UNIVERSIDAD COMPLUTENSE DE MADRID

FACULTAD DE CIENCIAS MATEMÁTICAS

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

TRABAJO DE FIN DE GRADO

Algoritmos de Aprendizaje Automático aplicados a problemas de Ciberseguridad

Presentado por: Pablo Jiménez Poyatos

Dirigido por: Luis Fernando Llana Diaz

Grado en Matemáticas

Curso académico 2023-24

Agradecimientos

Estas páginas ponen punto y final a una de las etapas más complicadas de mi vida. Por eso, estos agradecimientos se van a quedar cortos.

En primer lugar, quiero agradecer y dedicar este trabajo a la sístole y diástole de mi vida. A mi compañera en lo bueno y en lo malo. Sara, infinitas gracias. Sin tu apoyo, no habría sido posible. Sin tu aliento, nunca habría podido llegar a la meta. Sin tu consejo, no habría encontrado la salida. Sin tu cariño y paciencia, esto no se habría podido materializar. Gracias y mil veces gracias.

A mis padres, Ana y Francisco. Por dejarme decidir. Por confiar en mí. Ha sido un camino lleno de altibajos, de pensar que no se podía, que era demasiado difícil,... y lo fue, pero vosotros nunca me habéis exigido nada, y no puedo estar más agradecido y orgulloso de ser vuestro hijo. Desde aquella primera alegría desmesurada en el puerto de Gandía hasta hoy: ha sido todo por y para vosotros.

A mis amigos: Nacho, Sara, Marina y Rocío. De una u otra forma, sois parte de esto. Gracias por apoyarme siempre.

A Julián, por darme y enseñarme las herramientas que han hecho esto posible.

A mi gran y querido compañero de beca. Nunca pensé que sería becario en la universidad, pero si tuviera que repetir la experiencia, que fuera siempre contigo, Jhimmy. He aprendido y aprendo mucho de ti. Por muchas más charlas de programación, matemáticas e informática, y que nunca dejemos de construir castillos en el aire pensando en esas super empresas que se van a pelear por nosotros.

A mis compañeros del equipo de UTM en Indra, por resolver todas y cada una de mis dudas. En especial a Marta, por prestarse para la entrevista y guiar mis inicios como Ingeniero de Sistemas.

A mi tutor, Rafa del Vado. Por su implicación desde aquel día que entró en el aula para darme clase, hasta el día en el que entrego esta memoria. Muchas gracias, eres una fuente de inspiración como matemático y como persona.

Quiero mencionar también a algunas profesoras y profesores que han marcado mi camino: Pilar Cembranos, Rosa Alonso, Ángel Manuel Ramos, Ildefonso Díaz, Martín Avendaño, María del Mar Jiménez, Natalia López y Jose Luis Vega. Seguro que olvido a alguien. Han sido muchos los que han marcado este camino. Gracias por vuestra forma de trasmitir las matemáticas.

A Rober Tomé por hacer las matemáticas accesibles. Por contar lo abstracto como si fuera una película. Gracias.

Last but not least, I wanna thank me. I wanna thank me for believing in me. I wanna thank me for doing all this hard work. - Snoop Dogg

Resumen

El objetivo principal de esta investigación es la aplicación de los algoritmos genéticos al problema de diseño de vertipuertos. Basándonos en el estado del arte sobre diseño de vertipuertos y movilidad aérea urbana, complementado por las perspectivas obtenidas de una entrevista con una Ingeniera Aeroespacial experta en estos ámbitos, sentamos las bases del modelo matemático para el problema de óptimización, que resolveremos mediante heurísticas evolutivas, en concreto, algoritmos genéticos. Mediante un análisis de sensibilidad de los parámetros del algoritmo genético, hemos observado que tiende a converger hacia óptimos locales, lo cual destaca la necesidad de explorar estrategias adicionales para mejorar su capacidad de búsqueda global.

Palabras clave: *Vertipuerto, Algoritmos Genéticos, Óptimización, Heurísticas evolutivas, Movilidad Aérea Urbana.*

Abstract

The main objective of this research is the application of genetic algorithms to the vertiport design problem. Based on the state of the art in vertiport design and urban air mobility, complemented by the insights obtained from an interview with an Aerospace Engineer expert in these areas, we establish the foundations of the mathematical model for the optimization problem that we will solve using evolutionary heuristics, specifically, genetic algorithms. Through a sensitivity analysis of the genetic algorithm parameters, we have observed that it tends to converge towards local optima, which highlights the need to explore additional strategies to improve its global search capability.

Keywords: *Vertiport, Genetic Algorithms, Optimization, Evolutionary Heuristics, Urban Air Mobility.*

Índice general

Capítulo 1

Introducción

Las ciudades modernas enfrentan el reto de la movilidad urbana a diario y desde diferentes puntos de vista. No solo la congestión de sus carreteras forma parte de la problemática, las consideraciones sobre ecología y salud deben estar también presentes en los planes de movilidad de cualquier gran ciudad. En este primer capítulo, vamos a definir el concepto de *movilidad aérea urbana*, una posible solución integral al problema, ahondando en la infraestructura protagonista de este tipo de movilidad. Introduciremos el reto que supone diseñarla y hacer de ella una infraestructura eficiente desde el punto de vista de la optimización matemática. Finalmente, describiremos la estructura de la memoria, así como sus contribuciones más importantes. Las principales fuentes de consulta que han servido para la elaboración de este capítulo son: [**?**, **?**, **?**, **?**].

1.1. Motivación y objetivos del trabajo

Durante mis primeros meses de trabajo como Ingeniero de Sistemas en la división de UTM (de sus siglas en inglés, *Unmanned Traffic Management*) en Indra Sistemas S.A., me destinaron a uno de los proyectos europeos más ambiciosos en los que se está trabajando bajo el consorcio SESAR RJU[1]: EUREKA [**?**]. En este proyecto se están definiendo los flujos de trabajo dentro de vertipuertos, que es para los drones lo que un aeropuerto es para un avión, pero me llamó la atención que las diferentes infraestructuras que lo conforman no se colocaban según un cierto criterio.

El problema que vamos a tratar en esta memoria se conoce como el *problema de diseño de vertipuertos*. Su objetivo principal es encontrar el diseño *óptimo* de un vertipuerto basándose en la geometría del espacio que se ha elegido para su construcción, en términos de maximizar la capacidad y el beneficio neto del operador del vertipuerto. Por tanto, enfrentamos un *problema de óptimización matemática* de interés, que surge para abordar un problema complejo en el sector aeroespacial. El problema trata de encontrar la posición óptima de las infraestructuras del vertipuerto, incluyendo las áreas de aterrizaje y despegue, puertas, calles de rodaje y terminales de pasajeros dentro del espacio delimitado para su instalación. Además, el sector se caracteriza por un estándar muy exigente

[1]Para más información `https://www.sesarju.eu`.

en cuanto a los *requisitos de seguridad operacional*. Por todo ello, el uso de *algoritmos de optimización exactos* enfrenta varios retos relativos a la complejidad computacional, ya que las posibles soluciones crecen de manera exponencial, como los retos *hardware*, donde es necesario ser capaz de encontrar un balance óptimo entre tiempos de cómputo y una solución factible.

Este tipo de problemas tiene un gran número de variables y posibles combinaciones de la solución. Por ello, su complejidad computacional es NP-difícil [**?**], los algoritmos exactos suponen una mala elección a la hora de buscar el equilibrio del que hablábamos anteriormente.

Sin embargo, estos retos suponen también una oportunidad única para aplicar *heurísticas evolutivas*: algoritmos flexibles que se aplican a una gran variedad de problemas; estos algoritmos son robustos, ya que pueden encontrar buenas soluciones en espacios complejos con múltiples óptimos locales; permiten restricciones de diversos tipos, lo cual se hace indispensable cuando modelamos un problema real. Así mismo, permite incorporar el conocimiento experto que se tiene de un problema concreto: dado lo novedoso del problema, a tratar en este trabajo, el uso de estas técnicas heurísticas es también un camino sin recorrer, lo cual hace todavía más interesante su estudio.

Motivada la elección de la temática del trabajo nos planteamos partir de los siguientes objetivos:

- Asimilar conceptos complejos sobre movilidad aérea urbana y el negocio UTM.
- Aprender sobre las heurísticas evolutivas para poder llevar a cabo la implementación correcta y eficiente de un algoritmo genético.
- Aplicar y evolucionar mis conocimientos sobre Python.
- Que dicha implementación sea capaz de obtener soluciones factibles en un tiempo razonable, con el fin de extraer conclusiones sobre su validez.
- Tratar de mejorar la solución ofrecida en el artículo [**?**], donde se propone un problema de diseño de vertipuertos *mediante programación entera*.
- Trabajar otros problemas del negocio UTM, como puede ser el *problema de flujos de operación en un vertipuerto*, que introduciremos en la subsección **??**.
- Investigar y aprender sobre el mejor ajuste de configuración en algoritmos bioinspirados, como los algoritmos genéticos.
- Realizar un estudio de sensibilidad que permita incrementar la calidad de las soluciones, sus trazas de validez, y su aplicabilidad a otros posibles escenarios que involucren el manejo de una mayor cantidad de información.

1.2. Contexto y antecedentes del trabajo

En esta subsección vamos a introducir formalmente los conceptos de *movilidad aérea urbana*, *vertipuertos* y *optimización heurística*, además del *problema de optimización de diseño de un vertipuerto*.

1.2.1. Movilidad Aérea Urbana

El futuro de la movilidad urbana pasa por utilizar medios de trasporte que nos permitan desplazarnos en un entorno fuertemente congestionado. Se prevé que más de 340 millones de personas vivan en entornos urbanos para 2030 [**?**]. La *Movilidad Aérea Urbana* (UAM, de sus siglas en inglés, *Urban Air Mobility*), es un concepto que propone una posible solución para que las urbes del futuro puedan afrontar el reto que supone la congestión del tráfico actual [**?**]. Se trata de un nuevo sistema de transporte aéreo, que puede ser utilizado tanto por personas como para trasportar cargas. En este sentido, está especialmente orientada a operar en zonas densamente pobladas, y en un entorno con una gran cantidad de obstáculos. El progreso de los vehículos de propulsión eléctrica que despegan de forma vertical (eVTOL, de sus siglas en inglés, *electric Vertical Take-Off and Landing*), sugiere que este tipo de movilidad está cada vez más cerca de ser una realidad [**?**].

La *Agencia Europea de Seguridad en la Aviación* (EASA, de sus siglas en inglés, *European Union Aviation Safety*) espera que la movilidad aérea urbana se convierta en una realidad en un plazo de 3 a 5 años. Baterías con más capacidad, redes 5G y nuevas tecnologías aplicadas al despegue vertical van a hacer posible su desarrollo.

Para gestionar este futuro próximo, una nueva infraestructura va a ser sin duda necesaria en nuestras ciudades, dando lugar a conceptos como el de los *vertipuertos*. Estos serán complejos dotados de las herramientas necesarias para llevar a cabo todas las fases operacionales de los eVTOL, como el almacenaje, recarga de baterías, aterrizaje, despegue y gestión de pasajeros y/o cargamento.

Este trabajo pretende abordar el uso de la optimización heurística como herramienta para resolver los problemas asociados al diseño y gestión de los vertipuertos. Debido a la dimensión de estos problemas, utilizar esta herramienta permitirá buscar soluciones en tiempos computacionalmente asumibles y sin necesidad de explorar a fuerza bruta todas y cada una de las posibilidades, en base a un conocimiento experto del problema particular abordado.

1.2.2. Vertipuertos

Las regulaciones europeas trabajan para que este tipo de operación UAM comience y finalice en un vertipuerto [**?**]. Por este motivo, empresas privadas, organismos públicos y académicos, están dedicando recursos económicos y humanos en el desarrollo de lo que será la infraestructura principal de los sistemas UAM. Los vertipuertos, por tanto, serán centros para aeronaves eVTOL, como los taxis aéreos o los drones.

Según la EASA, un vertipuerto se define como: *"Un área de tierra o agua o una estructura utilizadas, o previstas para ser utilizadas, para el aterrizaje y el despegue de aeronaves VTOL"* [**?**].

Un vertipuerto consiste en un *lado aire* y un *lado tierra*. El lado tierra contendrá la terminal de pasajeros, mientras que el lado aire, incluirá diferentes infraestructuras. Entre ellas, las principales son:

1. **FATO**: de sus siglas, en inglés, *Final Approach and TakeOff*, son zonas de aproximación y despegue.

2. **Puerta de embarque**: zonas que permiten aparcar, recargar las baterías, el mantenimiento de la aeronave, y la carga y descarga de pasajeros u otros elementos.

3. **Calles de rodadura** (o *zona de taxi*[2]): zonas de ruta a lo largo del vertipuerto para que la aeronave pueda desplazarse entre las zonas FATO y las puertas de embarque.

Dependiendo de la localización del vertipuerto, existen:

1. **Vertipuertos dentro de un aeropuerto**: quedarían alejados de las pistas de la aviación tripulada del aeropuerto, pero dentro de las instalaciones.

2. **Vertipuertos en superficie**: construidos sobre un terreno a la altura del suelo.

3. **Vertipuertos en cubierta**: en este caso, estarían a la altura del suelo, pero elevados gracias a una estructura que podría permitir, por ejemplo, un aparcamiento disuasorio[3] bajo ella.

4. **Vertipuertos en altura**: construidos sobre algún tipo de edificación como, hospitales o centros comerciales.

En este trabajo, partiremos de suponer la localización y el tamaño del vertipuerto como dados.

Finalmente, introducimos la *topología del vertipuerto*, es decir, cómo aseguramos que las FATO están interconectadas con las puertas de embarques mediante calles de rodadura. En este punto, encontramos dos grandes familias:

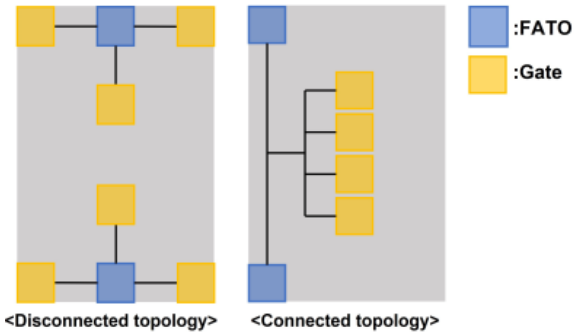

Figura 1.1: Tipos de *topologías* [**?**].

1. **Diseño desconectado**: en este caso, las calles de rodadura se utilizan sólo para conectar cada zona FATO con las diferentes puertas de embarque. De este modo, las FATO no están conectadas entre sí.

2. **Diseño conectado**: todo queda interconectado mediante las calles de rodadura, existiendo caminos entre los diferentes FATO con los que cuenta el vertipuerto.

Podemos ver un croquis de ambas situaciones en la Figura **??**.

En este trabajo, optaremos por un diseño conectado.

[2]Esta expresión viene del inglés, ya que "hacer taxi", en este idioma, hace referencia a una aeronave que se desplaza por el suelo utilizando su tren de aterrizaje.

[3]Estacionamiento ubicado en la corona metropolitana de las ciudades para aquellos pasajeros que intentan acceder al centro de la ciudad.

1.2.3. Optimización del diseño de un vertipuerto

Como nueva infraestructura que formará parte de nuestras ciudades, el vertipuerto presenta varios retos. En este trabajo se han abordado los siguientes:

- Optimización de su diseño.
- Optimización de la planificación.

En ambos casos, la optimización se hará sobre la figura principal del vertipuerto, su *operador*, ya sea para maximizar su beneficio o para mejorar la toma de decisiones de su personal. Definiremos el *operador del vertipuerto* como la entidad responsable de la gestión segura y eficiente de los recursos del vertipuerto. El *operador* del vertipuerto puede tener autoridad sobre la capacidad del operador UAM para aterrizar y despegar [**?**]. Los trabajos que se han consultado para fundamentar los objetivos de esta memoria, [**?**, **?**], proponen la utilización de una optimización heurística para dar respuesta a ambos problemas.

Más concretamente, los autores de [**?**] buscan maximizar el beneficio neto del operador del vertipuerto. Para ello, en primer lugar, se debe elegir una ubicación geográfica. Esta debe ser próxima a áreas donde se exprima el potencial del UAM. Estas áreas representan varios retos, como el elevado precio del suelo y una densidad urbana que se caracteriza por una concentración notable de edificaciones, entre las cuales se encuentran algunas construcciones notablemente altas. Por esto, los vertipuertos deben diseñarse optimizando tanto el espacio como la seguridad. Asumiendo que la localización, el tamaño y la forma (topología) vienen dados, el problema radica en encontrar su diseño óptimo, respetando a su vez las *condiciones de seguridad*.

Durante las operaciones que se llevan a cabo en un vertipuerto, las aeronaves invierten tiempo, por ejemplo, a la hora de embarcar o desembarcar pasajeros. Cada infraestructura de las que conforman un vertipuerto tiene asociado un tiempo de servicio. Dependiendo de las infraestructuras que posea el vertipuerto, este tendrá unos tiempos de servicio que determinarán su capacidad. Igualmente, las operaciones que se llevan a cabo en las FATO dependerán de ciertos criterios de seguridad, como, por ejemplo, la separación mínima entre aeronaves. Por ello, es importante equilibrar también el número de infraestructuras que posee el vertipuerto para conseguir maximizar su rendimiento. En resumen, el número de infraestructuras y su correspondiente rendimiento determinarán la *capacidad en el lado aire*, que se define como el número estimado de operaciones en un determinado tiempo. Este tipo de restricciones se denominan *restricciones operacionales*.

Finalmente, el diseño de vertipuertos tiene asociadas una serie de topologías tradicionales (lineal, satelital o embarcadero [**?**]). No obstante, estas topologías no responden adecuadamente a las demandas de seguridad y las restricciones asociadas a las infraestructuras. Por tanto, un VDP (de sus siglas en inglés *Vertiport Desing Problem*) buscará el diseño óptimo que maximice la capacidad y satisfaga las restricciones operacionales desde el principio del proyecto.

En la Figura **??** ilustramos en un diagrama los diferentes procesos que hemos mencionado hasta obtener el diseño óptimo. Podemos resumir el proceso en tres pasos, que, de acuerdo con [**?**], serían:

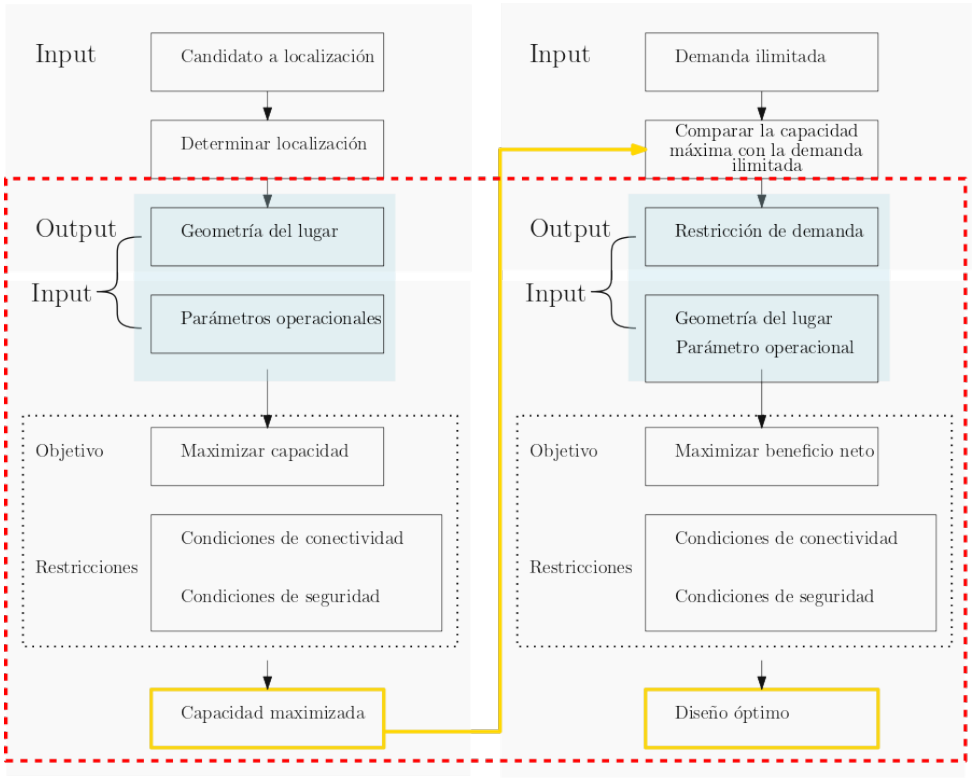

Figura 1.2: Proceso de diseño de un vertipuerto y alcance del trabajo [**?**].

1. Identificar la *capacidad máxima del vertipuerto*.

2. Calcular las *restricciones de demanda*. Suponiendo que la demanda UAM es suficiente, de modo que las restricciones de demanda sean iguales a la capacidad máxima del vertipuerto.

3. Encontrar el diseño óptimo que permita *maximizar el beneficio neto* desde la perspectiva del operador del vertipuerto.

Lo novedoso de estos trabajos radica en la idea de acomodar, dentro del modelo, las restricciones operacionales, mejorando la toma de decisiones del operador del vertipuerto, en contraposición a las topologías tradicionales. Además, contribuirá a realizar una previsión más precisa de la demanda UAM, ya que en estudios precedentes esta demanda se suponía sin restricción o aleatorizada.

Por otro lado, el reto al que se hace frente en [**?**] es la gestión del volumen de aeronaves que hacen uso de las instalaciones. En esta ocasión, la figura principal será el *Vertiport Manager* [**?**], en adelante VM, el cual aún se está definiendo con precisión. No obstante, sus funciones serán parecidas a las de un ATCO (de sus siglas en inglés *Air Traffic Control Officer*), limitadas al vertipuerto en el que desempeñe sus funciones. El objetivo será que las decisiones del VM se sustenten en metodologías que secuencien, de forma adecuada, el flujo de aeronaves entre las componentes del vertipuerto.

En este estudio, se presentan dos modelos matemáticos que, haciendo uso de la optimización heurística, pretenden optimizar las salidas y llegadas. Este problema consiste en asignar y secuenciar los eVTOL en las componentes del vertipuerto: calles de rodadura,

FATO, etc. Para ello, se tienen en cuenta dos objetivos:

- Maximizar el rendimiento.
- Minimizar la desviación del tiempo de llegada/salida respecto del que se había programado inicialmente.

Se tienen en cuenta *condiciones operativas*, como la separación mínima entre eVTOL, o impedir que dos aeronaves ocupen la misma infraestructura de forma simultánea. Respecto a las componentes del vertipuerto introducidas en la subsección **??**, los autores de [**?**] introducen una zona que recuerda a los hangares de un aeropuerto tradicional, y que darían cabida a aquellos eVTOL que estén fuera de servicio.

El esquema elegido es el de un diseño conectado, con una separación entre FATO que permite operaciones simultaneas. Se ha divido el vertipuerto en los siguientes 4 niveles:

- **Nivel 1**: Calle de rodadura 1.
- **Nivel 2**: Puertas de embarque.
- **Nivel 3**: Calle de rodadura 2.
- **Nivel 4**: FATO.

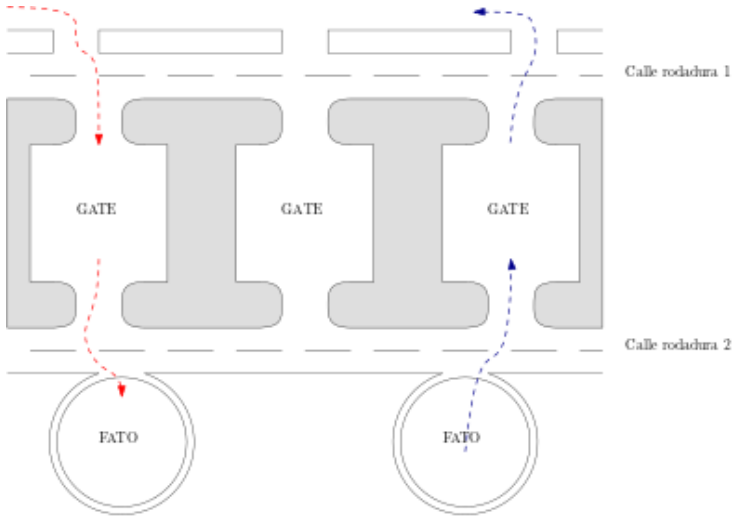

Figura 1.3: Flujos de trabajo en un vertipuerto [**?**].

Como se puede ver en la Figura **??** los eVTOL cuentan con dos operativas básicas:

- *Marcada por las flechas rojas*. Esta operativa consiste en *hacer taxi* hasta el nivel 2 dónde recibirá la carga o se montarán los pasajeros y llegar hasta el nivel 4 para despegar.
- *Marcada por las flechas azules*. Esta operativa es la opuesta a la anterior. La aeronave aterriza en el nivel 4, deposita la carga o los pasajeros en las *gates* y finalmente pasa a la zona de hangares.

En cuanto a las restricciones que se plantean en este problema, destacamos:

- Dos aeronaves no pueden permanecer simultáneamente en la misma infraestructura

dentro del vertipuerto, por razones de seguridad. Asimismo, deberán esperar que el componente esté disponible para transferirse a la siguiente zona.

- Los autores denominan *restricciones de bloqueo* a aquellas situaciones en las que aeronaves con diferente propósito (salida vs. llegada) quieran ocupar la misma calle de rodadura, y quedan, por tanto, confrontadas. Finalmente, en la zona de despegue y aterrizaje (FATO), debe prevalecer un criterio de separación mínima.
- Las limitaciones tecnológicas en las baterías de la aeronave hacen necesario añadir *restricciones relativas a los tiempos de carga*.
- Posibles prioridades para aeronaves de los servicios de emergencia.

De este modo, se introducen los problemas que se denominan HFS, de sus siglas en inglés *Hybrid Flowshop Scheduling*. Estos problemas plantean un entorno en el cual hay un conjunto de tareas que se deben realizar por etapas, de entre las cuales al menos una tendrá más de una *máquina de trabajo en paralelo*.

- En nuestro contexto, las máquinas en paralelo son las diferentes infraestructuras que conforman el vertipuerto.
- Por otro lado, las aeronaves son los trabajos que deben ser asignados y secuenciados en las infraestructuras (máquinas).

Presentados ambos problemas, cabe destacar que:

- El problema VDP queda dentro de los problemas de programación entera IP (de sus siglas en inglés *Integer Programing*) en los cuales el tiempo de cálculo aumenta exponencialmente a medida que el tamaño del problema se incrementa, debido a la naturaleza NP-difícil del problema de programación entera.
- Los problemas HFS, que, en su mayoría de casos, son NP-difíciles [**?**]. Esto se debe a que, incluso en configuraciones relativamente simples (como dos etapas de procesamiento, con dos máquinas en una etapa y una en la otra), el problema sigue siendo NP-difícil [**?**].

En la siguiente subsección, **??**, presentaremos la optimización heurística, siendo está una herramienta adecuada para enfrentar computacionalmente ambos problemas.

1.2.4. Optimización heurística

La *optimización* es una herramienta matemática que ayuda, especialmente, en la toma de decisiones [**?**]. Perteneciente a la rama de la *Investigación Operativa*, forma parte de la vida cotidiana de cualquier empresa, y se utiliza para llegar a una decisión más o menos precisa con ayuda de un modelo matemático. Generalmente, estos problemas se resuelven de forma exacta, ya que se reducen a problemas mixtos de programación lineal, entera o cuadrática, para los que existe software comercial competente como CPLEX de la empresa IBM o Gurobi Optimization. No obstante, algunos de estos problemas no permiten ser abordados mediante estas estrategias, debido a un coste computacional demasiado elevado [**?**, **?**].

Un *problema de optimización* está compuesto de [**?**]:

- Un conjunto de *variables de decisión* que representan los parámetros variables del problema.
- Las *restricciones* las cuales nos dan relaciones entre las variables de decisión y sus valores, permitiendo diferenciar soluciones válidas de las que no lo son.
- Las *funciones objetivo* son una representación matemática de los logros industriales que se desean conseguir.

Podemos entender, entonces, el proceso de optimización como aquel que trata de encontrar los valores de las variables de decisión que cumplan con todas las restricciones, y que *maximizan* o *minimizan* las funciones objetivo. Dependiendo de cómo estos valores se adapten al problema, tendremos diferentes tipos de soluciones [**?**]:

- *Factible*: satisface todas las restricciones.
- *Óptima*: solución factible que alcanza el mejor valor posible de la función objetivo.
- *No factible*: no satisface alguna de las restricciones.
- *No acotada*: produce un valor de la función objetivo que tiende a infinito (*max.* o *min.* respectivamente).

Para dar respuesta a estos problemas, que suponen un coste computacional tan elevado, sería deseable encontrar soluciones factibles lo suficientemente buenas en un tiempo razonable. De esta forma, si bien el objetivo de esta disciplina matemática es encontrar el óptimo del problema, sacrificaríamos dicho valor por una aproximación suficientemente buena del óptimo, en favor de tiempos de cómputo más aceptables. Entre las técnicas capaces de resolver los problemas de esta forma destacan los *algoritmos de optimización heurística*.

El término *heurística* proviene de la palabra griega *eureka*, cuyo significado está relacionado con el concepto de *encontrar*, el cual se vincula a la supuesta exclamación de Arquímedes (287 a.C. - 212 a.C.) al descubrir su famoso principio [**?**]. Según la RAE (Real Academia Española), la heurística se define como:

> *En algunas ciencias, manera de buscar la solución de un problema mediante métodos no rigurosos, como por tanteo, reglas empíricas, etc.*

Estos conceptos serán tratados en el Capítulo **??**. En este trabajo nos centraremos en un tipo concreto de algoritmo heurístico basado en la propia biología, los algoritmos genéticos, debido a que son particularmente útiles en problemas de optimización combinatoria debido a su capacidad para explorar eficientemente espacios de soluciones grandes y complejos, encontrando soluciones óptimas o casi óptimas cuando los métodos de búsqueda tradicionales pueden quedarse atrapados en óptimos locales o ser computacionalmente inviables.

1.3. Estructura de la memoria

La memoria se ha estructurado en seis capítulos. El capítulo actual contiene una introducción dirigida tanto a motivar los objetivos del trabajo como a contextualizar el problema

de estudio. En los capítulos **??** y **??** encontramos descripciones formales tanto de la optimización heurística, como del problema de vertipuertos. Posteriormente, en el Capítulo 4 se encuentran las bases de implementación para algoritmos genéticos en lenguaje Python que sirve, además, como explicación detallada de los algoritmos genéticos que hemos introducido anteriormente. El Capítulo **??** se ha destinado a la puesta en práctica del algoritmo descrito en el Capítulo **??** para la resolución del modelo presentado en el Capítulo **??** con su correspondiente análisis. Finalizamos la memoria con un capítulo de conclusiones y la sugerencia de nuevas líneas de investigación derivadas del trabajo realizado.

En el Apéndice **??** de esta memoria, encontraremos la transcripción de la entrevista previa realizada a una experta en UTM.

1.4. Contribuciones

El diseño de vertipuertos presenta un complejo desafío, ya que involucra múltiples criterios de optimización, que iremos abordando en este trabajo. Tradicionalmente, las metodologías para abordar este problema han recurrido a enfoques exactos, limitando así la exploración de soluciones innovadoras y óptimas en tiempos de cómputo igualmente eficientes. No obstante, la adopción de técnicas de optimización heurística marca un enfoque novedoso y prometedor, ofreciendo una metodología flexible y adaptable capaz de explorar el complejo espacio de soluciones posibles que nos brindan los problemas de optimización combinatoria. Estas nuevas técnicas, como los algoritmos genéticos, permiten identificar configuraciones de diseño altamente eficientes con un mejor coste computacional.

Sin embargo, cabe reconocer que, al ser métodos aproximativos, la optimización heurística puede no siempre alcanzar la solución exacta con toda la precisión deseada. La naturaleza probabilística implica una aceptación del equilibrio entre la calidad de la solución y el tiempo computacional requerido. Además, la posibilidad de aplicar el conocimiento experto es vital cuando nos movemos en escenarios de diseño urbano, donde las variables de decisión y las restricciones pueden ser tan numerosas y complejas que la búsqueda de una solución exacta se vuelve, prácticamente, inviable.

En este trabajo se presenta una implementación en Python, completamente original, de un modelo matemático realista del VDP. Esta implementación puede servir para experimentar, comparar con otras soluciones algorítmicas y, también, como punto de partida para futuras investigaciones.

Finalmente, en el desarrollo del modelo matemático que fundamenta este trabajo, he aplicado de manera integral los conocimientos y competencias adquiridos durante mi formación académica en el grado de Matemáticas. Esta base educativa ha sido esencial para abordar la complejidad del diseño, permitiéndome formular un modelo sólido que incorpora tanto principios de optimización como técnicas formales de programación. Este enfoque ha permitido traducir requisitos prácticos y restricciones técnicas en formulaciones matemáticas, estableciendo así un puente entre la teoría matemática pura y su aplicación en la resolución de un problema realista. Particularmente relevante ha sido mi comprensión de la teoría de complejidad computacional y la naturaleza de los problemas NP-difíciles, adquirida durante mi grado. Este conocimiento ha sido indispensable para buscar estrate-

gias de optimización eficientes. Este trabajo me ha enfrentado a un problema que permite valorar la alternativa a los métodos exactos adoptando un enfoque heurístico, en particular aplicando una estrategia novedosa, para los VDP, como los algoritmos genéticos.

Capítulo 2

Optimización Heurística

Este capítulo está dedicado a la optimización, en particular, a su vinculación con la rama de las heurísticas. Dedicaremos la atención a las *heurísticas evolutivas*, en concreto, a los *algoritmos genéticos*. Introduciremos, en primer lugar, esta técnica matemática desde el punto de vista histórico, presentando sus principales componentes, para, finalmente, describir su estructura general. Las principales fuentes de consulta que han servido para la elaboración del capítulo son [**?**, **?**, **?**, **?**, **?**, **?**].

2.1. De la realidad a la solución

En la subsección **??** comentamos brevemente los ingredientes principales de un problema de optimización, así como el concepto de *heurística*. Si bien conocer los esquemas de optimización es importante, que estos respondan a un *modelo* es crucial dentro de la disciplina. Según la RAE, este modelo se define como:

> *"Esquema teórico, generalmente en forma matemática, de un sistema o de una realidad compleja (por ejemplo, la evolución económica de un país), que se elabora para facilitar su comprensión y el estudio de su comportamiento."*

Modelar, por tanto, es la acción de construir un modelo, de "encorsetar" la realidad [**?**]. Un modelo debe plantearse nivelando su cantidad de detalles con la factibilidad de encontrar alguna técnica para resolverlo. Su desarrollo se divide en las siguientes etapas:

- **Identificación del problema**. Consiste en buscar y analizar la información del problema. Hay que ser preciso, de forma que lo recopilado pueda expresarse en lenguaje matemático. Es una etapa fundamental, pues de ella depende la calidad de las conclusiones futuras.
- **Especificación matemática y formulación**. Se escribe el problema de optimización definiendo sus variables, ecuaciones, función objetivo y parámetros.
- **Resolución**. En esta etapa se utilizará un algoritmo para llegar a la solución numérica (idealmente cercana a la matemática) óptima o casi-óptima. Este algoritmo puede tener un propósito general (e.g., el *método del Simplex*) o específico para el modelo.

- **Verificación, validación y refinamiento**. En esta fase se pretende eliminar los errores que se hayan podido dar en la codificación (esto es, verificar). Es necesario comprobar la validez de las simplificaciones realizadas utilizando las soluciones obtenidas (esto es, validar), o comprobando que los resultados son coherentes en la realidad.
- **Interpretar y analizar resultados**. Se proponen las soluciones. Se realiza un análisis de sensibilidad de los parámetros de entrada, lo que permite conocer el comportamiento del modelo. También se comprueba la robustez de la solución óptima.
- **Implementación, documentación y mantenimiento**. El objetivo de esta etapa es garantizar la difusión a través de una documentación clara, precisa y completa, así como un código bien documentado que facilite su mantenimiento.

A su vez, en [**?**] se presenta, en la Figura **??**, el uso de modelos matemáticos para ayudar en la toma de decisiones en situaciones del mundo real, y lo dividen en cuatro etapas:

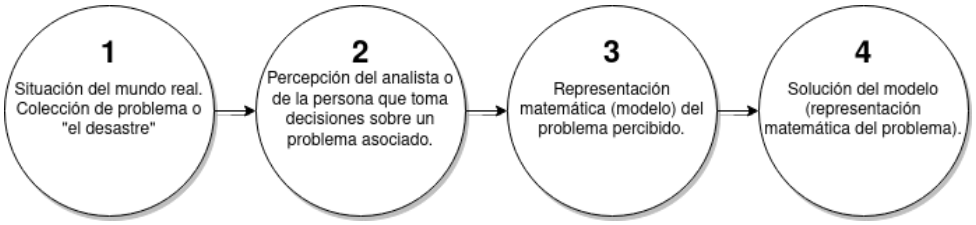

Figura 2.1: De la situación real a la solución.

Finalmente, en este camino, necesitaremos dos ingredientes más: un conjunto de datos que permita crear las diferentes iteraciones del modelo, y saber de qué forma resolvemos el problema. Desgraciadamente, la complejidad del mundo real nos lleva a modelos igualmente complejos que no pueden ser abordados con métodos exactos.

2.2. Algoritmos exactos de optimización matemática

La *optimización* representa el proceso de encontrar la *mejor solución* a un problema entre un conjunto muy grande de soluciones posibles [**?**]. La teoría de optimización clásica está enfocada en encontrar e identificar al mejor candidato de entre un conjunto de alternativas, sin evaluarlas explícitamente; por ello, un *problema de optimización* es, en general, un *problema de decisión* [**?**].

Un *problema de optimización* es un proceso donde se quiere encontrar el valor óptimo $\mathbf{x} = (x_1, \ldots, x_d) \in \mathbf{X}$ que *minimiza* o *maximiza* la función objetivo $f(\mathbf{x})$, donde $\mathbf{X}$ representa el conjunto de soluciones o *espacio de búsqueda*. Este espacio de búsqueda puede estar limitado inferiormente por el valor l_i, y superiormente por un valor u_i, de forma que $\mathbf{X} = \{\mathbf{x} \in \mathbb{R}^d \,|\, l_i \leq x_i \leq u_i,\ i = 1, \ldots, d\}$. Finalmente, tendrá una serie de restricciones que pueden ser de desigualdad o de igualdad. Formalmente:

$$\begin{aligned} &\text{mín} \,/\, \text{máx} \\ &\qquad f(\mathbf{x}),\ \mathbf{x} \in \mathbb{R}^n \\ &\text{sujeto a las restricciones} \\ &\qquad g_i(\mathbf{x}) \leq 0, \quad i = 1, \ldots, q \\ &\qquad h_j(\mathbf{x}) = 0, \quad j = q+1, \ldots, k \\ &\qquad l_l \leq \mathbf{x}_l \leq u_l, \quad l = 1, \ldots, n \end{aligned} \tag{2.1}$$

Problemas de optimización hay tantos como formas de expresar $f(\mathbf{x})$. Las *técnicas de optimización* nos permiten encontrar el óptimo en el espacio de búsqueda mediante procesos iterativos. Existen dos grandes familias de algoritmos a los que vamos a hacer referencia en este capítulo: *esquemas clásicos* y *esquemas metaheurísticos*.

Los métodos clásicos, como los *métodos Newtonianos*, utilizan el gradiente de la función $f(\mathbf{x})$ para el proceso de iteración, mientras que los métodos heurísticos, como los *algoritmos evolutivos*, ignoran la derivada y orientan la búsqueda gracias a unas *reglas* que permiten acotar y dirigir la exploración.

2.3. Optimización heurística

En esta subsección, daremos el marco general de las heurísticas para centrarnos en las heurísticas evolutivas.

2.3.1. Heurística y Metaheurística

Existen diferentes tipos de algoritmos heurísticos, que se diferencian en su forma de *buscar* y *construir* las soluciones. Según [**?**], las cuatro familias más importantes en las que se pueden dividir son:

- *Heurísticas de relajación*: son aquellas heurísticas que obtienen la solución del problema de optimización disminuyendo la información capturada por el modelo, es decir, simplificando el problema original.
- *Metaheurísticas constructivas*: están formadas por métodos que obtienen la solución del problema analizando y seleccionando, iterativamente, las componentes que la forman.
- *Metaheurísticas de búsqueda* son métodos que recorren el espacio de soluciones del problema, transformando de forma iterativa y, por medio de alguna regla "inteligente", una solución de partida en otra.
- *Metaheurísticas evolutivas*: son procedimientos que, mediante operadores, hacen cambiar los valores de un conjunto de soluciones, generando así nuevos conjuntos cada vez más aptos, y de entre los cuales se extraerá, al final del proceso iterativo, la *mejor* solución para ser la óptima del problema.

En las siguientes subsecciones del capítulo profundizaremos en las heurísticas evolutivas, ya que la técnica elegida para resolver el problema propuesto en el Capítulo **??** pertenece a este grupo.

2.3.2. Heurísticas evolutivas

Las metaheurísticas evolutivas establecen estrategias para dirigir, en el espacio de búsqueda, la evolución de un conjunto (llamado, habitualmente, *población*) de posibles soluciones del problema, con la intención de acercar los elementos de dicho conjunto a la solución óptima del problema [**?**]. Las características principales son:

- Exploran el espacio de soluciones, utilizando, en cada iteración, un conjunto de soluciones.

- Los operadores que modifican los valores de cada solución tienen en cuenta al resto de las soluciones de la población, y las funciones de ajuste asociadas a las mismas. De esta forma, las soluciones de la población interactúan entre sí, y evolucionan teniendo en cuenta las posibles interacciones. No obstante, en ocasiones se incorporan algunos artificios adicionales, que modifican las soluciones existentes sin tener en cuenta los valores del resto, para llegar a zonas del espacio de búsqueda que no son alcanzables por un camino más ortodoxo. La idea subyacente es evitar el problema del *óptimo local*. Un *óptimo local* es un vector de decisión factible que representa un óptimo local de la función objetivo si su valor en ese punto no es peor que el de sus vecinos. Esto puede hacer que nos quedemos atascados entre esos *vecinos*.

- La población se sustituye al final de cada iteración por una nueva, formada por las soluciones más aptas que han sido creadas. En algunos algoritmos, se utiliza una memoria, que guarda las mejores soluciones, de forma que puedan ser tenidas en cuenta durante la evolución.

- Las diferentes metaheurísticas evolutivas se distinguen por la forma en la que combinan la información proporcionada por los elementos de la población [**?**].

La Figura **??** presenta el proceso de optimización que siguen las heurísticas evolutivas [**?**]. Un algoritmo mantiene una población $\mathbf{P}^k(\mathbf{x}_1^k, \mathbf{x}_2^k, \ldots, \mathbf{x}_N^k)$ de N soluciones candidatas, las cuales *evolucionarán* un máximo de `nIter` iteraciones desde la *población inicial* hasta la solución final (minimiza o maximiza la aptitud obtenida de la población final). Este tipo de algoritmos se inicializan con una población de candidatos elegidos de forma aleatoria en $\mathbf{X}$. Para cada generación, un conjunto de operadores (que son distintos, dependiendo de la filosofía del algoritmo) se aplica sobre la población para obtener la nueva población, $\mathbf{P}^{k+1}$. La calidad de la solución se obtiene al evaluar la función objetivo $f(\mathbf{x}_i^k)$, con $i \in \{1, 2, \ldots, N\}$, para cada uno de los integrantes de la población. Así, la mejor solución m aportada por todas las soluciones candidatas mantendrá un lugar especial. Al final del proceso, dicha posición supondrá la solución del problema.

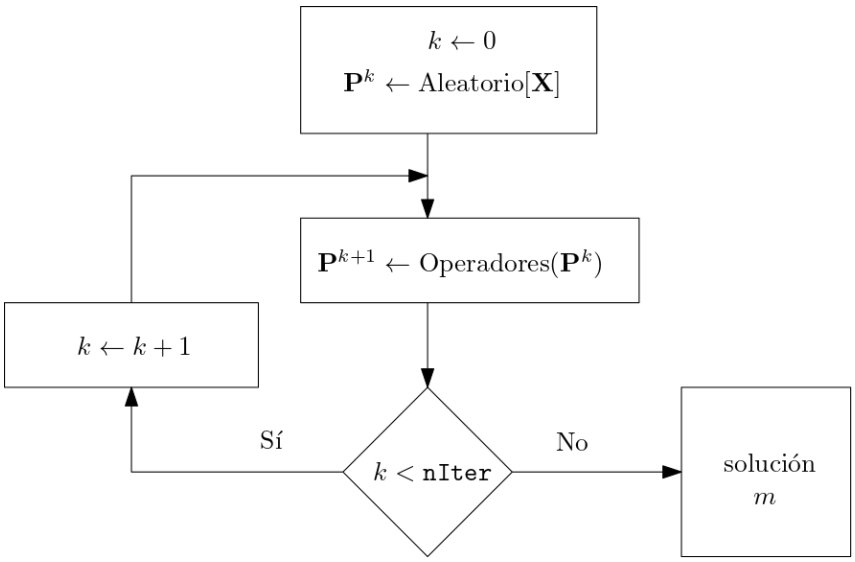

Figura 2.2: Proceso de optimización.

2.4. Algoritmos genéticos

Los algoritmos genéticos pertenecen a las metaheurísticas evolutivas que se definieron en la subsección **??**. En esta subsección, vamos a profundizar en esta técnica, que será la elegida para la implementación del Capítulo 4.

Los *algoritmos genéticos*, en adelante GA (del inglés, *Genetic Algorithms*), son un grupo de algoritmos que se inspiran en la teoría de la selección de la especies de Darwin (1809 - 1882). La selección natural es la base de todo el cambio evolutivo. Es el proceso a través del cuál, los organismos mejor adaptados a su entorno tienen mayor probabilidad de sobrevivir y reproducirse. Esto es debido a la acumulación de las características más positivas en la población a lo largo del tiempo. La información genética de los hijos es heredada de los padres durante el proceso de *cruce*, aunque en algunos casos es ligeramente modificada debido a algún mecanismo de *mutación* [**?**].

El artículo en el que John Holland (1929 - 2015) introdujo los Algoritmos Genéticos se titula *Outline for a logical theory of adaptive systems* [**?**]. Fue publicado en 1962 y sentó las bases para el desarrollo de los algoritmos genéticos como una metodología para la optimización y la búsqueda basada en los principios de la evolución biológica y la selección natural. En la Figura **??** podemos ver las relaciones conceptuales que estableció Holland.

Partimos de un problema de optimización con restricciones como el de la Definición **??**. El GA pretende encontrar la solución óptima de este mediante una serie de funciones que denominados *operadores genéticos* [**?**]:

- **Selección**: debe permitir que los hijos mejor adaptados tengan, a su vez, mayor descendencia.
- **Cruce**: combina los valores en los genes de los hijos para crear nuevos individuos.
- **Mutación**: modificará a algunos de los hijos de forma aleatoria.

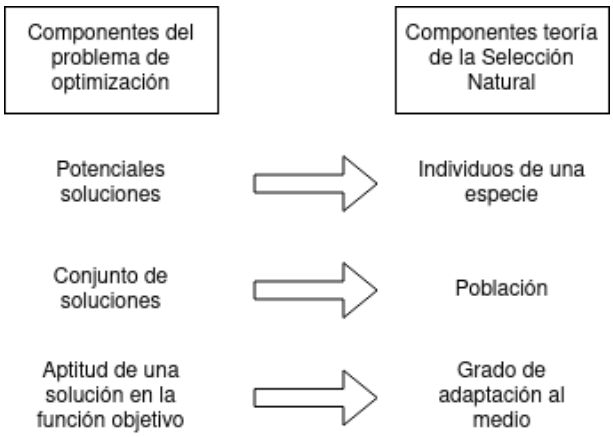

Figura 2.3: Relación de conceptos entre la selección natural y los GA.

Podemos ver en el Algoritmo **??** el pseudocódigo básico de un GA, y en la Figura **??** su diagrama de trabajo, con el que podemos observar cómo se interrelacionan los operadores. En el pseudocódigo encontramos: x, que representa a la población (es decir, un conjunto de soluciones del problema) actual, que suele tener un tamaño prefijado y constante a lo largo de la ejecución; por otro lado, *Parents* representa un conjunto de parejas de individuos seleccionados a partir de la población actual; finalmente, *Children* serán los nuevos individuos creados a partir del cruce de las parejas de padres. En cuanto a las funciones, tenemos, en primer lugar a *Generate*(), cuya función es dar una población inicial; *Select*(), que elige las parejas de padres, según un cierto método, de entre la población actual x; *Cross*() y *Mutate*(), cruzan y mutan, respectivamente; *Recombine*() se encargará de, en función de la aptitud de cada individuo, elegir entre las soluciones de la población x y los hijos; finalmente, *Stop*() tendrá codificada la condición de parada para finalizar el algoritmo.

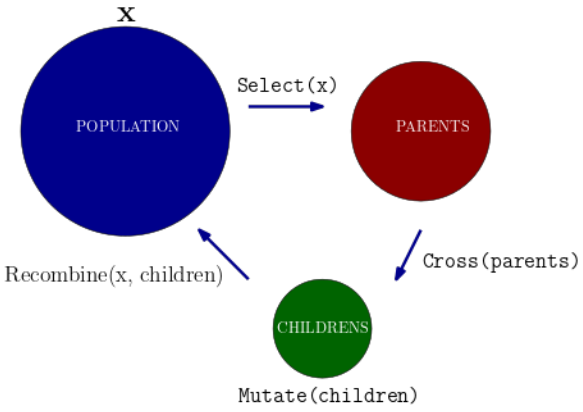

Figura 2.4: Diagrama de trabajo de un GA.

[h] FGenGenerate FSelectSelect FCrossCross FMutateMutate FRecombineRecombine FStopStop FnFunction: FnProcedure: genetic_algorithm(Generate, Select, Cross, Mutate, Recombine, Stop) $x =$ x $parents = x$ $children = parents$ $children = children$ $x = x, children$

Capítulo 3

Optimización del Diseño de un Vertipuerto

En este capítulo nos vamos a centrar en la modelización matemática del problema de diseño de un vertipuerto. En primer lugar, se presentan las consideraciones técnicas para el diseño de la infraestructura para, posteriormente, presentar el modelo matemático, así como una explicación detallada de cada una de sus componentes. Las principales fuentes de consulta que han servido para la elaboración del capítulo son [**?**, **?**, **?**]. Además, en este capítulo destacaremos la aportación especial de la entrevista realizada a una de las responsable técnicas del equipo en el que trabajo actualmente en la empresa *Indra Sistemas S.A*. Esta entrevista puede ser consultada en el Apéndice **??**, y nos servirá para comprender mejor cómo se han modelizado las restricciones del modelo matemático.

3.1. Problema de Diseño de un Vertipuerto

En esta subsección, vamos ver cómo se caracteriza la solución del problema de diseño de un vertipuerto en la línea de trabajos previos relacionados con [**?**]. Además, se darán algunas consideraciones generales del diseño de un vertipuerto, así como la descripción matemática del modelo.

3.1.1. Caracterización de la solución

Vamos a presentar, a continuación, la formulación matemática del modelo que pretende resolver el problema de diseño. En [**?**], los autores caracterizan el problema a través de una cuadrícula, como se describe en la Figura **??**, cuya unidad básica denominaremos *celda*. Esta puede ser asignada a cualquier infraestructura del vertipuerto (es decir, una celda puede representar una FATO, una puerta de embarque, etc.). Estas celdas son cuadrados de 20 metros de largo, formando una cuadrícula de 8×5 celdas (las dimensiones de un problema real serán 100 metros de ancho por 160 metros de largo).

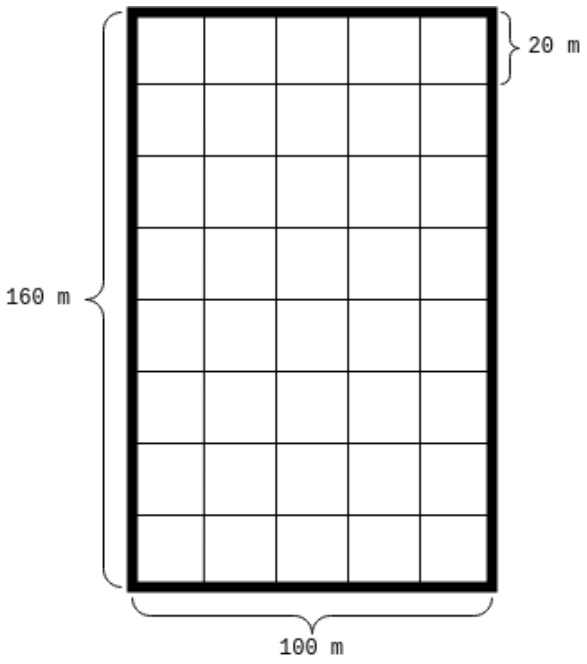

Figura 3.1: Caracterización del problema a través de una cuadrícula.

3.1.2. Consideraciones del diseño de optimización

Para definir adecuadamente el modelo matemático, hay que tener en cuenta algunas consideraciones previas en materia de seguridad operacional y desempeño de cada infraestructura. Para ello, los autores de [**?**] utilizan el documento *Draft EB 105, Vertiport Design* del 21 de septiembre de 2022, elaborado por la FAA (*Federal Aviation Administration*) de Estados Unidos. Dicho documento también es mencionado por la experta en UTM en la entrevista incluida en el Apéndice **??**. En este documento, se proponen las dimensiones de las infraestructuras, así como las *condiciones de seguridad* y conectividad[1]. En lo sucesivo, CD (o simplemente D), representará la dimensión de control estándar que tomaremos como el diámetro del círculo circunscrito sobre el VTOL en la proyección horizontal del plano, cuando el aparato está en su configuración de despegue o aterrizaje. También se puede definir como la mayor distancia entre dos puntos opuestos de la aeronave (véase la Figura **??**).

En base a este concepto, los autores tienen en cuenta los siguientes factores:

- El ancho y largo mínimos de una FATO es 2CD unidades de la aeronave de referencia.
- Aunque las dimensiones de las vías y las puertas de embarque no están presentes en [**?**], los autores tomaron como base el documento *Heliport Design* [**?**], del cual se infieren las siguientes dimensiones:
  - El ancho mínimo de una vía será 2CD.
  - El ancho y largo mínimos de una puerta de embarque será 1CD.

[1]Europa tiene un documento similar llamado *Vertiports Prototype Technical Specifications for the Design of VFR Vertiports for Operation with Manned VTOL-Capable aeronave Certified in the Enhanced Category*, publicado por la EASA en 2022 [**?**].

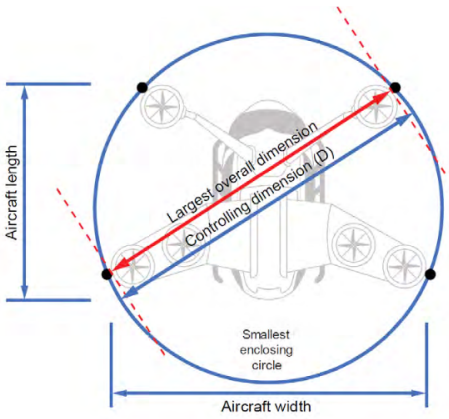

Figura 3.2: Dimensión de control [**?**].

- La separación mínima entre puertas de embarque será $\frac{1}{3}$CD.
- La terminal de pasajeros es más grande que las puertas de embarque.

En base a esto, las celdas que componen la cuadrícula serán 2CD $\times$ 2CD.

Respecto a las *condiciones de seguridad*, los autores las consideran como las más importantes a tener en cuenta, lo cual es coherente con la filosofía del sector. Se tienen en cuenta las siguientes consideraciones:

- Los VTOL deben aproximarse al vertipuerto esquivando los edificios que conforman el entorno urbano. Por este motivo, las FATO deben ubicarse teniendo en cuenta las rutas de aterrizaje y despegue. En este estudio, se identifica un conjunto de ubicaciones potenciales para las FATO mediante la evaluación de los obstáculos alrededor del sitio candidato para ubicar el vertipuerto.
- Por el propio concepto de UAM, el impacto nocivo que pueda generar un vertipuerto en los alrededores de su ubicación debe ser también un factor tenido en cuenta. En particular, las instalaciones terrestres no deben estar ubicadas alrededor de las rutas de aproximación/salida.

Finalmente, respecto a las condiciones de conectividad, se presentan tanto desde la perspectiva de aeronaves como de pasajeros del UAM. Estas condiciones se dividen en dos: las orientadas al vehículo, que aseguran que las FATO deben estar interconectadas con las puertas, para permitir a las aeronaves dirigirse desde las zonas de embarque hasta las FATO, y viceversa; y las que se centran en los pasajeros, que aseguran que se exijan áreas mínimas para las terminales de pasajeros adyacentes a la puerta de embarque; esto se debe a que los pasajeros sólo pueden acceder a la aeronave por las puertas de embarque. Los autores de [**?**] toman, así, una dimensión equivalente a dos celdas.

3.1.3. Definición del problema de optimización

El problema de diseño de vertipuertos, que se formaliza a continuación, se presenta formalmente a través de un problema de optimización matemática de programación entera en **??** a **??**. Las *variables de decisión* elegidas $x_{i,j}^{(k)}$ serán binarias. Diremos que es igual a

1 si la infraestructura k está localizada en la fila i y columna j, es decir, la celda (i, j) de la cuadrícula; es igual a 0 en cualquier otro caso. Tendremos el conjunto de índices

$$\mathcal{T} = \{F, G, T, P\} \tag{3.1}$$

donde F son las FATO, G las puertas de embarque, T las calles de rodadura, y P las terminales de pasajeros. Finalmente, el conjunto $\mathcal{C}$ representa todas las celdas de la cuadrícula. Se han elegido estas basándose principalmente en [**?**] y en la entrevista realizada en el Apéndice **??**.

La *función objetivo* (que se formaliza, a continuación, en **??**) maximiza el beneficio del operador del vertipuerto, el cual se modeliza fácilmente como la diferencia entre los costes e ingresos totales. Estos se calculan del modo siguiente:

- **Ingresos**: son calculados como el producto de la *tasa de uso por avión* (f), el *número de aeronaves* que están usando el vertipuerto (i.e., capacidad del vertipuerto, v), y el ciclo de vida de la infraestructura (l).
- **Costes**: el *coste de construcción* de la infraestructura de tipo k se denota por $c_{\text{con}}^{(k)}$, mientras que el *coste de operación del vertipuerto por ventana temporal* (*slot*) es c_{ops}.

3.2. Modelo de optimización

Una vez se han definido las componentes principales del modelo matemático, el problema de optimización viene dado por la siguiente formulación matemática:

Maximizar

$$f \cdot l \cdot v - \left[\sum_{k \in \mathcal{T}} c_{\text{con}}^{(k)} \sum_{(i,j) \in \mathcal{C}} x_{i,j}^{(k)} + c_{\text{cops}} \cdot l \right] \tag{3.2}$$

Sujeto a las restricciones

$$v \leq \frac{t_{\text{win}}}{t^{(F)}} \sum_{(i,j) \in \mathcal{C}} x_{i,j}^{(F)} \tag{3.3}$$

$$v \leq \frac{t_{\text{win}}}{t^{(G)}} \sum_{(i,j) \in \mathcal{C}} x_{i,j}^{(G)} \tag{3.4}$$

$$\sum_{k \in \mathcal{T}} x_{i,j}^{(k)} \leq 1, \quad \forall (i,j) \in \mathcal{C} \tag{3.5}$$

$$x_{i-1,j}^{(P)} + x_{i+1,j}^{(P)} + x_{i,j-1}^{(P)} + x_{i,j+1}^{(P)} \geq x_{i,j}^{(G)}, \quad \forall (i,j) \in \mathcal{C} \tag{3.6}$$

$$x_{i-1,j}^{(P)} + x_{i+1,j}^{(P)} + x_{i,j-1}^{(P)} + x_{i,j+1}^{(P)} \geq x_{i,j}^{(P)}, \quad \forall (i,j) \in \mathcal{C} \tag{3.7}$$

$$x_{i-1,j}^{(T)} + x_{i+1,j}^{(T)} + x_{i,j-1}^{(T)} + x_{i,j+1}^{(T)} \geq x_{i,j}^{(G)}, \quad \forall (i,j) \in \mathcal{C} \tag{3.8}$$

$$x_{i-1,j}^{(T)} + x_{i+1,j}^{(T)} + x_{i,j-1}^{(T)} + x_{i,j+1}^{(T)} \geq x_{i,j}^{(F)}, \quad \forall (i,j) \in \mathcal{C} \tag{3.9}$$

$$\sum_{k\in\mathcal{T}\setminus\{P\}} (x_{i-1,j}^{(k)} + x_{i+1,j}^{(k)} + x_{i,j-1}^{(k)} + x_{i,j+1}^{(k)}) \geq 2x_{i,j}^{(T)}, \quad \forall (i,j) \in \mathcal{C} \tag{3.10}$$

$$\sum_{k\in\mathcal{T}} \sum_{(i,j)\in\mathcal{D}} x_{i,j}^{(k)} = 0 \tag{3.11}$$

$$\sum_{k\in\mathcal{T}\setminus\{T\}} \sum_{(s,t)\in\mathcal{N}(i,j)} x_{st}^{(k)} \leq M \cdot (1 - x_{i,j}^{(F)}), \quad \forall (i,j) \in \mathcal{C} \tag{3.12}$$

$$\sum_{k\in\mathcal{T}\setminus\{T\}} \sum_{(s,t)\in\mathcal{M}(i,j)} x_{st}^{(k)} \leq M \cdot (1 - x_{i,j}^{(F)}), \quad \forall (i,j) \in \mathcal{C} \tag{3.13}$$

$$x_{i,j}^{(k)} \in \{0,1\}, \quad \forall (i,j) \in \mathcal{C}, \forall k \in \mathcal{T} \tag{3.14}$$

donde, los *parámetros del modelo* que aparecen en las dos primeras restricciones se definen como sigue

Parámetro	**Descripción**	**Unidad**
t_{win}	ventana de tiempo	minutos
$t^{(F)}$	separación mínima	minutos
$t^{(G)}$	cambio de rumbo	minutos

Cuadro 3.1: Parámetros del modelo.

Función objetivo

En primer lugar, a partir de la Figura[2] **??**, vamos a explicar con mayor profundidad tanto la función objetivo como cada una de las restricciones del problema.

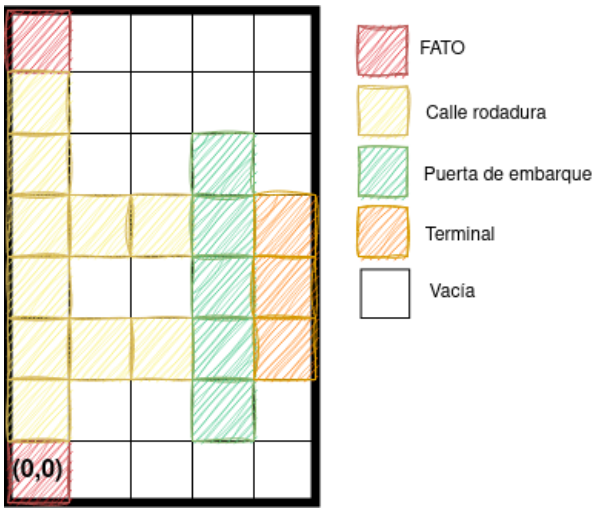

Figura 3.3: Representación del diseño del ejemplo.

Partimos de la función objetivo (**??**). Una de las fuentes principales que ayudan a entender la formulación de su definición se extrae de la propia entrevista realizada en el Apéndice

[2]Esta representación sirve para acompañar la explicación del modelo, por lo que no pretende ser una solución concreta que cumpla las restricciones del modelo.

??. Encontramos las constantes f, l y v, que tomarán los valores de forma acorde a la ubicación y criterios del operador del vertipuerto. A continuación, tenemos la suma del coste de la infraestructura k por aquellas celdas en las que se encuentra ubicada. Finalmente, se suma el producto del coste por ventana de tiempo por el ciclo de vida de la infraestructura. En el desglose del sumatorio, para nuestro ejemplo concreto de la Figura **??**, y según el modelo de su infraestructura, se expandiría como sigue:

- **FATO**

$$s_{\mathrm{F}} = c_{\mathrm{con}}^{(F)} \cdot \left(x_{00}^{(F)} + x_{09}^{(F)} \right) \tag{3.15}$$

- **Calle de rodadura**

$$s_{\mathrm{T}} = c_{\mathrm{con}}^{(T)} \cdot \left(x_{01}^{(T)} + x_{02}^{(T)} + x_{03}^{(T)} + x_{04}^{(T)} + x_{05}^{(T)} + x_{06}^{(T)} + x_{12}^{(T)} + x_{14}^{(T)} + x_{22}^{(T)} + x_{24}^{(T)} \right) \tag{3.16}$$

- **Puerta de embarque**

$$s_{\mathrm{G}} = c_{\mathrm{con}}^{(G)} \cdot \left(x_{41}^{(G)} + x_{42}^{(G)} + x_{43}^{(G)} + x_{44}^{(G)} + x_{45}^{(G)} \right) \tag{3.17}$$

- **Terminales de pasajeros**

$$s_{\mathrm{P}} = c_{\mathrm{con}}^{(P)} \cdot \left(x_{52}^{(P)} + x_{53}^{(P)} + x_{44}^{(P)} \right) \tag{3.18}$$

Juntando todos los términos, obtendremos que evaluar la función objetivo corresponde a evaluar la siguiente expresión

$$f \cdot l \cdot v - (s_{\mathrm{F}} + s_{\mathrm{T}} + s_{\mathrm{G}} + s_{\mathrm{P}} + c_{\mathrm{cops}} \cdot l) \tag{3.19}$$

Restricciones

Las restricciones (**??**) y (**??**) limitan la capacidad del vertipuerto en función de las FATO y las puertas de embarque, respectivamente. En estas restricciones encontramos los parámetros de las operaciones del Cuadro **??**.

Una restricción propia de la caracterización que se ha llevado cabo es la que encontramos en (**??**), asegurando que cada celda de la cuadrícula no se asignará a más de un tipo de la infraestructura.

Las restricciones (**??**)-(**??**) fueron comentadas anteriormente como *condiciones de conectividad*. Éstas se han divido en dos grupos, de acuerdo a la entrevista realizada en el Apéndice **??**: *condiciones orientadas a los pasajeros*, donde cada puerta debería ser adyacente a una terminal de pasajeros (la terminal de pasajeros debe constar de, al menos, dos celdas), y *condiciones orientadas a las aeronaves*, en las que se pretende asegurar la interconectividad entre las infraestructuras que conforman el vertipuerto. Las variables de decisión $x_{i,j}^{(k)}$, con $k \in \mathcal{T}$, vimos que determinan si en la celda (i, j) se construye (o no) la infraestructura k, que puede ser una FATO, una puerta de embarque, una calle de rodadura, o una terminal de pasajeros. Las restricciones restantes se interpretan del siguiente modo:

- **Restricciones** (**??**) y (**??**): si hay una celda de puerta de embarque en (i, j), al menos una de las celdas adyacentes debe ser una celda de pasajero. En la Figura **??** podemos ver que la restricción se cumple en

$$x_{22}^{(P)} + x_{42}^{(P)} + x_{31}^{(P)} + x_{33}^{(P)} \geq x_{32}^{(G)} \Rightarrow 0 + 1 + 0 + 0 \geq 1 \tag{3.20}$$

  pero no es así para

$$x_{21}^{(P)} + x_{41}^{(P)} + x_{30}^{(P)} + x_{32}^{(P)} \geq x_{31}^{(G)} \Rightarrow 0 + 0 + 0 + 0 \geq 1 \tag{3.21}$$

  por tanto, observamos que el ejemplo contiene una casilla que no verificaría la restricción (**??**).
- **Restricción** (**??**): cada puerta debe estar conectada a una calle de rodadura.
- **Restricción** (**??**): cada FATO debe estar conectado a una calle de rodadura.
- **Restricción** (**??**): si hay una celda de terminal de pasajeros en (i, j), al menos dos de las celdas adyacentes pueden ser de cualquier tipo menos de terminal de pasajeros, evitando así callejones sin salida.

Las restricciones que acabamos de comentar suponen un problema para las celdas en los extremos. Por ejemplo, la celda (0,0) no tiene celdas adyacentes ni a su izquierda ni debajo. Para solucionar esta situación, en [**?**] se propone añadir un conjunto de celdas ficticias, denominadas *dummy*. En la Figura **??**, se observa cómo quedaría la nueva situación al añadir este tipo de celdas, representadas mediante puntos discontinuos.

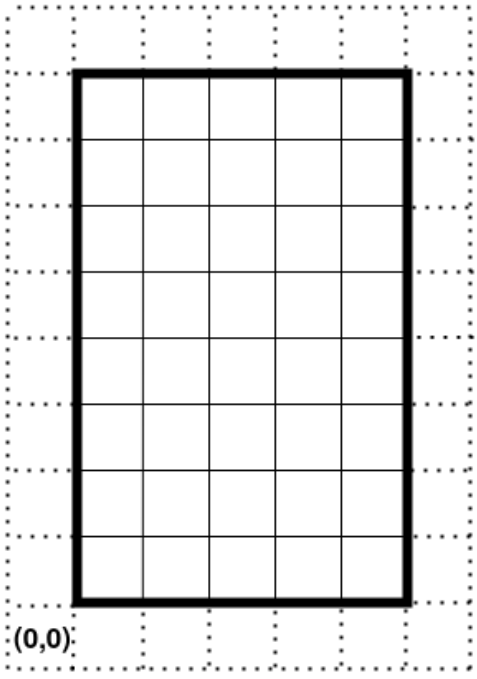

Figura 3.4: Celdas *dummy*.

El conjunto $\mathcal{D} \subset \mathcal{C}$ estará formado por todas estas celdas *dummy*.

Finalmente, las restricciones (**??**) y (**??**) utilizan un valor M suficientemente grande para representar términos *big-M*, una técnica usual para este tipo de problemas[3]. La primera

[3]Se puede consultar más información sobre este método en la referencia [**?**].

de las restricciones asegura que una FATO en las celdas directamente adyacentes no puede tener infraestructuras críticas (i.e., solo calles de rodadura). Se define el *conjunto de celdas adyacentes* a la celda (i, j) como

$$\mathcal{N}(i,j) = \{(i-1,j),\ (i+1,j),\ (i,j-1),\ (i,j+1)\} \tag{3.22}$$

Matemáticamente, se suman todos los tipos de celdas excepto las de tipo T (las calles de rodadura). Si esa celda (i, j) es una FATO, la suma de las celdas adyacentes de tipos no deseados deberá ser 0. El término $M \cdot (1 - x_{i,j}^{(F)})$ es una estrategia matemática habitual para representar la siguiente afirmación: si $x_{i,j}^{(F)} = 1$ (es decir, la celda es una FATO), el término *big-M* se anula, haciendo que la suma pase a ser 0. Por último, la restricción (**??**) garantiza que las FATO, puertas de embarque y terminales de pasajeros, no se coloquen en (o cerca de) los caminos de llegada/salida.

Capítulo 4

Implementación en Python de un GA para el VDP

En este capítulo, vamos a implementar un algoritmo genético que de respuesta al *Problema de Diseño de un Vertipuerto* (VDP), que presentamos en el Capítulo **??**. En primer lugar, vamos a caracterizar la solución de forma que se adapte adecuadamente al GA, discutiremos su estructura *genómica*, y daremos detalles de cómo se ha implementado esta en Python. Introduciremos los distintos *operadores genéticos* que emplean los GA y veremos una forma de calcular la *aptitud* de los individuos mediante la penalización asociada a las distintas restricciones del modelo matemático descrito en el Capítulo **??**. Finalmente, veremos dos formas de *recombinar* las poblaciones, es decir, de como se actualiza la población a través de las generaciones hasta llegar a la función de parada del GA. Las principales fuentes de consulta que han servido para la elaboración del capítulo son [**?**, **?**, **?**, **?**, **?**].

4.1. Algoritmos genéticos para el VDP

En esta subsección, vamos a ver cómo se ha caracterizado la solución, el proceso para implementar los operadores genéticos, cómo se extrae la aptitud de un individuo, y cómo se penaliza dicho valor si el individuo no cumple alguna de las restricciones. Finalmente, veremos diferentes opciones para detener la ejecución del algoritmo diseñado.

4.1.1. Caracterización de la solución

Una de las características diferenciales en los GA es ver como la codificación de las soluciones afecta al tipo de operador[1]. Algunas de las codificaciones más usuales son las siguientes:

- **Binaria**. En este grupo, las variables de decisión del problema se caracterizan me-

[1]Durante el resto de capítulo, *operador* hará referencia a los métodos de *cruce* y *mutación*.

diante cadenas de *bits* finitas[2].

- **Entera**. Las variables de decisión se representan con cadenas de números enteros. La longitud de la cadena dependerá, de nuevo, del tipo de variable.
- **Real**. Se utilizan los tipos que brindan los lenguajes de programación (`float` y `double`).
- **Codificación Natural**. En ocasiones, se puede utilizar estructuras de datos directamente ligadas al problema, como grafos o árboles. La elección de una codificación íntimamente relacionada con el problema puede evitar que los operadores de cruce y mutación, que son los responsables de crear y modificar los valores de las posibles soluciones del problema, generen soluciones no factibles [**?**].

Es importante destacar que Holland presentó estos métodos pensando únicamente en la representación binaria, ya que ofrece un paralelismo *ad hoc*, llegando a demostrar de forma teórica su conveniencia frente a otro tipo de representaciones. No obstante, la aplicación práctica, por su parte, ha demostrado que el resto de representaciones son también exitosas. Por ejemplo, en [**?**] utilizan una codificación basada en permutaciones, obteniendo *soluciones satisfactorias en tiempos razonables* para un problema de *despliegue y planificación de sistemas de tiempo real distribuidos*.

El objetivo de esta sección es definir una codificación adecuada para el problema propuesto en la Subsección **??**. Para ello, se ha trabajado con la librería `numpy`, la cual proporciona la estructura de datos `array`. Esta estructura cuenta con funciones predefinidas para extraer, manipular y realizar operaciones vectorizadas de forma eficiente.

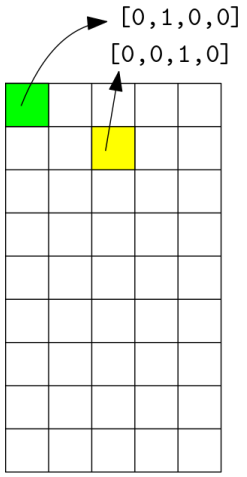

Figura 4.1: Caracterización de la solución.

Después de realizar diferentes pruebas, con distintos tipos de caracterización, se decidió tomar una matriz de tres dimensiones: i, j y k. Particularizando en el problema de diseño de vertipuertos **??**, partimos de una malla que tendrá $i = 1, \ldots, 8$ filas y $j = 1, \ldots, 5$

[2]La longitud viene determinada por el tipo de variable de decisión. Por ejemplo, las variables reales con precisión infinita necesitan una cadena binaria de longitud infinita. Como esta no es posible, la longitud de la cadena se determina teniendo en cuenta los valores máximo y mínimo de la variable real, y la precisión necesaria para representar sus valores [**?**].

columnas. La tercera dimensión, $k = 1, \ldots, 4$, hará referencia a una lista que contenga cada una de las celdas de la malla. En esa lista estará codificada la información del tipo de infraestructura presente en dicho espacio. Como se puede ver en la Figura **??**, si la celda representa una *zona de taxi* (utilizando el código de colores presente en la leyenda de la Figura **??**), la *lista* estará formada por 0 en las posiciones 0, 1 y 3, mientras que en la posición 2 será un 1. De esta forma, las matrices se definen *ad hoc* para expresar la información de las variables de decisión $x_{i,j}^{(k)}$ del modelo. En un GA, la codificación de una solución estará compuesta por:

- **Alelo**: cada valor binario en la posición. Tipos: *int; binario* $\{1, 0\}$.
- **Gen**: para una celda en una posición dada (i, j). Tipo: *lista de alelos*.
- **Individuo**: formado por toda la información contenida en la cuadrícula. Tipo: *array*.

Cabe destacar que no utilizaremos las celdas *dummy* que se proponen en [**?**], ya que la propia estructura de datos permitirá recorrerla sin este artificio.

Una vez elegida la forma de codificación de un individuo, podemos definir una función en Python para crear individuos aleatorios. No obstante, siempre que contemos con información que permita centrar la búsqueda, deberemos utilizarla; por ello, vamos a imponer que toda la población inicial cumpla la restricción **??** del modelo, ya que nos permite generar individuos más aptos de partida. Esto es, obligaremos a que los genes contengan un único tipo de infraestructura. El código completo se encuentra accesible en [**?**].

```
1
2  def generar_individuo_nuevo():
3      """
4      Estos individuo ya cumplen la restriccion g_3, es decir, que en una
       celda hay, a lo sumo, un tipo de infraestructura.
5      """
6      # Se genera un individuo formado por ceros
7      individuo = np.zeros((N_ROW, N_COL, N_INF), dtype=int)
8      # Creamos un individuo en 2 dimensiones, siendo 0 -> FATO, 1 -> Gate
       , 2 -> Taxi, 3 -> Terminal y 4 -> Celda vacia
9      elecciones_infraestructura = np.random.randint(
10         0, N_INF+1, size=(N_ROW, N_COL))
11     # Ahora hay que asignar al array que hay en cada celda un 1 en caso
       de que tenga una infraestructura de ese tipo
12     for tipo_infra in range(4):
13         individuo[:, :, tipo_infra] = (
14             elecciones_infraestructura == tipo_infra)
15
16     return individuo
```

Listing 4.1: Función para generar un individuo.

4.1.2. Función de generación

Como ya se ha comentado, su función es construir la población inicial de la que parte el proceso de evolución. Se construye en función del conocimiento que se tiene del espacio de soluciones:

- En caso de que no se disponga de información sobre el valor de las soluciones, la estrategia adecuada será construirlas siguiendo una distribución uniforme sobre los valores de la codificación elegida. La idea es que la población inicial esté compuesta de individuos distribuidos de forma equiprobable sobre el espacio de búsqueda.

- Si se conoce en torno a qué valores se localizan las soluciones, la estrategia sería construir la población inicial siguiendo una distribución Gaussiana entorno a ellos.

Para facilitar una buena convergencia, hay que evitar que la población converja a un único valor. Se dice que la población inicial debe ser lo suficientemente *dispersa*. En este sentido, se puede incluir un proceso de generación intermedio que añada algún *migrante*, esto es, una solución, o conjunto de ellas, que compitan contra los individuos generados por el cruce y mutación, para formar la población de la siguiente iteración.

En nuestro caso de estudio, para almacenar la información de las diferentes generaciones, utilizaremos los *DataFrames* de la librería `Pandas`. Así, la función que genera la población devolverá un *DataFrame* (véase la Figura **??**), que contiene las columnas *id_Individuo*, *Individuo*, *Aptitud_Obj* y *Aptitud_Penalizada*. Las columnas que contienen la información de la aptitud serán tratadas en la Subsección **??**, cuando se describa la función objetivo del modelo.

```
def creacion_poblacion(tam_pob, c, alfa, beta):
    """
    Retornamos un DF con:
      id_individuo
      Individuo
      Aptitud sin penalizar
      Aptitud penalizada
    """
    # Tamano igual al de la poblacinn
    pob = pd.DataFrame(index=range(tam_pob))
  # Asociamos un numero a cada individuo
    pob['id_Individuo'] = range(0, tam_pob)
  # Generamos un individuo para cada fila
    pob['Individuo'] = [generar_individuo_nuevo()
                        for _ in range(len(pob))]
  # Calculamos su aptitud sin penalizar
    pob['Aptitud_obj'] = pob['Individuo'].apply(fitness)
  # Calculamos su aptitud sin penalizar
    pob['Aptitud_total'] = pob['Individuo'].apply(
        lambda x: fitness_penalizado_dinamica(x, c, alfa, beta, 1))

    return pob
```

Listing 4.2: Función para crear la población.

En la precondición encontramos, entre los parámetros, el `tam_pob`, que hace referencia al tamaño de la población, el cual será constante a lo largo de todas las generaciones. El resto de parámetros serán definidos más adelante.

4.1.3. Función de aptitud

La *función de aptitud* del problema arroja un valor que mide cómo de *bueno* es un individuo dado, pero debemos tener en cuenta que existen una serie de restricciones en el modelo. Estas restricciones son las que deben guiar la solución dentro del espacio de búsqueda. La manera habitual de tratar las restricciones en un GA es aplicar una penalización al valor obtenido por el individuo en la función de aptitud.

En [**?**] se define la función que mide si se incumple una restricción como sigue

$$v_j(\mathbf{x}) = \begin{cases} \text{máx}\{0, g_j(\mathbf{x})\} & \text{si } 1 \leq j \leq q \\ |h_j(\mathbf{x})| & \text{si } q+1 \leq j \leq k \end{cases} \tag{4.1}$$

donde $\mathbf{x}$ será el *array* que conforma a un individuo.

Veamos dos ejemplos de cómo definir las funciones g_j a partir de nuestro problema concreto **??**, ya que, como se comentará más adelante, no tendremos funciones del tipo h_j. En primer lugar, la restricción **??** pasa a ser la función

$$g_1(\mathbf{x}) = v - \frac{t_{\text{win}}}{t^{(F)}} \sum_{(i,j)\in\mathcal{C}} x_{i,j}^{(F)} \tag{4.2}$$

Para el segundo, tomaremos una de las restricciones del conjunto de condiciones de vecindad; por ejemplo, la **??** pasa a ser la función

$$g_4(\mathbf{x}) = x_{i,j}^{(G)} - (x_{i-1,j}^{(P)} + x_{i+1,j}^{(P)} + x_{i,j-1}^{(P)} + x_{i,j+1}^{(P)}), \quad \forall (i,j) \in \mathcal{C} \tag{4.3}$$

Algunas observaciones:

- Dado que la codificación de la solución prescinde de las celdas *dummy* que se utilizaban en el modelo, la restricción **??** no formará parte de la implementación, siendo, además, la única función de tipo h_j de las definidas en **??**.
- Tampoco estará dentro del código la restricción **??**, dado que solo se pretende ejemplificar el uso de los algoritmos genéticos. Esta restricción hace referencia a un conjunto de celdas que se desprenden de un estudio que los autores de [**?**] hacen sobre las *rutas de aterrizaje y despegue*, en función de posibles obstáculos en la ciudad concreta de Samseong (Seúl).

Por tanto, se definen funciones g_j, para $j = 1, \ldots, 9$, que corresponden con las restricciones **??** - **??**, y la restricción **??**, respectivamente. El código de **??** sería la siguiente:

```
1 def g_4(individuo):
2     """
3     Cada gate debe ser adyacente (arriba, abajo, izq o dr) a una
    terminal de pasajeros.
4     """
5     penalizacion_total = 0
6     filas, columnas, _ = individuo.shape
7     for i in range(1, filas):
8         for j in range(1, columnas):
```

```
            vecinos = obtener_vecinos(i, j, filas, columnas)
            suma_vecinos = sum(individuo[s, t, 3] for s, t in vecinos)
            if suma_vecinos < individuo[i, j, 1]:
                penalizacion_total += 1
    return max(0, penalizacion_total)
```

Listing 4.3: Ejemplo de cabecera para una restricción g_j.

A lo largo del proceso de ajuste del algoritmo, se ha añadido una restricción adicional al modelo. Debido al funcionamiento de los algoritmos genéticos y, en especial, el mecanismo de penalización de individuos, que describiremos a continuación, un individuo *vacío* (aquel que no tiene nada construido) cumple las restricciones del problema y no es penalizado. Para ello, introducimos la restricción **??**.

$$\sum_{(i,j)\in\mathcal{C}} \prod_{k\in\mathcal{T}} (1 - x_{i,j}^{(k)}) \leq 39 \tag{4.4}$$

Tomando la numeración de los índices del modelo, no la nativa de Python, para cada celda tendremos un producto de los elementos de la lista, que contendrá cómo se expresa, para $\{x_{1,j}^{(k)} \mid j = 1, \ldots, 5 \wedge k \in \mathcal{T}\}$ en **??**:

$$\left[\prod_{k\in\mathcal{T}} (1 - x_{1,1}^{(k)}) + \prod_{k\in\mathcal{T}} (1 - x_{1,2}^{(k)}) + \prod_{k\in\mathcal{T}} (1 - x_{1,3}^{(k)}) + \prod_{k\in\mathcal{T}} (1 - x_{1,4}^{(k)}) + \prod_{k\in\mathcal{T}} (1 - x_{1,5}^{(k)})\right] + \cdots \tag{4.5}$$

Este producto se anulará siempre que se encuentre una construcción, como podemos ver al expandir $k \in \mathcal{T}$ en la ecuación **??**:

$$\left[(1 - x_{1,1}^{(1)}) \cdot (1 - x_{1,1}^{(2)}) \cdot (1 - x_{1,1}^{(3)}) \cdot (1 - x_{1,1}^{(4)}) + \ldots\right] \tag{4.6}$$

El valor máximo, es decir, el valor de un individuo *vacío*, será de 40; por eso, la restricción evalúa si el individuo está por debajo, o en, el valor 39. Para implementarlo, debe ser descrita como una función g_j, como puede verse en **??**:

$$g_{10}(\mathbf{x}) = 39 - \sum_{(i,j)\in\mathcal{C}} \prod_{k\in\mathcal{T}} (1 - x_{i,j}^{(k)}) \tag{4.7}$$

Si todas las restricciones son del mismo orden de magnitud, o se han normalizado, el *incumplimiento total* de las restricciones vendrá dado por cualquiera de las siguientes opciones:

$$v(\mathbf{x}) = \sum_{j=1}^{k} \omega_{\mathbf{j}} \cdot v_j(\mathbf{x}) \quad v(\mathbf{x}) = \max_j v_j(\mathbf{x}) \tag{4.8}$$

donde $\omega_{\mathbf{j}}$ es un vector de pesos que podemos asociar a las diferentes restricciones en función de su importancia. El valor de aptitud quedará penalizado según

$$f_p(\mathbf{x}) = f(\mathbf{x}) + v(\mathbf{x}) \tag{4.9}$$

La teoría sobre GA arroja diferentes formas de aplicar **??**. Para este trabajo, se ha optado por una penalización *dinámica*, es decir, un tipo de penalización que permite incrementar los *parámetros de penalización* de forma correlativa al proceso de evolución. Existen varios esquemas dentro de la penalización dinámica; tomaremos el descrito en [**?**], propuesto previamente por J. A. Joines y C. R. Houck en [**?**], el cual viene dado por

$$f_p(\mathbf{x}) = f(\mathbf{x}) + (c \cdot gen)^{\alpha} \cdot v(\mathbf{x}) \tag{4.10}$$

cuya función de penalización viene, a su vez, dada por

$$v(\mathbf{x}) = \sum_{j=1}^{k} v_j^{\beta}(\mathbf{x}) \tag{4.11}$$

donde

- $\boldsymbol{\alpha}$ es el parámetro que determina el grado de aumento de la penalización a medida que avanza el proceso evolutivo.
- $\boldsymbol{\beta}$ es el parámetro que influye en cómo se incorpora la penalización en la función de aptitud.
- **c** es el parámetro que define un factor de escala con el que relacionar la penalización con el número de generaciones (*gen* en la ecuación).

Volviendo a la implementación realizada, tendremos el *DataFrame* que vimos en la subsección anterior, las columnas *Aptitud_Obj* que contienen el valor de aptitud calculado según **??**, y la *Aptitud_Penalizada* según **??**. Un ejemplo de ejecución arroja el *DataFrame* de la Figura **??**.

	id_Individuo	Individuo	Aptitud_obj	Aptitud_total
0	0	[[[0, 0, 1, 0], [0, 1, 0, 0], [0, 0, 1, 0], [1...	369717631	369717631
1	1	[[[1, 0, 0, 0], [0, 0, 0, 1], [0, 0, 1, 0], [0...	369719305	369717631
2	2	[[[0, 1, 0, 0], [0, 1, 0, 0], [1, 0, 0, 0], [0...	369720816	369717631
3	3	[[[0, 0, 1, 0], [1, 0, 0, 0], [1, 0, 0, 0], [0...	369714130	369717631
4	4	[[[0, 1, 0, 0], [0, 0, 1, 0], [0, 0, 1, 0], [0...	369714210	369717631
5	5	[[[1, 0, 0, 0], [0, 0, 0, 1], [0, 0, 1, 0], [0...	369716134	369717631
6	6	[[[0, 0, 1, 0], [1, 0, 0, 0], [1, 0, 0, 0], [0...	369713493	369717631

Figura 4.2: Ejemplo *DataFrame* de la población inicial.

4.1.4. Operadores genéticos: Selección

Los *algoritmos de selección* serán los encargados de escoger qué individuos van a disponer de oportunidades de reproducirse y cuáles no [**?**]. Como ya se ha comentado anterior-

mente, los individuos más aptos deben tener una mayor probabilidad de reproducirse, sin embargo, los menos dotados no deben ser eliminados por completo, o la población sería demasiado homogénea, lo que daría lugar al estancamiento en óptimos locales, una convergencia temprana, y a una *pérdida de información genética*[3].

En el Algoritmo **??** se presenta el funcionamiento general del operador de selección:

[H]

FSelectSelect FSelectBasicSelectBasic FValidvalid FnFunction:*parents*

f_seleccion $parents = x, N\_p \;\; i = 1 \;\; N \;\; val(i) = Obj(x(i)) \quad i = 1 \;\; i < N_p \; parents(i, 1) = x, val \;\; parents(i, 2) = x, val \;\; parents(i, :) \; i = i + 1$

Se eligen N_p parejas de padres de entre las N posibles de la población x. En las líneas 1 a 3 del algoritmo, se representa el cálculo de aptitud del individuo en el que profundizaremos en la subsección anterior. Posteriormente, se eligen los dos padres de la pareja, donde $SelectBasic(x, val)$ es un método de selección que debe ser adecuado para el problema en cuestión. Por último, y de forma "opcional", dependiendo del método de selección elegido, se verifica si estos padres son una *pareja válida* (es decir, si no son el mismo padre, o son dos individuos muy similares).

Los métodos de selección se basan en una probabilidad relacionada con el valor de mérito. De entre los métodos existentes, hemos elegido tres de forma que, posteriormente, podamos comparar resultados. A continuación, vamos a profundizar en ellos.

En primer lugar, se programa el **método de la ruleta**. Vamos a ver en qué consiste a partir de la población **??**:

1. Calcular el sumatorio de la aptitud de la población. En este caso, se obtiene

$$S = \texttt{2588023417}$$

2. Calcular r, un número aleatorio, según una distribución uniforme entre 0 y S. En el ejemplo, la ejecución arrojó el valor

$$r = \texttt{1844855395.3541374}$$

3. Recorrer la población sumando su aptitud, hasta que la suma sea igual o mayor que r:

```
acumulado = 369717631 . id_individuo 0
acumulado = 739435262 . id_individuo 1
acumulado = 1109152893 . id_individuo 2
acumulado = 1478870524 . id_individuo 3
acumulado = 1848588155 . id_individuo 4
```

Figura 4.3: Proceso para encontrar el individuo cuya aptitud es mayor que r.

[3]Si todos los individuos son iguales, el cruce y la mutación no dará lugar a soluciones innovadoras.

4. El primer individuo que supera el valor de r es elegido como padre. En este caso, se corresponde con el índice 4.

Luego, calculado el primero de los padres, y repitiendo este proceso, encontraríamos el segundo. Se implementan también:

- **Métodos de torneo**. En estos casos se eligen los padres en dos etapas:
  1. Primero, hay que tomar un subconjunto (sin tener en cuenta la aptitud de los individuos) de tamaño T de entre todos los de la población actual.
  2. Después, uno de los individuos del subconjunto será elegido, teniendo en cuenta los valores en la función de mérito. Podemos diferenciar:
     - **Torneo determinístico**. Entre los individuos del subconjunto se elige el más óptimo. Cuanto más grande es T, mayor es la presión de selección de los individuos mejores.
     - **Torneo probabilístico**. En este caso, hay una etapa intermedia en la que, con probabilidad $p > 0.5$, se determina si el individuo elegido es el más apto o el menos apto del subconjunto. Cuanto menor es el valor de p, mayor es la presión de selección de los individuos mejores.

Han quedado, por tanto, implementados estos tres métodos: ruleta, torneo determinístico y torneo probabilístico, para poder hacer pruebas y ver si existen diferencias significativas en cuanto a los resultados que arroja el algoritmo genético, dependiendo de cuál use como método de selección.

4.1.5. Operadores genéticos: Cruce

Seleccionados los dos padres, los algoritmos de cruce operan de dos formas diferentes, de acuerdo con [**?**]. Se puede optar por una estrategia *destructiva*, en la que los descendientes pasan a formar parte de la nueva población, sin comparar con el mérito de sus padres. Por contra, una estrategia *no destructiva* supone que la descendencia pasará a la siguiente generación, si supera el mérito de sus padres. Ambas estrategias pretenden que aquello que hace *bueno* a un padre se trasmita a los hijos.

Existen diferentes algoritmos para este operador, pero nos centramos en el que se ha implementado en nuestro código, el **cruce uniforme**. Se determina, de forma aleatoria y a partir de una probabilidad p, si cada alelo está o no en el hijo 1. En caso de que no esté en el hijo 1, estará en el hijo 2.

En la Figura **??** vemos el comportamiento de este operador a partir de dos individuos, `Padre_1` y `Padre_2`, y de una *matriz de decisión*, `m_decisión`, del mismo tamaño que un individuo. El contenido de la matriz se calcula según una distribución uniforme, siendo `True` si el valor es menor que el porcentaje de cruzamiento:

- Si `m_decisión` es `True` en la celda (i, j), entonces `Hijo_1` tendrá el mismo alelo que `Padre_1` en la celda (i, j).
- Si `m_decisión` es `False` en (i, j), entonces `Hijo_1` debe completarse con el alelo de la celda (i, j) del `Padre_2`.

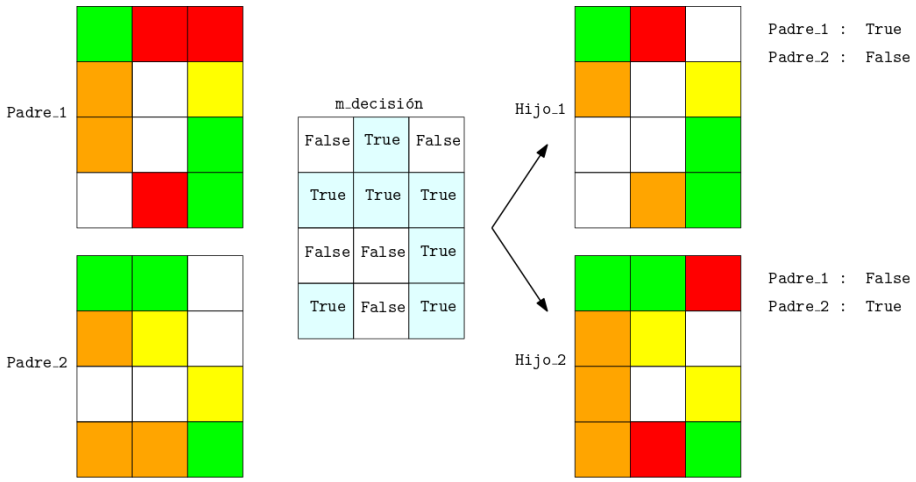

Figura 4.4: Ejemplo de cruce uniforme según se ha definido en la implementación.

Para obtener el `Hijo_2`, se procede a la inversa.

La elección del algoritmo de cruce uniforme, como operador de selección en nuestro algoritmo genético, se fundamenta en su capacidad para modular la alteración de patrones a través del parámetro configurable p.

4.1.6. Operadores genéticos: Mutación

La *mutación* de un individuo supone la transformación de alguno de sus genes, es decir, algunas de las partes de ese individuo se verán afectados por una alteración. Este operador puede actuar en solitario, o ser parte del operador de cruce. En nuestro caso será independiente, es decir, una vez que toda la descendencia ha sido creada, se elige un conjunto de estos para ser candidatos a mutar, en función de una cierta probabilidad p_m. A pesar de tener un carácter secundario dentro del algoritmo, es un proceso importante para añadir *diversidad* y, así, alcanzar regiones distintas del espacio de búsqueda.

Podemos subdividir el proceso en dos etapas [**?**]:

1. Se calculan las posiciones que serán mutadas utilizando una función aleatoria que determina, en función de p_m, si tiene que ser mutado o no.

2. Se calculan los nuevos valores para los elementos que mutan. En el caso binario se toma el valor contrario.

Veamos como funciona este operador en nuestra implementación. Partimos de un individuo de la población de hijos y la matriz de decisión `m_i`, del mismo tamaño que un individuo. El contenido de la matriz se calcula según una distribución uniforme, siendo `True` si el valor es menor que el porcentaje de mutación. Si `m_i` es `True` en la celda (i, j), entonces el individuo propuesto para mutar deberá cambiar su alelo de la celda (i, j) (véase la Figura **??**). La creación del nuevo alelo se hará a partir de una función que devuelva una lista con 4 elementos binarios, donde solo uno de ellos será un 1.

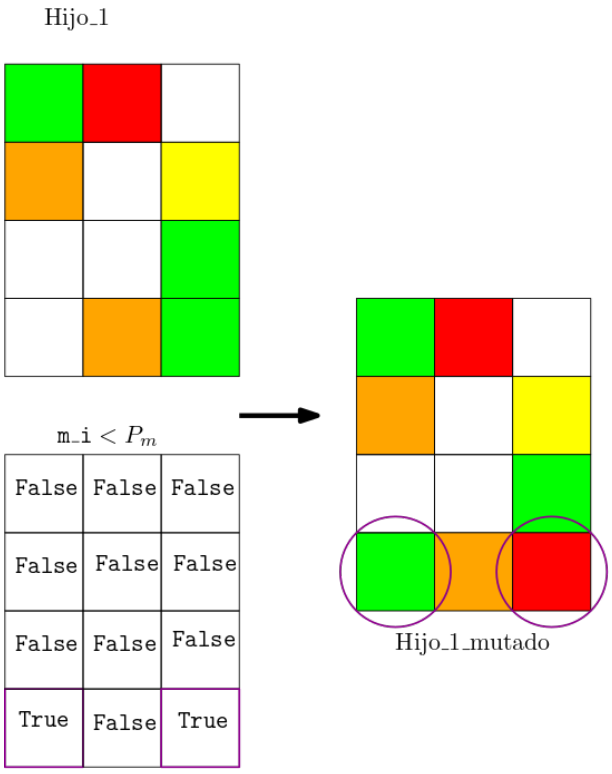

Figura 4.5: Ejemplo de proceso de mutación de un individuo.

4.1.7. Función de recombinación

La *función de recombinación* se encarga de, una vez que los operadores de cruce y mutación han generado los nuevos individuos de la población, reorganizarlos para crear la población que pasa a la siguiente generación. Por tanto, necesitamos tomar decisiones sobre cuáles de los individuos antiguos (o nuevos) formarán parte de la población en la próxima generación. Para ellos existen una gran cantidad de métodos, según las diferentes situaciones que se pueden dar. En nuestro caso, hemos implementado la de **sustitución generacional**.

Entenderemos por *nueva población* a la formada por los hijos obtenidos del operador de cruce, y que, posteriormente, han podido sufrir una mutación. La *sustitución generacional* consiste en reemplazar, directamente, la población actual por la nueva. Para mantener constante el tamaño de la población, es necesario crear tantos nuevos individuos como tamaño tenga la población. Esta estrategia puede llevar a perder la mejor solución encontrada hasta el momento; para evitar esta pérdida, la mejor solución obtenida a lo largo de todas las generaciones la hemos guardado en una estructura auxiliar.

Se ha elegido este tipo de recombinación por simplicidad en la implementación.

4.1.8. Función de parada

Para que el algoritmo pueda parar de generar las distintas generaciones, es necesario definir una *condición de parada*. Las versiones más habituales son:

- **Límite de generaciones**. Se establece un parámetro que limita el número total de generaciones; una vez se alcanza este límite, el algoritmo termina, independientemente de la calidad de las soluciones.
- **Convergencia poblacional**. El algoritmo también puede terminar cuando la diferencia de los valores de la cadena, o de las funciones de mérito de la población, es menor que un valor prefijado; o bien cuando las mejores soluciones, o sus funciones de mérito, no cambian de forma significativa durante un número prefijado de generaciones [**?**].

En nuestra implementación, se ha elegido la limitación generacional, ya que su puesta en marcha no requiere de ningún ajuste adicional.

Capítulo 5

Resultados y Análisis

En este capítulo presentamos brevemente los resultados obtenidos por el algoritmo genético implementado en el Capítulo **??** para realizar un análisis de la sensibilidad de los parámetros que configuran el algoritmo. La principal fuente de consulta que han servido para la elaboración del capítulo es [**?**].

5.1. Datos del caso práctico

Para poner en práctica el algoritmo tomaremos los datos que presentan los autores de [**?**] para un problema de diseño de vertipuertos particular. Los datos concretos se pueden consultar en **??**.

Parámetro	**Descripción**	**Valor**	**Unidad**
l	Ciclo de vida de la infraestructura	10	Años
t_{win}	Ventana de tiempo	60	Minutos
t_F	Tiempo de retorno de una FATO	77	Minutos
t_G	Tiempo de retorno en el vertipuerto	10	Minutos
f	Tasa por uso del vertipuerto	55	Dólares por aeronave
$c_{\text{con}}(F)$	Coste de construcción FATO	77	Dólares por m^2
$c_{\text{con}}(G)$	Coste de construcción puerta de embarque	2284	Dólares por m^2
$c_{\text{con}}(T)$	Coste de construcción celda de taxi	77	Dolares por m^2
$c_{\text{con}}(P)$	Coste de construcción terminal de pasajeros	2284	Dolares por m^2
c_{ops}	Coste operacional	1563379	Dolares por año
N_{row}	Número de filas	8	-
N_{col}	Número de columnas	5	-
CD	Dimensión de control de una aeronave	10	Metros
V	Capacidad de aeronaves por hora	80	Aeronaves por hora

Cuadro 5.1: Datos del caso práctico.

5.2. Resultados

Para la realización de este trabajo, exceptuando la subsección **??**, se ha contado con un ordenador portátil con las siguientes características técnicas:

- Marca y modelo de la CPU: Intel Core i5-8250U 1,60GHz.
- Memoria RAM: 8 GB DDR4.
- Almacenamiento: SSD 512 GB.
- S.O. y distribución: Linux, Ubuntu 22.04.3 LTS.

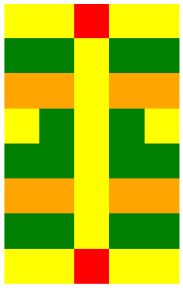

Figura 5.1: Solución final hallada en [**?**].

Los valores de aptitud de nuestras soluciones están cercanos a los de la solución propuesta en [**?**]. No obstante, no siempre obtenemos soluciones dentro del rango factible. Este desafío es habitual para este tipo de técnicas, y es la conclusión a la que se ha llegado en la subsección de análisis de sensibilidad **??** de este trabajo.

La solución, definitiva, propuesta por los autores podemos verla en la Figura **??**. Dicha solución arroja un valor de aptitud de 369754576.

Por nuestra parte, la mejor solución obtenida la podemos ver en la Figura **??**. Esta solución, con un valor de 369781060, mejora la aptitud en un 0.007 % a la mejor solución obtenida por los autores de [**?**]. No obstante, incumple:

- g_6 ya que dos de las puertas de embarque no son adyacentes a una celda de *taxi*.
- g_7 ya que la celda $(0, 0)$ no es adyacente a una celda de taxi y, además, queda rodeada por otras FATO, por lo que también incumple g_9.

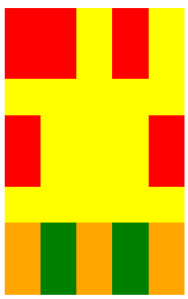

Figura 5.2: Mejor solución.

Tomando 10 ejecuciones de nuestro algoritmo, la media de aptitud obtenida es 369760415.2,

lo cual supone un desvío del 0.001 % respecto del valor de aptitud obtenido por los autores del artículo. No obstante, mediante la Figura **??**, en la que se presentan 2 de las 10 soluciones obtenidas, podemos constatar que no siempre entran dentro de la región de factibilidad. En la Tabla **??** se presenta un resumen de los tiempos y aptitudes de la batería de ejecuciones. En dicha tabla encontramos valores de aptitud muy próximos, por lo que la convergencia del algoritmo hacia un óptimo queda puesta de manifiesto, pese a que este no siempre sea factible. Por tanto, lo más probable es que estemos ante una convergencia prematura, lo que nos podría indicar que el algoritmo se ha estancado en un máximo local.

	Mej_Fit	**Tiempo [seg]**
0	369761505	45.2
1	369761505	52.0
2	369763866	58.6
3	369761736	49.5
4	369751505	76.2
5	369764020	56.8
6	369761813	51.7
7	369757245	48.2
8	369763789	47.9
9	369757168	56.3
Media	369760415.2	54.31

Cuadro 5.2: Batería de ejecuciones.

De estos resultados, se desprende la necesidad de ajustar correctamente los parámetros del algoritmo genético, de modo que calibremos adecuadamente el comportamiento que tiene éste respecto de las restricciones del modelo.

5.3. Comparación entre algoritmos de selección

Como comentamos en la subsección anterior, para las pruebas que hemos realizado en esta subsección, hemos contado con un equipo informático distinto, con prestaciones superiores. Las características técnicas del equipo en cuestión son las siguientes:

- Marca y modelo de la CPU: Intel Core i9-13900H 5.40GHz.
- Memoria RAM: 16 GB DDR4.
- Almacenamiento: SSD 1 TB.
- S.O. y distribución: Windows, 11 Home.

En la subsección **??**, vimos los tres algoritmos de selección que hemos implementando en el trabajo, estos son: ruleta, torneo probabilístico y torneo determinístico. Como hemos comentado, el equipo informático que hemos utilizado para comparar estos métodos es, significativamente, más potente. Esto se ve reflejado, especialmente, en los tiempos de ejecución.

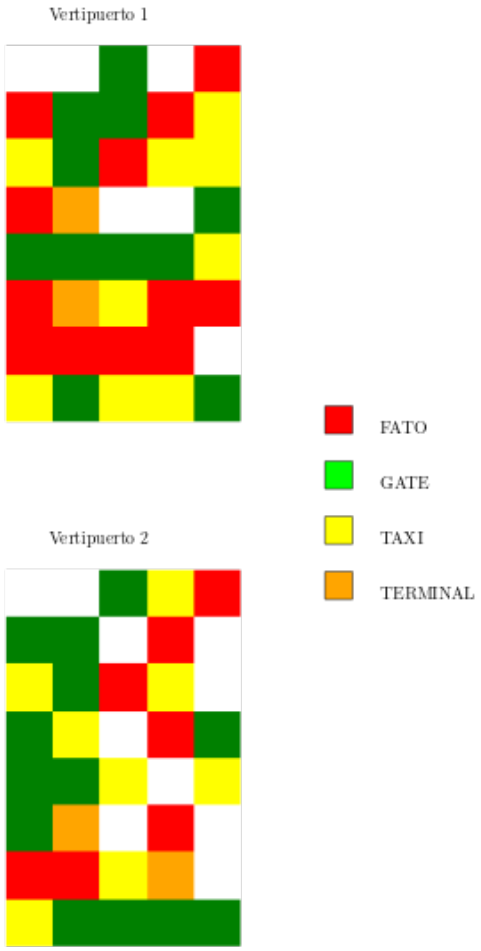

Figura 5.3: Ejemplos de soluciones.

Para comparar los algoritmo, hemos creado tres *DataFrames* con 50 ejecuciones por algoritmo, de modo que, concatenándolos tenemos un *DataFrame* con 150 observaciones. A partir de este, hemos construido el *DataFrame* de estadísticas con los atributos: media, mediana y desviación estándar. Gracias al método `groupby` de la libreria `pandas`, hemos agrupado los resultados por *algoritmo de resolución* y calculado cada medida estadística según la mejor aptitud sin penalizar (`Mej_Fit`), la mejor aptitud penalizada (`Mej_Fit_Pen`) y el tiempo de ejecución (`Tiempo_transcurrido`).

En primer lugar, si nos fijamos en el gráfico de la Figura **??**, las barras de error son relativamente grandes, lo que sugiere una variabilidad considerable en los resultados de cada algoritmo. El algoritmo de torneo probabilístico tiene el promedio y la mediana más altos, lo que podría indicar un mejor rendimiento en términos de ajuste. Sin embargo, también tiene la mayor variabilidad, como se muestra en la longitud de su barra de error. Finalmente, el algoritmo de ruleta tiene el menor promedio y mediana, y su variabilidad también es grande. El algoritmo de torneo determinístico se sitúa entre los dos en términos de rendimiento y variabilidad.

A continuación, veamos qué ocurre con la aptitud penalizada. Para ello, observamos la gráfica de la Figura **??**. Aquí, la variabilidad es notablemente menor que en la métrica sin penalización de la aptitud. El torneo probabilístico ya no muestra una ventaja clara;

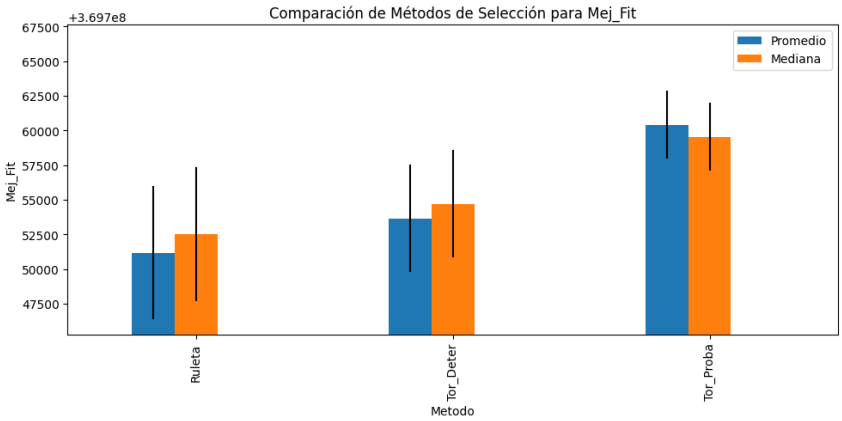

Figura 5.4: Comparación por aptitud sin penalizar.

de hecho, tiene el promedio más bajo, pero su mediana es competitiva con la del torneo determinístico. El torneo determinístico muestra un mejor promedio que el torneo probabilístico, pero su mediana es más baja que la de la ruleta. Finalmente, la ruleta tiene la mediana más alta, lo que sugiere que cuando se considera la penalización, puede ser un método consistentemente bueno, a pesar de su promedio más bajo.

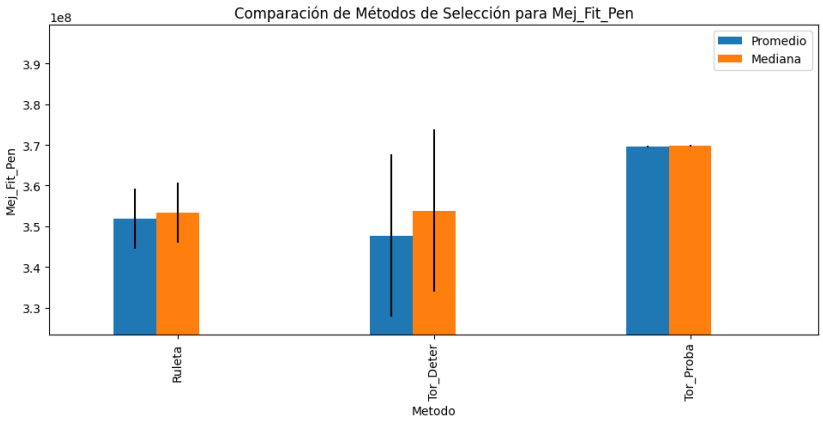

Figura 5.5: Comparación por aptitud penalizada.

Para terminar, se puede observar en la gráfica de la Figura **??**, la variabilidad en el tiempo es menor entre las ejecuciones. El torneo probabilístico tiene el tiempo promedio más bajo y la media más alta, lo que sugiere que es el más rápido pero con variabilidad en los resultados. Finalmente, ruleta y torneo determinístico tienen tienen tiempos promedio más altos que el probabilístico y similares entre sí.

Estos resultados nos llevaron a tomar la decisión de utilizar el torneo probabilístico para continuar con las pruebas.

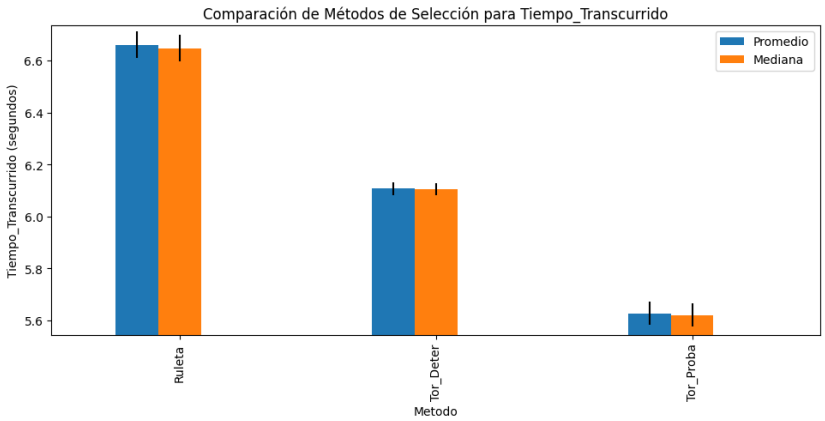

Figura 5.6: Comparación por tiempo de ejecución.

5.4. Análisis de Sensibilidad

Como hemos visto en la subsección **??**, el algoritmo no siempre es capaz de orientar la busqueda de forma que se cumplan las restricciones. Por este motivo, vamos a comenzar por el ajuste de los parámetros de la penalización, de forma que el tratamiento de las restricciones del modelo sea más eficiente. Posteriormente, ajustaremos los parámetros del algoritmo genético. Para ello diseñaremos dos tipos de experimentos, uno en el que compararemos la tendencia de los resultados ajustado el valor de los parámetros α y β, y otro mediante un modelo de regresión lineal que nos permita llevar a cabo un contraste de hipótesis.

Dado que vamos a comenzar ajustando los parámetros de penalización, inicialmente, los parámetros del GA los vamos a tomar dentro de unos valores que permitirán, en base a la experiencia con nuestro algoritmo, hacer pruebas en un tiempo razonable de ejecución. Estos valores se pueden consultar con más detalle en la Tabla **??**.

	Valor	Rango
Número de generaciones	500	$[100, 500]$
Tamaño poblacional	100	$[50, 200]$
Tasa de cruzamiento	0.7	$[0.6, 0.8]$
Número de parejas	20	$1 \leq \texttt{tam\_pob} \leq \texttt{tam\_pob}/2$
Número de individuos mutados	30	$\lfloor 0{,}3 \cdot \texttt{tam\_pob} \rceil$
Tasa de mutación	0.02	$[0.01, 0.05]$

Cuadro 5.3: Valores de referencia.

Ajuste de parámetros de penalización

Los valores teóricos que se proponen en [**?**] para los parámetros de la penalización, que describimos en la subsección **??**, son: $\alpha = \beta = 2$ y $c = 0{,}5$. A partir de estos valores, vamos a intentar mejorar la aptitud del mejor individuo de la población para diferentes valores de α. Estudiaremos este parámetro en el rango $\alpha \in [1, 5]$ sobre un vector equiespaciado

de 20 elementos. Se obtienen los resultados mostrados en la Figura **??**.

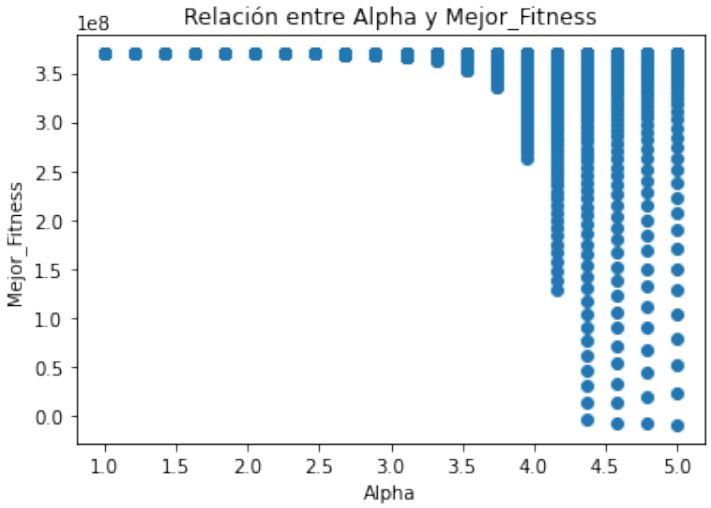

Figura 5.7: Valores de aptitud en función del valor de α.

El gráfico muestra que, para valores bajos de α (cerca de 1.0), la mejor aptitud obtenida tiende a ser relativamente estable y alta. Sin embargo, a medida que el valor de α aumenta, se observa una mayor variabilidad en el mejor valor de aptitud, con muchos valores bajos de aptitud apareciendo, lo que sugiere que un valor de α alto puede no ser favorable para obtener buenos valores de aptitud con este algoritmo genético en particular. Debido a que hay una tendencia clara a partir de 3 hacia valores de aptitud muy pequeños, acotamos el valor de α hasta 3.25. No obstante, los resultados son parecidos **??**.

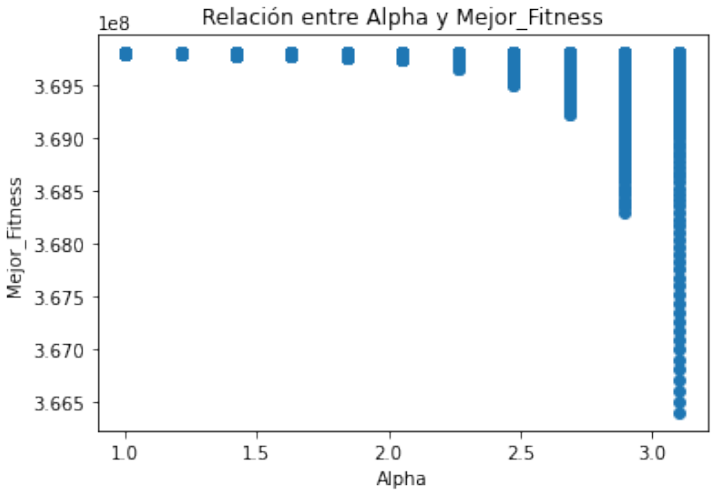

Figura 5.8: Valores de aptitud en función del valor de α.

Si bien los valores de aptitud son mejores, y están todos en un orden de magnitud parecido, la tendencia continúa apuntado a que los valores óptimos se obtendrán cuando α se aleja de los valores más altos. Tras varias pruebas llegamos al gráfico de la Figura **??**.

Los resultados revelan que el rango de α entre 1.0 y 2.0 mantiene una relación estable con los valores de aptitud obtenidos a lo largo de las generaciones, sugiriendo así una

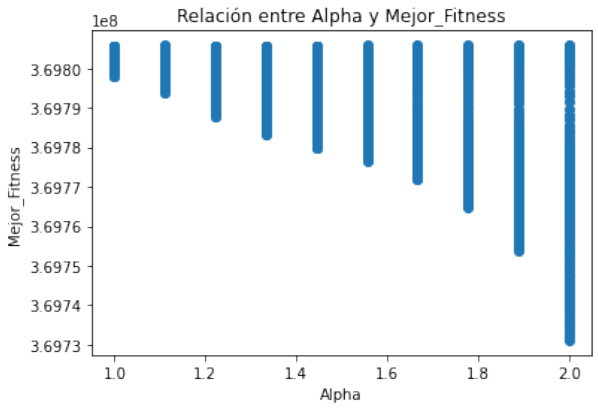

Figura 5.9: Valores de aptitud en función del valor de α.

sensibilidad reducida del algoritmo a variaciones dentro de este intervalo. Observamos también que, a pesar de una ligera tendencia hacia la sobrepenalización, el valor de aptitud óptimo se conserva, lo cual implica que la capacidad exploratoria del algoritmo no se ve comprometida con estas configuraciones. En consecuencia, se adoptará un valor de α dentro del intervalo mencionado, aceptando que este rango equilibra adecuadamente la exploración y la viabilidad de las soluciones generadas. Por tanto, tomaremos $\alpha = 1{,}5$.

A continuación, estudiamos la sensibilidad para el valor de β. Dada la relación que hay entre ambos valores [**?**], los resultados son similares a la Figura **??**, por lo que se decide tomar el mismo valor que para α.

Finalmente, debemos ajustar el valor del parámetro c. Su propósito es actuar como un coeficiente para ajustar la penalización basada en el desempeño de la población actual respecto a las restricciones. Este parámetro ayuda a calibrar el nivel de penalización aplicado en función de cómo la población actual está satisfaciendo las restricciones. Si comparamos el valor de aptitud del individuo que se presenta como mejor solución en [**?**], el valor 369754576.0, con las aptitudes que maneja el algoritmo, vemos que estamos en el mismo orden de magnitud, e incluso mejorando dicho valor. No obstante, la solución que se presenta en [**?**] es *más factible*, ya que se comporta mejor respecto de las restricciones del problema. En esta ocasión, vamos a realizar un contraste de hipótesis:

$$\begin{cases} H_0: & \text{Aumentar } c \text{ no tiene efecto en la factibilidad.} \\ H_1: & \text{Aumentar } c \text{ mejora la factibilidad.} \end{cases} \tag{5.1}$$

Tomaremos como métrica, para medir la factibilidad de las soluciones, el número de restricciones violadas, ya que es un buen indicador cuantitativo de la factibilidad de las soluciones. Para ello, definimos dos conjuntos: en el primero de ellos tomaremos un valor de c cercano al recomendado en [**?**], y en el otro tomaremos un valor ligeramente más agresivo, $c = 1$. Entre la bibliografía específica del Capítulo **??**, encontramos diversos valores utilizados para el parámetro c, oscilando entre 0 y 2. En caso de que ir aumentando su valor hasta $c = 1$ sea conveniente, se deberá seguir estudiando, repitiendo este proceso, hasta conseguir ajustar el valor. Finalmente, buscando un equilibrio entre los tiempos de ejecución y el tamaño de la muestra, tomaremos una muestra de 30 ejecuciones por grupo.

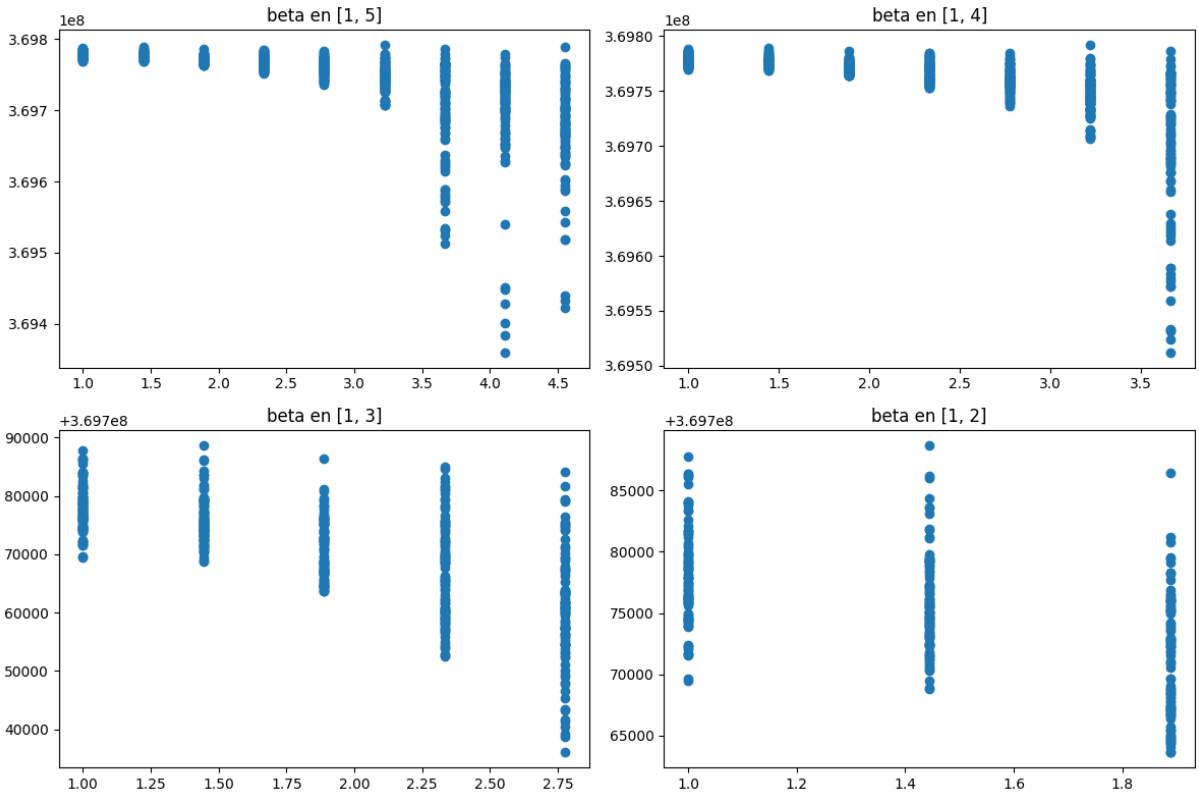

Figura 5.10: Valores de aptitud en función del valor de β.

Mediante el *modelo de regresión lineal simple*, $y = \beta_0 x$, donde x es el valor del parámetro c, e y la métrica de la factibilidad, se presentan los resultados obtenidos en las Tablas **??**, **??** y **??**.

	coef	**std err**	t	$P > \|t\|$
const	2.8879	0.242	11.930	3008272e-17
Valor_c	2.0320	0.306	6.636	1.193014e-08

Cuadro 5.4: Coeficientes y p-valores del modelo.

En primer lugar, en la Tabla **??**, vemos que los p-valores son menores que 0.05, lo que indica que ambos son estadísticamente significativos.

	R-squared	**Adj. R-squared**	**AIC**	**BIC**
Results	0.432	0.422	109.5	113.7

Cuadro 5.5: Ajuste del modelo.

En el ajuste del modelo, que podemos ver en la Tabla **??**, el valor de 0.432 indica que el modelo puede explicar el 43.2 % de la variabilidad de la variable dependiente `N_Res_Violadas`. Dado que no es un valor alto, podría estar indicando que hay más variables que no se han tenido en cuenta, y tienen impacto en el valor que estamos estudiando. De cara a comparar con otros modelos de regresión, sería interesante apuntar los valores obtenidos

de los *criterios bayesiano y de Aikaike* para comparar y poder observar qué otras variables pueden ayudar a explicar mejor el modelo.

	Durbin-Watso	**Jarque-Ber**	**Prob. JB**	**Skew**
Results	1.796	4.147	0.126	0.040

Cuadro 5.6: Hipótesis del modelo.

Finalmente, centrándonos en los valores de la Tabla **??**, respecto a las hipótesis del modelo, podemos ver que el valor de 4.147, con un p-valor de 0.126 del *criterio de Jarque-Bera*, sugiere que no hay suficiente evidencia para rechazar la hipótesis de normalidad de los residuos, ya que este p-valor es mayor que el umbral común de 0.05. El valor del *criterio de Durbin-Watson* está por debajo de 2, por lo que no hay una autocorrelación significativa de los residuos y, esto hace que sea poco probable la heterocedasticidad[1] del modelo.

De este modo, y en función de los resultados de la Tabla **??**, vemos que el p-valor asociado al valor de c es menor que el nivel de significancia usual, por lo que se rechaza la hipótesis nula H_0, y, en consecuencia, aceptamos la hipótesis de cambio, concluyendo que el aumento del parámetro c permite mejora la factibilidad de las soluciones.

Como ya comentamos, sería conveniente repetir este proceso hasta acotar todo lo posible el valor de c. En este caso, nos quedaremos con $c = 1$, quedando así cerrado el estudio de los parámetros de penalización.

Ajustes de los parámetros del GA

El ajuste efectivo de los parámetros de un algoritmo de optimización se puede abordar de diferentes maneras, como se muestra en [**?**]. En este caso, optaremos por un enfoque sistemático que permita evaluar y ajustar de manera iterativa el impacto de cada parámetro en el rendimiento del algoritmo. Este proceso se divide en diferentes etapas:

- **Selección Inicial**: comenzamos con una selección inicial de parámetros basada en [**?**]. De esta forma, podemos entender su impacto aislado en el rendimiento del algoritmo haciendo ajustes individuales. En este sentido, utilizamos los valores predeterminados de la Tabla **??**.
- **Ajuste Secuencial**: es el paso que vamos a introducir en esta subsección. Consiste en ajustar los parámetros de manera secuencial, manteniendo los demás constantes. Este enfoque permite identificar interacciones y dependencias entre parámetros. Es crucial evaluar cómo la modificación de un parámetro afecta al rendimiento en presencia de los ajustes previos, permitiendo así una comprensión más profunda de las dinámicas del sistema.
- **Retroalimentación**: después de una pasada inicial por los parámetros, el proceso de optimización se beneficia de un enfoque iterativo, donde se revisan y ajustan nuevamente los parámetros en función de los resultados obtenidos. Este ciclo de refinamiento se basa en la retroalimentación del rendimiento del algoritmo, donde

[1] Un modelo de regresión presenta homocedasticidad cuando la varianza de los residuos es constante. Si no tiene esta propiedad diremos que el modelo presenta heterocedasticidad.

los ajustes anteriores pueden requerir modificaciones. Este enfoque iterativo asegura un ajuste fino, y una optimización más precisa de los parámetros, pero requiere de un tiempo de trabajo considerable.

- **Evaluación**: en cada iteración, se debe evaluar el rendimiento del algoritmo considerando todos los parámetros en conjunto. Esta evaluación integral ayuda a identificar configuraciones óptimas que maximicen el rendimiento global del algoritmo.

Actualmente existen *técnicas de optimización* automatizadas [**?**]: búsqueda en cuadrícula (*grid search*), búsqueda aleatoria (*random search*), o algoritmos evolutivos específicos para la selección de parámetros. Estas técnicas pueden explorar, de manera más eficiente, el espacio de parámetros, y ofrecer retroalimentación valiosa para el ajuste.

Partimos de la creación de un *DataSet* con 81 observaciones de las variables:

- `n_gen`: representa el número de generaciones que tendrá cada ejecución. El rango de valores es un vector equiespaciado entre 40 y 200: `np.linspace(40, 200, 3, dtype=int)`.
- `tam_pob`: representa el tamaño, constante, de la población, para cada ejecución. El rango de valores es un vector equiespaciado entre 50 y 200: `np.linspace(50, 200, 3, dtype=int)`.
- `tasa_cruz`: representa el porcentaje de cruzamiento para cada ejecución. El rango de valores es: `[0.6, 0.7, 0.8]`.
- `tasa_mut`: representa la tasa de cruzamiento para cada ejecución. El rango de valores es: `[0.01, 0.03, 0.05]`.
- `mej_fit_pen`: representa la aptitud del mejor individuo al final de la ejecución.

Establecemos el número de parejas, y de individuos mutados, de forma dependiente del tamaño de la población:

$$\texttt{num\_parejas} = \texttt{tam\_pob // 2},$$
$$\texttt{num\_indiv\_mutados} = \texttt{round(0.3 * tam\_pob)}$$

Comenzamos viendo las correlaciones entre variables. Inicialmente observamos que existe una fuerte correlación negativa entre el número de generaciones y la aptitud del individuo final; véase la Figura **??**. Esto es interesante, ya que se esperaría, con más generaciones, que la aptitud mejorara, a menos que esté influenciada por el sobreajuste o por una convergencia prematura.

Por otro lado, el resto de variables no parece tener un impacto significativo en la aptitud del individuo.

Fijémonos ahora en la variable aptitud de forma aislada, véase Figura **??**. Observamos que la mediana de la distribución se encuentra cerca del cuartil superior, indicando que más de la mitad de los valores están agrupados hacia el extremo superior del rango. El rango intercuartílico es relativamente corto, lo que sugiere una baja dispersión y, por tanto, una consistencia en los valores de aptitud alrededor de la mediana. No se observan valores

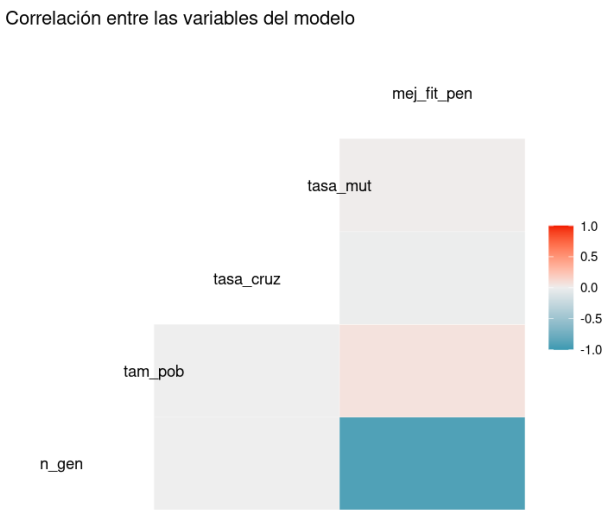

Figura 5.11: Correlación entre `n_gen` y `mej_fit_pen`.

atípicos significativos, lo que indica que no hay resultados extremos que distorsionen la distribución. La concentración de los datos sugiere un rendimiento estable del algoritmo en términos de aptitud final, a través de las ejecuciones analizadas.

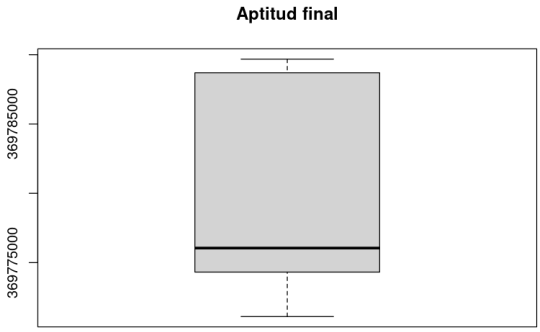

Figura 5.12: *Boxplot* `mej_fit_pen`.

Construimos un *modelo de regresión lineal múltiple*, ya que, a pesar de que la correlación entre variables indica que la única significativa es el número de generaciones, mediante el análisis de varianza (ANOVA) podemos identificar qué variable son estadísticamente significativas.

En vista de los resultados obtenidos por el el análisis de la varianza, véase la Tabla **??**, con un nivel de significancia del 0.05, observamos que, **tam_pob**, **tasa_cruz** y **tasa_mut** no son estadísticamente significativas en el modelo, es decir, su coeficiente de regresión es nulo. Por tanto, comenzaremos configurando el tamaño de la población.

Así, el *modelo de regresión* que vamos a estudiar es $y = \beta_0 x$, donde x es el valor de la variable `n_gen`, e y el valor de aptitud `mej_fit_pen`. No obstante, este modelo resulta no cumplir la hipótesis de normalidad en los residuos. Hemos asumido que $\varepsilon_i \sim N(\mathbf{0}, \sigma^2\mathbf{I})$,

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
n_gen	1	3072041526.90	3072041526.90	409.53	0.0000
tam_pob	1	17906963.67	17906963.67	2.39	0.1265
tasa_cruz	1	838109.03	838109.03	0.11	0.7391
tasa_mut	1	728195.12	728195.12	0.10	0.7562
Residuals	76	570105747.23	7501391.41		

Cuadro 5.7: ANOVA modelo completo.

denominando a este modelo, *modelo con error normal*. Pero la función de distribución de los residuos que observamos en la Figura **??** no sugiere un comportamiento normal.

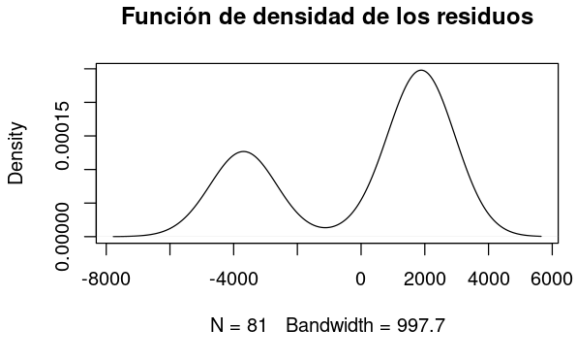

Figura 5.13: Función de densidad de los residuos.

Por otro lado, si los residuos se distribuyen normalmente, en un gráfico que expresa los cuántiles de los residuos, los puntos en el gráfico deben estar aproximadamente en línea recta. Sin embargo, como vemos en la Figura **??**, esto no sucede.

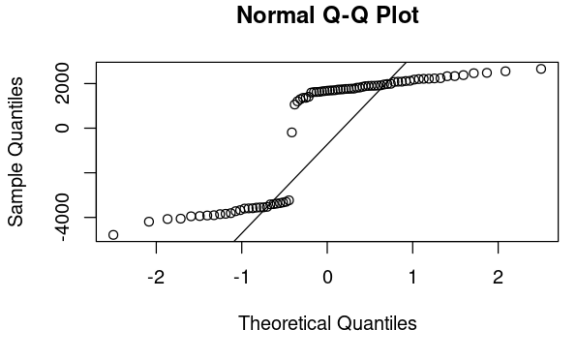

Figura 5.14: Normal Q-Q gráfico.

Podemos corroborarlo mediante inferencia con el *contraste de Safiro-Wilk* **??**.

$$\begin{cases} H_0: & \varepsilon_i \sim N(\mathbf{0}, \sigma^2\mathbf{I}) \\ H_1: & \varepsilon_i \not\sim N(\mathbf{0}, \sigma^2\mathbf{I}) \end{cases} \tag{5.2}$$

Test de normalidad	
Prueba	Shapiro-Wilk
Datos	residuos
Estadístico	0.71961
***p*-valor**	3.8e-11

Cuadro 5.8: Resultados test Shapiro-Wilk.

El resultado del test de normalidad de Shapiro-Wilk, para los residuos **??**, indica que la hipótesis nula, en la que los residuos se distribuyen normalmente, debe ser rechazada, ya que el p-valor es prácticamente nulo, por debajo de cualquier nivel de significancia al uso.

Llegados a este punto, una estrategia a seguir consistiría en tomar un valor intermedio de generaciones, y volver a construir un *DataSet* configurando el resto de parámetros en sus valores más óptimos.

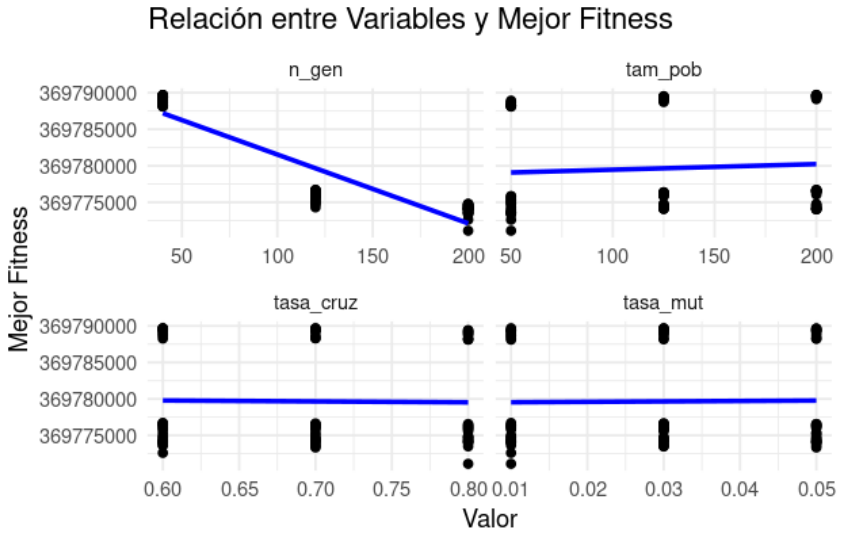

Figura 5.15: Comparación de las variables con la Aptitud.

Dado el estudio completo que se ha llevado a cabo en este capítulo, parece haber una convergencia prematura en el algoritmo lo que lleva a que algunas soluciones *no evolucionen lo suficiente*. Algunas de las estrategias que se podrían seguir son las siguientes:

- **Selección de individuos diversa**: utilizar algoritmos de selección que favorezcan la diversidad genética en la población. Por ejemplo, *la selección por torneo* puede modificarse para incrementar la probabilidad de seleccionar individuos menos adaptados.
- **Cruzamiento y mutación adaptativos**: ajustar las tasas de cruzamiento y de mutación dinámicamente durante la ejecución del algoritmo, en función de la diversidad genética de la población. Si la diversidad disminuye, aumentar la tasa de mutación puede ayudar a introducir nueva variabilidad genética en la población.
- **Inserción de nuevos individuos**: periódicamente, introducir nuevos individuos en la población, generados al azar (o mediante otros métodos de optimización), puede ayudar a mantener o aumentar la diversidad genética.

- **Uso de múltiples objetivos**: implementar una función de aptitud que considere múltiples objetivos, en lugar de trabajar con penalizaciones.

Estos son sólo algunos de los caminos que se podrían tomar para mejorar esa convergencia prematura detectada y, posteriormente, volver sobre el análisis de los parámetros.

Capítulo 6

Conclusiones y Trabajo Futuro

En este capítulo final, resumiremos los principales hallazgos de nuestra investigación sobre la aplicación de algoritmos genéticos al problema de diseño óptimo de vertipuertos, destacando las contribuciones más relevantes, y cómo estas permiten avanzar en el conocimiento del campo. Discutiremos también las implicaciones prácticas de nuestro trabajo, junto con las limitaciones encontradas durante el proceso de investigación. En base a estas reflexiones, presentaremos posibles direcciones para un futuro estudio, que puedan abordar las brechas identificadas, explorar nuevas preguntas que surjan durante su desarrollo, y aplicar los conocimientos adquiridos en contextos más amplios o específicos.

6.1. Conclusiones

Los objetivos que se marcaron al principio de la investigación fueron la profundización en la movilidad aérea urbana y el negocio UTM, enfocándonos en dominar y aplicar heurísticas evolutivas, como los algoritmos genéticos, para mejorar las soluciones existentes en [**?**]. Planeamos mejorar las habilidades de programación en Python aprendidas durante el Grado para implementar estos algoritmos de manera efectiva, buscando soluciones factibles y mejores a problemas del mundo del UTM, basándonos en la optimización heurística. Además, queríamos comparar los resultados obtenidos con nuestra implementación, para finalmente dedicar esfuerzos en el ajuste de parámetros que permitiera realizar un estudio de sensibilidad, con el fin de incrementar la calidad de las soluciones propuestas.

Los objetivos principales de aprendizaje sobre algoritmos genéticos, negocio UTM e implementación se han cumplido. El trabajo realizado ha permitido dos cosas, por un lado, entender mejor lo que podía estar fallando en la implementación para futuras mejoras; y por otro, analizar mejor el modelo matemático, así como esbozar pequeños cambios en las restricciones que puedan mejorar considerablemente la implementación. No debe olvidarse que éstas fueron concebidas pensando en métodos de resolución exactos.

Cabe destacar que el uso de algoritmos genéticos ha supuesto un desafío que, al punto en el que queda esta investigación, aún conllevaría un trabajo importante para poder igualar la calidad de las soluciones que pueden ofrecer métodos exactos tradicionales. Si bien el tiempo de cómputo de estos métodos puede ser elevado, podría verse compensado por el

esfuerzo que supone la optimización del modelo matemático, de los parámetros y de la implementación. Y es que, sin lugar a dudas, uno de los fuertes de las heurísticas es el uso de conocimiento experto para el desarrollo del algoritmo. Pero a la vez, considero, lo convierte en una herramienta poco trasversal y escalable a otros problemas, mientras que un *software* comercial tiene un uso más universal. Si nos fijamos en los valores de aptitud que hemos conseguido no distan mucho de los que ofrecen en [**?**]. Existen numerosos estudios que utilizan los algoritmos genéticos para optimizar la localización geográfica de un vertipuerto [**?**, **?**], en cambio, las referencias sobre el uso de este tipo de técnicas de optimización para el VPD es más limitado. Por tanto, representa un avance, que aunque modesto, permite ampliar el conocimiento que se posee sobre este. La exploración realizada puede servir de punto de partida para futuros estudios con los que abrir nuevas líneas de investigación, contribuyendo así al desarrollo continuo de este campo de vanguardia.

Finalmente, a nivel personal, el aprendizaje de estas técnicas ha supuesto un reto a nivel teórico y práctico. Desde las primeras lecturas de publicaciones se pudo observar la complejidad que supone modelar un problema real en el ámbito aeroespacial. Respecto a la implementación, si bien Python es un lenguaje "amigo", quiero destacar que la imposibilidad de depurar una ejecución debido al gran número generaciones, dentro de las cuales se hacen, a su vez, multitud de operaciones, ha sido un verdadero desafío. Por otro lado, enfrentarme a una implementación que roza las 800 líneas de código ha supuesto un verdadero aprendizaje para un estudiante del Grado de Matemáticas. Principalmente, debido a la complejidad inherente del proyecto, la necesidad de comprender y optimizar cada fragmento del código, así como asegurar la integración efectiva de todas las partes, con el fin de que funcione de forma cohesionada e integrada.

Esta investigación supone también la culminación de mis estudios en el Grado de Matemáticas y, concretamente, del itinerario de Computación. Los contenidos de este trabajo me han permitido aplicar gran parte de los conocimientos adquiridos en dicho itinerario: estructuras de datos, diseño de algoritmos, recursividad, son sólo algunas de las principales herramientas que se han desplegado a lo largo de este trabajo; pero nada de esto habría sido posible sin la sólida base matemática que he adquirido durante el grueso de mi estudios. Muchas disciplinas matemáticas han formado parte de este trayecto: optimización, análisis multivariable, álgebra y modelos estadísticos, entre otras, son, de nuevo, sólo algunas de ellas. Ha sido muy gratificante poder "unirlo" todo; en ocasiones, "los árboles no nos dejan ver el bosque", y las matemáticas son un bosque enormemente rico y lleno de matices que, en su conjunto, se convierte en una herramienta poderosa que no tiene límites.

6.2. Trabajo futuro

En primer lugar, una de las mayores limitaciones que se han presentado a la hora de realizar este trabajo, ha sido el equipo informático. Esto queda de manifiesto en el contraste entre los tiempos de ejecución de las subsecciones **??** y **??**. Por tanto, las pruebas y ajustes del algoritmo se han visto afectadas. En el futuro, realizar el análisis de sensibilidad completo con un equipo informático con mejores prestaciones puede ayudar a encontrar una configuración para los parámetros más eficiente.

En este trabajo hay tres pilares que requieren de atención: el estudio de la penalización de las restricciones, la convergencia prematura y la optimización de los parámetros.

En cuanto a la penalización de las restricciones, existen numerosos métodos, más o menos sofisticados, para abordar este problema [**?**]. Sería conveniente explorar diferentes estrategias, e incluso trabajar en adaptar alguna a la situación concreta del modelo del problema presentado en esta memoria.

Respecto a la convergencia prematura, se han dado las siguientes vías de investigación: selección de individuos diversa mediante otros métodos de selección, cruzamiento y mutación adaptativos, inserción de nuevos individuos generados al azar o por otros métodos de optimización y, finalmente, el uso de múltiples objetivos como alternativa a la penalización por incumplimiento de las restricciones; véase el Capítulo **??** para una información más detallada.

Finalmente, la optimización de los parámetros podría ser mejorada. Por ejemplo, mediante métodos específicos para algoritmos genéticos, o con técnicas generales más dirigidas, las cuales, además, optimicen el tiempo dedicado a esta parte del proceso.

Sería también una gran aportación implementar y comparar los resultados obtenidos con diferentes técnicas de optimización como los algoritmos de enjambres de partículas, u otros algoritmos bio-inspirados. Esto no sólo pretende buscar un método algorítmico mejor, si no que también se pueden utilizar sus soluciones como migrantes de las poblaciones del algoritmo genético y, de esta forma, acotar de forma más precisa el espacio de búsqueda.

Me gustaría, especialmente, tratar en mayor profundidad el problema de optimización de operaciones en un vertipuerto [**?**]. Si bien la infraestructura es vital, en mi puesto de trabajo actual, el principal problema es cómo van a interactuar las aeronaves dentro de ella. En este sentido, creo que puede ser una buena vía de investigación para el futuro.

Finalmente, las restricciones del modelo pueden ser revisadas con mayor detalle. Los propios autores de [**?**] comentan la idoneidad de buscar nuevas restricciones de colectividad que no se han tenido en cuenta como, por ejemplo, que todas las celdas de taxi estén conectadas.

Apéndice A

Entrevista previa realizada

Durante el desarrollo de esta investigación, tuve la oportunidad de entrevistar a Marta García, responsable técnica del equipo encargado de los proyectos europeos de los que se hace cargo la división de UTM (de sus siglas en inglés, *Unmanned Traffic Management*), equipo del que formo parte en *Indra Sistemas* desde mayo del 2023. Marta es Ingeniera Aeroespacial y Máster en Sistemas de Trasporte Aéreo por la Universidad Politécnica de Madrid. Actualmente, ocupa el puesto de Ingeniera Senior de Sistemas U-space[1] y ATM (de sus siglas en inglés, *Air Traffic Management*) con amplia experiencia en proyectos relacionados con el UTM. La entrevista se llevó a cabo con el objetivo de obtener perspectivas valiosas sobre el diseño de vertipuertos debido al profundo conocimiento que Marta tiene de este tipo de infraestructuras. En concreto, Marta lleva la carga principal del proyecto EUREKA [**?**], el cual está destinado a desarrollar los procesos de llegada, salida y cambio de dirección para los vertipuertos. Sus aportaciones han sido útiles no sólo para la elaboración de esta investigación, sino también para mi crecimiento profesional como Ingeniero de Sistemas en Indra. A continuación, se presenta la transcripción completa de la entrevista.

P: **¿Qué factores influyen en el beneficio de un vertipuerto?**

R: *Para poder medir con la máxima veracidad posible el beneficio de un vertipuerto habría que tener en cuenta qué tasa de uso tiene en cuanto a la cantidad de operaciones tanto de carga de mercancías como de pasajeros se consiguen cubrir. Al hilo de esto mismo, dependerá mucho la capacidad del vertipuerto. Si se prepara un vertipuerto que tiene la capacidad suficiente para abarcar la demanda esperada, el beneficio será mucho mayor que si tiene más capacidad que demanda o viceversa. De hecho, otro factor importante es saber que el vertipuerto está construido con vistas a poder servir el tráfico no solo actual, sino también que depara en lo siguientes años. Siempre se pueden llevar a cabo modificaciones en cuanto a futuras demandas, pero el estar preparado para ellas en la medida que se puede, aporta gran beneficio a la infraestructura global.*

P: **¿Cuáles son las infraestructuras básicas que conforman un vertipuerto?**

[1] *El denominado "concepto U-space" engloba un conjunto de sistemas, servicios y procedimientos específicos que han sido diseñados para permitir el acceso seguro, eficiente y asequible al espacio aéreo de operaciones de UAS numerosas o complejas, sobre la base de desarrollos técnicos con un alto grado de digitalización y automatización* [**?**].

R: *Un vertipuerto tiene que constar, como mínimo, de las siguientes infraestructuras:*

- *FATO(s).*
- *Zona de aparcamiento o stand(s). En algunos casos, podría coincidir la FATO con el stand, pero para un vertipuerto en el que se espera cierta carga de tráfico, esto no sería viable.*
- *Calle de rodaje o taxi, en el caso habitual de que la FATO no coincidiera con el STAND.*
- *Hangar(es) con zona de carga de baterías y donde se podrían incluir oficinas para operadores.*
- *Terminal o zona de espera para pasajeros.*
- *Torre de control o similar para alojar a los operadores del vertipuerto. Al igual que sucede con aviación tripulada, se podría analizar el caso de torres remotas y, en ese caso, incluso poder alojarse junto a controladores de aviación tripulada.*

P: **¿Qué factores influyen en el coste de la infraestructura?**

R: *El coste de la infraestructura estará delimitado por lo siguientes factores:*

- *El primero de ellos, el coste de planificación. Antes de iniciar los trabajos de edificación, debe existir un estudio que analice la demanda de pasajeros y operaciones, llevando a cabo los respectivos pronósticos, de manera que se pueda tomar una primera decisión sobre la inversión total a realizar.*
- *El segundo, y siguiendo la línea de la pregunta anterior, los costes de construcción y correspondiente mano de obra de las propias infraestructuras.*
- *El tercero y último correspondería a los costes de testeo, validación y certificación del vertipuerto por parte de las autoridades competentes.*

P: **¿Cuáles son los principales factores que afectan el rendimiento operativo de un vertipuerto?**

R: *El rendimiento operativo de un vertipuerto puede llegar a estar limitado por múltiples factores, sobre todo cuando se habla del manejo de VTOL como un vehículo que, hasta ahora, no se ha tratado en una infraestructura independiente junto a otros vehículos. Principalmente se podría decir que los factores de mayor afección son:*

- *Disponer de una red segura para poder cerciorarse de que los VTOL pueden operar bajo los niveles de seguridad esperados. Esto es, proteger adecuadamente las zonas de rodaje, stands, carga de baterías, FATO, etc. Esto no solo se refiere a disponer estas áreas en zonas libres de obstáculos, sino a señalizarlas correctamente y a que tengan las separaciones adecuadas entre unas y otras para poder evitar potenciales conflictos en las operaciones.*
- *Disponer de suficientes centros de carga. El rendimiento se puede ver enormemente decrementado si un vehículo no puede realizar su carga a tiempo porque no dispone del servicio adecuado.*
- *Se ha de disponer de un servicio de mantenimiento 24 horas de cara a posibles problemas que puedan sufrir los vehículos en cualquier fase de su operación.*
- *Disponer del suficiente personal para operar en el vertipuerto. Esto comprende tanto a los gestores/controladores del mismo, como al staff necesario en FATOS e infraestructuras adyacentes.*

- *Disponer de las ayudas adecuadas de radiovigilancia y navegabilidad, de manera que se puedan gestionar correctamente las operaciones de los VTOL.*

P: **¿Cómo se podría medir la eficiencia operativa de un vertipuerto?**

R: *Bajo mi opinión, la eficiencia operativa de un vertipuerto debe tener en cuenta diferentes factores y no consolidarse con un cálculo único en el cual puedan faltar ciertas métricas de elevado interés. Digamos que se trataría de asignar ciertos indicadores operativos que podrían formarse. Por ejemplo:*

- *Número de operaciones de drones por hora/media hora en términos tanto de despegue como de aterrizaje. El hecho de calcularlo de 30 minutos en 30 minutos radica en el hecho de que estos vehículos tardan mucho menos tiempo en realizar el rodaje, despegue y aterrizaje que las aeronaves tripuladas.*
- *Movimiento diario de pasajeros.*
- *Número de operaciones que se pueden gestionar simultáneamente. En función del tamaño del vertipuerto, varios drones podrían despegar o aterrizar al mismo tiempo (similitud con el caso de las aproximaciones paralelas de aviación tripulada).*

P: **¿Cómo se aborda la disposición de las diferentes infraestructuras que conforman el vertipuerto?**

R: *En el momento de abordar la disposición de las infraestructuras, se requiere realizar un estudio previo en el que se pueda analizar la demanda esperada, el tipo de drones que se espera que operen en el vertipuerto, la geografía del lugar y la cercanía a áreas críticas como puede ser un aeropuerto. Además de esto, se debería tener en cuenta la cercanía a áreas urbanas siempre evitando potenciales molestias de ruido, emisiones y/o interferencias con otras infraestructuras ya que la mayor parte de las operaciones esperadas de los VTOL serán de gran utilidad en estas zonas (tanto transporte de pasajeros como de carga).*

P: **¿Qué factores intervienen a la hora de ubicar las infraestructuras que conforman el vertipuerto?**

R: *Los factores a tener en cuenta en el momento de ubicar las infraestructuras básicas de un vertipuerto se pueden dividir en lo que respecta a, tal y como llamamos en el caso de aeropuerto, lado tierra (afección a pasajeros) y lado aire (afección a vehículos). De esta manera, para el lado aire, los factores que intervienen en esa ubicación serían:*

- *Las servidumbres de seguridad de la zona.*
- *El espacio aéreo adicional necesario para la fase de aproximación y salida de drones.*
- *La disposición de stands respecto a terminales de pasajeros debe estar correctamente dispuesta de manera que cada una conecte con un stand.*
- *Los stands han de estar conectados de manera adecuada con las zonas de calles de rodaje de VTOL.*
- *De la misma manera, esas calles de rodaje han de estar correctamente conectadas con una o varias FATO.*
- *Las calles de rodaje deberán estar debidamente distanciadas y la conexión entre ellas no debería abarcar ángulos muy pequeños siendo, como mínimo, de 90 grados.*

Y para el lado tierra:

- *Las zonas de acceso al vertipuerto deberán estar posicionadas cerca de la terminal de carga de pasajeros para evitar largos trayectos.*
- *Las terminales tendrán que disponer de tantas zonas de embarque como sean necesarias para poder cubrir la demanda esperada.*

P: **¿Existen desafíos particulares o estrategias que se emplean para garantizar tanto la comodidad de los pasajeros como la eficiencia en el movimiento de las aeronaves dentro de las instalaciones?**

R: *Aparte de la ya mencionada guía publicada por la FAA, actualmente cada país está buscando su propia estrategia para tratar de acomodarse tanto a las necesidades de los vehículos como a las de los pasajeros. Se busca con ello tener una aproximación similar a la que se emplea en aeropuertos. Esto es, de cara a los pasajeros:*

- *Disponer de zonas de descanso en las propias terminales.*
- *Posicionar los vertipuertos en zonas en las que se pueda facilitar un aparcamiento gratuito o de bajo coste.*
- *Distribuir pantallas en la terminal y en el propio VTOL de manera que se pueda visualizar la información relativa a la operación a realizar y que así el pasajero pueda sentirse cómodo.*

Y de cara al movimiento de vehículos:

- *Evitar que las calles de rodaje se encuentren muy cerca unas de otras.*
- *Evitar ángulos agudos de conexiones entre calles para no tener que realizar grandes giros constantemente en vuelo.*
- *Disponer las FATO, stands y calles de rodaje de manera que la rodadura no sea muy extensa.*
- *Facilitar las zonas de salida de hangares tras la debida carga de los vehículos y que el acceso a las calles de rodaje sea sencillo.*

P: **¿Cómo se aborda la distribución espacial de las áreas críticas para garantizar la seguridad y eficiencia de las operaciones?**

R: *La FAA publicó recientemente la primera guía de diseño y disposición de vertipuertos. Aunque en Europa se está empezando a trabajar en generar algo similar, es una buena pauta a seguir en el momento de conocer cómo abordar la estructura espacial de las áreas críticas. El principal factor que se ha de tener en cuenta es que debe existir una cierta separación entre todas y cada una de las infraestructuras anteriormente mencionadas, y esto dependerá del tipo de vehículo (peso y dimensiones, principalmente) que va a operar en el vertipuerto. Como comentaba anteriormente, la señalización de áreas críticas es imprescindible. Al igual que ocurre en los aeropuertos, las separaciones entre infraestructuras, las zonas donde el VTOL debe encontrarse en cada una de ellas, zonas prohibidas, restringidas, etc. Han de estar siempre debidamente señalizadas. Además de todo esto, todas aquellas áreas que supongan zonas de actividad crítica, deberán estar protegidas de un buffer de seguridad, de manera que si el vehículo en algún momento se desvía de los límites establecidos, la seguridad de las operaciones, pasajeros y del mismo vehículo no se vean comprometidas.*

P: **¿Existen restricciones específicas de seguridad en el UAM que deban considerarse?**

R: *En el momento de hablar de seguridad, a la hora de construir la infraestructura, se deben tener, como mínimo, ciertas consideraciones en cuenta:*

- *Las FATO deben estar lo suficientemente distancias, no solo de otras FATOs, sino también de cualquier otra infraestructura en la que pueda encontrarse otro vehículo (stand) o pasajeros en espera (terminal de pasajeros).*
- *La zona de fase final de aterrizaje así como la de inicial de despegue debe estar completamente despejada de infraestructuras para poder permitir la operación eficiente y segura de los VTOL.*