora considerablemente a tan solo 0,2 segundos.

Parámetro	Valor
Mes	Abril
Año	2025
Algoritmo	Genético
Solución inicial	Greedy
Tamaño población	5
N generaciones	5
Prob. cruce	0,6
Prob. mutación	0,3
Tiempo (s)	0,2
Estado	Optimal
Coste (€)	35 820,00
Violaciones	0

Tabla 7.8: Resultados Algoritmo Genético (5,5)

La figura 7.3 muestra la asignación de turnos obtenida con el algoritmo genético configurado con 5 generaciones y 5 individuos para el mes de abril. En ella se visualiza claramente la distribución diaria de los turnos para cada empleado, destacando el cumplimiento de todas las restricciones impuestas en el Capítulo 4, tales como la asignación de días libres o descanso entre turnos y la correspondencia de turnos según el área de trabajo. Se observa una asignación similar a la obtenida con el algoritmo greedy, especialmente en la distribución de los días libres, que están más concentrados en los primeros días y menos repartidos a lo largo de las semanas. Esto es esperable, dado que el algoritmo genético parte de soluciones generadas por dicha heurística y las mejora mediante la mutación y el cruce de estas soluciones.

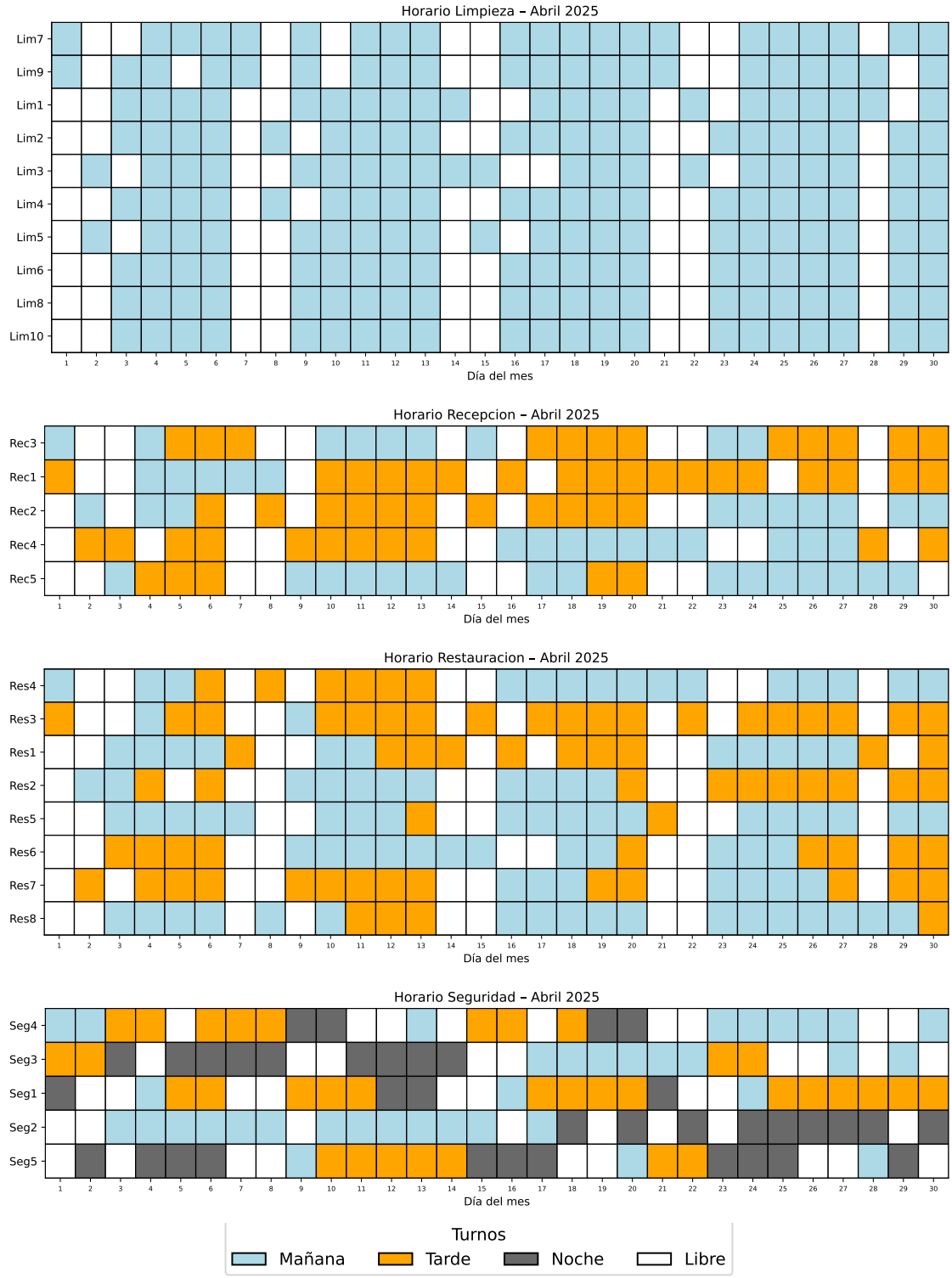

Figura 7.3: Planificación de turnos Algoritmo genético - Abril 2025

7.2. Validación de los métodos

Con el objetivo de verificar el cumplimiento de las restricciones definidas para todos los empleados, se ha implementado una función de validación. La imagen siguiente muestra de forma gráfica los resultados desde el año 2025 hasta el 2050 de los métodos tratados en este trabajo: el modelo exacto, la mejora greedy, greedy con multi-start, y por último el algoritmo genético, tomando como solución inicial el método greedy. El enfoque multi-start se ejecuta con 20 arranques y el algoritmo genético con 5 individuos, 5 generaciones y probabilidades de cruce y mutación de 0,6 y 0,3 respectivamente.

La figura 7.4 se interpreta de la siguiente manera: cada casilla grande representa un año completo y está dividida en doce subcasillas, una por cada ejecución de cada mes. Las subcasillas en verde indican que el método en cuestión cumple todas las restricciones para el mes dado, y si están en rojo, significa que se ha obtenido una solución no factible.

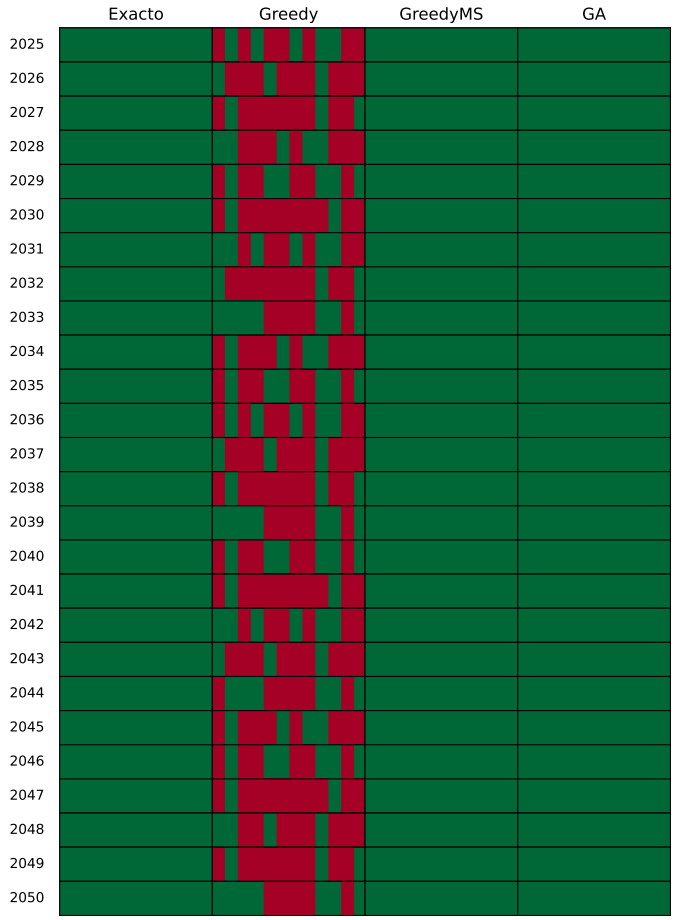

Figura 7.4: Pruebas de validación de resultados

En el gráfico se puede observar que la heurística greedy da soluciones no factibles para

la mayoría de los meses. En cambio, tanto la versión con multi-start como el algoritmo genético consiguen eliminar esas violaciones y garantizan siempre la factibilidad para los 25 años probados. Por lo tanto, consideramos válidos el método exacto, la heurística greedy con multi-start y el algoritmo genético.

7.3. Comparación de los modelos validados

Finalmente se ejecutan todos los modelos para cada mes del año 2025, comparando de coste y tiempo de ejecución entre los tres métodos validados: el modelo exacto, la heurística greedy con multi-start y el algoritmo genético.

Primero tomamos como ejemplo un hotel de 150 habitaciones, tal y como se ha desarrollado el trabajo hasta ahora. Ejecutamos los modelos con las mismas configuraciones empleadas en la validación de los resultados: 20 arranques para el enfoque multi-start y un algoritmo genético con 5 individuos, 5 generaciones, probabilidad de cruce del 60 % y probabilidad de mutación del 30 %.

Mes	Exacto Coste	Exacto Tiempo (s)	GreedyMS Coste	GreedyMS Tiempo (s)	GA Coste	GA Tiempo (s)
Enero	29 570,0	0,95	29 790,0	0,28	29 955,0	0,47
Febrero	27 735,0	0,63	28 475,0	0,25	28 585,0	0,47
Marzo	34 520,0	0,86	34 750,0	0,29	35 220,0	0,56
Abril	35 370,0	1,00	35 570,0	0,31	35 970,0	0,57
Mayo	37 050,0	0,80	37 380,0	0,32	37 960,0	0,69
Junio	34 320,0	1,23	34 670,0	0,33	35 325,0	0,64
Julio	34 520,0	0,92	34 820,0	0,29	34 955,0	0,56
Agosto	34 520,0	0,79	34 740,0	0,41	35 220,0	0,57
Septiembre	30 645,0	0,82	31 040,0	0,35	31 040,0	0,57
Octubre	32 100,0	0,75	32 400,0	0,29	32 500,0	0,53
Noviembre	29 280,0	0,67	29 950,0	0,38	30 625,0	0,50
Diciembre	29 570,0	0,71	29 910,0	0,28	30 200,0	0,53

Tabla 7.9: Resumen Coste y tiempo ejecución 2025 - 150 habitaciones

Como puede verse en la tabla 7.9, con el modelo exacto se obtienen los costes mínimos que van desde 27 735 € (febrero) hasta 37 050 € (mayo), mientras que en la heurística greedy multi-start y el algoritmo genético los costes son más elevados en cada uno de los meses, siendo el algoritmo genético el que tiene el coste más alto de los tres. En cuanto al tiempo de ejecución, para este caso con una capacidad de 150 habitaciones, los tiempos son casi inmediatos, siendo el modelo exacto el único que supera el segundo de ejecución en abril y junio. Seguidamente, en la tabla 7.10 se decide realizar una estimación puntual de cada modelo normalizando el coste y el tiempo por los días de cada mes, incluyendo también un intervalo de confianza y la desviación típica de cada modelo. Además, en la ta-

bla 7.11 se han calculado los p-valores obtenidos mediante el test de diferencia de medias con el objetivo de comprobar si existen diferencias significativas entre los modelos.

Modelo	Coste/día			Tiempo/día (s)		
	Medio (€)	IC 95 %	Desv. (%)	Medio	IC 95 %	Desv. (%)
Exacto	1 065,84	[1 010,24 – 1 121,44]	8,21	0,0277	[0,0243 – 0,0311]	19,47
GreedyMS	1 077,74	[1 023,85 – 1 131,63]	7,87	0,0104	[0,0095 – 0,0113]	14,34
GA	1 088,84	[1 032,53 – 1 145,15]	8,14	0,0182	[0,0169 – 0,0195]	10,92

Tabla 7.10: Estimación puntual - 150 habitaciones, año 2025

Modelo	Variable	Diferencia	p-value
Exacto vs GreedyMS	Coste/día medio	-11,90	0,735
Exacto vs GA	Coste/día medio	-23,00	0,522
GreedyMS vs GA	Coste/día medio	-11,10	0,754
Exacto vs GreedyMS	Tiempo/día medio (s)	0,02	<0,001
Exacto vs GA	Tiempo/día medio (s)	0,01	<0,001
GreedyMS vs GA	Tiempo/día medio (s)	-0,01	<0,001

Tabla 7.11: Comparación entre modelos (150 habitaciones)

En la tabla 7.10 se observa que, tras haber normalizado por día, el modelo exacto sigue manteniendo el coste medio más bajo (1 065,84 €) aunque con una variación ligeramente más elevada que los otros dos métodos. La heurística tiene un coste medio diario de 1 077,74 €, el cual es el más moderado y, finalmente, el algoritmo genético posee un coste medio mínimo de 1 088,84 €, siendo este el más costoso de los tres métodos. No obstante, los intervalos de confianza al 95 % se solapan entre sí, por lo que no existen diferencias estadísticamente significativas en cuanto al coste diario medio, lo cual se respalda con los p-valores obtenidos en la tabla 7.11 ya que todos son muy elevados y, por lo tanto, no significativos (>0.001).

En cuanto al tiempo de ejecución, vemos que el modelo exacto tarda aproximadamente 0,01 segundos más que los otros dos métodos, siendo la heurística la más rápida de todas (0,0104 s/día). En este caso, el modelo exacto tiene una mayor variabilidad (19,47 %). Por último, los intervalos de confianza no se solapan y los p-valores son prácticamente cero, lo cual indica que existe una diferencia significativa entre el tiempo de ejecución de los tres modelos.

A continuación, para evaluar la escalabilidad de los métodos, se repite la misma comparativa aumentando el tamaño del hotel a 5 000 habitaciones (tabla 7.12).

Mes	Exacto Coste	Exacto Tiempo (s)	GreedyMS Coste	GreedyMS Tiempo (s)	GA Coste	GA Tiempo (s)
Enero	714 820,0	29,93	715 400,0	24,11	716 030,0	10,64
Febrero	713 975,0	62,30	719 680,0	21,30	723 860,0	7,02
Marzo	902 150,0	28,18	904 660,0	37,61	904 990,0	14,10
Abril	930 060,0	70,70	931 445,0	42,51	931 815,0	14,55
Mayo	1 005 000,0	24,11	1 006 990,0	46,55	1 007 410,0	15,70
Junio	907 800,0	19,52	913 590,0	41,10	914 145,0	14,47
Julio	929 320,0	23,34	930 495,0	39,68	930 495,0	14,11
Agosto	894 670,0	20,64	897 230,0	38,26	897 875,0	13,97
Septiembre	799 860,0	19,06	801 675,0	31,30	803 605,0	12,52
Octubre	811 180,0	68,67	811 975,0	30,63	812 965,0	11,98
Noviembre	762 060,0	17,76	769 095,0	29,74	774 105,0	12,77
Diciembre	715 920,0	18,01	716 485,0	23,62	717 330,0	10,53

Tabla 7.12: Resumen Coste y tiempo ejecución 2025 - 5000 habitaciones

Estos resultados nos muestran cómo, al aumentar la capacidad del hotel, los resultados cambian drásticamente. El coste más bajo sigue siendo el que obtiene el método exacto, que es superado por algunos miles de euros en los otros dos métodos. Sin embargo, el tiempo de ejecución de la heurística multi-start, que antes era el más bajo, ahora parece igualar al del modelo exacto. De igual manera que antes, se realiza una estimación puntual normalizando por día para una mejor interpretación y se calculan los p-valores correspondientes.

Modelo	Coste/día			Tiempo/día (s)		
	Medio (€)	IC 95 %	Desv. (%)	Medio	IC 95 %	Desv. (%)
Exacto	27 625,37	[25666.53 – 29584.21]	11,16	1,1106	[0.6613 – 1.5599]	63,67
GreedyMS	27 714,22	[25754.36 – 29674.08]	11,13	1,1116	[0.9449 – 1.2783]	23,60
GA	27 758,88	[25813.50 – 29704.26]	11,03	0,4165	[0.3692 – 0.4638]	17,88

Tabla 7.13: Estimación puntual - 5000 habitaciones, año 2025

Modelo	Variable	Diferencia	p-value
Exacto vs GreedyMS	Coste/día medio	-88,85	0,943736
Exacto vs GA	Coste/día medio	-44,66	0,971605
GreedyMS vs GA	Coste/día medio	-133.51	0.915232
Exacto vs GreedyMS	Tiempo/día medio (s)	-0,00	0,996335
Exacto vs GA	Tiempo/día medio (s)	0,69	<0,001
GreedyMS vs GA	Tiempo/día medio (s)	0,70	<0,001

Tabla 7.14: Comparación entre modelos (5000 habitaciones)

En la tabla 7.13 podemos ver que las desviaciones típicas de los tres modelos en

cuanto al coste diario son casi idénticas. Además, el coste medio mínimo sigue siendo el obtenido por el modelo exacto, seguido por la heurística greedy multi-start y, por último, el algoritmo genético. Sin embargo, los intervalos de confianza para el coste medio siguen solapándose para esta capacidad, lo que indica que no existen diferencias estadísticamente significativas entre los modelos, lo cual se confirma por los valores superiores a 0,9 de los p-valores.

Por otro lado, ahora el algoritmo con el mayor tiempo de ejecución es la heurística *multi-start*, que llega a superar al modelo exacto por aproximadamente 0,001 segundos al día. El algoritmo genético tarda menos de la mitad de tiempo que los otros modelos para esta ejecución (0,04165 s/día). Asimismo, el algoritmo genético tiene la menor variación, con una desviación típica de 17,88 % frente al 63,67 % y 23,60 % del modelo exacto y la heurística, respectivamente. Además, el modelo exacto y la heurística no muestran diferencias estadísticamente significativas en cuanto al tiempo de ejecución, ya que los intervalos de confianza se solapan y el p-valor obtenido es muy elevado. Por el contrario, sí existen diferencias significativas de ambos modelos con respecto al tiempo de ejecución del algoritmo genético (p-val <0.001). Por ello, se concluye que el algoritmo genético es significativamente más rápido que los otros dos modelos.

Se ha generado un gráfico que permite ver de forma visual el comportamiento de los algoritmos en cuanto al tiempo de ejecución con respecto al número de habitaciones (y por tanto, el número de empleados).

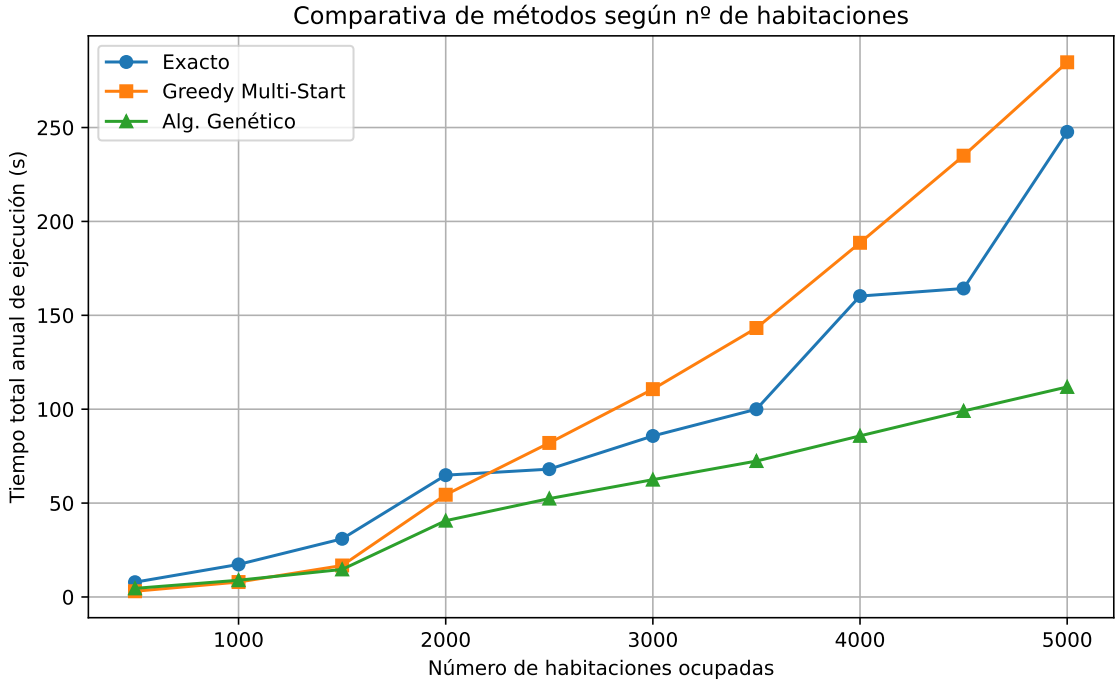

Figura 7.5: Tiempo ejecución vs nº empleados 2025

La figura 7.5 muestra cómo, hasta aproximadamente 1 000 habitaciones, los tiempos de la heurística con multi-start y la metaheurística van prácticamente a la par, siendo ligeramente superior el de la metaheurística. A partir de ese punto, el tiempo de ejecución de la heurística con múltiples arranques comienza a elevarse hasta el punto en el que el modelo exacto consigue superar el rendimiento de la heurística con una capacidad de aproximadamente 2 200 habitaciones.

8. Discusión de los resultados y conclusiones

A lo largo de este trabajo se han comparado cuatro modelos de generación de horarios: un método exacto, una heurística *greedy* con y sin versión *multi-start* y un algoritmo genético.

El método exacto obtiene siempre la mejor solución en cuanto a coste, y para un hotel con un número no muy elevado de habitaciones, se ejecuta en un tiempo más que razonable, no superando ni el segundo de ejecución en las pruebas realizadas. En estos casos no sería necesario recurrir a enfoques heurísticos o metaheurísticos. Sin embargo, si el problema crece, el tiempo de cálculo se eleva drásticamente y deja de ser la mejor opción.

Por otra parte, la heurística *greedy* construye el horario día a día en mucho menos tiempo y cumpliendo con las restricciones diarias. Esto es ideal para problemas más grandes y para los cambios de horario por bajas, vacaciones, nuevas contrataciones o cualquier otro tipo de improviso. Su punto débil está en las restricciones a largo plazo: al no revisar lo ya asignado, a veces deja a empleados sin días libres semanales. Es por esto que este modelo solo es válido en el caso en el que las restricciones globales no tengan que cumplirse estrictamente. Por eso, aunque resulta muy rápido, es necesario reforzarlo con otro método que cumpla todos los requisitos. El enfoque *multi-start* repite la heurística anterior varias veces con diferentes órdenes iniciales, buscando soluciones que puedan salir de óptimos locales. Con un número suficiente de arranques, el algoritmo ha conseguido devolver una asignación factible para todas las pruebas realizadas.

Los resultados obtenidos muestran que el algoritmo genético, a pesar de ofrecer la solución con el coste más alto de los tres métodos evaluados, tiene un buen rendimiento en términos de eficiencia computacional, siendo significativamente el método más rápido para hoteles de gran volumen. Por tanto, este algoritmo es la opción más adecuada para hoteles o cadenas hoteleras con miles de empleados, donde es crucial obtener las asignaciones en el menor tiempo posible.

La principal dificultad a lo largo del trabajo ha sido compatibilizar las restricciones diarias con los requisitos globales (semanales o mensuales). Ya que, si se reservan primero los días libres para asegurar los descansos semanales, luego surgían huecos que no se podían rellenar debido a incompatibilidades en la cobertura, ya que algunos turnos no tenían empleados disponibles que no rompiesen las reglas de descanso. Finalmente, la solución más eficaz fue primero generar una solución que minimiza las violaciones, y luego aplicar una mejora, ya sea con el enfoque *multi-start* o con el algoritmo genético, que refinan el resultado eliminando las incompatibilidades que pudieran quedar.

Volviendo a los objetivos que se plantearon al inicio de esta memoria, podemos ver que el objetivo (O1) se ha cumplido con éxito, ya que se ha diseñado un modelo de optimización que minimiza el coste del hotel y se han planteado diferentes restricciones aplicadas a la gestión de un hotel cumpliendo con los requisitos mínimos legales. El objetivo (O2) también se ha conseguido al implementar los métodos que consiguen resolver este problema (exacto, heurística y metaheurística). Por último, el análisis comparativo realizado en el capítulo 7 muestra el cumplimiento del objetivo (O3).

8.1. Trabajo Futuro

De cara a futuros proyectos, convendría reforzar la heurística *greedy* para que sea capaz de obtener soluciones de calidad, cumpliendo todas las restricciones, para así no sacrificar su rendimiento. También se podrían incluir otro tipo de restricciones que mejoraran la calidad de las planificaciones como, por ejemplo, incluir un máximo de días consecutivos de trabajo seguidos o un mínimo de domingos libres para los empleados. Siguiendo en la misma línea, con la intención de mejorar la calidad de los horarios, se podría tener más en cuenta el bienestar de los trabajadores, incorporando variables que les permitan introducir preferencias personales en cuanto a turnos y días al modelo. De esta forma, se favorecería la reducción de la rotación de personal y un mejor clima laboral.

Asimismo, sería interesante desarrollar una interfaz que permita llevar el algoritmo a la práctica y evaluarlo en un hotel real. El responsable del personal del hotel podría interactuar con una aplicación web que calcule la planificación y proponga nuevas asignaciones que garanticen tanto la cobertura requerida como los descansos del personal. De esta forma, sería muy sencillo tener en cuenta cambios como la rotación de personal, las vacaciones o bajas de los empleados, de manera instantánea.

Referencias

[1] Y. Alaouchiche, Y. Ouazene, F. Yalaoui, and H. Chehade. Workload balancing for the nurse scheduling problem: A real-world case study from a french hospital. *Socio-Economic Planning Sciences*, 2024.

[2] C. Artigues, M. Gendreau, and L-M. Rousseau. *A flexible model and a hybrid exact method for integrated employee timetabling and production scheduling*, volume 3867 of *Lecture Notes in Computer Science*. Springer-Verlag, 2007.

[3] Alain Billionnet. Integer programming to schedule a hierarchical workforce with variable demands. *European Journal of Operational Research*, 114:105–114, 1999.

[4] Instituto Nacional de Estadística. Encuesta hotelera, 2024.

[5] Instituto Nacional de Estadística. Estadísticas - inebase, 2024.

[6] J. Van den Bergh, J. Beliën, P. De Bruecker, E. Demeulemeester, and L. De Boeck. Personnel scheduling: A literature review. *European Journal of Operational Research*, 226(3):367–385, 2013.

[7] S. Rajakumar, V. Arunachalam, and V. Selladurai. Workflow balancing strategies in parallel machine scheduling. *International Journal of Advanced Manufacturing Technology*, 23:366–374, 2004.

[8] S. Rajakumar, V. Arunachalam, and V. Selladurai. Workflow balancing in parallel machines through genetic algorithm. *International Journal of Advanced Manufacturing Technology*, 33:1212–1221, 2007.

[9] Chang-Chun Tsai and Sherman Li. A two-stage modeling with genetic algorithms for the nurse scheduling problem. *Expert Systems with Applications*, 36:9506–9512, 2009.

[10] T. C. Wong, M. Xu, and K. S. Chin. A two-stage heuristic approach for nurse scheduling problem: A case study in an emergency department. *Computers & Operations Research*, 51:99–110, 2014.

Siglas

GA Algoritmos genéticos. 6

ILP Programación Lineal Entera. 2

INE Instituto Nacional de Estadística. 1, 7

MILP Programación Lineal Entera Mixta. 2, 4, 5, 10

NSP Problema de programación de turnos en enfermería. 5, 6

A. Apéndice: Código Python desarrollado

Modelo Exacto

```
start_time = time.time()

# Problema y Variables de Decision
prob = LpProblem("Planificacion_Mensual", LpMinimize)

# variable binaria: Para cada empleado, dia y turno disponible segun el area
y = {}
for e in lista_empleados:
    if area_empleado[e] == 'Seguridad':
        ops = turnos_seguridad
    elif area_empleado[e] == 'Limpieza':
        ops = turnos_limpieza
    else:
        ops = turnos # Recepcion y restauracion
    y[e] = {d: {op: LpVariable(f"y_{e}_{d}_{op}", cat='Binary') for op in ops} for d in range(D)}

# Restricciones

# Restriccion 1: Cada empleado debe tener exactamente un unico turno por dia.
for e in lista_empleados:
    for d in range(D):
        prob += lpSum(y[e][d][op] for op in y[e][d]) == 1, f"Un_opcion_{e}_{d}"



# Restriccion 2: Cada empleado tendra como maximo 9 dias libres al mes
lista_dias = range(D)

for e in lista_empleados:
    prob += (
        lpSum(y[e][d]['Libre'] for d in lista_dias) <= 9,
        f"Max9Libres_mes_{e}"
    )
```

```
# Restriccion 3: Cada empleado debe tener 1 dias libres por semana
primer_dia, D = calendar.monthrange(ano, num_mes)
primer_dia_mes = primer_dia + 1 # 1=Lunes, ..., 7=Domingo
# Se calcula indice de semana para cada indice de dia
offset = primer_dia_mes - 1  # Dias hasta completar la primera semana de lunes a domingo

i = {}
for d in range(D):  # d=0..D-1
    dia = d + 1  # dia del mes
    sem = (offset + (dia - 1)) // 7 + 1
    i[d] = sem

# Agrupamos indices de dia por semana
dias_por_semana = {}
for d, sem in i.items():
    dias_por_semana.setdefault(sem, []).append(d)

# Restriccion para cada empleado y cada semana
for e in lista_empleados:
    for sem, dias_i in dias_por_semana.items():

        suma_libres = lpSum(y[e][d]['Libre'] for d in dias_i)

        if len(dias_i) == 7:

            prob += suma_libres >= 1, f"Libres2_semana{sem}_{e}"



# Restriccion 4: Se deben garantizar 12 horas de descanso entre turnos.
for e in lista_empleados:
    for d in range(D - 1):
        if 'Noche' in y[e][d] and 'Manana' in y[e][d+1]:
            prob += y[e][d]['Noche'] + y[e][d+1]['Manana'] <= 1, f"Descanso_12h_1_{e}_{d}"
        if 'Tarde' in y[e][d] and 'Manana' in y[e][d+1]:
            prob += y[e][d]['Tarde'] + y[e][d+1]['Manana'] <= 1, f"Descanso_12h_2_{e}_{d}"
        if 'Noche' in y[e][d] and 'Tarde' in y[e][d+1]:
```

```
            prob += y[e][d]['Noche'] + y[e][d+1]['Tarde'] <= 1, ↩
    f"Descanso_12h_3_{e}_{d}"


# Restriccion 5: Prohibimos los cambios de turno que impidan que ↩
    haya un descanso de menos de 36h si hay un solo dia libre de ↩
    por medio
# Prohibir secuencias Tarde -> Libre -> Manana | Noche -> Libre -> ↩
    Manana | Noche -> Libre -> Tarde
for e in lista_empleados:
    for d in range(D - 2):
        # Solo si el empleado tiene esas tres variables en los dias ↩
    d, d+1 y d+2
        if all(
            turno in y[e][dia]
            for turno, dia in [('Tarde', d), ('Libre', d+1), ↩
    ('Manana', d+2)]
        ):
            prob += (
                y[e][d]['Tarde']
              + y[e][d+1]['Libre']
              + y[e][d+2]['Manana']
              <= 2,
                f"No_Tarde_Libre_Manana_{e}_{d}"
            )

        if all(
            turno in y[e][dia]
            for turno, dia in [('Noche', d), ('Libre', d+1), ↩
    ('Manana', d+2)]
        ):
            prob += (
                y[e][d]['Noche']
              + y[e][d+1]['Libre']
              + y[e][d+2]['Manana']
              <= 2,
                f"No_Noche_Libre_Manana_{e}_{d}"
            )

        if all(
            turno in y[e][dia]
```

```
            for turno, dia in [('Noche', d), ('Libre', d+1), ↩
    ('Tarde', d+2)]
        ):
            prob += (
                y[e][d]['Noche']
              + y[e][d+1]['Libre']
              + y[e][d+2]['Tarde']
              <= 2,
                f"No_Noche_Libre_Tarde_{e}_{d}"
            )

# Restriccion 6: Minimo de empleados trabajando en los fines de ↩
    semana (ocupacion mas alta)
# Consideramos que como minimo en cada turno en los fines de semana ↩
    debe haber el 50% de los empleados totales dividido entre los ↩
    turnos

for d in range(D):
    dia = (primer_dia_mes + d - 1) % 7
    if dia in [5,6]:  # sabado o domingo
        for turno in ['Manana', 'Tarde']:
            empleados_restauracion_trabajando_turno = lpSum(
                y[e][d][turno] for e in lista_empleados
                if area_empleado[e] == 'Restauracion'
                if turno != 'Libre'
            )
            prob += empleados_restauracion_trabajando_turno >= ↩
    min_empleados_por_turno_Rest[turno], ↩
    f"Min_restauracion_{turno}_fin_de_semana_{d}"
        for turno in ['Manana', 'Tarde']:
            empleados_recepcion_trabajando_turno = lpSum(
                y[e][d][turno] for e in lista_empleados
                if area_empleado[e] == 'Recepcion'
                if turno != 'Libre'
            )
            prob += empleados_recepcion_trabajando_turno >= ↩
    min_empleados_por_turno_Recep[turno], ↩
    f"Min_recepcion_{turno}_fin_de_semana_{d}"
        for turno in ['Manana']:
            empleados_limpieza_trabajando_turno = lpSum(
                y[e][d][turno] for e in lista_empleados
```

```
                if area_empleado[e] == 'Limpieza'
                if turno != 'Libre'
            )
            prob += empleados_limpieza_trabajando_turno >= ↩
    min_empleados_por_turno_Limp[turno], ↩
    f"Min_limpieza_{turno}_fin_de_semana_{d}"
        for turno in ['Manana','Tarde','Noche']:
            empleados_seguridad_trabajando_turno = lpSum(
                y[e][d][turno] for e in lista_empleados
                if area_empleado[e] == 'Seguridad'
                if turno != 'Libre'
            )
            prob += empleados_seguridad_trabajando_turno >= ↩
    min_empleados_por_turno_Segu[turno], ↩
    f"Min_seguridad_{turno}_fin_de_semana_{d}"


# Restriccion 7: Cobertura minima entre semana
# Minimo de empleados por turno de lunes a viernes 25% de los ↩
    empleados contratados

for d in range(D):
    dia = (primer_dia_mes + d - 1) % 7
    if dia not in [5, 6]:  # Si NO es sabado (5) o domingo (6)
        for turno in ['Manana', 'Tarde']:
            empleados_restauracion_trabajando_turno = lpSum(
                y[e][d][turno] for e in lista_empleados
                if area_empleado[e] == 'Restauracion' and turno != ↩
    'Libre'
            )
            prob += empleados_restauracion_trabajando_turno >= ↩
    min_empleados_por_turno_Rest[turno], ↩
    f"Min_restauracion_{turno}_laboral_{d}"

        for turno in ['Manana', 'Tarde']:
            empleados_recepcion_trabajando_turno = lpSum(
                y[e][d][turno] for e in lista_empleados
                if area_empleado[e] == 'Recepcion' and turno != 'Libre'
            )
            prob += empleados_recepcion_trabajando_turno >= ↩
    min_empleados_por_turno_Recep[turno], ↩
```

```
    f"Min_recepcion_{turno}_laboral_{d}"

        for turno in ['Manana']:
            empleados_limpieza_trabajando_turno = lpSum(
                y[e][d][turno] for e in lista_empleados
                if area_empleado[e] == 'Limpieza' and turno != 'Libre'
            )
            prob += empleados_limpieza_trabajando_turno >= ↩
    min_empleados_por_turno_Limp[turno], ↩
    f"Min_limpieza_{turno}_laboral_{d}"

        for turno in ['Manana','Tarde','Noche']:
            empleados_seguridad_trabajando_turno = lpSum(
                y[e][d][turno] for e in lista_empleados
                if area_empleado[e] == 'Seguridad' and turno != 'Libre'
            )
            prob += empleados_seguridad_trabajando_turno >= ↩
    min_empleados_por_turno_Segu[turno], ↩
    f"Min_seguridad_{turno}_laboral_{d}"


# Restriccion 8: Cobertura minima 1 por turno por dia
# Se requiere que al menos 1 persona trabaje en un turno y dia dado ↩
    para casos en los que hay pocos trabajadores
for area, lista_emps in empleados.items():
    if area == 'Seguridad':
        turnos = ['Manana', 'Tarde', 'Noche']
    elif area == 'Limpieza':
        turnos = ['Manana']
    else:
        turnos = ['Manana', 'Tarde']
    for d in range(D):
        for turno in turnos:
            prob += (
                lpSum(y[e][d][turno] for e in lista_emps)
                >= 1,
                f"Cover_{area}_{turno}_dia_{d+1}"
            )


# Funcion objetivo: minimizar coste total de los turnos
```

```
coste = {}
for e in lista_empleados:
    area = area_empleado[e]
    for d in range(D):
        for op, var in y[e][d].items():
            coste[(e,d,op)] = coste_turno[area][op]

# Funcion objetivo
prob += lpSum(
    coste[(e,d,op)] * y[e][d][op]
    for e, dias in y.items()
    for d, ops in dias.items()
    for op in ops
), "CosteTotal"



# Resultados
status = prob.solve(PULP_CBC_CMD(msg=True))

tiempo_calculo = time.time() - start_time

df = pd.DataFrame([{
    "Mes": mes_actual,
    "Anyo": ano,
    "Tiempo(s)": f"{tiempo_calculo:.1f}",
    "Estado": LpStatus[status],
    "Coste": f"{value(prob.objective):.2f}"
}])
```

Heurística greedy

```
class HeuristicaGreedy:
    def __init__(self,
                 df,
                 meses,
                 coste_turno: dict,
                 min_fds: dict,
                 min_sem: dict,
                 turnos_area: dict,
                 PSecuencias: list,
                 PTransiciones: set,
                 seed=None):
        """
        Datos de entrada:

        df: data frame n empleados por area y mes
        meses: diccionario con el nombre del mes (numeros 1-12)
        coste_turno: dict[area][turno] - coste de cada turno por area
        min_fds: dict[area][turno] - minimo empleados fines de semana
        min_sem: dict[area][turno] - minimo empleados entre semana
        turnos_area: dict[area] - lista de turnos validos por area
        PSecuencias: lista de secuencias prohibidas
                    - conjunto de tres turnos con descanso <36h
        PTransiciones: conjunto de pares (t_prev, t_next)
                    - transiciones prohibidas por violar <12h
        seed: semilla

        """
        self.df            = df
        self.meses         = meses
        self.seed          = seed
        self.coste_turno   = coste_turno
        self.min_fds       = min_fds
        self.min_sem       = min_sem
        self.turnos_area   = turnos_area
        self.PSecuencias   = PSecuencias
        self.PTransiciones = PTransiciones

    def generar_horario(self, mes_actual, ano, seed=None):
        sem = seed if seed is not None else self.seed
```

```
    random.seed(sem)

    # Datos de mes y empleados
    num_mes = self.meses[mes_actual]
    D = calendar.monthrange(ano, num_mes)[1]
    df_mes = self.df[self.df['Mes'] == mes_actual].iloc[0]

    empleados_por_area = {
        area: [f"{area[:3]}{i+1}" for i in ↩
range(int(df_mes[area]))]
        for area in self.turnos_area
    }

    # Inicializar contadores y estructuras
    count_turnos     = {area: {e: {t:0 for t in ↩
self.turnos_area[area]} for e in emps}
                        for area, emps in ↩
empleados_por_area.items()}
    turno_anterior   = {area: {} for area in empleados_por_area}
    consecutivos     = {area: {e:0 for e in emps} for area, emps ↩
in empleados_por_area.items()}
    libres_semana    = {area: {e:0 for e in emps} for area, emps ↩
in empleados_por_area.items()}
    libres_mes       = {area: {e:0 for e in emps} for area, emps ↩
in empleados_por_area.items()}

    horario = {d: {area: {} for area in empleados_por_area} for ↩
d in range(1, D+1)}
    coste   = 0
    infeasible = False

    for day in range(1, D+1):
        d = calendar.weekday(ano, num_mes, day)
        fin = d >= 5
        reqs = self.min_fds if fin else self.min_sem

        # Reset semanal (lunes - d==0)
        if d == 0:
            for area in empleados_por_area:
                for e in empleados_por_area[area]:
                    libres_semana[area][e] = 0
```

```
        dias_semana_rest = 6 - d

        for area, emps in empleados_por_area.items():
            req = reqs[area]
            turns = sorted(self.turnos_area[area],
                           key=lambda t: self.coste_turno[area][t])
            ptr = {t: random.randrange(len(emps)) for t in turns}

            def rompe_36h(emp, turno):
                if day >= 3:
                    seq = [
                        horario[day-2][area].get(emp),
                        horario[day-1][area].get(emp),
                        turno
                    ]
                    return seq in self.PSecuencias
                return False

            # Asignacion cobertura minimos
            for t in turns:
                for _ in range(req.get(t, 0)):
                    candidatos = [e for e in emps if e not in ↩
horario[day][area]]
                    # filtrar <12 horas
                    candidatos = [
                        e for e in candidatos
                        if (turno_anterior[area].get(e), t) not ↩
in self.PTransiciones
                    ]
                    # filtrar <36 horas
                    candidatos = [e for e in candidatos if not ↩
rompe_36h(e, t)]
                    if not candidatos:
                        candidatos = [e for e in emps if e not ↩
in horario[day][area]]
                        ptr[t] = 0
                    e = candidatos[ptr[t] % len(candidatos)]
                    horario[day][area][e] = t
                    count_turnos[area][e][t] += 1
                    if t=='Libre':
                        consecutivos[area][e]=0
```

```
                        libres_semana[area][e]+=1
                        libres_mes[area][e]+=1
                    else:
                        consecutivos[area][e]+=1
                    ptr[t]+=1

            # Asignar el resto de turnos
            for e in emps:
                if e not in horario[day][area]:
                    if consecutivos[area][e] >= 6 and ↩
libres_mes[area][e] < 9: # Dia libre cada semana
                        t_sel = 'Libre'
                    else:
                        faltan = 1 - libres_semana[area][e]
                        if faltan > 0 and dias_semana_rest + 1 ↩
== faltan and libres_mes[area][e] < 9:
                            t_sel = 'Libre'
                        elif libres_semana[area][e] >= 2 or ↩
libres_mes[area][e] >= 9:
                            dispo = [
                                t for t in turns if t != 'Libre'
                                and ↩
(turno_anterior[area].get(e), t) not in PTransiciones
                                and not rompe_36h(e, t)
                            ]
                            if dispo:
                                t_sel = dispo[0]
                            else:
                                t_sel = turns[0]

                        else:
                            t_sel = 'Libre'

                    horario[day][area][e] = t_sel
                    count_turnos[area][e][t_sel] += 1
                    if t_sel == 'Libre':
                        consecutivos[area][e] = 0
                        libres_semana[area][e] += 1
                        libres_mes[area][e] += 1
                    else:
                        consecutivos[area][e] += 1
```

```
                # Guardar turno anterior para la proxima iteracion
                for e, t in horario[day][area].items():
                    turno_anterior[area][e] = t

            # Calcular coste diario
            for area in empleados_por_area:
                for e, t in horario[day][area].items():
                    coste += self.coste_turno[area][t]

    chk = self.check_viol_horario(horario, mes_actual, ano)
    viol = len(chk['violaciones'])
    if coste >= 1e6 or viol != 0:
        infeasible = True
    status = 'Infactible' if infeasible else 'Optimal'

    return {
        'horario': horario,
        'status':  status,
        'coste':   coste,
        'violaciones': viol
    }
def multi_start_greedy(self, mes_actual, ano, n_starts):
    best = None
    best_cost = float('inf')
    best_viol = float('inf')
    infeasible = False

    for i in range(n_starts):
        self.seed = i
        sol_dict = self.generar_horario(mes_actual, ano)
        horario = sol_dict['horario']

        chk = self.check_viol_horario(horario, mes_actual, ano)
        cost = chk['coste']
        viol = len(chk['violaciones'])

        if (viol < best_viol) or (viol == best_viol and cost < ↩
best_cost):
            best = horario
            best_cost = cost
```

```
            best_viol = viol

    if best_cost >= 1e6 or best_viol!=0:
        infeasible = True

    status = 'Infactible' if infeasible else 'Optimal'
    return {
        'horario': best,
        'coste': best_cost,
        'status': status,
        'violaciones': best_viol,
    }
```

Algoritmo genético

```
class AlgoritmoGenetico:
    def __init__(self, heuristic, mes_actual, ano,
                 poblacion_size, generaciones,
                 crossover_prob, mutation_prob):
        self.heuristic      = heuristic
        self.mes_actual     = mes_actual
        self.ano            = ano
        self.pop_size       = poblacion_size
        self.generaciones   = generaciones
        self.crossover_prob = crossover_prob
        self.mutation_prob  = mutation_prob



        res = self.heuristic.generar_horario(mes_actual, ano)
        base_horario = res['horario']
        # inicializar poblacion clonando la solucion greedy
        self.poblacion = [deepcopy(base_horario) for _ in ↩
    range(self.pop_size)]

    def fitness(self, horario):
        chk = self.heuristic.check_viol_horario(horario, ↩
    self.mes_actual, self.ano)
        return chk['coste'] + len(chk['violaciones']) * 10

    def seleccion_torneo(self, k=3):
        candidatos = random.sample(self.poblacion, k)
        return min(candidatos, key=lambda h: self.fitness(h))

    def crossover(self, padre, madre):
        dias = sorted(padre.keys())
        punto = random.randint(1, len(dias)-1)
        h1, h2 = {}, {}
        for i, d in enumerate(dias):
            if i < punto:
                h1[d] = deepcopy(padre[d]); h2[d] = deepcopy(madre[d])
            else:
                h1[d] = deepcopy(madre[d]); h2[d] = deepcopy(padre[d])
        return h1, h2
```

```
    def mutacion(self, horario):
        day = random.choice(list(horario.keys()))
        for area in horario[day]:
            emps = list(horario[day][area].keys())
            if len(emps) > 1:
                e1, e2 = random.sample(emps, 2)
                horario[day][area][e1], horario[day][area][e2] = ↩
    horario[day][area][e2], horario[day][area][e1]
                break
        return horario
    def ejecutar(self):
        sin_mejora = 0
        mejor_previo = float('inf')

        for gen in range(self.generaciones):
            # nueva poblacion
            nueva = []
            while len(nueva) < self.pop_size:
                p, m = self.seleccion_torneo(), self.seleccion_torneo()
                if random.random() < self.crossover_prob:
                    o1, o2 = self.crossover(deepcopy(p), deepcopy(m))
                else:
                    o1, o2 = deepcopy(p), deepcopy(m)
                if random.random() < self.mutation_prob: o1 = ↩
    self.mutacion(o1)
                if random.random() < self.mutation_prob: o2 = ↩
    self.mutacion(o2)
                nueva.extend([o1, o2])
            self.poblacion = nueva[:self.pop_size]

            # mejor horario de la generacion
            mejor_actual = min(self.poblacion, key=lambda h: ↩
    self.fitness(h))
            fitness_actual = self.fitness(mejor_actual)

            # comprobamos si ha habido mejora
            if fitness_actual < mejor_previo:
                mejor_previo = fitness_actual
                sin_mejora = 0
            else:
                sin_mejora += 1
```

```
        mejor = min(self.poblacion, key=lambda h: self.fitness(h))
        return {'horario': mejor, 'fitness': self.fitness(mejor)}
```