Diseño Avanzado de Algoritmos

Actividad 2 – Planificación multiobjetivo de rutas de drones en entornos urbanos

Báchelor en Informática – Universidad UNIPRO

Alumno: Eduardo ROBLEDO MARTÍNEZ

Repositorio GitHub: https://github.com/erobmar/dronerouting

Profesor: Bernardo RONQUILLO JAPÓN

Fecha: 05/01/2026

Tabla de contenido

Índice de tablas

Índice de figuras

1 – Introducción

Uno de los problemas clásicos de la optimización combinatoria es el de la planificación de rutas para vehículos autónomos. En el caso de los drones de reparto se presentan, además, restricciones adicionales como áreas con distinto nivel de riesgo y zonas de exclusión aérea que, junto a las limitaciones de capacidad de la batería, complican significativamente el problema.

La presente actividad plantea este problema simplificado al uso de un único dron que, partiendo de un hub central debe visitar el conjunto de todos los clientes y regresar al hub, minimizando simultáneamente distintos indicadores; la distancia recorrida, el riesgo acumulado y el número de recargas.

Debido al carácter combinatorio del problema el espacio de soluciones crece en relación factorial con el número de clientes. Esto hace que los métodos exactos sólo sean viables para mapas de tamaño reducido. Por este motivo se incluyen tres enfoques complementarios; un algoritmo exacto basado en Branch & Bound para mapas reducidos, heurísticas geométricas rápidas para obtener soluciones iniciales factibles y una metaheurística de Simulated Annealing para mejorar la calidad de las soluciones en mapas más grandes.

En este trabajo se aborda el análisis del comportamiento de estos métodos sobre un conjunto de mapas de diversos tamaños. Se evaluará el compromiso entre calidad de la solución y tiempo de ejecución y la escalabilidad de cada enfoque.

2 – Modelado del problema

2.1 – Modelado de la ciudad y restricciones geométricas

El modelado de las ciudades se realiza como un espacio bidimensional en el que se definen distintos elementos relevantes:

- **Nodos**: Puntos por los que pasará previsiblemente la ruta. Se distinguen tres tipos:
  - **Hub**: Nodo único en cada mapa que actúa como punto de partida y llegada del dron. Podría asimilarse a la sede de la compañía de transportes.
  - **Clientes**: Cada uno de los usuarios que esperan una entrega y que deben ser visitados, al menos, en una ocasión.
  - **Nodos de recarga**: Donde el dron puede recargar su batería completamente
- **Zonas prohibidas (no-fly zones):** Regiones poligonales que el dron no tiene permitido atravesar.
- **Zonas de riesgo (risk-zones):** Regiones poligonales que tienen un factor de riesgo asociado, haciendo que el coste de riesgo aumente cuando el dron las sobrevuela.

Cada nodo está definido con un identificador único y sus coordenadas cartesianas mientras que las zonas se definen como una secuencia de vértices en el plano.

Una trayectoria rectilínea se considera factible si el segmento que une sus vértices no atraviesa una zona prohibida. Adicionalmente, se tiene una limitación energética: Cada desplazamiento consume batería proporcionalmente a la distancia recorrida. Los nodos de recargan restauran la batería al valor máximo establecido.

2.2 – Grafo de navegación y generación implícita de aristas

El problema se ha modelado utilizando un grafo de navegación implícito cuyos nodos corresponden a los definidos en el mapa. Las aristas se irán generando bajo demanda usando cada uno de los algoritmos.

Una arista entre dos nodos u y v únicamente existe si el dron puede desplazarse entre ellos cumpliendo todas las restricciones. Esta factibilidad se evalúa de manera dinámica en cada nodo mediante una función de transferencia que realiza una primera comprobación rápida para verificar que no se cruzan zonas prohibidas con bound boxes para descartar casos muy evidentes con el menor cómputo posible.

Si el desplazamiento es factible, se calcula el coste asociado al mismo, compuesto por la distancia recorrida, el riesgo acumulado en caso de atravesar alguna zona de riesgo y el consumo de batería.

En el caso de que la batería no sea suficiente, se permite insertar visitas intermedias a nodos de recarga, siempre que exista una secuencia de desplazamientos factibles que permitan alcanzar el destino final.

Utilizando este enfoque se evita construir un grafo completo, cuyo tamaño crecería cuadráticamente con el número de nodos y se permite integrar restricciones geométricas y energéticas de manera natural. Su formato modular permite, además, su reutilización para cualquiera de los algoritmos estudiados.

2.3 – Funciones objetivo y dominancia de Pareto

Una solución completa del problema es un recorrido completo que parte del hub, visita todos los nodos una vez y regresa al hub. Cada solución llevará asociado un vector de costes compuesto de tres componentes que se deben minimizar; distancia total recorrida, riesgo acumulado y número total de recargas.

Dado que estos objetivos pueden estar contrapuestos, se afronta el problema desde una perspectiva multiobjetivo aplicando el concepto de dominancia de Pareto, según la cual una solución domina a otra si no es peor en ninguno de los objetivos y es estrictamente mejor en al menos uno de ellos. El conjunto de soluciones no dominadas constituye lo que se conoce como Frontera de Pareto del problema.

Este enfoque es especialmente adecuado para el algoritmo Branch & Bound, donde hay que mantener un conjunto de soluciones no dominadas. En el caso de heurísticas y metaheurísticas resulta menos práctico, dado que los objetivos se agregan en una función de coste escalar para simplificar el proceso.

3 – Arquitectura de la implementación

3.1 – Organización modular del sistema

El programa se ha desarrollado en Python 3 siguiendo una arquitectura modular que posibilite la reutilización de componentes y la comparación entre enfoques. Los módulos se han organizado en función de su responsabilidad del siguiente modo:

- **common**: Contiene las utilidades compartidas por todos los algoritmos. Incluye las clases para el manejo de archivos JSON, operaciones geométricas y grafos.
- **exact_bb:** Implementa el algoritmo Branch & Bound exacto
- **geo_heuristics:** Agrupa las heurísticas geométricas para obtener soluciones factibles de forma rápida. Incluye el vecino más cercano factible y una variante voraz ponderada.
- **metaheuristics:** Incluye la metaheurística de Simulated Annealing, empleada para mejorar soluciones iniciales.
- **experiments:** Contiene los scripts de ejecución de experimentos que ejecutan de forma automática los distintos métodos en cada uno de los mapas almacenados en el directorio data y recopila los resultados en un archivo CSV.

Esta organización permite ejecutar y evaluar cada instancia y algoritmo de forma independiente evitando duplicidad de código.

3.2 – Representación de soluciones y coste

Una solución, como ya se ha mencionado con anterioridad, consiste en la secuencia en la que el dron debe recorrer cada uno de los destinos, comenzando y finalizando en el hub y visitando los nodos de recarga necesarios.

El coste asociado a cada solución se realiza de manera acumulativa, evaluando cada desplazamiento mediante la función de transferencia descrita anteriormente. Para cada tramo, se comprueba la factibilidad, se actualizan los acumulados de distancia y riesgo y se actualiza la batería restante, permitiendo añadir visitas intermedias a nodos de recarga. Este coste se expresa como un vector que recoge la distancia recorrida, el riesgo acumulado y el número de recargas. Es utilizado directamente por el algoritmo exacto para realizar comparaciones en términos de dominancia de Pareto, y sirve como base para heurísticas y metaheurísticas.

4 – Algoritmos

4.1 – Algoritmo exacto Branch & Bound multiobjetivo

Este algoritmo explora el espacio de soluciones definido por las permutaciones de los clientes. Por esta razón sólo es computacionalmente viable en mapas de tamaño reducido. Cada nodo del árbol corresponde a una solución parcial definida por un prefijo del orden de visita y los nodos ya visitados.

Durante la exploración del árbol, la ruta se va construyendo incrementalmente con cada solución parcial. Si en algún punto la solución parcial es inviable, la rama correspondiente se poda. Se

mantiene, además, un conjunto de soluciones no dominadas para aplicar poda por dominancia. Si el coste acumulado de una solución está dominado por alguna solución ya encontrada, la rama se poda.

Con este enfoque se consigue reducir considerablemente el espacio de búsqueda respecto a un algoritmo exhaustivo, aunque su complejidad aumenta de manera factorial con el número de clientes. Por esto, se emplea solo en mapas de tamaño reducido donde resulta viable obtener la frontera de Pareto óptima.

4.2 – Heurísticas geométricas

Se han definido dos heurísticas geométricas basadas en criterios voraces. Ambas construyen la ruta de manera incremental, seleccionando en cada paso el siguiente nodo a visitar a partir del nodo actual.

La primera de estas heurísticas, de “vecino factible más cercano”, selecciona en cada paso al cliente alcanzable cuya distancia geométrica desde el nodo actual sea mínima. Este método prioriza trayectos cortos y permite obtener soluciones válidas con un coste computacional reducido, aunque no presta especial atención al riesgo del trayecto.

La segunda heurística es una variante voraz ponderada. En ella los distintos objetivos se combinan mediante una función de coste escalar. El siguiente cliente a visitar se decide minimizando este coste. Esto permite tener en cuenta información del entorno en la toma de decisiones.

El tiempo de ejecución de ambas heurísticas es realmente bajo, incluso para mapas de gran tamaño y son válidas tanto para obtener soluciones finales como para obtener soluciones aproximadas con las que iniciar métodos de metaheurísticas.

4.3 – Metaheurística: Simulated Annealing

Este método permite abordar mapas de mayor tamaño que el algoritmo exacto, explorando el espacio de soluciones de forma probabilística, aceptando soluciones teóricamente peores para escapar de óptimos locales.

A partir de una solución inicial factible generada por una de las heurísticas geométricas, se construyen soluciones vecinas aplicando pequeñas modificaciones en el orden de la solución (empíricamente, intercambiando las posiciones de dos clientes). Cada una de estas nuevas soluciones es evaluada mediante una función de coste que combina los objetivos de distancia y riesgo.

Se emplea una variable de temperatura, que se inicializa a un valor elevado. Esta variable modela la probabilidad de que el algoritmo acepte soluciones que empeoren el coste actual y va disminuyendo conforme a un patrón de enfriamiento. Cuanto mayor sea esta temperatura mayor probabilidad de aceptación y cuanto menos sea, más conservador será el comportamiento del algoritmo. De este modo se favorece la exploración al inicio de la búsqueda y la explotación de las mejores soluciones a medida que el algoritmo converge.

Este algoritmo permite explorar el espacio de soluciones a partir de soluciones iniciales obtenidas por las heurísticas geométricas, aunque en los experimentos realizados no se observa una mejora sistemática de dichas soluciones.

5 – Experimentos y resultados

Como parte empírica del trabajo se proponen cuatro mapas de distintos tamaños (10, 15, 20 y 25 clientes) que serán ejecutados sobre cada uno de los algoritmos siempre que sea computacionalmente viable. Para cada mapa se han ejecutado las heurísticas geométricas y la metaheurística de Simulated Annealing mientras que el algoritmo exacto de Branch & Bound sólo se aplicó al mapa de menor tamaño. Para cada método, además, se ha medido el tiempo de ejecución y el coste asociado a la solución encontrada a efectos de análisis.

Se observa que, para el mapa de menor tamaño, "01_gotham.json" con 10 nodos cliente, el algoritmo exacto ha obtenido la frontera de Pareto óptima. La metaheurística de Simulated Annealing ha encontrado una solución coincidente con la del algoritmo exacto y los métodos de heurísticas geométricas obtienen soluciones factibles, aunque de menor calidad que la ya mencionada, eso sí, en tiempos de ejecución prácticamente despreciables.

Para los mapas de tamaño intermedio (02_mordor.json y 03_valladolid.json), de 15 y 20 nodos cliente, el algoritmo exacto deja de ser computacionalmente viable, por lo que se descarta su uso a partir de este punto. Las heurísticas geométricas ofrecen resultados factibles en tiempos de ejecución muy reducidos, y la metaheurística de Simulated Annealing iguala e incluso mejora el coste de dichas soluciones a costa de un tiempo de ejecución mayor, aunque aún tolerable.

Finalmente, en el mapa más grande (04_andorra.json), con 25 nodos cliente, se observa que el tiempo de ejecución de la metaheurística de Simulated Annealing aumenta significativamente. Sin embargo, no se observa que, en los experimentos realizados, consiga mejorar las soluciones obtenidas por las heurísticas geométricas llegando, en el mejor de los casos, a igualarlas. Esto pone de manifiesto que, para mapas grandes, el coste computacional adicional de Simulated Annealing no siempre se traduce en mejores soluciones.

A continuación, se presentan los resultados de la experimentación empírica en forma de tabla.

Tabla 1 - Resultados de experimentos

Mapa	N	Método	Tiempo	Distancia	Riesgo	Recargas
01_gotham.json	10	nearest_feasible	0,0007	63,3157	75,4142	0
01_gotham.json	10	greedy_weighted	0,0006	61,9887	49,9557	0
01_gotham.json	10	simulated_annealing	0,4458	49,0371	35,7145	0
01_gotham.json	10	exact_bb	0,8872	49,0371	35,7145	0
02_mordor.json	15	nearest_feasible	0,0028	79,7831	31,2090	0
02_mordor.json	15	greedy_weighted	0,0026	92,9422	43,5827	0
02_mordor.json	15	simulated_annealing	1,2817	92,9422	43,5827	0
03_valladolid.json	20	nearest_feasible	0,0097	149,9289	214,5367	0
03_valladolid.json	20	greedy_weighted	0,0083	111,3257	61,8444	0
03_valladolid.json	20	simulated_annealing	2,5194	112,2700	56,9053	0
04_andorra.json	25	nearest_feasible	0,0377	128,2304	49,2053	0
04_andorra.json	25	greedy_weighted	0,0346	135,9778	58,7001	0
04_andorra.json	25	simulated_annealing	8,7326	135,9778	58,7001	0

6 – Discusión

Los resultados de la fase de experimentación recogidos en la Tabla 1 permiten analizar el comportamiento de cada algoritmo en función del tamaño del mapa y los objetivos considerados.

Para el mapa más pequeño, el algoritmo exacto basado en Branch & Bound llega a una solución óptima que permite validar el funcionamiento de los algoritmos basados en heurísticas y metaheurísticas. Precisamente, la metaheurística de Simulated Annealing es capaz de alcanzar la misma solución en un tiempo significativamente menor – alrededor de la mitad -. Por su parte, las heurísticas geométricas son capaces de obtener soluciones menos óptimas en tiempos de ejecución ínfimos.

A medida que se aumenta el número de clientes, el algoritmo exacto deja de ser computacionalmente viable debido al crecimiento factorial del espacio de búsqueda. Para mapas de tamaño mediano y grande, las heurísticas geométricas destacan por alcanzar soluciones no óptimas en tiempos de procesado muy reducidos. Entre las dos heurísticas incluidas, el algoritmo voraz presenta, en general, mejores resultados que el algoritmo de vecino factible más cercano. Esto pone de manifiesto la importancia de incorporar información del entorno en la toma de decisiones.

Por su parte, la metaheurística de Simulated Annealing presenta un comportamiento más costoso desde el punto de vista computacional y, es de esperar que ese uso extra de recursos se vea traducido en una mejora de los resultados. No obstante, y a la luz de los datos empíricos recogidos, de forma general no se aprecia una mejora sistemática respecto a las soluciones proporcionadas por las heurísticas geométricas. De hecho, en ocasiones mejora la solución de una de la heurística de “nearest feasible” pero no la voraz y, en el mejor de los casos, llega a la misma solución óptima alcanzada por estas heurísticas, pero con un tiempo de procesado, aproximadamente, 200 veces mayor. Este comportamiento sugiere que, por lo menos en los casos con los que se ha experimentado, el aumento de esfuerzo computacional no siempre se traduce en mejor calidad de la solución. Si bien, también es cierto que no se pueden extraer conclusiones categóricas de un grupo de mapas tan reducido y, para llegar a una conclusión más general debería realizarse una experimentación más intensiva, con muchos más mapas, de tamaños más diversos y ejecutado en diferentes configuraciones de hardware.

Por otro lado, se ha comprobado que para todos los mapas y con todos los métodos, se ha encontrado una solución que recorra todos los nodos con una única carga de batería. Esto indica que la solución óptima encontrada por los algoritmos ha conseguido recorrer todos los nodos sin pasar por ningún nodo de recarga y que, soluciones de menor calidad que podrían requerir recargas no forman parte de la frontera óptima observada en los experimentos.

En resumen, los experimentos apuntan a que no hay un método universalmente superior para la resolución de este tipo de problemas y que la elección del enfoque depende en gran medida de diversos factores, como el tamaño del mapa o los requisitos de calidad de la respuesta y tiempo de procesado.

7 – Conclusiones y trabajo futuro

En este trabajo se ha afrontado un problema de planificación de rutas para drones en entornos urbanos con restricciones espaciales y energéticas combinando un enfoque exacto y otros aproximados con el uso de heurísticas. Se ha modelado un sistema que permite integrar zonas de exclusión aérea, áreas de riesgo y limitaciones energéticas y una arquitectura modular que facilite la reutilización de código y la comparación entre mapas y algoritmos.

Los resultados empíricos muestran que el algoritmo exacto basado en Branch & Bound es adecuado para mapas pequeños, obteniendo soluciones óptimas y validando las heurísticas. Sin embargo, su aplicación es inviable para mapas de tamaño mediano y grande. En este contexto las heurísticas geométricas han destacado por su eficacia en la búsqueda de soluciones óptimas y su eficiencia temporal.

La metaheurística de Simulated Annealing, por su parte, presenta un tiempo de ejecución mucho mayor y en los experimentos no ha mostrado una mejora sustancial y sistemática de los resultados obtenidos por otros métodos. Esto sugiere que las heurísticas ya obtienen una solución cercana al óptimo (si no, directamente, éste) y dejan poco margen de mejora para este método.

Los resultados obtenidos sugieren que un modelado geométrico adecuado, junto con heurísticas bien diseñadas, pueden reducir la necesidad de algoritmos más complejos para refinar los resultados dado que las soluciones que produces son muy cercanas al óptimo incluso en mapas de tamaño considerable.

Por otro lado, llama la atención que todas las soluciones ofrecidas por el programa hayan conseguido encontrar una ruta completa sin necesidad de recargar la batería, lo que indica que la planificación es eficiente desde el punto de vista de las restricciones energéticas.

Como trabajo futuro se podría continuar la experimentación incluyendo mapas de tamaños muy diversos para comprobar si, a partir de cierto tamaño, comienza a ser más rentable usar metaheurísticas o el cómputo extra no llega nunca a compensar el encontrar una solución mejor. Sería interesante, también, estudiar cómo influye el paralelismo en este tipo de problemas y si ayuda a mejorar la solución y la eficiencia de los algoritmos o añade demasiada carga de sincronización como para que la diferencia sea notable.

8 – Anexos

8.1 – Anexo A: Formato de mapas JSON

El formato de los archivos JSON sigue la siguiente estructura:

- Parámetros generales: Como el nombre, número de clientes, batería máxima…
- Nodos: Hub, clientes y puntos de recarga
- Zonas prohibidas: Descritas como un conjunto de puntos que generan un polígono
- Zonas de riesgo: Descritas del mismo modo, aunque estas es posible atravesarlas

A continuación se muestra un mapa de ejemplo:

```
{
  "city_name": "Gotham City",
  "description": "Instancia valida con 10 clientes, 3 no-fly, 2 risk zones, y ecargas.",
  "max_battery": 120.0,

  "nodes": [
    { "id": "H",  "type": "hub",      "x": 0.0,   "y": 0.0 },

    { "id": "R1", "type": "recharge", "x": 4.0,   "y": 0.0 },
    { "id": "R2", "type": "recharge", "x": 8.0,   "y": 0.0 },
    { "id": "R3", "type": "recharge", "x": 12.0,  "y": 0.0 },

    { "id": "C1",  "type": "client",  "x": 2.0,   "y": 2.0 },
    { "id": "C2",  "type": "client",  "x": 3.0,   "y": -2.0 },
    { "id": "C3",  "type": "client",  "x": 6.0,   "y": 3.0 },
    { "id": "C4",  "type": "client",  "x": 6.0,   "y": -3.0 },
    { "id": "C5",  "type": "client",  "x": 9.0,   "y": 2.0 },
    { "id": "C6",  "type": "client",  "x": 9.0,   "y": -2.0 },
    { "id": "C7",  "type": "client",  "x": 13.0,  "y": 2.0 },
    { "id": "C8",  "type": "client",  "x": 13.0,  "y": -2.0 },
    { "id": "C9",  "type": "client",  "x": 16.0,  "y": 3.0 },
    { "id": "C10", "type": "client",  "x": 16.0,  "y": -3.0 }
  ],

  "forbidden_zones": [
    {
      "id": "NF1",
      "polygon": [
        { "x": 4.5, "y": -1.0 },
        { "x": 5.5, "y": -1.0 },
        { "x": 5.5, "y":  1.0 },
        { "x": 4.5, "y":  1.0 }
      ]
    },
    {
      "id": "NF2",
      "polygon": [
        { "x": 10.5, "y": -1.2 },
        { "x": 11.5, "y": -1.2 },
        { "x": 11.5, "y":  1.2 },
        { "x": 10.5, "y":  1.2 }
      ]
    },
    {
      "id": "NF3",
      "polygon": [
        { "x": 14.5, "y": 1.0 },
        { "x": 15.5, "y": 1.0 },
        { "x": 15.5, "y": 3.5 },
        { "x": 14.5, "y": 3.5 }
      ]
    }
  ],

  "risk_zones": [
    {
```

```
      "id": "RZ1",
      "risk_factor": 2.0,
      "polygon": [
        { "x": 2.0, "y": 1.0 },
        { "x": 7.0, "y": 1.0 },
        { "x": 7.0, "y": 4.0 },
        { "x": 2.0, "y": 4.0 }
      ]
    },
    {
      "id": "RZ2",
      "risk_factor": 1.5,
      "polygon": [
        { "x": 9.0, "y": -4.0 },
        { "x": 14.0, "y": -4.0 },
        { "x": 14.0, "y": -1.0 },
        { "x": 9.0, "y": -1.0 }
      ]
    }
  ]
}
```

8.2 – Anexo B: Representación gráfica de mapas y rutas

Las figuras incluidas en este anexo han sido generadas mediante un script auxiliar desarrollado en Python, utilizando la librería matplotlib a partir de los archivos JSON de los mapas y el archivo CSV con los resultados de los experimentos. Dado que ese script no forma parte de los requisitos del enunciado no se incluye como parte del código de la actividad en el repositorio.

Las rutas representadas corresponden al orden de visita de los clientes y tienen fines ilustrativos. La validación geométrica se lleva a cabo durante la ejecución de los algoritmos por lo que las trayectorias reales pueden diferir de las representadas.

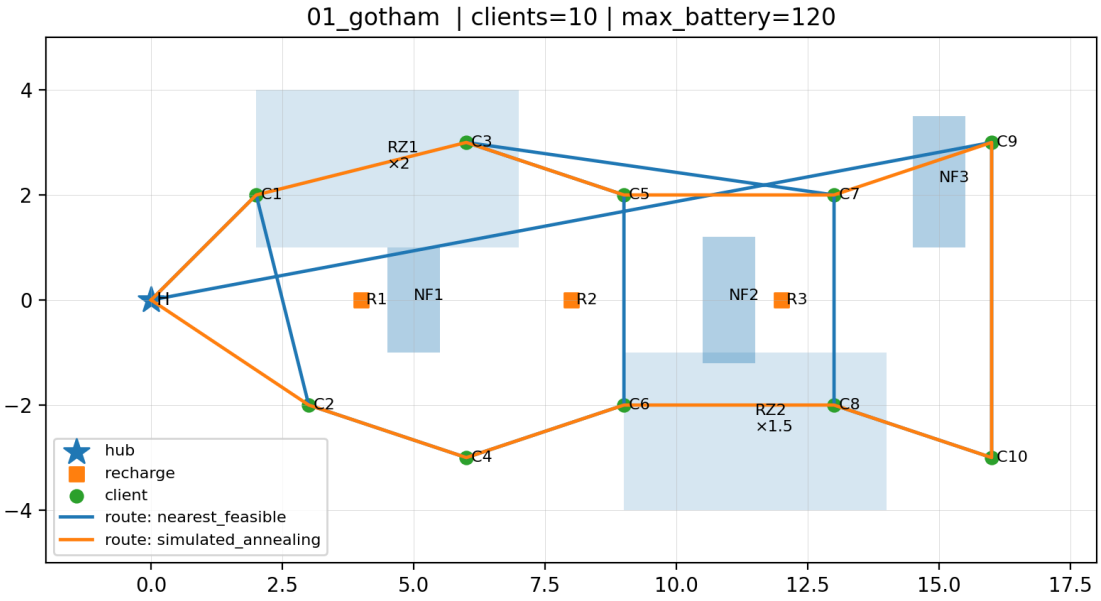

Figura 1 - Representación geométrica del mapa 01_gotham y rutas obtenidas por los distintos métodos.

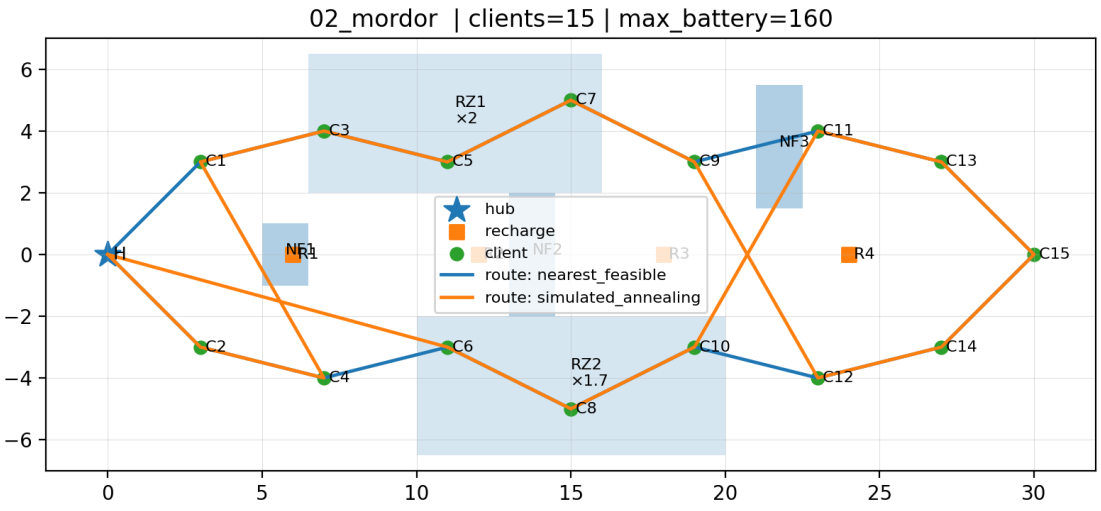

Figura 2 -Representación geométrica del mapa 02_mordor y rutas obtenidas por los distintos métodos.

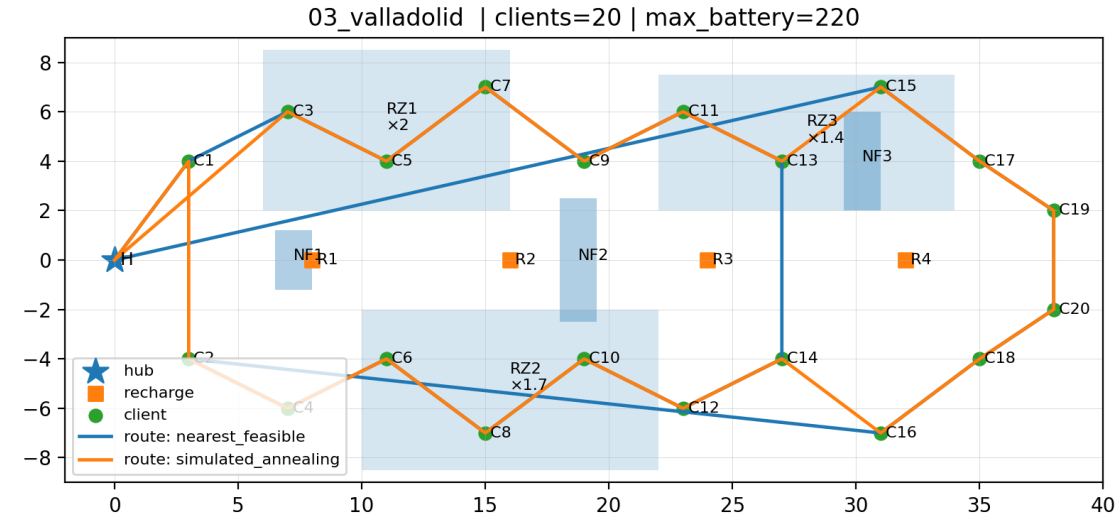

Figura 3 - Representación geométrica del mapa 03_valladolid y rutas obtenidas por los distintos métodos

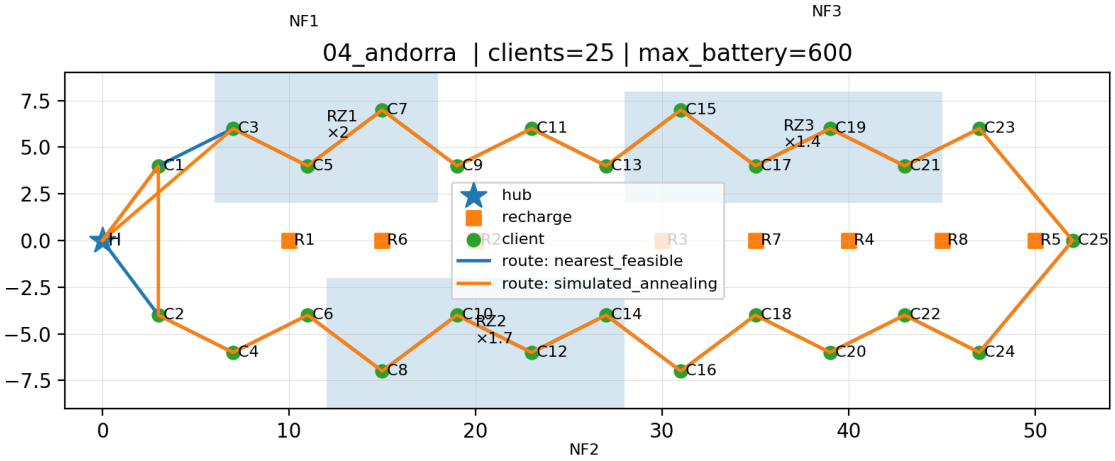

Figura 4 - Representación geométrica del mapa 04_andorra y rutas obtenidas por los distintos métodos

8.3 – Manual de usuario

El proyecto se puede descargar desde el siguiente repositorio de GitHub: https://github.com/erobmar/dronerouting

El programa tiene un punto de entrada a través del archivo main.py del directorio raíz. Este dará la bienvenida al usuario y un menú simple que permite ejecutar el script run_experiments.py de manera sencilla.

Para ello, basta con ejecutar el programa desde la raíz del proyecto tecleando:

```
python3 main.py
```