Trabajo Fin de Grado
Grado en Ingeniería de Tecnologías Industriales

Coordinación y dimensionamiento de flota de drones en problemas TSP a través de Python

Autor: Francisco Javier Gómez Lozano
Tutor: José Miguel León Blanco

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

Sevilla, 2021

Trabajo Fin de Grado
Grado en Ingeniería de Tecnologías Industriales

Coordinación y dimensionamiento de flota de drones en problemas TSP a través de Python

Autor:
Francisco Javier Gómez Lozano

Tutor:
José Miguel León Blanco
Profesor Contratado Doctor

Dpto. de Organización Industrial y Gestión de Empresas I
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2021

Agradecimientos

A mi familia por su apoyo incondicional.

A José Miguel por su total implicación y entrega en el proyecto.

Resumen

El éxito empresarial se podría resumir a su extremo como la forma de conseguir máximas ganancias minimizando los costes. En el día a día de muchas empresas, esos costes lo forman mayoritariamente aquellos empleados en logística. Con el objetivo de minimizarlos, surgieron dos problemas que hoy en día pueden definirse como clásicos: *Traveling Salesman Problem* y *Vehicle Routing Problem*.

Una de sus grandes peculiaridades, y el motivo por lo que se han asentado como base de estudio de otros problemas de optimización, es que, a pesar de su fácil comprensión, para dimensiones considerables estos problemas son irresolubles de forma exacta. Es por ello por lo que se emplean diferentes algoritmos de aproximación al óptimo, ya sean propios de estos problemas (heurísticas) como otros de corte genérico (metaheurísticas).

Dentro del sector de la distribución, el uso de los drones para el reparto se ha convertido en una de las líneas de trabajo con mayor potencial para revolucionar el sector. Abordaremos la implementación del reparto llevado a cabo por un camión y un dron, de manera conjunta, así como la extensión del problema a varios drones, resolviendo un caso concreto y analizando los resultados obtenidos, con el objetivo final de dimensionar la flota de drones óptima.

Abstract

Business succes can be summed up in a nutshell as the way to maximize profits minimizing costs. In the day-to-day business of many companies, these costs are mainly incurred by those employed in logistics. In order to minimise these costs, two problems arose that today can be defined as classics: Traveling Salesman Problem and Vehicle Routing Problem.

One of their great peculiarities, and the reason why they have become the basis for the study of other optimization problems, is that, in spite of their easy comprehension, for considerable dimensions these problems are unsolvable in an exact way. This is why different algorithms are used to approximate the optimum, whether they are specific to these problems (heuristics) or others of a generic nature (metaheuristics).

Within the distribution sector, the use of drones for delivery has become one of the lines of work with the greatest potential to revolutionize the sector. Therefore, we will discuss how to deploy a drone to assist some target customers on a truck route, ending by extending the problem to multiple drones, exposing for a specific case the optimal drone fleet size.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Notación

TFG	Trabajo Fin de Grado
TSP	Traveling Salesman Problem
VRP	Vehicle Routing Problem
mTSP	Multiple Traveling Salesman Problem
TSPTW	Traveling Salesman Problem with Time Windows
PTSP	Probabilistic Traveling Salesman Problem
CVRP	Capacited Vehicle Routing Problem
DCVRP	Distance Capacited Vehicle Routing Problem
VRPTW	Vehicle Routing Problem with Time Windows
VRPB	Backhauls Vehicle Routing Problem
PVRP	Periodic Vehicle Routing Problem
FAA	Administración Federal de Aviación
FSTSP	The flying sidekick TSP
NN	Nearest Neighbor Algorithm
NI	Nearest Insertion Algorithm
PSO	Particle Swarm Optimization
GA	Genetic Algorithm
ACO	Ant Colony Optimization

1 INTRODUCCIÓN

Traveling Salesman Problem y *Vehicle Routing Problem* son dos de los problemas más estudiados y con más aplicaciones dentro del mundo de los problemas de optimización combinatoria, constando a sus espaldas una infinitud de aplicaciones tanto prácticas como teóricas.

El objetivo de este Trabajo Fin de Grado (en adelante *TFG*), y lo que me ha motivado a realizarlo, no es otro que el de adentrarse en esta problemática, analizando en detalle sus características, sus orígenes y sus aplicaciones. Trataremos de abordar las distintas formas de resolución del mismo, analizando las ventajas e inconvenientes de las mismas y exponiendo una serie de metaheurísticas aplicables a diferentes problemas.

Además, a través de una variante muy concreta y con una gran proyección futura, propondremos su implementación en Python, redescubriendo un lenguaje de programación en auge y ampliamente utilizado en una gran variedad de empresas y situaciones.

A fin de dar mayor claridad al proyecto, la estructura del documento es la siguiente:

- *Capítulo 1: Introducción.* Breve explicación de lo que deparará este TFG, dejando constancia de la motivación para la realización del mismo.
- *Capítulo 2: Problemas de Rutas*. Comenzamos a adentrarnos en la problemática. Detallaremos las características intrínsecas tanto del TSP como del VRP, así como su historia y orígenes. Expondremos un breve modelo matemático del TSP a modo de ejemplo explicativo y numeraremos las variantes más típicas de ambos problemas.
- *Capítulo 3: Nuestro Problema.* Explicación de la inclusión de un dron en los problemas de ruteo de vehículos, así como la adaptación para múltiples drones.
- *Capítulo 4: Algoritmos clásicos de resolución.* Desgranaremos las tres clases de algoritmos para la resolución de los problemas en cuestión: Métodos Exactos, Heurísticas y Metaheurísticas, exponiendo diversos ejemplos de los mismos.
- *Capítulo 5: Nuestros Algoritmos.* De forma más detallada, trataremos de exponer los algoritmos elegidos para la resolución de nuestro problema, así como el por qué de dicha elección, de forma comparativa.
- *Capítulo 6: Resultados y Conclusión.* Exposición y comparación de los resultados. Conclusión y futuras parcelas de estudio.
- *Anexo:* Código en Python[1], [2] detallado paso a paso.

2 PROBLEMAS DE RUTAS

Entendemos, grosso modo, por el término logística industrial[3] como el estudio de la organización del desplazamiento y la manutención de materiales de cualquier índole, asegurando la mayor eficiencia y eficacia posible. Dentro del mercado en el que nos encontramos en la actualidad, la logítica industrial es utilizada por las distintas empresas con el fin de conseguir ventajas competitivas. Por ello, reducir los costes tanto de aprovisionamiento como de distribución se ha convertido en los últimos años en una cuestión capital dentro de cualquier compañía.

En este contexto, el establecimiento de rutas para vehículos de la forma óptima generó un incipiente crecimiento en el estudio de la materia, encajando los problemas de TSP y VRP (analizados en profundidad más adelante) a la perfección.

Si bien es cierto que la aplicación de dichos problemas al mundo de la logística es la más utilizada y estudiada, resuelta interesante detenerse un momento en analizar otros campos de estudio con gran potencial en los próximos años.[4]

En la industria, por ejemplo, siguiendo la misma estrategia de la obtención de la ruta óptima, el problema TSP es utilizado en situaciones tales como la secuenciación de tareas de una máquina, entendiendo tal como el recorrido de menor coste (tiempo, distancia, gasto económico, etc) a realizar por una máquina sabiendo que debe emplearse en diversas tareas; la producción de circuitos eléctricos en tareas concretas tales como el perforado de placas, en el cual, siguiendo con la analogía característica de esta introducción, es vital saber el camino óptimo entre los distintos puntos a perforar; la impresión 3D, de forma similar a la perforación de placas, etc. [5][6]

Debido a la facilidad de comprensión del problema TSP, así como sus múltiples aplicaciones, es ampliamente utilizado como plataforma de estudio de diversos métodos generales de resolución, que, posteriormente, pueden ser aplicados a numerosos problemas discretos de optimización.

A continuación, en este capítulo abordaremos en detalle los problemas mencionados anteriormente (TSP y VRP), analizando en profundidad sus características, su historia, así como algunas de las variantes clásicas de los mismos.

2.1 Traveling Salesman Problem*[7]*

El problema del viajante (TSP debido a sus siglas en inglés, *Traveling Salesman Problem*) es uno de los problemas más conocidos y estudiados del campo de la matemática computacional, así como de los problemas de optimización combinatoria, caracterizado por la contraposición entre su simplicidad en definición y entendimiento y su extrema dificultad en su resolución. De hecho, si nos ceñimos en términos de eficiencia, el problema aún no ha sido resuelto.

Su definición es bastante trivial. El problema consiste en, dado un listado de nodos (por ejemplo,

ciudades) encontrar el trayecto de menor coste (ya sea en términos de distancia o tiempo) en el cual se visiten todos y cada uno de los nodos una sola vez, regresando finalmente al nodo inicial.

Figura 2-1. Ejemplo TSP[8]

Antes de entrar en profundidad en la problemática, hagamos una pequeña retrospección a sus inicios.

Aunque previamente, en un manual alemán para viajantes de 1832, se había abordado el problema, no fue hasta 1857 cuando se hace su primera formulación matemática, de la mano del matemático irlandés William Rowan Hamilton a través de su *Icosian Game*. El juego consistía en un dodecaedro regular cuyo objetivo era visitar cada uno de los veinte vértices que formaban el poliedro, una sola vez cada uno y acabando en el vértice de partida. Al ciclo formado al acabar dicho juego se le denominó ciclo hamiltoniano. Obviamente, el juego no consistía en encontrar la ruta óptima, pero se puede considerar el TSP como el ciclo hamiltoniano de menor longitud en un grafo, lo cual supuso el punto de partida para el estudio del problema en cuestión.[9]–[11]

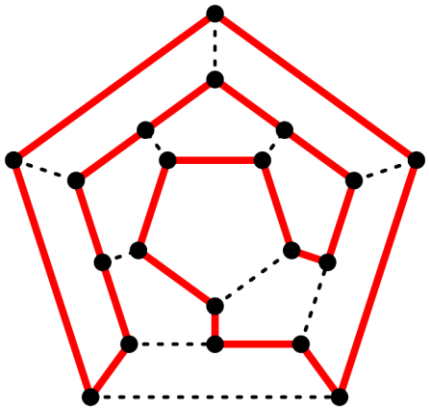


Figura 2-2. Ciclo Hamiltoniano. [12]

Figura 2-3. Juego Icosiano.[10]

Aunque el origen del estudio del problema concreto no está del todo claro, parece que no fue hasta la década de 1930 cuando el matemático austríaco Karl Menger estudió y definió por primera vez directamente dicho problema, denominándolo *Problema del Mensajero*[13]. Este problema consistía en encontrar la mínima longitud de la unión de un conjunto de puntos, sabiendo previamente

la longitud entre cada par de puntos. Para su resolución, consideró el algoritmo de "fuerza bruta", que básicamente se trata de probar todas las combinaciones posibles hasta encontrar la mejor. Como veremos más adelante, para una dimensión del problema no excesivamente grande, la utilización de dicho algoritmo es, sencillamente, inviable.

A pesar de que la acuñación del término *Traveling Salesman Problem* pertenece al matemático estadounidense Hassler Whitney, el mérito de la promulgación y difusión del TSP, dentro de la comunidad matemática, recae sobre el matemático estadounidense Merrill Flood[14], quien, gracias a su estudio del TSP tratando de encontrar la optimalidad en rutas escolares, presentó la problemática en la Corporación RAND, cuya reputación y autoridad hizo que se ampliase exponencialmente la popularidad del problema, siendo esta corporación el centro intelectual de los estudios sobre el TSP en aquella época.

Desde ese momento, infinidad de estudios, realizados por científicos de toda índole, han versado sobre el problema en cuestión, otorgando al TSP el estatus de ser uno de los problemas más famosos de la matemática computacional.

2.1.1 ¿A qué clase de complejidad pertenece el TSP? *[15]*

La teoría de la complejidad computacional siendo un apartado dentro de la teoría de computación, basa su estudio en la clasificación y asignación de las distintas problemáticas computacionales, en función de la dificultad para su resolución.

Asignar el TSP a un tipo de problema puede resultar más confuso de lo que cabría esperar pues, la simple diferenciación entre problemas P y NP es un estudio denso y arduo. No concierne dentro del alcance de este trabajo el abordar dicha disyuntiva, por lo que me limitaré a definir brevemente los distintos tipos, explicar a cuál pertenece el TSP y por qué.

2.1.1.1 Tipo P

Los problemas de complejidad P son aquellos problemas de decisión que pueden ser resueltos en tiempo polinómico a través de la entrada por una máquina de Turing determinista. En otras palabras, cuando el problema en cuestión puede ser resuelto en un *tiempo polinómico*, entendiendo por tal el formado por una función polinómico utilizando las variables implicadas.

Hablamos de los problemas más sencillos o abordables.

2.1.1.2 Tipo NP

Siguiendo con la definición utilizada en P, los problemas tipo NP son aquellos que pueden ser resueltos en un tiempo polinómico por una máquina de Turing no determinista. Dando una definición más coloquial, un problema NP es aquel cuya solución puede ser verificada en un tiempo polinomial.

Dentro de la categoría de problemas NP, encontramos los que se conocen como **NP-complete**. Un problema es NP-complete si dicho problema pertenece al conjunto de problemas NP y, además, cualquier problema NP es transformable al primero. Para afirmar que un problema es NP-complete, bastaría con que fuese equivalente a otro y, de forma automática, lo sería a todos los ya demostrados.

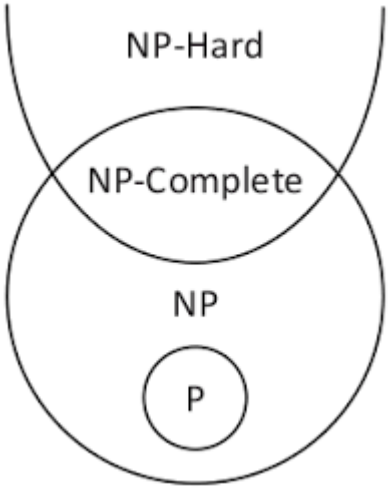


Figura 2-4. Clases de complejidades.[16]

Esta clase de problema es considerada como la más compleja de todas, en el cual aún no se ha encontrado ningún problema que la valide.

2.1.1.3 Tipo NP-hard

A pesar de la similitud en el nombre, esta clase de complejidad no pertenece al conjunto NP. Se podría definir un problema NP-hard como aquel que, dando una máquina oráculo de este (máquina abstracta capaz de resolver ciertos problemas de decisión en una simple operación), todos los demás problemas en NP pueden ser resueltos en un tiempo polinomial.

Los problemas NP-hard son al menos tan costosos como un problema NP-complete, pero no tienen por qué pertenecer al conjunto de problemas NP. Dicho de otro modo, los problemas NP-complete serían la intersección entre los problemas NP y los NP-hard

El TSP pertenece a esta clase de problemas. Pero ¿por qué? ¿por qué no NP-complete? La respuesta podría ser tan sencilla como que el TSP no puede pertenecer al conjunto de problemas NP.

Retomando la defición coloquial de los problemas NP, decíamos que son aquellos cuya solución puede verificarse en un tiempo polinomial. Supongamos que partimos de una potencial solución de un problema TSP de cuatro nodos. La única forma de saber si nuestra solución es la óptima sería comparándola con cada una de las combinaciones posibles, 6 en nuestro ejemplo ((N-1)! combinaciones para el caso general, siendo N el número de nodos). Puede parecer asequible su comparación, pero ¿y si partimos de 10 nodos? ¿y de 100? Por poner un sencillo ejemplo, si aplicásemos un TSP a las escasas 8 capitales de provincia andaluzas, una situación bastante verosímil, tendríamos que resolver un total de 5.040 combinaciones, surgiendo un problema que, aunque aún sea factible obtener la solución de manera exacta, se torna bastante más complicado.

Obviamente, la comprobación de que la solución de un TSP, de una dimensión no necesariamente muy grande, es la óptima, no podría hacerse en un tiempo polinomial, por lo que el TSP no pertenece al conjunto de problemas NP, siendo un problema NP-hard.

La dificultad implícita en estos problemas insta a la necesidad de abrir un abanico de posibilidades en cuanto a algoritmos de resolución se refiere, siendo fundamental la inclusión de algoritmos heurísticos y metaheurísticos de los cuales obtendríamos soluciones aproximadas en un

tiempo razonable. Hablaremos de todo ello en los próximos capítulos.

2.1.2 Modelización y formulación*[17]*

A lo largo de los años, el TSP ha sido modelizado y formulado de diversas formas, adaptándose a las características intrínsecas de la variante objeto. Nosotros expondremos una de las formulaciones más comunes y sencillas del problema, la cual nos sirva a modo de ejemplo explicativo.

Antes de comenzar, cabe destacar los dos tipos de TSP clásicos en cuanto a estructura se refiere:

- *Simétrico:* El coste entre dos nodos es el mismo independientemente del sentido. Tomando por nodos un conjunto de ciudades, la distancia de la ciudad *i* a la ciudad *j* será la misma que de la *j* a la *i*. En esta casuística, la matriz de costes es simétrica.
- *Asimétrica:* El coste entre dos nodos puede diferir en función del sentido en el que se visiten. La distancia de la ciudad *i* a la *j* no tiene por qué ser la misma que de la *j* a la *i*, e incluso puede que uno de los dos trayectos no sea posible. En este caso, la matriz de costes será asimétrica.

Abordaremos una formulación del problema asimétrico. Para ello previamente debemos definir una serie de términos, de acuerdo con la teoría de grafos: Tenemos un grafo G = (V, A, C), donde V es el conjunto de vértices del grafo, A el conjunto de aristas y C es la matriz de costes (c_{ij} es el coste de ir de *i* a *j*) y *n* el número de nodos.

Definimos también un ciclo simple aquel en el que el vértice inicial del mismo coincide con el final, siendo cada vértice visitado una única vez. El ciclo hamiltoniano, ya definido previamente, sería el ciclo simple que visita cada uno de los vértices. En este punto, tratamos de localizar el ciclo hamiltoniano de menor coste.

Comencemos con las variables de decision del problema. Denominamos x_{ij} a la variable que representa la arista que conecta el vértice *i* con el *j*, tomando el valor de 1 si esa arista es parte de la solución final del problema y 0 en caso contrario. A modo de inicialización, cada c_{ii} tomarán valores muy grandes, para así asegurarnos que la arista que va desde un nodo hasta sí mismo nunca será parte de la solución del problema. Llegados a este punto, la función objetivo del problema sería la siguiente:

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$$

Cuyas restricciones que implican que a cada vértice llegará una única arista y de cada vértice saldrá una única arista son las siguientes:

$$\sum_{j=1}^{n} x_{ij} = 1$$

$$\sum_{i=1}^{n} x_{ji} = 1$$

En este punto surge un problema el cual las restricciones definidas hasta el momento no consiguen solventar: la creación de lazos o subconjuntos. Nótese que si dentro de un mismo grafo se crean varios subciclos en los cuales se cumplen las restricciones de entrada y salida única en cada vértice, no hay ninguna restricción que se incumpla hasta el momento, pero no define el TSP. Por ello, debemos incluir ciertas restricciones que obliguen a romper esos posibles lazos.

Una forma de hacerlo sería a través de la inclusión de nuevas variables de decisión denominadas u_i. Estas variables representan el momento de la solución en el que el vértice *i* es visitado, pudiendo tomar valores de 2 a *n*, reservando el valor 1 al nodo de partida (lo que sería el almacén). Con ello, tendríamos la siguiente restricción:

$$u_i - u_j \leq (\mathrm{n} - 1)(1 - x_{ij}) - 1$$

En el caso de que x_{ij} pertenezca a la solución, nos quedaría $u_i - u_j \leq -1$, lo cual se cumple ya que el nodo *i* se visita justo antes que el nodo *j*. En caso contrario, llegaríamos a $u_i - u_j \leq n - 2$, inecuación que se cumple para todos los posibles valores de u_i y u_j por lo que la inecuación se cumple. En el caso de la existencia de lazos, esta restricción no podría cumplirse ya que no podría establecerse una secuencia de visitas a los nodos por no estar relacionados entre sí.

2.1.3 Variantes del TSP*[18]*

Pronto, tras la acuñación del TSP y su posterior estudio, se hizo evidente que el problema era adaptabale a una infinidad de situaciones, con sus características específicas, así como la propia evolución que sufrió a medida que el paso del tiempo iba generando nuevas necesidades. A continuación, se detallan algunas de esas variables

2.1.3.1 Multiple Traveling Salesman Problem*[19]*

Se podría considerar esta variante como precusor del VRP, el cual se abordará más adelante en este mismo capítulo. Como adelanto, si considerásemos los vehículos propios del VRP con capacidad infinita o lo suficientemente grandes como para que la capacidad no suponga una restricción del problema, entonces mTSP y VRP coincidirían.

Definimos el *Multiple Traveling Salesman Problem* (mTSP) como aquel en el que habiendo una serie de nodos (ejemplifiquémoslos como ciudades) y un conjunto m de "*salesmen*", obtengamos las distintas rutas recorridas por todos los vendedores, de modo que todas las ciudades sean visitadas por un único vendedor, siendo el coste del trayecto el mínimo posible.

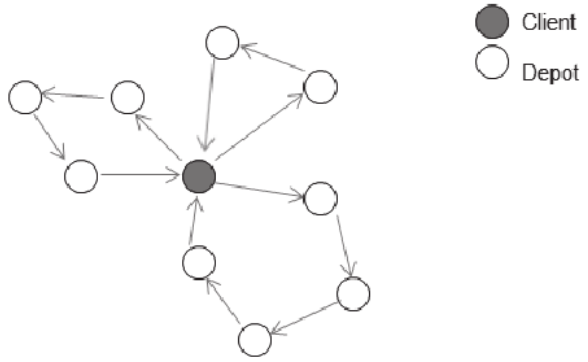


Figura 2-5. mTSP[20]

2.1.3.2 Traveling Salesman Problem with Time Windows*[21]*

Partamos, de nuevo, del TSP clásico. Contamos con un conjunto de nodos (que, de nuevo, ejemplificamos como ciudades y así lo mantendremos a lo largo de la explicación de las distintas variantes), y necesitamos conocer la ruta de menor coste en la que recorramos todos las ciudades una sola vez. En lo que difiere el *Traveling Salesman Problem with Time Windows* (TSPTW) es que cada nodo consta de un instante de "apertura" (denominado a_i) y de "cierre" (b_i). Bajo estos términos, el problema incluye la restricción de que cada ciudad i debe ser visitada en el intervalo de tiempo creado entre ai y bi. En el caso en que la ciudad sea visitada antes del instante ai, nuestro vendedor deberá esperar, con el correspondiente aumento del coste, hasta dicho instante, siendo imposible la visita a la ciudad en un instante posterior a bi.

Dicha variante puede ser combinada con el mTSP, con la única diferencia de que el problema constará de más de un vendedor.

2.1.3.3 Probabilistic Traveling Salesman Problem

En el *Probabilistic Traveling Salesman Problem* (PTSP), añadimos una nueva diferencia que, de igual modo, es combinable con las anteriores. En este caso, la consecución de la ruta de menor coste debe tener en cuenta que cada ciudad lleva asociada una probabilidad de que en ellas haya o no un consumidor que dé sentido a la visita.

2.2 Vehicle Routing Problem

A medida que la sociedad iba evolucionando, surgieron numerosos nuevos conflictos, los cuales debían ser solventados aplicando nuevas herramientas o reiventando las ya existentes. Del mismo modo, el TSP se generalizó añadiendo nuevas condiciones o restricciones, creando un nuevo tipo de problema el cual, desde su primera formulación hace más de 50 años, se convirtió en uno de los principales problemas de la Investigación Operativa.

El Problema de Enrutamiento de Vehículos (VRP por sus siglas en inglés *Vehicle Routing*

Problem) consiste en encontrar las rutas óptimas entre uno o varios depósitos o almacenes y un conjunto de ciudades (nodos), constando dichas rutas con el depósito como punto común y final del problema y siendo visitadas cada ciudad una única vez. Queda claro, desde este punto, que el VRP es una extensión del TSP.

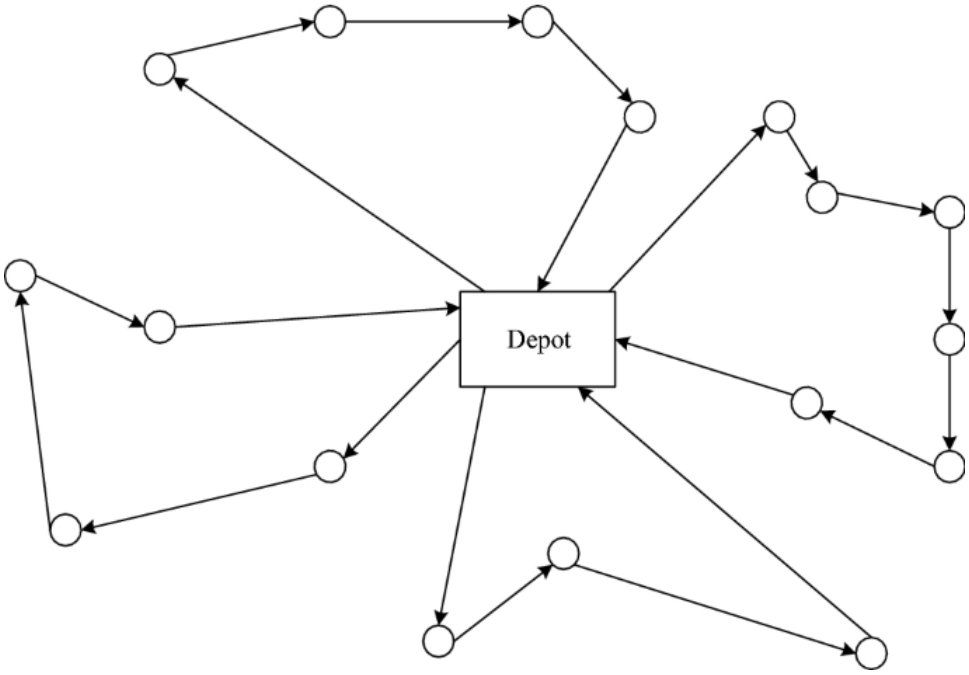


Figura 2-6. Solución de VRP[22]

Como ya hicimos en la explicación del TSP, antes de detallar de manera concisa las características del VRP, veamos de manera superficial sus orígenes.

No fue hasta 1959 cuando los matemáticos estadounidenses George Dantzig y John Ramser formularon matemáticamente por primera vez esta problemática, en su célebre "*The Truck Dispatching Problem*"[23]. Para ello, aplicaron el problema a la entrega de combustible por parte de una flota N de camiones a una serie de estaciones de servicios, partiendo de un único terminal. Con este fin propusieron una formulación de programación lineal de la cual obtuvieron una solución cercana a la óptima. La importancia que recae sobre dicho estudio no es únicamente la resolución de un VRP de dimensión considerable, sino la utilización de métodos de resolución tales como *Branch and bound*, el cual abordaremos en profundidad más adelante, así como la demostración de que, a pesar de la complejidad de los problemas de optimización combinatoria, su resolución era posible.[24]

Destacar además que el VRP se trata de un problema NP-hard, cuya explicación es análoga a la ya expuesta en el caso del TSP.

2.2.1 Variantes del VRP*[18]*

Desde el nacimiento del problema, al igual que con el TSP, surgieron una infinidad de variantes del mismo cuya finalidad era dar respuesta a las distintas situaciones reales a las que se podría enfrentar el mismo.

El TSP generalizado, conocido como VRP o CVRP (*Capacited VRP*) introduce la primera característica esencial de los problemas de enrutamiento de vehículos y es que la capacidad de la flota de vehículos comienza a ser una restricción más del problema, pues no es infinita. Con lo cual, queda definido el CVRP como un problema que busca la obtención de una serie de rutas, cuya suma de costes entre los nodos que la forman sea mínima, siendo cada nodo visitado una única vez por un único vehículo y la suma de las demandas de cada nodo no excede la capacidad intrínseca del vehículo. Como ya sucedía en el TSP, podríamos diferenciar entre el CVRP simétrico, en el cual el coste de ir de la ciudad *i* a la ciudad *j* es el mismo que el de la ciudad *j* a la *i*, y el CVRP asimétrico, donde esta igualdad no tiene por qué cumplirse.

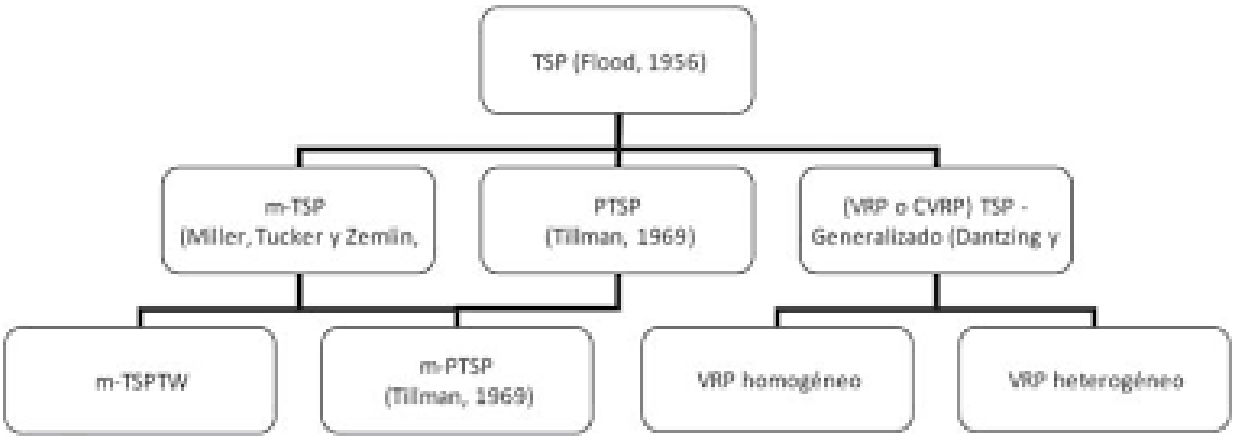


Figura 2-7. Variantes TSP-VRP[18]

A raíz de dicho problema inicial surgen dos grandes familias, las cuales a su vez tienen numerosas ramificaciones con características propias. Nos adentraremos a continuación a explicar las características propias de cada familia, así como a mostrar algunos casos concretos a modo de ejemplos.

2.2.1.1 VRP Homogéneo

Los problemas pertenecientes a la familia del VRP Homogéneo comparten una serie de características comunes en las que todas las ciudades o nodos controlan un mismo recurso. En función de los recursos controlados tendremos un tipo de problema. Procederemos a explicar algunos de ellos.

2.2.1.1.1 Distance VRP

En este caso, la restricción se produce en la máxima longitude admissible de cada ruta. Dicho de otro modo, la suma de distancias entre los distintos nodos que forman una ruta no puede exceder un tamaño de longitude máximo de la ruta, establecido previamente.

En el supuesto de que, además de dicha restricción exista una limitación en la capacidad de los vehículos como la previamente explicada, el problema pasaría a llamarse *DCVRP*.

2.2.1.1.2 VRP with Time Windows

Restricción ya abordada en el TSP con Ventana de tiempos, el VRPTW, incluye la novedad de que cada consumidor debe ser visitado en un interval de tiempo establecido previamente, siendo

imposible la visita al mismo una vez que dicho interval finalice y con la correspondiente penalización de espera en el caso de que se visite antes de que comience el intervalo

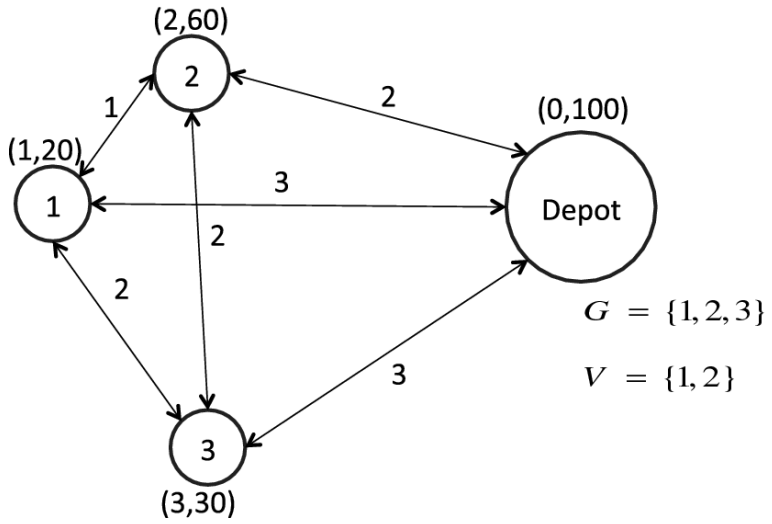


Figura 2-8. VRP with Time Windows[25]

En función de las características de esas ventanas de tiempo, encontramos a su vez nuevas variantes, como la VRPTD, en las que el intervalo es completamente rígido, VRPMTW, en la cual los intervalos de tiempo de los distintos nodos no son de la misma dimensión, o el VRPSTW, donde los límites de los intervalos no son completamente rígidos.

2.2.1.1.3 VRP with Backhauls

En esta casuística abreviada como *VRPB*, los clientes pueden devolver ciertas mercancías al vehículo, por lo que antes de visitar a un cliente cuyo deseo sea devolver una mercancía hay que tener en cuenta si en ese momento el vehículo tiene la capacidad necesaria para ello. Un caso extremo sería aquel en el que para que una mercancía pueda ser recogida, primero han de realizarse todas las entregas de la ruta.

En este caso quedan diferenciados dos tipos de clientes, los que van a recoger un pedido o los que van a entregarlos.

2.2.1.1.4 Split Delivery VRP

Relajación del VRP en el cual se permite que un mismo nodo sea visitado por distintos vehículos, bajo la condición de que dicha operación reduzca el coste total.

2.2.1.2 VRP Heterogéneos

En la familia de los VRP Heterogéneos, se hace referencia a componentes desiguales en los que los distintos nodos controlan recursos distintos. Algunos ejemplos son los siguientes:

2.2.1.2.1 Periodic VRP

A diferencia del VRP donde el periodo de planificación es de un único día, en el *PVRP* dicho periodo se extendiende a M días

2.2.1.2.2 Multi-Trip VRP

En este caso, un mismo vehículo dentro del periodo de planificación puede llevar a cabo varias rutas, por lo que no solo hay que tener en cuenta el diseño intrínseco de las rutas, sino además la disponibilidad de los diversos vehículos.

2.2.1.2.3 Multi-Objective VRP

Consiste en utilizar diferentes objetivos, ya sean referentes a las rutas, a los arcos, a los nodos o a los recursos, y obtener la solución en función del cómputo de objetivos.

3 NUESTRO PROBLEMA

Tal y como se ha explicado en el anterior capítulo de este TFG, tanto el TSP como el VRP constan de una infinidad de variantes que se adaptan a las necesidades y características que demande el autor. En ese aspecto, el espectro de maniobra es tan amplio que el número de casuísticas posibles crece de forma constante. Nosotros hemos escogido una concreta, elaborada dentro del *Departamento de Ingeniería Industrial y de Sistemas* de la *Universidad de Auburn*, Estados Unidos, por parte de la alumna *Amanda Chu* y el profesor *Chase Murray*, en el artículo denominado ***"The Flying Sidekick Traveling Salesman Problem: Optimization of Drone-assisted Parcel Delivery"***[26], el cual, en adelante en este TFG, lo mencionaremos siempre como el Artículo, el cual propone una variante del TSP que combina el reparto a través de drones y camiones de forma coordinada.

En este capítulo desgranaremos el Artículo, explicándolo en profundidad, haciendo una *review* del mismo, pero, antes de comenzar, detengámonos en analizar el por qué de la elección.

El avance y la sofisticación en cuanto a los modernos sistemas de reparto o distribución está adquiriendo en los últimos tiempos una velocidad de vértigo. El ahorro en los costes asociados al reparto de mercancías genera una ventaja competitiva que puede llegar a ser un factor diferencial entre las compañías del sector. En este sentido, el reparto con la ayuda de drones se encuentra a la vanguardia de las últimas investigaciones.

El reparto asistido por drones está siendo apoyado e incentivado por algunas de las mayores potencias globales, compañías tales como *Amazon*, *Google* o *Walmart*, por lo que la viabilidad de dicho reparto crece a pasos agigantados. Pero, a pesar de ello, se enfrenta a una serie de hándicaps que necesitan superar.

Figura 3-1. Reparto con dron de la compañía Walmart[27]

El ahorro en costes que supondrían el reparto asistido por drones choca frontalmente con ciertas limitaciones técnicas, tales como la autonomía de los drones, el peso máximo a transportar por un dron, la dificultad para evitar tendidos eléctricos o diversos obstáculos aéreos, etc. Aunque, sin lugar a duda, la mayor oposición a esta forma de reparto la encontramos en los términos legales, y en la preocupación social por la falta de privacidad a la que puede desembocar dicha situación. Sin detenernos en exceso en esta problemática, recalcar que la consecución en 2.019 por parte de la compañía *UPS* de la primera autorización por parte de la *Administración Federal de Aviación* (***FAA***) de EEUU para el reparto con drones ha supuesto un enorme avance en la resolución de este conflicto[28]–[30]

A modo de curiosidad, en el distrito madrileño de Villaverde, al tratarse de un entorno seguro para ello, se ha comenzado a probar el reparto a domicilio con drones.[31]

Centrándonos en lo que nos concierne, una vez se superen dichas adversidades y se implante de manera habitual el reparto asistido por drones, la máxima rentabilidad de este sistema se convertirá en un factor clave, adquiriendo los métodos de optimización y problemas como el TSP y el VRP una gran importancia. Es por ello, la elección del estudio del Artículo.

3.1 The flying sidekick TSP

En el Artículo se exponen dos clases distintas de TSP: ***The flying sidekick TSP*** y ***The parallel drone scheduling TSP***. Nosotros únicamente abordaremos el primero de ellos, el que posteriormente aplicaremos a través de Python. Queda como futuros estudios la aplicación del segundo.

The flying sidekick TSP (FSTSP) consta de un camión, un dron y un conjunto de clientes, los cuales deben ser visitados una sola vez, ya sea por el camión o por el dron. Debido a distintas razones como pueden ser el peso del paquete a entregar o la localización, algunos clientes solo pueden ser visitados por el camión. [32]

Cada salida del dron consta de tres puntos concretos: el lugar de salida, el cual puede ser el depósito o la localización de un cliente, con su correspondiente tiempo de servicio asociado para el cambio de baterías del dron; el cliente a visitar por el dron; y el lugar final del trayecto, el cual, de nuevo, puede ser o el depósito o el camión, donde se le debe sumar otro tiempo de servicio de recogida del dron. Cada vuelo del dron debe realizarse en un tiempo que nunca supere el tiempo de autonomía del mismo. Es importante destacar que en lo que dura el trayecto del dron, el camión puede visitar varios clientes.

El FSTSP asume ciertas condiciones para su aplicación, como, por ejemplo, que el dron realiza todo su vuelo, salvo la entrega al cliente, a velocidad constante, por lo que no puede detenerse en ningún punto intermedio con el objetivo de ahorrar batería, al tiempo que tampoco es factible la recogida del dron en un lugar que no sea un cliente ni en un cliente ya visitado.

3.1.1 Modelización y formulación

A continuación, nos dispondremos a analizar la función objetivo y las restricciones necesarias para el cumplimiento de las características expresadas con anterioridad, mostrando el modelo matemático completo. Con el fin de realizarlo de la forma más clara posible, iremos introduciendo las distintas variables a medida que aparezcan en el modelo. No trataremos de abordar todas y cada una

de las restricciones, descritas en detalle en el Artículo, sólo nos detendremos en las que consideramos más relevantes. Comencemos por la función objetivo:

$$Min\ t_{c+1}$$

Denotando por ***t*** al tiempo de llegada del camión a los distintos nodos, y siendo ***C={1, 2, …,c}*** el conjunto de clientes a visitar, la función objetivo trata de minimizar el tiempo de llegada del camión al cliente c+1, siendo los "clientes" 0 y c+1 igual al depósito. Una vez definida la función objetivo, vayamos a las restricciones.

En primer lugar, el modelo consta de las restricciones intrínsecas a cualquier TSP, como las descritas en el capítulo anterior. Con ello, se obliga al modelo a visitar cada cliente una sola vez, a que el camión salga y vuelva al depósito, así como a que, si llega a un nodo, salga después desde el mismo nodo y a eliminar las posibles subrutas o lazos. Por ello, aunque no volvamos a mostrar dichas restricciones, presentaremos la notación empleada, pues nos será necesaria en adelante.

El modelo define el conjunto ***N = {0, 1, …, c+1}*** al conjunto de todos los nodos visitables, diferenciando entre ***N_0={0, 1, …, c}*** como el subconjunto de clientes de partida del camión, y ***N_+={1, 2, …, c+1}***, el subconjunto de clientes de llegada del camión. Las variables de decisión serían ***x_{ij}***, cuyo valor sería igual a 1 si es parte de la solución el trayecto del camión entre los nodos ***i-j***, e ***y_{ijk}***, que valdría 1 si fuese parte de la solución que un dron partiese desde el nodo ***i***, visitase al cliente ***j***, y aterrizase en ***k***. En este punto, queda claro que ***i,j*** y ***k*** son los índices del problema. El conjunto P incluye a las tuplas ***{i, j, k}*** que cumplen las condiciones para que formen el vuelo de un dron.

$$\sum_{\substack{j\in C\\ j\neq i}} \sum_{\substack{k\in N_+\\ \{i,j,k\}\in P}} y_{ijk} \leq 1 \quad \forall i \in N_0$$

$$2y_{ijk} \leq \sum_{\substack{h\in N_0\\ h\neq i}} x_{hi} + \sum_{\substack{l\in C\\ l\neq k}} x_{lk} \quad \forall i \in C, j \in \{C: j \neq i\}, k \in \{N_+: \{i,j,k\} \in P\}$$

Comienzan ahora una serie de restricciones cuyo objetivo es limitar y caracterizar el vuelo del dron, así como las condiciones a las que se ve sometido el camión por el mismo. Por exponer dos ejemplos, la primera de las dos restricciones anteriores expone la obligatoriedad de que el dron visite a un mismo cliente como máximo una vez p. En cuanto a la segunda, restringe que, si un dron despega del nodo ***i*** y aterriza en el ***k***, el camión debe parar tanto en ***i*** como en ***k***.

$$t_i' \geq t_i - M * (1 - \sum_{\substack{j\in C\\ j\neq i}} \sum_{\substack{k\in N_+\\ \{i,j,k\}\in P}} y_{ijk}) \quad \forall i \in C$$

$$t_i' \leq t_i + M * (1 - \sum_{\substack{j \in C \\ j \neq i}} \sum_{\substack{k \in N_+ \\ \{i,j,k\} \in P}} y_{ijk}) \quad \forall i \in C$$

$$t_k' \geq t_k - M * (1 - \sum_{\substack{i \in N_0 \\ i \neq k}} \sum_{\substack{j \in C \\ \{i,j,k\} \in P}} y_{ijk}) \quad \forall k \in N_+$$

$$t_k' \leq t_k + M * (1 - \sum_{\substack{i \in N_0 \\ i \neq k}} \sum_{\substack{j \in C \\ \{i,j,k\} \in P}} y_{ijk}) \quad \forall k \in N_k$$

Las cuatro ecuaciones anteriores cobran una importancia particular dentro de nuestro problema en cuestión.

En la definición de la función objetivo, quedó ***t*** como el vector de tiempos de llegada del camión a los distintos nodos, siendo t_i, la llegada al nodo ***i***. De forma análoga, definimos el vector ***t'***, como el vector de tiempos de llegada del dron a los nodos, siendo t'_i quien determina el momento de llegada del dron al nodo ***i***.

Estas cuatro ecuaciones tienen el objetivo de definir una coordinación entre dron y camión en cuanto a tiempos se refiere. Si justo anteriormente habíamos expuesto las ecuaciones que imponían relaciones de posición, pues si el dron salía de ***i***, el camión de forma obligada debía pasar por dicho nodo (del mismo modo con la llegada del dron), ahora ambos deben llegar en el mismo momento al nodo ***i***, y del mismo modo, llegar ambos a la vez al nodo ***k***.

Para conseguir que las anteriores funcionen, nos hemos valido de la variable ***M***, que no es más que un número al menos igual o más grande al último instante de tiempo en el que el camión y el dron regresaron al depósito. Esta variable nos será de utilidad en las próximas restricciones.

$$t_k \geq t_h + \tau_{hk} + sl * (\sum_{\substack{l \in C \\ l \neq k}} \sum_{\substack{m \in N_+ \\ \{k,l,m\} \in P}} y_{klm}) + sr * (\sum_{\substack{i \in N_0 \\ i \neq k}} \sum_{\substack{j \in C \\ \{i,j,k\} \in P}} y_{ijk}) - M * (1 - x_{hk})$$

$$\forall h \in N_0, k \in \{N_+ : k \neq h\}$$

A través de esta ecuación se incluyen por primera vez los tiempos necesarios de recogida del dron por parte del camionero (***sr***) y de preparación para su lanzamiento (***sl***). Además, surge la variable τ_{ij}, la cual determina el tiempo que tarda el camión de ir desde el nodo ***i*** al ***j***. Básicamente, esta ecuación restringe el hecho de que para que el camión salga del punto ***k***, lugar donde también llega el dron, no podrá hacerlo hasta que el dron haya sido recogido y preparado para su lanzamiento, en caso de que sea necesario.

$$t'_k - \left(t'_j - \tau'_{ij}\right) \leq e + M * \left(1 - y_{ijk}\right) \quad \forall k \in N_+, j \in \{C : j \neq k\}, i \in \{N_0 : \{i,j,k\} \in P\}$$

Se limita en la anterior restricción el alcance del dron, denominando como ***e*** a la autonomía de vuelo del dron.

$$u_i - u_j \geq 1 - \ (c+2) * p_{ij} \quad \forall i \in C, j \in \{C; j \neq i\}$$

$$u_i - u_j \leq -1 + (c+2) * (1 - p_{ij}) \quad \forall i \in C, j \in \{C; j \neq i\}$$

$$p_{ij} + p_{ji} = 1 \quad \forall i \in C, j \in \{C; j \neq i\}$$

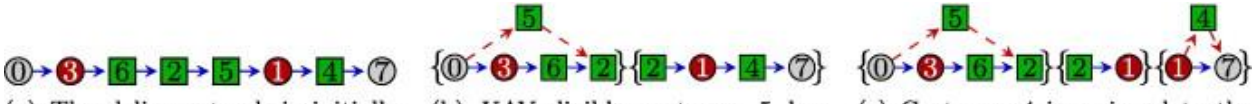
Surge una nueva variable, binaria en este caso, la cual se antoja necesaria para conseguir las limitaciones en cuanto a orden de visita, una vez que el dron se introduce en el trayecto. La variable p_{ij} determina que el nodo ***i*** ha sido visitado en algún momento antes que ***j***. Recalcar que, si el dron visita ambos nodos, no pueden ser visitados por el camión de manera secuencial.

$$t'_l \geq t'_k - M * (3 - \sum_{\substack{j \in C \\ \{i,j,k\} \in P \\ j \neq l}} y_{ijk} - \sum_{\substack{m \in C \\ m \neq i \\ m \neq j \\ m \neq k}} \sum_{\substack{n \in N_+ \\ \{l,m,n\} \in P \\ n \neq i \\ n \neq k}} y_{lmn} - p_{il}) \quad \forall i \in No\ , k \in \{N_+ : k \neq i\}, l \in \{C : l \neq i, l \neq k\}$$

En esta última restricción, se consigue el último paso en la limitación en cuanto a precedencia y ordenación se refiere. Si suponemos que el dron aterriza en algún punto al nodo ***k***, no podrá ser lanzado desde nodo ***l*** hasta que no se haya producido el aterrizaje en ***k***.

3.1.2 Heurística FSTSP

Llegados a este punto, tras la compresión del modelo matemático, surge la pregunta: ¿qué hace realmente esta variante? ¿cúal es la aplicación dentro del modelo? Para ello, nos ayudaremos de una imagen obtenida del mismo Artículo, la cual es bastante explicativa.



(a) The delivery truck is initially assigned to visit all customers. `truckSubRoutes` = `truckRoute` = $\{0, 3, 6, 2, 5, 1, 4, 7\}$. `Cprime` = $C' = \{6, 2, 5, 4\}$.

(b) UAV-eligible customer 5 has been removed from the truck's route and is assigned to the UAV. The UAV is launched from the depot (node 0) and is recovered by the truck at node 2. There are now two truck subroutes, $\{0, 3, 6, 2\}$ and $\{2, 1, 4, 7\}$; `Cprime` = $\{6, 4\}$.

(c) Customer 4 is assigned to the UAV. There are now three truck subroutes, and `Cprime` = $\{6\}$.

Figura 3-2. FSTSP Heurística[26]

El pseudo código propuesto en el Artículo y desarrollado en este TFG en el *Anexo 1*, muestra la implantación de lo que pretende reflejar esta imagen. Partimos de la ruta óptima del camión, calculada a través de distintos algoritmos que serán la temática de los dos prócimos capítulos del TFG. A su vez, tenemos definido los nodos a los que el dron le es factible visitar.

Para cada cliente potencialmente visitable por el dron, el código debe realizar un estudio de los costes que te supondrá la visita del dron, o el movimiento en el orden de visita de un determinado cliente, en contraposición con el ahorro o ganancias que genera. En caso de un ahorro mayor que el coste, se implementará dicho movimiento.

Con este objetivo, se desarrollarán una serie de funciones, las cuales procedo a explicar brevemente, siendo desgranadas a través de comentarios en el propio código:

- *Código principal del FSTSP:* En él, se produce la unión de las distintas funciones, efectuando un bucle que tenga en cuenta todos los posibles nodos a visitar por el dron.
- *calcSavings:* Calcula el ahorro de eliminar un cliente ***j*** de la ruta del camión. Destacar la posibilidad de un ahorro negativo.
- *calcCostTruck:* Coste de insertar un cliente ***j*** en otro punto de la ruta del camión.
- *calcCostUAV:* Coste de que, efectivamente sea el dron quien visite al cliente ***j***.
- *performUpdate:* Una vez decidido que hacer con el cliente ***j***, debemos actualizar todas las entradas del algoritmo, ya sea las distintas subrutas del camión, la ruta genérica, el tiempo de llegada del camión a cada nodo, etc.

3.1.3 Cuantificación de la flota de drones

Una vez elaborada la implementación del FSTSP, nos percatamos a través de los resultados obtenidos, los cuales se encuentran en detalle en el último capítulo de este TFG, que, en ocasiones, el dron se encuentra envuelto en distintas subrutas dentro de las cuales existen clientes potencialmente visitables, a su vez, por drones. En este sentido, llegamos a un punto en el que la heurística FSTSP sólo desarrolla su máximo potencial para problemáticas muy específicas, siendo, en el problema genérico, muchos clientes visitados por el camión cuando un dron podría realizar esa tarea.

En ese punto observamos que la adaptación de la heurística FSTSP al uso de varios drones podría generar unos beneficios bastante interesantes. Tal es así que, el mismo Chase Murray, esta vez junto a Ritwik Raj aborda dicha disyuntiva en otro artículo, "***The multiple flying sidekicks traveling salesman problem: Parcel delivery with multiple drones"***[33]. Cabe resaltar que nuestra implementación en Python con el uso de varios drones se aleja de la empleada por Murray y Raj. Nuestro cometido ha sido, el readaptar el FSTSP para varios drones, de manera simplificada. Utilizaremos, no obstante, el artículo a modo de fuente de información.

El mayor problema al ir introduciendo progresivamente más drones a la ecuación es la necesidad de una coordinación constante dron-camión y dron-dron. Y no únicamente nos referimos a lo expuesto anteriormente de la necesidad de que el camión se localice en los nodos donde despegan/aterrizan

todos los drones, sino que se antoja fundamental calcular la autonomía del dron, sabiendo el incremento de tiempo que supone la recogida y preparación de otros drones.

Al mismo tiempo, conseguir que cada cliente sea visitado una única vez cuando contamos con más de un dron, se convierte en una tarea cuanto menos compleja.

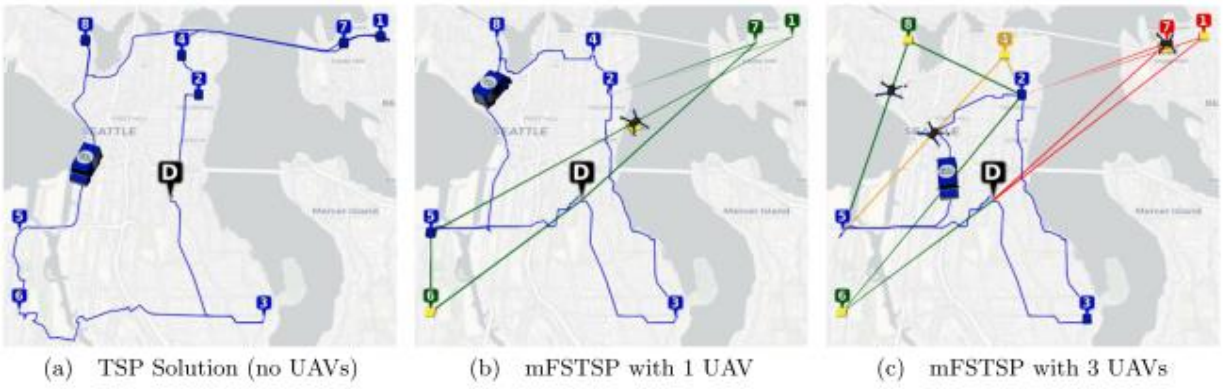


Figura 3-3. TSP con diferentes tamaños de flota de drones[33]

En el código que se muestra en el *Anexo 1*, desarrollamos la implementación de un número n de drones para el problema TSP, relajando en algunos aspectos las restricciones temporales y de coordinación. No obstante, lo que resulta interesante, a nivel conceptual, es la cuantificación de la flota de drones.

Aunque el reparto de drones, pensado para trayectos cortos, supone un ahorro en costes con respecto al reparto tradicional de camiones, no es tan sencillo el saber el tamaño óptimo de la flota de drones para una dimensión concreta del problema. Puede ocurrir que, para un conjunto de nodos relativamente pequeño, con clientes situados a distancias cortas entre sí, sobredimensionemos el problema de forma que un cierto número de drones ni siquiera sea necesario su uso o, utilizándolos, no supongan un ahorro tal que merezca la pena tenerlo.

Sin llegar abarcar el tema económico, sí que abordaremos distintos tamaños de flota de drones para varios problemas distintos, analizando las ventajas y desventajas de emplear más o menos drones.

4 ALGORITMOS CLÁSICOS DE RESOLUCIÓN

Una vez abordado tanto el problema TSP como el VRP en profundidad, resulta evidente la complejidad del mismo. Un problema de relativamente fácil compresión pero de una dificultad capital en cuanto a su resolución, pues el número de combinaciones va creciendo a ritmo de *(N-1)!*. Pero, ¿cómo se puede resolver?

La pregunta tiene una respuesta un tanto más extensa de lo que cabría esperar. Surgen entonces tres grandes grupos o familias de algoritmos de resolución, los cuales, a su vez, tienen sus propias características y sus campos de aplicación.

4.1 Métodos exactos*[34]*

Los métodos exactos pretenden encontrar la mejor solución del problema, en nuestro caso, la ruta o rutas óptimas en las que se visiten todos y cada uno de los nodos una única vez. En nuestro caso particular, como ya se ha explicado, al ser un problema de complejidad NP-hard, su aplicación se encuentra cuanto menos limitada, aunque dentro de los mismos algunos tienen un alcance mayor que otros.

4.1.1 Algoritmo de Fuerza Bruta*[35]*

Quizás, el más simple y robusto de los que se someterán a estudio. El algoritmo trata de encontrar la mejor solución del problema a través de la comprobación de todas y cada una de las combinaciones posibles.

Como ya se ha explicado con anterioridad, este método resulta útil para un número muy limitado de nodos pues, creciendo las combinaciones posibles a ritmo de *(N-1)/2!*, en el momento que llegamos a un número de nodos no excesivamente grande, su uso resulta inviable.

Al ser la mayoría de situaciones en las que se aplican el TSP y el VRP de una dimension mayor de la necesitada por el algoritmo de fuerza bruta, en la práctica este algoritmo es poco utilizado.

4.1.2 Programación Dinámica*[36]*

La Programación Dinámica, desarrollada por Richard Bellman en la década de 1950, se basa en descomponer el problema original en diversos subproblemas de manera recursiva, almacenando las soluciones de los mismos. De este modo, si al evaluar futuros subproblemas estos coinciden con subproblemas que ya habían sido solucionados con anterioridad, se podrán utilizar directamente dichas soluciones, ahorrando coste computacional y tiempo de resolución del algoritmo.

Con el objetivo previamente descrito, se producen relajaciones del problema original, de modo que los diferentes subproblemas sean similares entre sí y tengan utilidad.

De forma similar al algoritmo de fuerza bruta, la aplicación de la programación dinámica al TSP se limita a un número muy reducido de nodos, por lo que tampoco resulta uno de los métodos más

utilizados de resolución.

4.1.3 Programación Entera

En ciertos problemas, como pueden ser el caso del TSP y el VRP, las variables de decision deben ser valores enteros, denominándose *Problemas de Programación Entera* si es la única diferencia con respecto a los problemas de programación lineal, o, en caso de que solo algunas variables tengan la necesidad de ser valores enteros, se trataría de los *Problemas de Programación Entera-Mixta*.

Ya que, como hemos dicho, el TSP y el VRP pueden quedar enmarcados dentro de este subconjunto de problemas, las distintas soluciónes clásicas del subconjunto en cuestión son válidas para conseguir la resolución buscada.

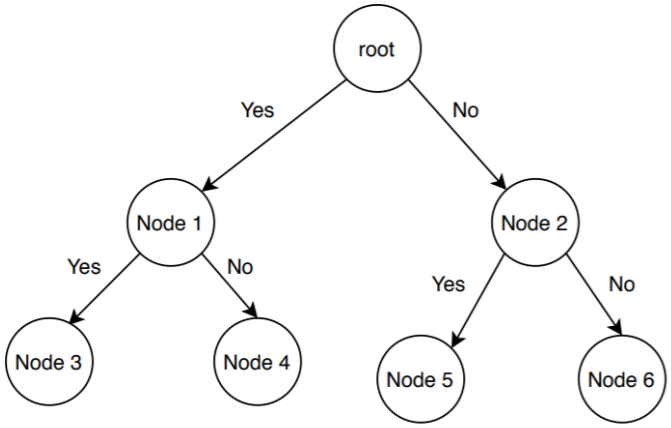


Figura 4-1. Simplificación visual de la técnica de Branch-and-Bound[37]

Para este tipo de problemas, la técnica más común es aquella denominada ***Branch-and-Bound***. Técnica surgida y refinada entre 1960 y 1970, además de ampliamente utilizada y estudiada, genera una solución exacta del problema actuando de la siguiente manera:

- En primer lugar, divide el problema en subproblemas cada vez más pequeños en los que se pueda evaluar la función objetivo de manera más rápida (*branch*).
- Acotando cómo de buena puede ser la solución óptima y descartando los diversos subconjuntos en los que la optimalidad no se produzca (*bound*)

Posteriormente, en la década de 1980, la utilización de los planos de corte supuso una mejora técnica del *Branch and Bound*, denominada ***Branch-and-Cut***, en la cual, gracias al corte de distintas ramas del árbol de decision generado, se produce un importante ahorro en tiempo computacional con respecto al *Branch and Bound*.

Si bien es cierto que estas técnicas pueden ser aplicadas para una mayor dimension del problema tratado, la naturaleza NP-Hard del TSP hace que los métodos exactos sean inoperables para dimensiones más cercanas a la realidad del problema, por lo que deben buscarse distintas vías de resolución.

4.2 Heurísticas*[38]*

Si flexibilizamos el objetivo de conseguir la optimalidad, dado el coste computacional a medida que aumentamos el número de nodos, aparecen una serie de algoritmos, únicamente utilizables para este problema particular, los cuales generan una solución buena, la cual no garantiza ser la óptima, pero alcanzable a un coste mucho menor.

Estos algoritmos son muy interesantes de cara a buscar soluciones para TSP o VRP de tamaños mayor, más cercanos a los que nos podemos encontrar en el día a día.

A pesar de la gran cantidad de heurísticas existentes, expondremos algunas de ellas a modo de ejemplo:

4.2.1 Greedy Algorithm*[39]*

Este nombre engloba a un conjunto de algoritmos con unas características específicas, el cual si que es reutilizable para algún otro problema distinto al TSP, aunque es en éste donde encuentra una solución más cercana a la óptima. La aplicación del algoritmo parte de la ordenación de los distintos nodos o ciudades a visitar, por pares de nodos a menor distancia. Siguiendo esta dinámica, el *greedy algorithm* siempre elige la mejor solución en cada iteración. Continuaríamos uniendo las ciudades, siempre buscando la menor distancia entre ellas, hasta que lleguemos a un ciclo cerrado en el que se incluyan todas las ciudades del problema.

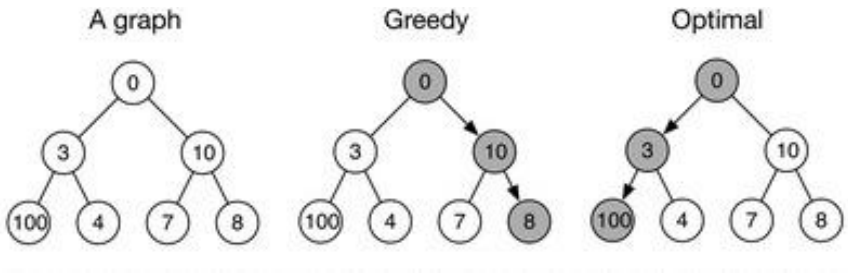


Figura 4-2. Greedy Algorithm[40]

4.2.2 Nearest Neighbor*[41]*

El *Nearest Neighbor Algorithm* (NN) es un caso particular de los algoritmos greedy. En el partimos de una ciudad o nodo específico del problema y, desde el mismo, vamos eligiendo como próximo nodo a visitar el que se encuentra a menor distancia del nodo precursor, de forma iterativa.

Aún dando una buena solución del problema, el principal inconveniente de esta heurística se localiza al final del algoritmo, cuando quedan pocas ciudades a visitar y, teniendo en cuenta la limitación de que cada ciudad sólo puede ser visitada una vez, nos vemos obligados a recorrer distancias muy grandes en estas uniones finales.

Definiendo la complejidad temporal como la cantidad de tiempo necesaria para ejecutar un algoritmo, estimada a través del número de operaciones elementales que emplea el algoritmo en cuestión, suponiendo un valor constante del tiempo empleado en las operaciones a realizar. En este sentido, el número de cálculos necesarios no crecerá más rápido que n^2 (complejidad temporal $O(n^2)$).

4.2.3 Nearest Insertion

Volvemos a una técnica similar a *NN*. En este punto, partimos de un ciclo o tour, formado por un número escaso de nodos. El algoritmo trata de localizar el nodo cuya inserción suponga el menor coste possible. En otras palabras, trata de localizar la ciudad más cercana a cualquiera de las ciudades que forman el tour, para así insertarla en el mismo. De igual modo que en el caso anterior, dichas inserciones se irán repitiendo de forma iterativa aumentando el tamaño del ciclo inicial, hasta que finalmente obtendríamos un ciclo que incluyese a todas las ciudades.

Al igual que ocurriría en el algoritmo NN, el algoritmo *NI* consta de una complejidad temporal de $O(n^2)$.

4.2.4 Christofides Algorithm*[42]*

El algoritmo de Christofides garantiza una aproximación de, como máximo, 3/2 veces mayor de la ruta óptima. Desarrollado en 1976 por el professor Nicos Christofides, este algoritmo, a pesar de la demostración de cuanto es capaz de acercarse a la solución óptima, suele utilizarse como un método de búsqueda de una solución inicial a la cual practicarle otras heurísticas de mejora, u otros métodos exactos como los expuestos en el 4.1.

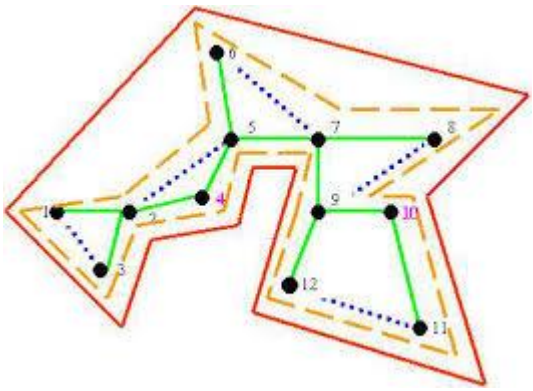


Figura 4-3. Christofides Algorithm.[42]

Dicho algoritmo se basa en la satisfacción, por parte de la función de costes de los distintos arcos entre nodos, de la desigualdad triangular, teorema de la geometría euclidiana que expone que, para cualquier triángulo, la suma de dos cualesquiera de los lados es siempre mayor que el lado restante. Para poder efectuar el algoritmo, la matriz de distancia entre ciudades debe ser simétrica.

En este caso, la complejidad temporal es de $O(n^4)$.

4.2.5 K-Optimal Algorithm

Un problema es denominado k-óptimo si no es posible mejorarlo intercambiando entre sí unos k arcos. Este tipo de algoritmos conllevan una complejidad temporal de $O(n^k)$.

En función del número de arcos a intercambiar, surgen una serie de algoritmos con características específicas, como podría ser el *2-Opt Algorithm*, el cual lo realiza intercambiando dos arcos cualesquiera, si este intercambio produce una mejora de la solución, partiendo de la afirmación

de que, si dos de los arcos se cruzan entre sí, esa solución jamás será la óptima del problema.

En este punto, cabe resaltar el ***Lin-Kernighan Heuristic,*** el cual es un algoritmo que trata de mejorar un k-tour inicial. Para ello, hace uso de secuencias de los algoritmos 2-Opt, 3-Opt, etc, con el fin de escapar de mínimos locales. Además, con el objetivo de llegar a la solución más cercana a la óptima del problema, permite que ciertos tours intermedios tengan un coste mayor que el coste inicial, algo no permitido en el algoritmo 2-Opt, por ejemplo.

4.3 Metaheurísticas*[43]*

Como última clase de algoritmos de resolución de problemas, aparecen las metaheurísticas, de gran utilidad en la práctica. Las metaheurísticas son aquellos algoritmos generales adaptables a una gran variedad de problemas. Las claras ventajas que le otorgan la maleabilidad que les caracteriza genera a su vez un detriment en la optimalidad de la solución, por ello se considera que lo que consiguen en una aproximación. Su ejecución se realiza de forma iterativa y es habitual que vayan acompañados de otros algoritmos de optimización.

Las metaheurísticas constan de unas características que las colocan como grandes candidatas para la resolución de problemas de optimización en la práctica. Su fácil implementación en cuanto a codificación se refiere y su bajo coste computacional hacen de estos algoritmos uno de los grupos más estudiados y de más interés en la actualidad.

Como ya se ha indicado, estas ventajas contrastan con su escasa garantía de optimalidad, tratándose de algoritmos probabilísticos, cuyo sustento teórico es relativamente pobre.

En función de su principal característica o de la inspiración del algoritmo, nos encontramos distintas familias de metaheurísticas. A continuación, procederemos a exponer dichas metaheurísticas, a excepción de dos de las más estudiadas y conocidas, como serían *Genetic Algorithm* y *Ant Colony Optimization*. Explicaremos estos dos algoritmos en un capítulo aparte, pues serán los utilizados para conseguir la ruta óptima del camión de nuestro problema, antes de someterlo al algoritmo de asignación de drones. Serán explicados al detalle en el próximo capítulo.

4.3.1 Metaheurísticas de Búsqueda Local*[44]*

Este tipo de metaheurísticas se basan en partir de una solución incial, donde, en su proceso iterativo, realizan una serie de movimientos de mejora a sus soluciones vecinas, otorgándoles las metaheurísticas un amplio abanico de actuación.

Dentro de esta familia, la metaheurística más conocida es la denominada ***Búsqueda Tabú***. La clave diferencial de la Búsqueda Tabú con respecto a las demás metaheurísticas de su familia es la facultad de memoria que se le otorga, la cual le permite escaper de óptimos locales. Una vez que el algoritmo detecta una potencial solución del problema, ésta se almacena es un espacio de memoria "tabú", de modo que el algoritmo no podrá volver a dicha solución. Es una forma de proporcionarle una especie de "inteligencia" al algoritmo, de modo que no itere una y otra vez sobre el mismo óptimo local.

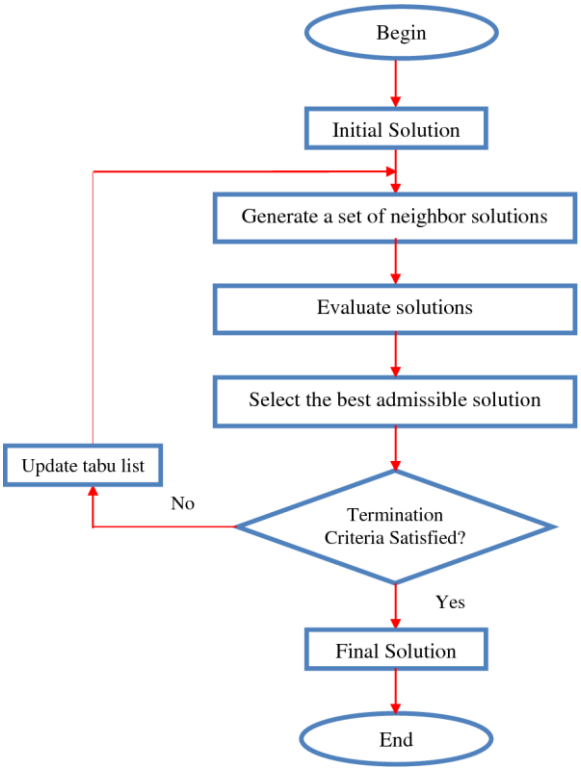


Figura 4-4. Diagrama de Flujos de la Búsqueda Tabú

La Búsqueda Tabú es ampliamente utilizada para la resolución de los problemas TSP.

Dentro de esta familia de metaheurísticas aparece otra de gran aplicación como es el ***Recocido Simulado***. Método descrito de forma independiente por Scott Kirkpatrick, C. Daniel Gelatt y Mario P. Vecchi en 1983 y por Vlado Černý en 1985, recibe su nombre debido a las similitudes con dicho proceso.

El proceso de recocido se basa en el calentamiento a altas temperaturas del material en cuestión, para luego enfriarlo de manera controlada a través de una tasa α, lo cual le otorga una serie de propiedades.

En este algoritmo partimos de una solución inicial aleatoria. A continuación, a través de cada iteración, se propone una nueva solución también aleatoria. Llegados a este punto tendríamos dos posibles escenarios: que la nueva solución sea mejor que la anterior, la cual se aceptaría; que la nueva solución sea peor, en cuyo caso se acepta como nueva solución del problema en función de una probabilidad que va disminuyendo a medida que avanzamos en las iteraciones.

La gran ventaja que otorga este algoritmo es su facilidad para escapar de óptimos locales.

4.3.2 Metaheurísticas Bioinspiradas*[45], [46]*

Particle Swarm Optimization (*PSO*), la metaheurística más conocida dentro de esta familia,

consiste en modelar cada robot del sistema de enjambre robótico como una partícula. Se trata de una técnica de computación evolutiva paralela, desarrollada en 1995. Esto significa que el conocimiento de cada paso de iteración se transfiere y se utiliza en el siguiente paso de cálculo.

Cada partícula inicia la búsqueda del espacio de búsqueda asignado en una posición aleatoria o, alternativamente, en una posición especificada con una velocidad aleatoria. A continuación, se desplaza por el área de búsqueda cambiando iterativamente su posición y velocidad. En cada paso, las partículas miden el valor de la función objetivo en su posición y lo comparan con su mejor experiencia personal, así como con la mejor experiencia global de las mediciones que se han realizado. Al ajustar su velocidad para el siguiente paso, se tienen en cuenta estas mediciones y los incrementos de conocimiento.

Como este procedimiento se repite para cada paso de iteración y para cada partícula, resulta un comportamiento de enjambre inteligente. Los parámetros de ajuste influyen en el impacto de las velocidades parciales. La siguiente fórmula explica matemáticamente el comportamiento de las partículas.

$$v_{i+1,j+1} = wv_{ij} + c_1 q \left[\frac{x_{ij}^{pb} - x_{ij}}{\Delta t} \right] + c_2 r \left[\frac{x_j^{sb} - x_{ij}}{\Delta t} \right]$$

Figura 4-5. Fórmula matemática PSO

La siguiente ilustración muestra el movimiento de una sola partícula de forma gráfica. Las contribuciones de cada parte de la fórmula se representan en verde. El primer término (I) es la velocidad anterior de la partícula multiplicada por un factor. El segundo término (II) considera la mejor velocidad personal de la partícula y la aporta a la velocidad total para la siguiente iteración. El tercer término (III), por último, tiene en cuenta la mejor posición global y aporta este conocimiento a la velocidad total de la partícula.

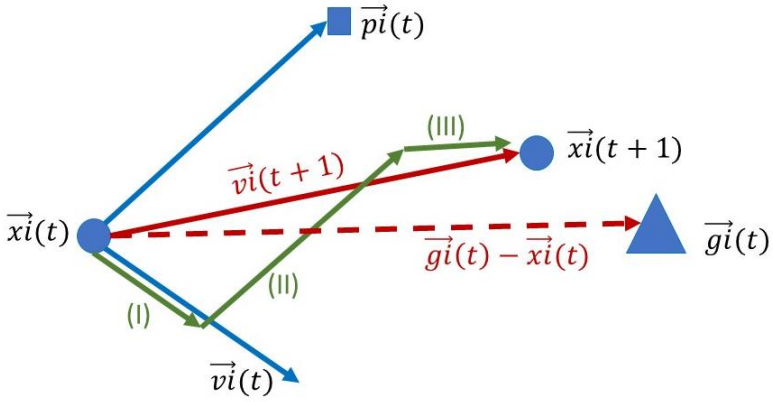


Figura 4-6. Descripción Gráfica PSO

Para terminar, a modo de resumen, adjuntamos un pequeño esquema de los algoritmos de resolución vistos a lo largo de este capítulo:

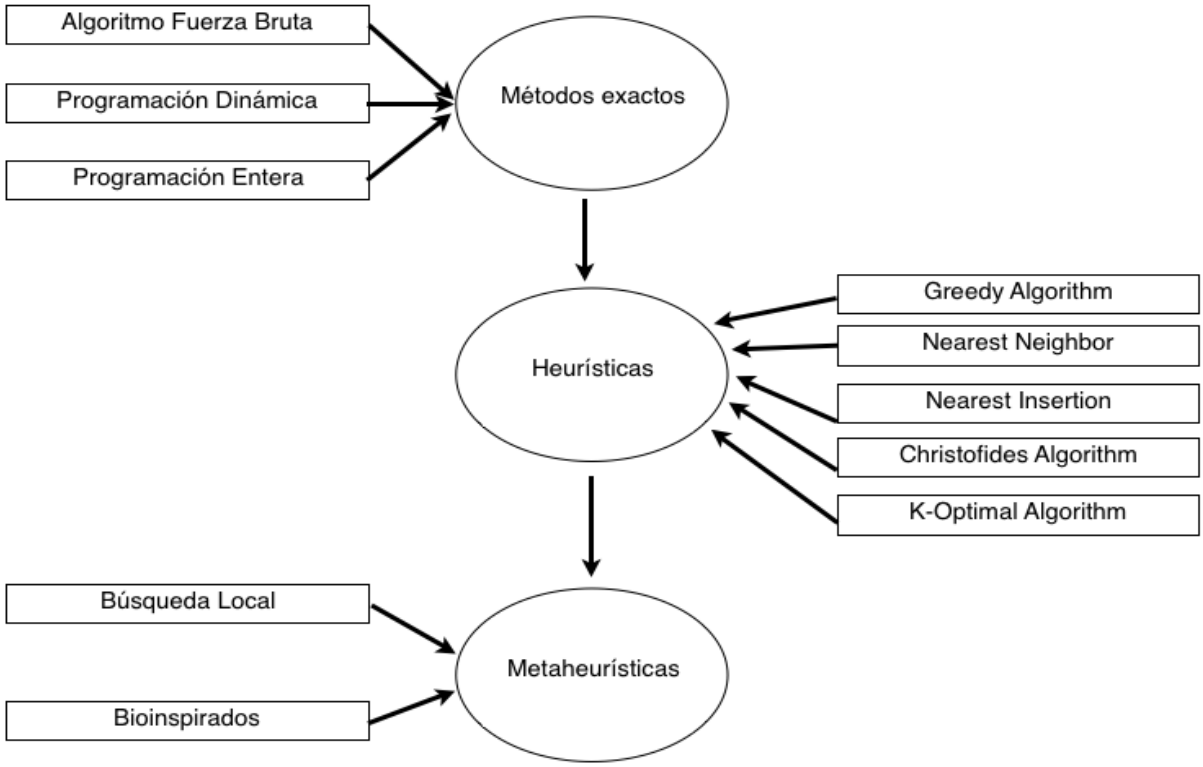


Figura 4-7. Esquema final de algoritmos para el TSP

5 NUESTROS ALGORITMOS

Una vez expuestos los diferentes algoritmos de resolución típicamente utilizados para resolver un problema tipo TSP, llega el turno de explicar los que serán utilizados para la resolución de nuestro caso en cuestión. Para ello, hemos hecho uso de dos de las metaheurísticas más utilizadas y estudiadas, *Genetic Algorithm* y *Ant Colony Optimization*.

5.1 Genetic Algorithm*[47]–[50]*

El Algoritmo Genético (GA) es una metaheurística introducida en la decada de 1960 por John Henry Holland, para problemas de optimización con y sin restricciones inspirado en la teoría de Charles Darwin, la selección natural, el proceso que impulsa la evolución genética. Esto significa que aquellas especies que pueden adaptarse a los cambios en su entorno son capaces de sobrevivir, reproducirse y pasar a la siguiente generación.

Como veremos, el proceso del GA tiene un marcado carácter aleatorio, por lo que esta metaheurística no te asegura encontrar el óptimo del problema.

Antes de comenzar a explicar paso a paso en qué consiste la metaheurística, resulta importante definer una serie de conceptos utilizados a lo largo del mismo:

- *Individuo*: Son las posibles soluciones del Sistema.
- *Población*: Conjunto de soluciones (individuos)
- *Función fitness*: Esta función tiene como objetivo evaluar a los distintos individuos, en relación con lo que se busca en el problema, para asignarles un valor en función de lo buena que sea la solución que generan.
- *Función de cruce*: Partiendo de dos individuos, se generan otros dos que surgen de la mezcla de los dos iniciales. Los primeros harían el papel de padres, mientras que los segundos serían los hijos. El cruce se realiza de modo que los "hijos" generen una mejor solución que los "padres". Se crea como una forma de mitigar la tendencia de este algoritmo de atascarse en máximos locales. La forma en la que se realiza dicho cruce puede ser muy variada.

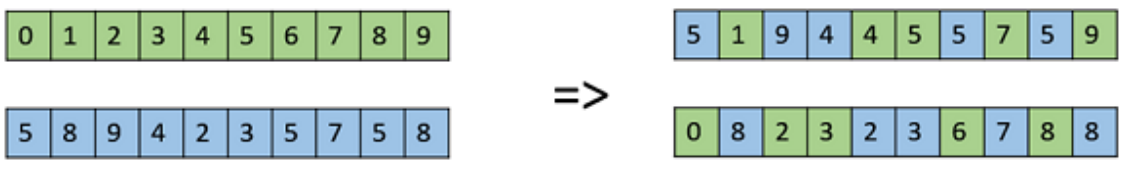


Figura 5-1. Ejemplo de un posible cruce[51]

Iniciaríamos el algoritmo con una solución inicial, una población, la cual suele encontrarse de forma aleatoria, si bien podemos tratar de localizar una población de partida "buena" para agilizar el proceso y conseguir una mejor solución final de nuestro problema.

El siguiente paso consistirá en evaluar la población a través de la función fitness, seleccionando los mejores de ellos.

Una vez tengamos los mejores individuos de la población inicial, iniciamos el proceso de reproducción hacienda uso de la función de cruce previamente explicada, formando una nueva población que sustituirá a la anterior.

Finalmente, asemejándose al proceso de selección natural, introducimos pequeñas modificaciones o mutaciones en los individuos de la nueva población. Dichas mutaciones se suelen generar de manera completamente aleatoria.

Llegados a este punto, solo quedaría repetir el proceso de forma iterativa todas las veces que hayamos decidido previamente.

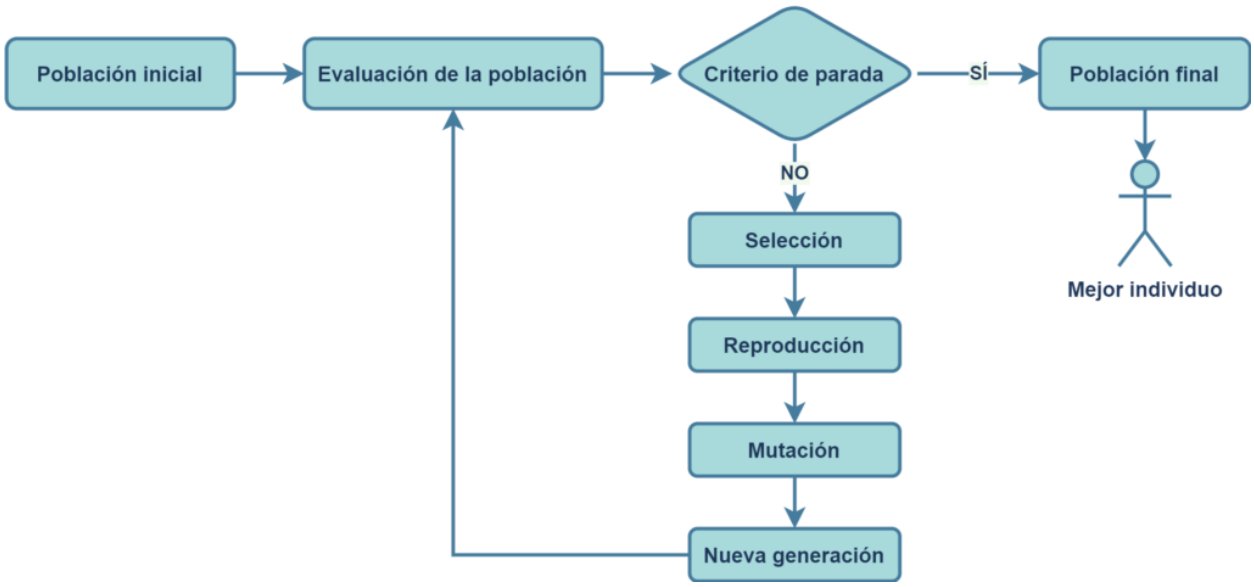


Figura 5-2. Diagrama del GA.[49]

El Algoritmo Genético posee grandes ventajas a la hora de implementarlos para ciertos tipos de problemas. Uno de ellos es su versatilidad y facilidad en la implementación. Como hemos podido comprobar, la estructura principal del algoritmo es idéntica sea cual sea el problema a tratar, exceptuando la función fitness.

Otra de las grandes ventajas del GA es su velocidad a la hora de localizar una solución aceptable del problema.

Por último, hay que destacar la capacidad del algoritmo para encontrar varias soluciones de forma simultánea, una característica idónea para los problemas de búsqueda y optimización, pues el rango de búsqueda se amplia a diferentes direcciones.

En cuanto a su implementación en Python, hemos hecho uso de la librería *DEAP*[52], ampliamente conocida, la cual incluye todas las diferentes funcionalidades necesarías para desarrollar el algoritmo genético. En particular, el código lo hemos extraído de un ejemplo concreto de resolución de un problema clásico de TSP a través de GA[53].

5.2 Ant Colony Optimization*[54]–[57]*

Propuesta por el científico italiano Marco Dorigo, en 1992, el algoritmo de Colonias de Hormigas (*ACO*, por sus siglas en ingés *Ant Colony Optimization*) es una metaheurística basada en el comportamiento de las hormigas y otros insectos. Este comportamiento consiste básicamente en la interacción y coordinación de diferentes organismos modificando el entorno.

Para entender el funcionamiento del algoritmo ACO, vamos a centrarnos primero en el concepto de estigmergia, definido como la colaboración a través del medio físico, aplicado al comportamiento de las hormigas. Las hormigas, al igual que otros insectos, no tienen la capacidad de ver, por lo que necesitan utilizar otras formas de comunicación, como el sonido, el tacto o la producción de sustancias químicas. De hecho, esta última es la base del ACO.

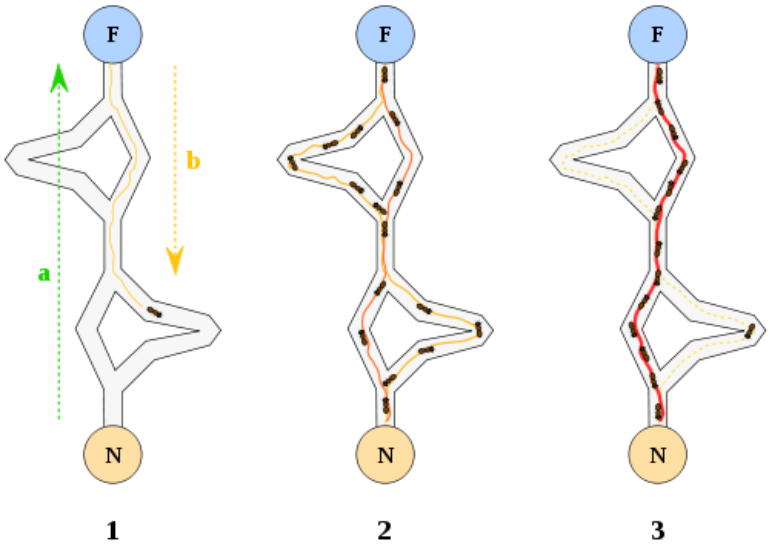


Figura 5-3. Comportamiento de las hormigas[58]

Cuando una colonia de hormigas comienza su viaje para conseguir comida, sus primeros miembros tienen diferentes caminos que pueden seguir, con la misma probabilidad entre ellos de ser elegidos. Tras la elección y el comienzo del viaje, empiezan a producir sustancias químicas, llamadas *feromonas*, que dejan a lo largo del camino como señal para que las demás hormigas sigan.

Las hormigas se sienten más atraídas a elegir un camino con mayor cantidad de feromonas, pero no por ello siempre será éste el elegido, por lo que toman la decisión basándose en probabilidades. Cuando la hormiga que eligió el camino más corto llegue a la comida, regresará al nido siguiendo el mismo camino y dejando en él más cantidad de feromonas, aumentando la probabilidad de ser elegido

por las siguientes hormigas.

Teniendo en cuenta la vaporización de las feromonas, después de algunas hormigas, el nivel de feromonas en el camino más corto será enorme, en comparación con los otros caminos, encontrando el camino óptimo para conseguir la comida. Vemos como, el hecho de que para el hormiguero consiga el camino óptimo hacia la fuente de comida depende de la repetición del proceso por parte de un número determinado de hormigas, volvemos a plantear un algoritmo iterativo.

El concepto del ACO queda íntimamente ligado con la resolución de los problemas TSP. De hecho, el primer algoritmo de colonia de hormigas, denominado *Ant System*, tenía como objetivo encontrar el óptimo del problema del viajante.

El algoritmo trata de hacer un calco del comportamiento previamente descrito. Para ello, y pensando directamente en la resolución del TSP, vemos las siguientes consideraciones que deberían tomar las "hormigas" a la hora de decidir qué ciudad visitar:

- Si la ciudad ha sido ya visitada o no.
- Se introduce el concepto de *visibilidad* como la inversa de las distancias entre ciudades. Cuanto más cerca esté la ciudad *i* con respecto a la *j*, mayor será la visibilidad y por tanto más probable será su elección.
- Programación del nivel de feromonas en los distintos caminos elegibles, así como la tasa de vaporización.

Como en el caso del Algoritmo Genético, hay multitud de ejemplos de resolución del TSP utilizando Ant Colony Optimization. Hemos utilizado uno de ellos a la hora de inicializar nuestro problema[59].

Llegados a este punto, y antes de comenzar a analizar los resultados de nuestro problema, conviene aclarar que, aunque nuestro código para la asignación de drones sea determinista, la solución inicial de la ruta óptima del camión a través de la resolución del TSP, tiene un claro componente aleatorio, originado por la utilización de metaheurísticas, en concreto GA y ACO. Ahí radica la importancia de una buena aproximación inicial a la hora de crear un modelo verdaderamente efectivo.

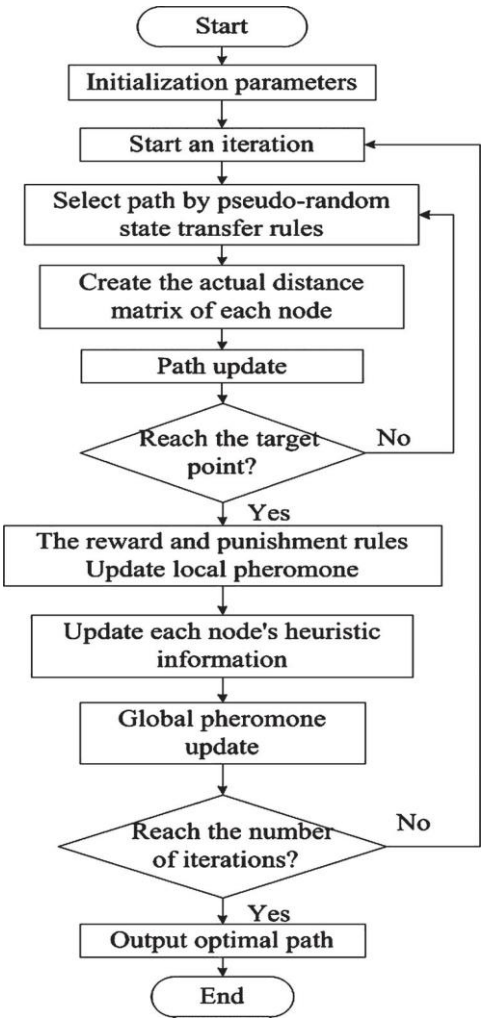


Figura 5-4. Posible diagrama de flujos de ACO[60]

6 RESULTADOS Y CONCLUSIÓN

6.1 Método de resolución

Antes de abordar los resultados obtenidos tras la experimentación a través tanto del GA como del ACO, aunque se definieron en el capítulo de *Nuestro Problema*, explicaremos con mayor detalle las funciones del código, con el fin de entender en profundidad como se consigue los resultados mencionados.

El código consta de una serie de conceptos que se erigen como fundamentales para su desarrollo. Uno de ellos, las *subrutas*, constan a su vez de dos clases, las subrutas asociadas a un dron (aquella que comienzan en el cliente donde despega el dron y termina en el cliente en el cual aterriza) y las subrutas en las que no constan ningún dron en vuelo a lo largo de las mismas.

Tal y como se expresó en el capítulo de *Nuestro Problema*, el código desarrollado para varios drones es una adaptación del pseudocódigo propuesto por Murray para un único dron, alejándonos de su propuesta para múltiples drones. Realizar esta adaptación entra en claro conflicto con las subrutas, pues cada dron dispondrá de diferentes de ellas, generando un resultado final fruto de la superposición de las subrutas de los distintos drones.

Para solucionar esta situación, generamos un diccionario de Python con tantas "keys" como número de drones *n* estudiemos. En este sentido, en función del dron elegido para el movimiento, se modificarán las subrutas únicamente asociadas a dicho dron, generando una solución fidedigna del problema.

- ***calcSavings*:**

Como ya se comentó, esta función tiene como objetivo el cálculo del ahorro que se produce en la ruta del camión cuando extraemos un cliente *j*. Para conseguirlo, simplemente se calcula tanto el tiempo empleado por el camión en recorrer todos los nodos, incluyendo a *j* como el mismo tiempo excluyendo a *j*, siendo la diferencia de ambos tiempos el ahorro producido en la ruta del camión.

Es importante tener en cuenta que este ahorro puede ser negativo, en el caso de que el coste de ir del cliente anterior a *j* al posterior de forma directa sea muy elevado. En este caso, no se ejecutará el código principal para dicho cliente, pues sabremos con certeza que no será rentable el cambio.

- ***calCostTruck:***

En esta función se obtiene el coste de insertar al cliente *j*, atendido por el camión, en una posición distinta de la ruta del mismo. Una vez que los drones comiencen a atender clientes, la ruta del camión tras la eliminación de dichos clientes puede no ser la óptima, pues la calculada mediante las metaheurísticas era para todo el conjunto de clientes. Por ello, se estudia el posible coste de insertar al cliente en diferentes posiciones de la ruta del camión.

La función tiene en cuenta que, para que dicho cambio pueda ser factible, los costes del

movimiento deben ser menores que los ahorros generados en *calcSavings*, así como que la autonomía del dron en vuelo sea suficiente para hacer frente a dicho cambio sin que se agote su batería antes de llegar a su destino.

- ***calCostUAV:***

De manera similar a lo explicado en la función anterior, se trata de obtener el coste asociado a que un cliente *j* sea atendido por un dron *k*. Una vez se obtengan los puntos de despegue y aterrizaje del dron, debe comprobarse que éste tiene la autonomía suficiente para realizar dicho trayecto.

Resaltar la inclusión en el coste el tiempo empleado en la recogida del dron por parte del transportista, así como el tiempo que tarda en prepararlo para su despegue.

- ***performUpdate:***

Una vez que se ha encontrado el movimiento o cambio que genera mayores ahorros al camión, es hora de actualizar las distintas subrutas, así como la ruta del camión, eliminando al cliente *j* en caso de que este sea atendido por un dron o cambiándolo de posición si así se requiere.

Con este objetivo surge la variable booleana *servedbyUAV*, la cual tendrá el valor "True" cuando el cliente sea visitado por el dron y "False" en caso contrario.

- ***Código principal:***

Para cada dron y para cada subruta del dron, comprobamos los distintos movimientos posibles asociados al cliente *j* de estudio. Comparamos todos los ahorros generados, eligiendo el de mayor valor, aplicándole a dicho dron y movimiento la función *performUpdate*.

6.1.1 Características destacables

Como último paso antes de abordar la base de datos utilizada y los resultados obtenidos, se expondrán algunas licencias y simplificaciones adquiridas.

En primer lugar, realizaremos el estudio asumiendo que no todos los clientes son factibles de ser visitados por drones, debido a que dichos clientes se encuentren en zonas de imposible acceso para los drones, que los paquetes a entregar sean demasiado pesados, etc. La elección personal de los clientes potenciales a ser visitados por drones es completamente aleatoria, quedando fuera del alcance de este TFG las consideraciones expuestas previamente.

Por otro lado, aunque el código generado sea completamente determinista, la solución inicial de partida consta de un componente pseudoaleatorio, pues se obtiene a través de metaheurísticas. Por ello, a fin de una correcta experimentación, realizamos veinte iteraciones para cada metaheurísticas.

En cuanto a las simplificaciones, hay dos factores a comentar. Primero, hemos obviado el depósito o almacén, el cual debería ser un punto fijo al inicio y final de toda la ruta. En segundo lugar, una vez que se comprueba que el dron tiene autonomía suficiente para realizar un determinado movimiento, aceptamos que este permanece factible a lo largo del tiempo, a pesar de que otras operaciones puedan aumentar el tiempo de vuelo requerido para el dron. La exclusión de dichas simplifaciones quedan como futuras líneas de estudio de este proyecto.

6.2 Resultados

Los clientes a estudiar se obtienen de la web GitHub[59], los cuales quedan expresados en la siguiente tabla:

Clientes	X	Y
1	10	0
2	38	94
3	83	27
4	29	99
5	48	71
6	95	61
7	54	18
8	73	21
9	47	76
10	99	47
11	87	94
12	44	76
13	83	61
14	27	29
15	29	11
16	40	52
17	56	24
18	80	13
19	63	82
20	79	69
21	71	61
22	92	30
23	54	47
24	36	12
25	2	9
26	85	66
27	99	9
28	18	58
29	92	80
30	16	80
31	36	30
32	63	85
33	56	15
34	34	69
35	39	97
36	38	34
37	43	86
38	6	92
39	82	21
40	97	23
41	24	49
42	53	76
43	91	9
44	59	1
45	95	42
46	63	35
47	53	35
48	45	76
49	82	76

Tabla 6-1. Clientes

Contamos con un total de 49 clientes, de los cuales únicamente 20 serán visitables por drones: 0, 5, 7, 10, 11, 12, 14, 15, 16, 23, 25, 27, 29, 30, 33, 36, 40, 44, 45 y 47. Para la aplicación del algoritmo, requerimos de convertir los datos de los clientes de modo que se indique la distancia relativa entre cada par de ellos. Con el fin de que concuerde con el dato utilizado que indica el tiempo requerido en la preparación del dron por parte del transportista, asumimos como igual la distancia como las unidades de tiempo empleadas en llegar del cliente *i* al *j*. Dicha conversión la realizamos a través del código *basedatos*, que se encuentra también en el anexo.

6.2.1 Resultados del GA

Tomando diferentes semillas de números pseudoaleatorios, los resultados obtenidos tomando con rutas iniciales del camión trayectos generados por GA son los que se muestran en la tabla 6-2, en la cual se indica lo siguiente, por columnas de izquierda a derecha: iteraciones del algoritmo; ruta del camión tras la aplicación del GA, sin haber aún aplicado el algoritmo de inclusión de los drones; coste inicial para dicha ruta inicial del camión (en color verde aparece el mejor resultado, así como en rojo el peor); número de drones sometidos a estudios, con una tamaño máximo de flota de tres drones; los clientes visitados por alguno de los drones; porcentaje de clientes visitados por algún dron con respecto a los clientes potencialmente visitables, 20 en nuestro caso (de igual modo aparece en verde el mayor porcentaje y en rojo el menor); coste final tras la inclusión de los drones, con la misma distribución de colores mencionada en columnas anteriores; ahorro generado por la inclusión de los drones, calculado con la resta entre el coste de la ruta inicial y el coste tras la aparición de los drones, destacando en verde el máximo ahorro producido; número de drones utilizados con respecto a los disponibles paras los distintos tamaños de flota; y, por último, los clientes a los que se les ha cambiado de posición dentro de la ruta del camión.

Iteraciones	Ruta camión	Coste inicial	Drones	Clientes visitados por dron	%drones	Coste final	Ahorro	Drones utilizados	Cambio pos
1	[18, 45, 38, 2, 5, 12, 20, 33, 1, 3, 34, 36, 11, 4, 22,15, 46, 24, 0, 43, 16, 17, 26, 42, 25, 19, 48, 10, 28, 9,44,21 39, 7, 6, 32, 23, 14, 13, 30, 35, 40, 27, 37, 29, 8, 47, 41, 31]	913	1	45, 5, 12, 33, 15, 0	30%	839	74	Todos	
			2	45, 5, 12, 33, 15, 0,16, 25, 10, 14, 29	55%	779	134	Todos	
			3	45, 5, 12, 33, 15, 0,16, 25, 10, 14, 29,47	60%	775	138	Todos	
2	[13, 30, 46, 45, 12, 25, 48, 19, 5, 44, 2, 22, 4, 8, 11, 33, 15, 17, 26, 42, 7, 6, 32, 23, 0, 14, 24, 29, 37, 3, 36, 41, 20, 31, 18, 47, 27, 35, 16, 43, 38, 39, 21, 9, 28, 10, 34, 1, 40]	1014	1	45, 5, 33, 15, 0, 16, 10	35%	936	78	Todos	
			2	45, 5, 33, 15, 0, 16, 10, 44, 7, 40, 27, 29, 47	65%	845	169	Todos	
			3	45, 5, 33, 15, 0, 16, 10, 44, 7, 40, 27, 29, 47	65%	845	169	Sólo 1 y 2	
3	[11, 36, 29, 33, 15, 20, 19, 48, 18, 34, 1, 31, 40, 24, 0, 32, 28, 10, 5, 26, 42, 6, 23, 14, 13, 30, 35, 46, 22, 12, 9, 44, 39, 45, 3, 37, 27, 17, 16, 7, 43, 17, 38, 2, 21, 25, 41, 4, 8, 47]	1140	1	45, 12, 36, 11, 15, 0, 25	35%	1059	81	Todos	
			2	45, 12, 36, 11, 15, 0, 25,10, 7, 14, 27, 29	60%	981	159	Todos	
			3	45, 12, 36, 11, 15, 0, 25,10, 7, 14, 27, 29	60%	981	159	Sólo 1 y 2	
4	[46, 6, 43, 32, 24, 0, 22, 31, 41, 4, 8, 11, 2, 38, 17, 7, 42, 26, 39, 21, 12, 25, 19, 20, 45, 16, 23, 14, 3, 1, 37, 29, 27, 35, 13, 40, 33, 47, 36, 34, 10, 28, 48, 5, 44, 9, 18, 15, 30]	1029	1	5, 11, 0, 25, 10, 44	30%	991	38	Todos	
			2	5, 11, 0, 25, 10, 44, 7, 14, 47	45%	937	92	Todos	30
			3	5, 11, 0, 25, 10, 44, 7, 14, 47	45%	937	92	Sólo 1 y 2	30
5	[12, 48, 31, 41, 33, 22, 46, 9, 5, 25, 19, 10, 28, 2, 38, 7, 24, 0, 15, 18, 34, 1, 36, 4, 8, 20, 45, 23, 14, 40, 35, 30, 13, 27, 3, 37, 29, 47, 11, 16, 6, 32, 43, 17, 21, 39, 26, 42, 44]	1042	1	12, 33, 0, 10, 44, 14, 27	35%	888	154	Todos	40
			2	12, 33, 0, 10, 44, 14, 27, 29	40%	885	157	Todos	40
			3	12, 33, 0, 10, 44, 14, 27, 29	40%	885	157	Sólo 1 y 2	40
6	[8, 29, 37, 3, 1, 34, 47, 4, 11, 36, 31, 18, 20, 46, 6, 17, 42, 9, 5, 19, 48, 10, 28, 12, 45, 22, 15, 33, 27, 40, 30, 13, 24, 0, 14, 32, 43, 23, 35, 16, 7, 39, 38, 2, 26, 21, 44, 25, 41]	892	1	45, 5, 12, 33, 36, 11, 25	35%	835	57	Todos	
			2	45, 5, 12, 33, 36, 11, 25, 0, 10, 23, 30, 27	60%	762	130	Todos	44
			3	45, 5, 12, 33, 36, 11, 25, 0, 10, 23, 30, 27,29	65%	759	133	Todos	44
7	[22, 15, 40, 46, 16, 23, 14, 0, 24, 13, 30, 35, 32, 6, 7, 38, 39, 26, 42, 45, 27, 29, 37, 3, 33, 4, 19, 28, 25, 44, 9, 2, 21, 17, 43, 10, 18, 8, 41, 1, 34, 36, 11, 47, 31, 48, 5, 12, 20]	930	1	5, 12, 33, 0, 16	25%	883	47	Todos	
			2	5, 12, 33, 0, 16,25, 10, 44, 40, 27	50%	791	139	Todos	
			3	5, 12, 33, 0, 16,25, 10, 44, 40, 27	50%	791	139	Sólo 1 y 2	
8	[23, 35, 15, 33, 4, 41, 12, 19, 31, 18, 20, 9, 44, 21, 39, 7, 32, 6, 0, 24, 14, 30, 13, 40, 27, 37, 3, 29, 8, 11, 22, 45, 2, 5, 25, 48, 28, 10, 34, 1, 36, 47, 42, 26, 17, 38, 16, 46, 43]	947	1	5, 12, 33, 11, 0	25%	878	69	Todos	
			2	5, 12, 33, 11, 0,16, 10, 23, 14, 30, 29	55%	818	129	Todos	
			3	5, 12, 33, 11, 0,16, 10, 23, 14, 30, 29	55%	818	129	Sólo 1 y 2	
9	[15, 16, 32, 43, 35, 29, 37, 3, 34, 1, 36, 19, 48, 10, 18, 31, 47, 11, 41, 21, 9, 28, 12, 20, 45, 40, 27, 25, 5, 44, 2, 38, 39, 42, 26, 17, 7, 30, 13, 14, 0, 24, 23, 6, 46, 22, 8, 4, 33]	978	1	45, 5, 12, 33, 15, 0, 25	35%	874	104	Todos	16
			2	45, 5, 12, 33, 15, 0, 25,10, 44, 14, 40, 27	60%	713	265	Todos	16
			3	45, 5, 12, 33, 15, 0, 25,10, 44, 14, 40,27	60%	713	265	Todos	16
10	[5, 25, 28, 10, 41, 4, 46, 16, 7, 38, 2, 21, 39, 26, 42, 0, 24, 35, 13, 23, 14, 32, 17, 9, 44, 12, 19, 31, 22, 30, 40, 27, 29, 11, 47, 37, 34, 36, 8, 48, 18, 3, 1, 33, 15, 6, 43, 45, 20]	1039	1	5, 0, 16, 25, 10, 44, 23	35%	965	74	Todos	30
			2	5, 0, 16, 25, 10, 44, 23,14, 30, 40, 27, 29	60%	884	155	Todos	47
			3	5, 0, 16, 25, 10, 44, 23,14, 30, 40, 27, 29	60%	884	155	Sólo 1 y 2	47
11	[35, 30, 16, 23, 14, 40, 1, 3, 37, 29, 27, 33, 22, 42, 26, 39, 9, 44, 21, 2, 6, 32, 43, 17, 38, 7, 45, 46, 13, 24, 0, 20, 48, 28, 10, 5, 12, 25, 41, 4, 47, 36, 11, 8, 19, 31, 18, 34, 15]	958	1	45, 5, 12, 33, 36, 15	30%	846	112	Todos	25
			2	45, 5, 12, 33, 36, 15,0, 16, 25, 10, 27	55%	771	187	Todos	
			3	45, 5, 12, 33, 36, 15,0, 16, 25, 10, 30,23, 14, 40, 27	75%	717	241	Todos	
12	[28, 25, 19, 31, 48, 22, 45, 38, 17, 42, 26, 46, 33, 47, 41, 34, 3, 1, 36, 37, 13, 30, 35, 9, 44, 20, 40, 27, 29, 18, 12, 2, 21, 39, 7, 43, 23, 0, 24, 14, 4, 8, 11, 15, 6, 32, 16, 5, 10]	1140	1	5, 12, 33, 0, 25, 10, 47	35%	1003	137	Todos	36
			2	5, 12, 33, 0, 25, 10, 47,44, 7, 14, 40, 29	60%	873	**267**	Todos	27,36
			3	5, 12, 33, 0, 25, 10, 47,44, 7, 14, 40, 29	60%	873	**267**	Sólo 1 y 2	27,36
13	[47, 4, 18, 48, 25, 5, 19, 41, 8, 37, 1, 34, 3, 29, 33, 22, 23, 30, 35, 27, 40, 15, 2, 20, 12, 28, 10, 45, 46, 32, 16, 7, 38, 6, 13, 14, 24, 0, 43, 42, 26, 39, 17, 21, 44, 9, 31, 36, 11]	1021	1	45, 5, 12, 36, 11, 15, 0	35%	952	69	Todos	
			2	45, 5, 12, 36, 11, 15, 0,16, 25, 10, 23, 14	60%	867	154	Todos	
			3	45, 5, 12, 36, 11, 15, 0,16, 25, 10, 23, 14,30, 27, 29, 47	80%	818	203	Todos	
14	[14, 24, 0, 43, 7, 2, 21, 9, 12, 20, 31, 18, 41, 25, 48, 5, 44, 39, 17, 38, 42, 26, 45, 46, 22, 29, 37, 3, 11, 33, 40, 35, 16, 32, 6, 13, 27, 34, 1, 36, 47, 10, 28, 19, 4, 8, 15, 30, 23]	953	1	5, 11, 0, 16, 25	25%	918	35	Todos	
			2	5, 11, 0, 16, 25,10, 23, 14, 30, 40, 27	55%	820	133	Todos	47
			3	5, 11, 0, 16, 25,10, 23, 14, 30, 40, 27,47	60%	817	136	Todos	
15	[13, 35, 2, 7, 16, 14, 23, 6, 46, 28, 10, 31, 20, 19, 12, 25, 48, 41, 34, 36, 11, 8, 33, 27, 40, 30, 32, 43, 17, 39, 26, 42, 38, 45, 21, 44, 9, 5, 18, 4, 15, 0, 24, 37, 29, 3, 1, 47, 22]	1004	1	45, 5, 12, 0, 25, 10, 27	35%	910	94	Todos	
			2	45, 5, 12, 0, 25, 10, 27,14, 29	45%	881	123	Todos	
			3	45, 5, 12, 0, 25, 10, 27,14, 29	45%	881	123	Sólo 1 y 2	
16	[28, 10, 31, 18, 41, 8, 4, 40, 27, 33, 3, 34, 1, 36, 47, 11, 29, 37, 46, 45, 9, 5, 19, 44, 21, 2, 38, 14, 23, 13, 24, 0, 35, 30, 16, 22, 15, 7, 43, 32, 6, 17, 42, 26, 39, 20, 12, 25, 48]	901	1	5, 33, 15, 0, 16, 10	30%	812	89	Todos	30
			2	5, 33, 15, 0, 16, 10,7, 23, 14, 30, 40, 27	60%	726	175	Todos	
			3	5, 33, 15, 0, 16, 10, 7, 23, 14, 30, 40, 27, 29	65%	723	178	Todos	
17	[27, 15, 22, 45, 32, 6, 14, 23, 16, 46, 47, 8, 4, 11, 2, 26, 39, 21, 38, 7, 41, 31, 18, 44, 9, 28, 10, 48, 19, 5, 12, 25, 20, 35, 13, 40, 24, 0, 30, 43, 42, 17, 33, 1, 34, 36, 3, 29, 37]	1015	1	45, 5, 12, 33, 36, 11, 0	35%	931	84	Todos	
			2	45, 5, 12, 33, 36, 11, 0, 16, 25, 10, 44, 7, 14, 40	70%	810	205	Todos	47, 30
			3	45, 5, 12, 33, 36, 11, 0, 16, 25, 10, 44, 7, 14, 40, 27, 29	80%	772	243	Todos	47, 30
18	[18, 1, 34, 37, 3, 36, 11, 47, 41, 15, 22, 45, 16, 6, 32, 7, 43, 39, 26, 42, 17, 14, 23, 30, 46, 8, 33, 35, 13, 40, 27, 29, 4, 28, 10, 20, 12, 25, 9, 44, 2, 0, 24, 38, 21, 5, 48, 19, 31]	1038	1	45, 5, 33, 15, 0, 25, 10	35%	970	68	Todos	
			2	45, 5, 33, 15, 0, 25, 10, 7, 23, 14, 30, 29	60%	861	177	Todos	
			3	45, 5, 33, 15, 0, 25, 10, 7, 23, 14, 30, 29	60%	861	177	Sólo 1 y 2	
19	[45, 2, 20, 41, 8, 4, 11, 47, 1, 36, 18, 31, 19, 25, 10, 28, 12, 5, 48, 34, 3, 37, 33, 29, 27, 24, 0, 23, 43, 14, 13, 7, 42, 26, 17, 40, 15, 9, 44, 21, 38, 39, 22, 30, 35, 6, 32, 16, 46]	1014	1	45, 5, 12, 33, 0, 25, 30	35%	885	129	Todos	
			2	45, 5, 12, 33, 0, 25, 30, 10, 14, 40	50%	827	187	Todos	
			3	45, 5, 12, 33, 0, 25, 30, 10, 14, 40	50%	827	187	Sólo 1 y 2	
20	[39, 7, 45, 16, 6, 25, 19, 20, 31, 18, 41, 47, 36, 1, 34, 8, 11, 4, 22, 21, 38, 17, 14, 24, 0, 2, 44, 9, 5, 30, 13, 29, 37, 3, 48, 28, 10, 12, 15, 35, 23, 42, 26, 43, 32, 40, 27, 33, 46]	1135	1	45, 5, 12, 33, 11, 15, 0, 16	40%	1050	85	Todos	23
			2	45, 5, 12, 33, 11, 15, 0, 16, 25, 10, 23, 27, 47	65%	985	150	Todos	
			3	45, 5, 12, 33, 11, 15, 0, 16, 25, 10, 23, 27, 47	65%	985	150	Sólo 1 y 2	

Tabla 6-2. Resultados GA

Como inicio de análisis de los resultados obtenidos, observamos que el mayor coste de las rutas iniciales del camión, extraídas directamente del GA, es de 1.140 unidades temporales, por las 892 u.t. del mínimo coste. Asimismo, observamos un coste promedio del TSP clásico de 1.005,15 u.t. con una desviación típica de 74,25 u.t., parámetros que serán relevantes para la comparación con los resultados del ACO.

Resulta importante resaltar que en una primera instancia se comenzó el estudio para una flota máxima de cuatro drones. Debido a la dimensión del problema tratado (49 nodos), escasas veces era utilizado el cuarto dron, no porporcionándonos información relevante para la experimentación, por lo que se decidió limitarlo a tres drones máximo.

```
Cliente a estudiar su visita con el dron 47
La nueva subruta asociada al vuelo del dron 1 es [13, 29, 37, 3] y el dron visita al cliente 47
```

Figura 6-1. Ejemplo de visita del cliente 47 por el dron 1 en Python

En el caso de que únicamente utilicemos un dron, resulta reseñable el porcentaje de clientes visitados por el dron con respecto a los potencialmente visitables, el cual es de media aproximadamente un 35%, teniendo incluso mínimos del 25%. Diferente situación se presenta para dos y tres drones, donde dicho porcentaje ronda de media el 60% con máximos de hasta 80%.

En la última columna de la tabla se muestran los clientes que han sido cambiados de posición dentro de la ruta del camión. Bajo este concepto podemos observar que, normalmente, se producen mayores ahorros visitando un cliente con el dron que cambiando dicho cliente de posición. Esto lo podemos ver por ejemplo en la iteración 14, donde para dos drones el cliente 47 no podía ser visitado por ninguno de ellos, por estar ambos en vuelo, por lo que se decide cambiarlo de posición. En cambio, al sumar el tercer dron, éste si puede visitarlo produciéndose un ahorro mayor.

El mayor ahorro, de 267 u.t., se produce en la iteración donde el coste inicial de la ruta del camión era mayor.

Para concluir, observamos que, de las 20 experimentaciones realizadas, en 12 de ellas el tercer dron no es requerido, constando, en promedio, un ahorro de únicamente 7,7 u.t. si se utilizan tres drones en lugar de dos, frente al ahorro de 88,15 u.t. si se utilizan tres drones en vez de uno o de 80,45 u.t. de dos drones en lugar de uno. Teniendo en cuenta los costes de adquisición de un dron, mantenimiento, baterías, licencias, etc, que quedan fuera del alcance de este TFG, a la vista de los resultados, para este dimensionamiento del problema lo más lógico sería la utilización de dos drones para el reparto a los clientes.

```
Nueva ruta del camión: [39, 7, 6, 19, 20, 31, 18, 41, 36, 1, 34, 8, 4, 22, 21, 38, 17, 14, 24, 2, 44, 9, 30, 13, 29, 37, 3, 48, 28, 35, 42, 26,
43, 32, 40, 46]
Nuevos tiempos de llegada a cada nodo del camión [0, 24, 43, 100, 111, 136, 139, 151, 165, 174, 177, 199, 204, 229, 271, 284, 292, 343, 370,
453, 472, 478, 543, 552, 604, 620, 644, 702, 713, 784, 843, 851, 892, 906, 953, 985]
Nodos de inicios de las subrutas en función del dron [[39, 19, 11, 17, 44, 37, 23, 32], [39, 20, 4, 24, 13]]
Clientes visitados por drones [[45, 5, 12, 33, 11, 15, 0, 16], [25, 10, 23, 27, 47]]
Las nuevas subrutas en función de los drones {0: [[39, 7, 6], [19, 20, 31, 18, 41, 36, 1, 34, 8], [4, 22, 21, 38], [17, 14, 24, 2], [44, 9, 30,
13, 29], [37, 3, 48, 28, 35], [42, 26, 43], [32, 40, 46]], 1: [[39, 7, 6, 19], [20, 31, 18, 41, 36, 1, 34, 8], [4, 22, 21, 38, 17, 14], [24, 2,
44, 9, 30], [13, 29, 37, 3], [48, 28, 35, 42, 26, 43, 32, 40, 46]]}
```

Figura 6-2. Ejemplo resultados en Python para dos drones

6.2.2 Resultado del ACO

Iteraciones	Ruta camión	Coste inicial	Drones	Clientes visitados por dron	%drones	Coste final	Ahorro	Drones utilizados	Cambio pos
1	[28, 10, 48, 19, 25, 12, 5, 9, 44, 21, 39, 26, 42, 17, 38, 2, 7, 45, 46, 16, 6, 32, 43, 23, 14, 0, 24, 13, 30, 35, 40, 27, 33, 4, 8, 47, 11, 41, 18, 31, 36, 1, 34, 3, 29, 37, 15, 22, 20]	608	1	45, 5, 0, 25, 10	25%	565	43	Todos	27
			2	45, 5, 0, 25, 10, 7, 27, 29	40%	538	70	Todos	
			3	45, 5, 0, 25, 10, 7, 27, 29	40%	538	70	Sólo 1 y 2	
2	[34, 1, 36, 11, 47, 8, 4, 41, 31, 18, 48, 28, 10, 20, 19, 25, 12, 5, 9, 44, 21, 39, 26, 42, 17, 38, 2, 7, 45, 46, 16, 6, 32, 43, 23, 14, 13, 30, 35, 15, 22, 0, 24, 40, 27, 33, 29, 37, 3]	644	1	45, 5, 33, 15, 0	25%	593	51	Todos	
			2	45, 5, 33, 15, 0, 25, 10, 7, 14, 40	50%	550	94	Todos	
			3	45, 5, 33, 15, 0, 25, 10, 7, 14, 40	50%	550	94	Sólo 1 y 2	
3	[10, 48, 19, 25, 12, 5, 9, 44, 21, 39, 26, 42, 17, 38, 2, 7, 45, 16, 6, 32, 43, 23, 14, 0, 24, 13, 30, 35, 46, 22, 15, 40, 27, 33, 11, 47, 8, 4, 41, 18, 31, 36, 1, 34, 3, 29, 37, 20, 28]	629	1	45, 5, 0, 16	20%	599	30	Todos	
			2	45, 5, 0, 16, 25, 10, 40, 27, 29	45%	534	95	Todos	
			3	45, 5, 0, 16, 25, 10, 40, 27, 29	45%	534	95	Todos	
4	[45, 46, 16, 6, 32, 43, 17, 38, 2, 21, 39, 26, 42, 7, 44, 9, 5, 25, 12, 19, 48, 28, 10, 31, 18, 41, 8, 47, 11, 4, 33, 15, 22, 35, 30, 13, 23, 14, 0, 24, 40, 27, 29, 37, 3, 34, 1, 36, 20]	631	1	45, 12, 33, 15, 0, 10	30%	573	58	Todos	
			2	45, 12, 33, 15, 0, 10, 7, 23, 14, 40	50%	528	103	Todos	
			3	45, 12, 33, 15, 0, 10, 7, 23, 14, 40	50%	528	103	Sólo 1 y 2	
5	[13, 30, 35, 15, 22, 46, 16, 6, 32, 43, 7, 38, 2, 21, 39, 26, 42, 17, 44, 9, 5, 25, 12, 19, 48, 28, 10, 31, 18, 41, 8, 47, 11, 4, 33, 27, 40, 29, 37, 3, 34, 1, 36, 20, 45, 24, 0, 14, 23]	663	1	45, 12, 15, 0	20%	630	33	Todos	
			2	45, 12, 15, 0, 10, 7, 23, 14	40%	564	99	Todos	27, 40
			3	45, 12, 15, 0, 10, 7, 23, 14, 40, 27	50%	540	123	Todos	
6	[8, 29, 37, 3, 1, 34, 47, 4, 11, 36, 31, 18, 20, 46, 6, 17, 42, 9, 5, 19, 48, 10, 28, 12, 45, 22, 15, 33, 27, 40, 30, 13, 24, 0, 14, 32, 43, 23, 35, 16, 7, 39, 38, 2, 26, 21, 44, 25, 41]	642	1	45, 5, 33, 15, 0, 25	30%	609	33	Todos	
			2	45, 5, 33, 15, 0, 25, 10, 7, 29	45%	552	90	Todos	
			3	45, 5, 33, 15, 0, 25, 10, 7, 29	45%	552	90	Sólo 1 y 2	
7	[35, 30, 13, 40, 27, 33, 11, 47, 8, 4, 41, 18, 31, 48, 28, 10, 20, 19, 25, 12, 5, 9, 44, 21, 39, 26, 42, 17, 38, 2, 7, 16, 6, 32, 43, 23, 14, 0, 24, 29, 37, 3, 34, 1, 36, 15, 22, 46, 45]	638	1	45, 5, 15, 0, 16	25%	603	35	Todos	
			2	45, 5, 15, 0, 16, 25, 10, 27, 29	45%	563	75	Todos	
			3	45, 5, 15, 0, 16, 25, 10, 27, 29	45%	563	75	Sólo 1 y 2	
8	[22, 46, 45, 16, 6, 32, 43, 7, 38, 2, 21, 39, 26, 42, 17, 44, 9, 5, 25, 12, 19, 48, 28, 10, 31, 18, 41, 8, 47, 11, 4, 33, 36, 1, 34, 3, 29, 37, 27, 40, 15, 35, 30, 13, 24, 0, 14, 23, 20]	640	1	45, 12, 33, 15, 0	25%	571	69	Todos	29
			2	45, 12, 33, 15, 0, 10, 7	35%	556	84	Todos	29
			3	45, 12, 33, 15, 0, 10, 7	35%	556	84	Sólo 1 y 2	29
9	[21, 39, 26, 42, 17, 38, 2, 7, 16, 6, 32, 43, 23, 14, 0, 24, 13, 30, 35, 46, 45, 22, 15, 40, 27, 33, 8, 47, 11, 4, 41, 18, 31, 36, 1, 34, 3, 29, 37, 10, 28, 48, 19, 25, 12, 20, 5, 9, 44]	633	1	45, 5, 0, 16, 25	25%	598	35	Todos	
			2	45, 5, 0, 16, 25, 10, 44, 40	40%	579	54	Todos	
			3	45, 5, 0, 16, 25, 10, 44, 40, 27, 29	50%	540	93	Todos	
10	[16, 6, 32, 43, 23, 14, 0, 24, 13, 30, 35, 46, 45, 22, 15, 40, 27, 33, 11, 47, 8, 4, 41, 18, 31, 36, 1, 34, 3, 29, 37, 20, 12, 25, 19, 48, 28, 10, 5, 9, 44, 21, 39, 26, 42, 17, 38, 2, 7]	625	1	45, 0, 16, 25, 10	25%	566	59	Todos	
			2	45, 0, 16, 25, 10, 7, 40, 27, 29	45%	511	114	Todos	
			3	45, 0, 16, 25, 10, 7, 40, 27, 29	45%	511	114	Sólo 1 y 2	
11	[3, 1, 34, 36, 47, 11, 8, 4, 41, 18, 31, 10, 28, 48, 19, 25, 12, 20, 5, 9, 44, 21, 39, 26, 42, 17, 38, 2, 7, 45, 46, 16, 6, 32, 43, 23, 14, 0, 24, 13, 30, 35, 15, 22, 33, 40, 27, 29, 37]	583	1	45, 5, 33, 15, 40	25%	535	48	Todos	
			2	45, 5, 33, 15, 40, 0, 25	35%	525	58	Todos	
			3	45, 5, 33, 15, 40, 0, 25, 10, 7, 27	50%	498	85	Todos	
12	[48, 19, 25, 12, 5, 9, 44, 21, 39, 26, 42, 17, 38, 2, 7, 45, 46, 16, 6, 32, 43, 23, 14, 0, 24, 13, 30, 35, 15, 22, 20, 28, 10, 31, 18, 41, 8, 47, 11, 4, 33, 27, 40, 29, 37, 3, 1, 34, 36]	602	1	45, 5, 36, 15	20%	569	33	Todos	
			2	45, 5, 36, 15, 0, 25, 10	35%	547	55	Todos	
			3	45, 5, 36, 15, 0, 25, 10, 7, 40, 27	50%	501	101	Todos	
13	[2, 38, 17, 42, 26, 39, 21, 44, 9, 5, 25, 12, 19, 48, 28, 10, 31, 18, 41, 8, 47, 11, 4, 33, 15, 22, 46, 16, 6, 32, 43, 23, 14, 0, 24, 13, 30, 35, 40, 27, 29, 37, 3, 34, 1, 36, 20, 45, 7]	608	1	45, 12, 33, 15, 0, 10	30%	556	52	Todos	
			2	45, 12, 33, 15, 0, 10, 7	35%	516	92	Todos	
			3	45, 12, 33, 15, 0, 10, 7	35%	516	92	Todos	
14	[14, 23, 30, 35, 46, 45, 16, 6, 32, 43, 7, 38, 2, 21, 39, 26, 42, 17, 44, 9, 5, 25, 12, 19, 48, 28, 10, 31, 18, 41, 8, 47, 11, 4, 33, 15, 22, 20, 36, 1, 34, 3, 29, 37, 27, 40, 13, 0, 24]	632	1	45, 12, 33, 15, 0	25%	577	55	Todos	
			2	45, 12, 33, 15, 0, 10, 7, 23	40%	557	75	Todos	
			3	45, 12, 33, 15, 0, 10, 7, 23, 14, 30, 29	55%	518	114	Todos	
15	[39, 26, 42, 17, 38, 2, 21, 44, 9, 5, 25, 12, 19, 48, 28, 10, 31, 18, 41, 8, 47, 11, 4, 33, 15, 22, 46, 45, 16, 6, 32, 43, 23, 14, 0, 24, 13, 30, 35, 40, 27, 29, 37, 3, 34, 1, 36, 20, 7]	616	1	45, 12, 33, 15, 0, 10	30%	556	60	Todos	
			2	45, 12, 33, 15, 0, 10, 7	35%	516	100	Todos	
			3	45, 12, 33, 15, 0, 10, 7	35%	516	100	Sólo 1 y 2	
16	[37, 3, 34, 1, 36, 11, 47, 8, 4, 41, 18, 31, 48, 28, 10, 20, 19, 25, 12, 5, 9, 44, 21, 39, 26, 42, 17, 38, 2, 7, 45, 16, 6, 32, 43, 23, 14, 0, 24, 13, 30, 35, 46, 22, 15, 40, 27, 33, 29]	581	1	45, 5, 33, 0, 16	25%	533	48	Todos	
			2	45, 5, 33, 0, 16, 25, 10, 40, 27	45%	494	87	Todos	
			3	45, 5, 33, 0, 16, 25, 10, 40, 27, 29	50%	457	124	Todos	
17	[26, 42, 17, 7, 38, 2, 21, 39, 44, 9, 5, 25, 12, 19, 48, 28, 10, 31, 18, 41, 4, 8, 47, 11, 36, 34, 1, 3, 37, 29, 27, 40, 33, 15, 22, 46, 45, 16, 6, 32, 43, 23, 14, 0, 24, 13, 30, 35, 20]	595	1	45, 12, 33, 0, 10	25%	533	62	Todos	
			2	45, 12, 33, 0, 10, 7, 40, 27	40%	509	86	Todos	
			3	45, 12, 33, 0, 10, 7, 40, 27	40%	509	86	Sólo 1 y 2	
18	[41, 8, 47, 11, 4, 33, 15, 22, 46, 16, 6, 32, 43, 23, 14, 0, 24, 13, 30, 35, 40, 27, 29, 37, 3, 1, 34, 36, 18, 31, 48, 19, 25, 12, 5, 9, 44, 21, 39, 26, 42, 17, 38, 2, 7, 45, 20, 28, 10]	604	1	45, 5, 33, 36, 15, 25	30%	565	39	Todos	
			2	45, 5, 33, 36, 15, 25, 0, 10, 7	45%	527	77	Todos	
			3	45, 5, 33, 36, 15, 25, 0, 10, 7	45%	527	77	Sólo 1 y 2	
19	[9, 44, 21, 39, 26, 42, 17, 38, 2, 7, 45, 46, 16, 6, 32, 43, 23, 14, 0, 24, 13, 30, 35, 15, 22, 20, 12, 25, 19, 48, 28, 10, 18, 31, 41, 8, 47, 11, 4, 33, 27, 40, 29, 37, 3, 1, 34, 36, 5]	630	1	45, 5, 36, 15	20%	545	85	Todos	
			2	45, 5, 36, 15, 0, 25, 10, 40	40%	501	129	Todos	
			3	45, 5, 36, 15, 0, 25, 10, 40, 7, 27	50%	476	154	Todos	
20	[22, 46, 45, 16, 6, 32, 43, 7, 17, 38, 2, 21, 39, 26, 42, 9, 44, 5, 25, 12, 19, 48, 28, 10, 31, 18, 41, 8, 47, 11, 4, 33, 15, 35, 30, 13, 14, 23, 0, 24, 40, 27, 29, 37, 3, 34, 1, 36, 20]	635	1	45, 5, 12, 33, 0, 10	30%	582	53	Todos	
			2	45, 5, 12, 33, 0, 10, 44, 7, 23, 14, 40	55%	531	104	Todos	
			3	45, 5, 12, 33, 0, 10, 44, 7, 23, 14, 40	55%	531	104	Sólo 1 y 2	

Tabla 6-3. Resultados ACO

La estructura de la tabla 6-3, en la cual se muestran los resultados tras la aplicación del ACO para la obtención de las rutas iniciales, es idéntica a la tabla 6-2, explicada anteriormente.

Como ya hicimos en el caso del GA, analicemos primero los resultados de la ruta del camión, previa inclusión de drones. En este caso, el mayor coste de entre las rutas obtenidas es de 663 u.t. frente a las 581 u.t. del menor coste. Además, según ACO, en promedio el camión tarda en visitar todos los clientes aproximadamente 622 u.t., con una desviación típica entre las muestras seleccionadas de aproximadamente 21,55 u.t.

Estos resultados ponen de manifiesto la primera conclusión que conseguimos a través del algoritmo, y es que el Algoritmo de Colonia de Hormigas obtiene mejores resultados que el Algoritmo Genético para un TSP clásico, sobre la base de datos de estudio. Por ello, en promedio, el costo aplicando el ACO es de aproximadamente 400 u.t. menos, además de producirse una menor fluctuación entre las distintas iteraciones.

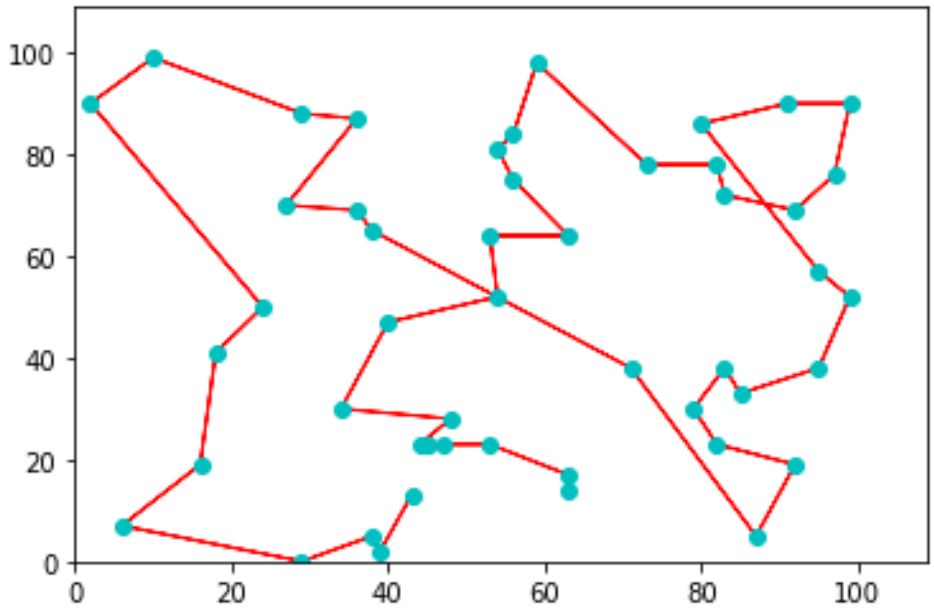


Figura 6-3. Ejemplo ruta del camión con ACO

Cabe destacar el bajo número de clientes a los cuales se les cambia de posición dentro de la ruta del camión con respecto a los mismos tras emplear GA. Al ser capaz el ACO de obtener mejores resultados que el GA, esto implica que el trayecto del camión resultante se encuentra más próximo al óptimo del problema, por lo que será más complicado encontrar cambios en las posiciones de los clientes que disminuyan el coste final.

Misma explicación encontramos como respuesta a la bajada en cuanto a porcentaje de clientes visitados por drones con respecto a los potencialmente visitables. Como podemos observar, para dos o tres drones como máximo se visitan el 55% de los clientes a los que el dron puede acceder, frente al 80% obtenido con el GA. Asimismo, para un único dron, se dan mínimos de hasta un 20%.

El mayor ahorro producido es de 154 u.t., considerablemente menor a las 267 u.t. del GA, lo cual se puede explicar, al igual que en los párrafos anteriores, por una mejor aproximación inicial por parte del ACO

En la siguiente figura se muestra el trayecto del camión (rojo) una vez que los drones 1 (verde), dos (azul) y tres (amarillo), visitan a sus respectivos clientes. El gráfico corresponde concretamente a la iteración 16 del ACO, donde se obtiene el menor coste.

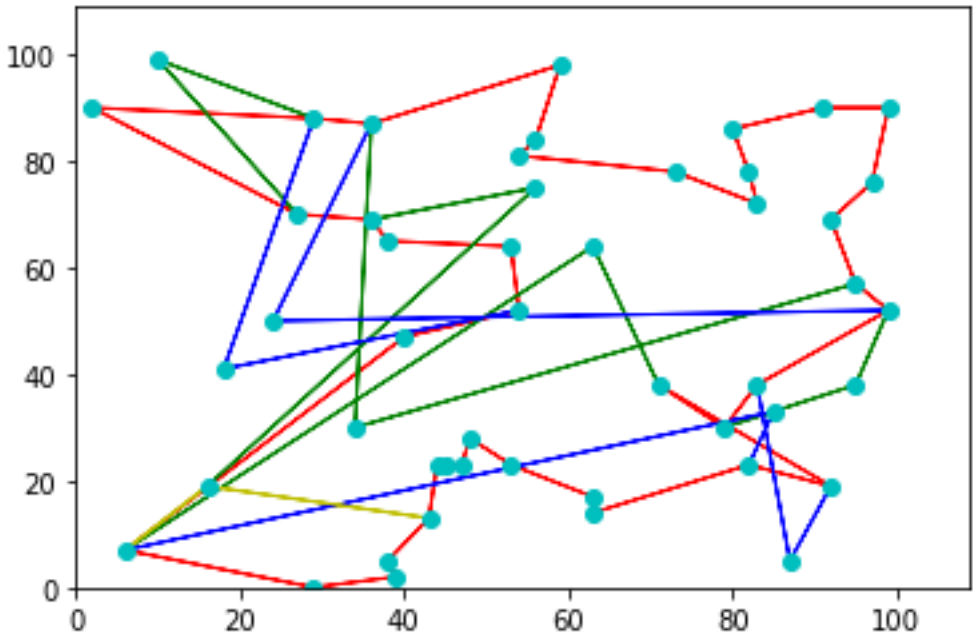


Figura 6-4. TSP con drones

A pesar del aumento del ahorro promedio al utilizar tres drones en lugar de dos, que se sitúa ahora en 11,85 u.t., éste sigue siendo muy bajo y, sumado a que únicamente es requerido el tercer dron en el 50% de los casos, para esta dimensión de problema llegamos a la misma conclusión que en el caso del GA, la mejor opción es la utilización de dos drones para el reparto a clientes.

6.3 Conclusiones

Tras analizar los resultados, surgen dos conclusiones generales:

- La importancia de la elección de la metaheurística adecuada para estimar la ruta inicial del camión. Observamos que, a pesar de los grandes ahorros obtenidos tras la ejecución del algoritmo de drones, el coste final sigue siendo sustancialmente mayor incluso que los costes de la ruta del camión obtenidos por ACO, sin la utilización de drones. Ésto pone en evidencia que partir de los mejores datos de partida posibles se consiguen los menores costes finales, particularizando los resultados del ACO como mejores que los de GA.
- El dimensionamiento de la flota de drones no siempre se rige por cuantos más, mejor. El caso estudiado expone con clarividencia la importancia de la realización de un estudio pormenorizado a la hora de seleccionar el número de drones. Por ejemplo, si incluíamos cuatro drones posibles, el cuarto nunca era utilizado, así como la utilización del tercero genera la duda de si los ahorros producidos son suficientes como para que compensen el sobrecoste asociado.

Como ya se mencionó anteriormente, el algoritmo generado para la inclusión de los drones es completamente determinista, pues, aunque parta de una ruta inicial obtenidas gracias a metaheurísticas, que llevan intrínsecas un componente pseudoaleatorio, de esta ruta se obtiene un único resultado final de los distintos drones utilizados. Una posible línea de estudio sería la de otorgarle dicho componente aleatorio también a la asignación de drones, incluyendo nuestro algoritmo a la propia metaheurística. Dicho cambio de estrategia podría generar unos resultados aún más verosímiles, y sería interesante analizar y comparar ambos modos (el empleado en este TFG y el recién explicado).

Sin duda alguna, la mayor dificultad encontrada a la hora de realizar el algoritmo se encontró en la adaptación de un único dron a la de un número a determinar *n* de drones. Ciertos conceptos que suponían la columna vertebral del algoritmo para un único dron (como las *subrutas*, explicadas anteriormente), debían ser reconvertidos a las exigencias de varios drones. Eso supuso la simplificación del modelo, aceptando que, si un movimiento en el momento de estudio es factible, permanece siéndolo a lo largo del tiempo, a pesar de que otros movimientos puedan generar un aumento del tiempo de vuelo de algún dron. El lograr una actualización continua de la capacidad de los drones para visitar los clientes asignados a pesar de los cambios producidos a posteriori es, sin lugar a dudas, algo esencial a la hora de aplicar este algoritmo a problemas reales.

REFERENCIAS

[1] "2021 Complete Python Bootcamp From Zero to Hero in Python | Udemy." [Online]. Available: https://www.udemy.com/course/complete-python-bootcamp/learn/lecture/9407966?start=0#overview. [Accessed: 29-May-2021].

[2] "Anaconda | The World's Most Popular Data Science Platform." [Online]. Available: https://www.anaconda.com/. [Accessed: 29-May-2021].

[3] "Definición y componentes de la logística industrial – Apuntes para universitarios." [Online]. Available: https://edukativos.com/apuntes/archives/3868. [Accessed: 13-Jun-2021].

[4] "TSP Applications." [Online]. Available: http://www.math.uwaterloo.ca/tsp/apps/index.html. [Accessed: 17-May-2021].

[5] S. Lin and B. W. Kernighan, "An Effective Heuristic Algorithm for the Traveling-Salesman Problem."

[6] P. C. Gilmore and R. E. Gomory, "Sequencing a One State-Variable Machine: A Solvable Case of the Traveling Salesman Problem," *Oper. Res.*, vol. 12, no. 5, pp. 655–679, Oct. 1964.

[7] S. Benhida and A. Mir, "||V (I) || PP 17-21 International organization of Scientific Research," 2018.

[8] "Using the Bing Maps Distance Matrix API for solving the Travelling Salesman Problem: the origin of routing problems - Nouss." [Online]. Available: https://www.noussintelligence.com/en/using-the-bing-maps-distance-matrix-api-for-solving-the-travelling-salesman-problem-the-origin-of-routing-problems/. [Accessed: 13-Jun-2021].

[9] J. Cooper and R. Nicolescu, "The Hamiltonian cycle and travelling salesman problems in cP systems," *Fundam. Informaticae*, vol. 164, no. 2–3, pp. 157–180, Jan. 2019.

[10] Félix García Merayo, "La magia de los grafos," 2002.

[11] "El problema del viajante - Asociación Nacional de Estudiantes de Matemáticas." [Online]. Available: http://www.anem.es/web/2018/09/el-problema-del-viajante/. [Accessed: 17-May-2021].

[12] "Camino hamiltoniano - Wikipedia, la enciclopedia libre." [Online]. Available: https://es.wikipedia.org/wiki/Camino_hamiltoniano. [Accessed: 13-Jun-2021].

[13] K. Menger, "Das Botenproblem," *Kolloquium 12*, 1930. [Online]. Available: https://www.google.com/search?q=Menger%2C+K.+Das+Botenproblem.+Kolloquium+12%2C+1930.&oq=Menger%2C+K.+Das+Botenproblem.+Kolloquium+12%2C+1930.&aqs=chrome.0.69i59.1129j0j7&sourceid=chrome&ie=UTF-8. [Accessed: 17-May-2021].

[14] M. M. Flood, "The Traveling-Salesman Problem," *Oper. Res.*, vol. 4, no. 1, pp. 61–75, Feb. 1956.

[15] "The Traveling Salesman Problem Is Not NP-complete." [Online]. Available: https://eklitzke.org/the-traveling-salesman-problem-is-not-np-complete. [Accessed: 15-May-2021].

[16] "NP-hard - Wikipedia, la enciclopedia libre." [Online]. Available: https://es.wikipedia.org/wiki/NP-hard. [Accessed: 13-Jun-2021].

[17] "TSP: Formulación." [Online]. Available: https://knuth.uca.es/moodle/mod/page/view.php?id=3416&forceview=1. [Accessed: 17-May-2021].

[18] L. Bibiana Rocha Medina, E. Cristina González La Rota, and J. Arturo Orjuela Castro, "Una revisión al estado del arte del problema de ruteo de vehículos: Evolución histórica y métodos de solución," 2011.

[19] "Multiple Traveling Salesman Problem (mTSP) | NEOS." [Online]. Available: https://neos-guide.org/content/multiple-traveling-salesman-problem-mtsp. [Accessed: 17-May-2021].

[20] "Multiple Traveling Salesmen Problem (m-TSP). Source: Bektas, 2006. | Download Scientific Diagram." [Online]. Available: https://www.researchgate.net/figure/Multiple-Traveling-Salesmen-Problem-m-TSP-Source-Bektas-2006_fig2_280974382. [Accessed: 13-Jun-2021].

[21] S. Trigos Muriel Daniel Utrera Jaén, "D6-TSPTW Problema del viajante con ventanas de tiempo."

[22] H. Zhang, H. Ge, J. Yang, and Y. Tong, "Review of Vehicle Routing Problems: Models, Classification and Solving Algorithms," *Arch. Comput. Methods Eng.*, pp. 1–27, Apr. 2021.

[23] G. B. Dantzig and J. H. Ramser, "The Truck Dispatching Problem," 1959.

[24] G. Dantzig, R. Fulkerson, and S. Johnson, "Solution of a Large-Scale Traveling-Salesman Problem," 1954.

[25] "An example of a vehicle routing problem with time windows | Download Scientific Diagram." [Online]. Available: https://www.researchgate.net/figure/An-example-of-a-vehicle-routing-problem-with-time-windows_fig1_333443449. [Accessed: 13-Jun-2021].

[26] C. C. Murray and A. G. Chu, "The flying sidekick traveling salesman problem: Optimization of drone-assisted parcel delivery," *Transp. Res. Part C Emerg. Technol.*, vol. 54, pp. 86–109, May 2015.

[27] "Los primeros drones de reparto de Walmart echarán a volar en Carolina del Norte." [Online]. Available: https://www.ticbeat.com/tecnologias/walmart-comienza-reparto-con-drones/. [Accessed: 13-Jun-2021].

[28] "La verdad sobre los drones repartidores: la tecnología no está tan madura como nos vendieron." [Online]. Available: https://theconversation.com/la-verdad-sobre-los-drones-repartidores-la-tecnologia-no-esta-tan-madura-como-nos-vendieron-131508. [Accessed: 30-May-2021].

[29] "Por qué Amazon, UPS o Domino's invierten en drones de reparto | Business Insider España." [Online]. Available: https://www.businessinsider.es/amazon-ups-dominos-invierten-drones-reparto-582105. [Accessed: 30-May-2021].

[30] "Reparto en casa con drones: la polémica que viene." [Online]. Available: https://www.lavanguardia.com/vida/20190504/462013093290/reparto-paquetes-domicilio-drones-quejas-ruido-intimidad.html. [Accessed: 30-May-2021].

[31] "Los drones de reparto ya vuelan en Villaverde: de paquetes a comida a domicilio." [Online]. Available: https://www.elespanol.com/omicrono/hardware/20210312/drones-reparto-vuelan-villaverde-paquetes-comida-domicilio/565194676_0.html. [Accessed: 30-May-2021].

[32] N. Agatz, P. Bouman, and M. Schmidt, "Optimization approaches for the traveling salesman problem with drone," *Transp. Sci.*, vol. 52, no. 4, pp. 965–981, Jul. 2018.

[33] C. C. Murray and R. Raj, "The multiple flying sidekicks traveling salesman problem: Parcel delivery with multiple drones," *Transp. Res. Part C Emerg. Technol.*, vol. 110, pp. 368–398, Jan. 2020.

[34] R. Baldacci, P. Toth, and D. Vigo, "Exact algorithms for routing problems under vehicle capacity constraints," *Ann. Oper. Res.*, vol. 175, no. 1, pp. 213–245, Feb. 2010.

[35] "Algortimos de fuerza bruta y BackTracking – Configuraciones para recordar." [Online]. Available: https://sebamawa.wordpress.com/2017/07/09/fuerza-bruta-y-backtracking/. [Accessed: 29-May-2021].

[36] "4.12. Programación dinámica — Solución de problemas con algoritmos y estructuras de datos." [Online]. Available: https://runestone.academy/runestone/static/pythoned/Recursion/ProgramacionDinamica.html. [Accessed: 29-May-2021].

[37] "Branch and Bound Algorithm | Baeldung on Computer Science." [Online]. Available: https://www.baeldung.com/cs/branch-and-bound. [Accessed: 13-Jun-2021].

[38] "11 Animated Algorithms for the Traveling Salesman Problem." [Online]. Available: https://stemlounge.com/animated-algorithms-for-the-traveling-salesman-problem/. [Accessed: 29-May-2021].

[39] "Greedy Algorithms | Brilliant Math & Science Wiki." [Online]. Available: https://brilliant.org/wiki/greedy-algorithm/. [Accessed: 29-May-2021].

[40] B. I. Simmons, C. Hoeppke, and W. J. Sutherland, "Beware greedy algorithms," *Journal of Animal Ecology*, vol. 88, no. 5. Blackwell Publishing Ltd, pp. 804–807, 01-May-2019.

[41] "A Simple Introduction to K-Nearest Neighbors Algorithm | by Dhilip Subramanian | Towards Data Science." [Online]. Available: https://towardsdatascience.com/a-simple-introduction-to-k-nearest-neighbors-algorithm-b3519ed98e. [Accessed: 29-May-2021].

[42] Ye Zheng and Eric Bach, "Advanced Algorithms and Data Structures: Christofides' Algorithm ," 2003.

[43] Francisco de Paula Pastrana Alcántara and María Rodríguez Palero, "Optimización de Rutas de Vehículos de Recogida de Basuras mediante MILP," Sevilla, 2020.

[44] A. Cirila and R. Cañari, "Conceptos, algoritmo y aplicación al problema de las N-reinas Capítulo3. Búsqueda de tabú," 2005.

[45] I. Cristian Trelea, "The particle swarm optimization algorithm: convergence analysis and parameter selection," 2003.

[46] Konstantinos E. Parsopoulos and Michael N. Vrahatis, "Multi-objective particle swarm optimization approaches." [Online]. Available: https://www.researchgate.net/publication/267752157_Multi-objective_particle_swarm_optimization_approaches. [Accessed: 29-May-2021].

[47] S. Ahmed Haroun, B. Jamal, and E. Hassani Hicham, "A Performance Comparison of GA and ACO Applied to TSP," 2015.

[48] C. Martinez, O. Castillo, and O. Montiel, "Comparison between ant colony and genetic algorithms for fuzzy system optimization," *Stud. Comput. Intell.*, vol. 154, pp. 71–86, 2008.

[49] "Algoritmos genéticos: cómo funcionan y para qué se utilizan." [Online]. Available: https://blogs.imf-formacion.com/blog/tecnologia/algoritmos-geneticos-como-funcionan-202010/. [Accessed: 29-May-2021].

[50] "Traveling Salesman Problem using Genetic Algorithm - GeeksforGeeks." [Online]. Available: https://www.geeksforgeeks.org/traveling-salesman-problem-using-genetic-algorithm/. [Accessed: 29-May-2021].

[51] "Algoritmos genéticos - Crossover." [Online]. Available: https://www.hebergementwebs.com/tutorial-sobre-algoritmos-geneticos/algoritmos-geneticos-crossover. [Accessed: 13-Jun-2021].

[52] "GitHub - DEAP/deap: Distributed Evolutionary Algorithms in Python." [Online]. Available: https://github.com/deap/deap. [Accessed: 15-Jun-2021].

[53] "deap/tsp.py at master · DEAP/deap · GitHub." [Online]. Available: https://github.com/DEAP/deap/blob/master/examples/ga/tsp.py. [Accessed: 15-Jun-2021].

[54] C. Blum, "Ant colony optimization: Introduction and recent trends," *Phys. Life Rev.*, vol. 2, pp. 353–373, 2005.

[55] M. Brand, M. Masuda, N. Wehner, and X.-H. Yu, *Ant Colony Optimization Algorithm for Robot Path Planning I I 2 1*. 2010.

[56] Marco Dorigo and Thomas Stützle, "Ant Colony Optimization | The MIT Press," 2004. [Online]. Available: https://mitpress.mit.edu/books/ant-colony-optimization. [Accessed: 29-May-2021].

[57] "Combinatorial Problems and Ant Colony Optimization Algorithm | Udemy." [Online]. Available: https://www.udemy.com/course/antcolonyoptimization/learn/lecture/11586972?start=0#overview. [Accessed: 29-May-2021].

[58] "Algoritmo de la colonia de hormigas - Wikipedia, la enciclopedia libre." [Online]. Available: https://es.wikipedia.org/wiki/Algoritmo_de_la_colonia_de_hormigas. [Accessed: 13-Jun-2021].

[59] "TSP-D-Instances/uniform-71-n50.txt at v1.2 · pcbouman-eur/TSP-D-Instances · GitHub." [Online]. Available: https://github.com/pcbouman-eur/TSP-D-Instances/blob/v1.2/uniform/uniform-71-n50.txt.

[Accessed: 12-Jun-2021].

[60] Y. Zheng, Q. Luo, H. Wang, C. Wang, and X. Chen, "Path planning of mobile robot based on adaptive ant colony algorithm," *J. Intell. Fuzzy Syst.*, vol. 39, no. 4, pp. 5329–5338, Jan. 2020.

ANEXO

- ***Basedatos:***

```
#Algoritmo para mostrar los datos como la diferencia relativa entre los clientes, siendo a una lista con las X de los clientes, y b una con las Y
mapa=[]
n=49
for x in range(0,n):
    mapa.append([])
for x in range(0,n):
    for y in range(0,n):
        numero=math.sqrt((a[x]-a[y])**2+(b[x]-b[y])**2)
        redondeado=round(numero)
        mapa[x].append(redondeado)
```

- ***calcSavings:***

```
def calcSavings(j):
    #Calcularemos el ahorro de quitar j de la ruta de camiones de un modo distinto al calculado en el código para un solo dron, debido a que se complica en exceso para varios drones
    ttruck_savings=[]
    truckRoutes_savings= truckRoutes[:]
    #Copiamos la ruta del camión en una nueva lista y eliminamos el cliente j
    truckRoutes_savings.remove(j)
    #Calculamos los tiempos de llegada a cada nodo sin el cliente j
    for i in range(0,len(truckRoutes_savings)):
        if (i==0):
            ttruck_savings.append(0)
        else:
            a=ttruck_savings[i-1]+t[truckRoutes_savings[i-1]][truckRoutes_savings[i]]
            ttruck_savings.append(a)
    #Los savings serán la diferencia entre el coste de llegar al último nodo con o sin j
    savings= ttruck[-1]-ttruck_savings[-1]
```

```
    return savings
```

- ***calCostTruck:***

```
#Calcula el coste de insertar j en una posición distinta de la ruta del camión
def calCostTruck (j,subruta):
    #Vemos cuales son los nodos de inicio y fin de la subruta, necesarios para el calculo posterior de ver si la autonomía del dron es suficiente
    a=subruta[0]
    maxsavings=0
    iprima=0
    kprima=0
    servedbyUAV = False
    indicea=truckRoutes.index(a)
    b=subruta[-1]
    indiceb=truckRoutes.index(b)
    #El for calcula el coste de meterlo entre x y x+1, de la ruta
    for x in range(indicea,indiceb):
        cost=t[truckRoutes[x]][j]+t[j][truckRoutes[x+1]]-t[truckRoutes[x]][truckRoutes[x+1]]
        #Primero hay que comprobar que lo que te gastas es menor de lo que te ahorras
        if (cost<savings):
            #Comprobamos que el UAV puede seguir volando ese tiempo
            if (ttruck[indiceb]-ttruck[indicea]+cost<=e):
                #y vemos si la diferencia entre ahorro y coste es mayor que el máximo ahorro, haciendo rentable la operación
                if(savings-cost>maxsavings):
                    servedbyUAV=False
                    maxsavings=savings-cost
                    iprima=truckRoutes[x]
                    kprima=truckRoutes[x+1]

    return maxsavings,j,iprima,kprima , servedbyUAV
```

- ***calCostUAV:***

```
#calCostUAV calcula el coste de servir al cliente j con un dron
```

```
def calCostUAV(j,subruta):
    #Nodos de inicio y fin de la subruta, similar a calcCostTruck
    iprima=0
    kprima=0
    servedbyUAV=True
    a=subruta[0]
    maxsavings=0
    indicea=truckRoutes.index(a)
    b=subruta[-1]
    indiceb=truckRoutes.index(b)
    #Doble for porque no sólo hay que tener en cuenta al inmediatamente después de x, sino todos los índices mayores que x
    for x in range(indicea,indiceb):
        for y in range(indicea,indiceb+1):
            #Para todo "x" que preceda a "y", siendo "j" distinto a "x" e "y"
            if (x<y and truckRoutes[y]!=j and truckRoutes[x]!=j):
                #Comprobamos si el dron tiene autonomía para ir al cliente "j"
                if (t[truckRoutes[x]][j]+t[j][truckRoutes[y]]<=e):
                    #Tiempo que se tarda en llegar al nodo k (el situado en y en la ruta) si quitamos j: tprimay
                    tprimay=0
                    for z in range(0,y):
                        if j!= truckRoutes[z] and j!= truckRoutes[z+1]:
                            tprimay += t[truckRoutes[z]][truckRoutes[z+1]]
                        elif j== truckRoutes[z+1]:
                            tprimay += t[truckRoutes[z]][truckRoutes[z+2]]
    # Después calculamos costes con esos máximos y comparamos savings y cost de forma muy similar a la anterior función
                    cost=max(0,max((tprimay-ttruck[x])+sr,t[truckRoutes[x]][j]+t[j][truckRoutes[y]]+sr)-(tprimay-ttruck[x]))
                    if savings-cost > maxsavings:
                        maxsavings=savings-cost
                        servedbyUAV=True
                        iprima=truckRoutes[x]
                        kprima=truckRoutes[y]
    return maxsavings,j,iprima,kprima, servedbyUAV
```

- ***performUpdate:***

```
def performUpdate (j,kprima,iprima, servedbyUAV,inicio_subrutasUAV,indicef):
  #Tenemos que saber si el cliente "j" será visitado por un UAV o si se cambio su posición en la ruta, a través de la variable servedbyUAV
  #Todo lo realizaremos para el dron indicef, el dron con mayor ahorro del código principal
  if servedbyUAV==True:
    #Elimino el nodo "j" de los nodos pendientes de estudio del dron
    Cprimecopia.remove(j)
    newSubRoute=[]
    newSubRoute2=[]
    #Genero la subruta
    for h in range(truckRoutes.index(iprima),truckRoutes.index(kprima)+1):
      if (truckRoutes[h]!=j):
        newSubRoute.append(truckRoutes[h])
    print("La nueva subruta asociada al vuelo del dron",indicef,"es",newSubRoute,"y el dron visita al cliente",j)
    #En el caso de que los nodos inicial y final de la nueva subruta sean dos puntos intermedios de la ruta del camión, se generan 3 subrutas, dos no visitadas por dron y una que sí, ahora genero la subruta de los nodos posteriores al nodo final de la nueva sub ruta
    final=truckRoutes[-1]
    for z in range(truckRoutes.index(kprima)+1,truckRoutes.index(final)+1):
      if (truckRoutes[z]!=j):
        newSubRoute2.append(truckRoutes[z])
    #Elimino de las subrutas el nodo con el dron, y todos los elementos de las nuevas subrutas
    truckRoutes.remove(j)
    for l in newSubRoute:
      for y in range(0,len(subrutas.get(indicef))):
        if (l in subrutas.get(indicef)[y]):
          subrutas.get(indicef)[y].remove(l)
    for k in subrutas.keys():
      for y in range(0,len(subrutas.get(k))):
        if (j in subrutas.get(k)[y]):
          subrutas.get(k)[y].remove(j)
```

```
        for x in newSubRoute2:
            for y in range(0,len(subrutas.get(indicef))):
                if (x in subrutas.get(indicef)[y]):
                    subrutas.get(indicef)[y].remove(x)
        #Añado la nueva subruta sin dron, las ordenaré posteriormente
        if newSubRoute2!=[]:
            subrutas.get(indicef).append(newSubRoute2)
        #Añado la nueva subruta a las subrutas
        subrutas.get(indicef).append(newSubRoute)
        #Genero una lista con todos los primeros nodos de las subrutas que tienen asociado un UAV. Contiene también los clientes visitados por el dron
        inicio_subrutasUAV[indicef].append(newSubRoute[0])
        clientes_UAV[indicef].append(j)
        #Elimino posibles subrutas vacías, por si el punto de aterrizaje del dron es el punto final de la ruta del camión
        for z in range(0,len(subrutas.get(indicef))-1):
            if  subrutas.get(indicef)[z] == []:
                subrutas.get(indicef).remove(subrutas.get(indicef)[z])
    else:
        #Elimino la j de la actual truck sub route
        for k in range(0,len(subrutas)):
            for y in range(0,len(subrutas.get(k))):
                if (j in subrutas.get(k)[y]):
                    subrutas.get(k)[y].remove(j)
            #La inserto entre i prima y k prima, sabiendo que son consecutivas
                if (kprima in subrutas.get(k)[y]):
                    subrutas.get(k)[y].insert(subrutas.get(k)[y].index(kprima),j)
        #Elimino el cliente j de donde estaba en la ruta del camión y lo inserto en la nueva posición
        truckRoutes.remove(j)
        truckRoutes.insert(truckRoutes.index(kprima), j)
    #Borro la anterior ttruck y genero la nueva
    for i in range(0,len(ttruck)):
        for j in range(0,len(ttruck)):
            if (ttruck !=[]):
```

```
            ttruck.remove(ttruck[i])
    for i in range(0,len(truckRoutes)):
        if (i==0):
            ttruck.append(0)
        else:
            a=ttruck[i-1]+t[truckRoutes[i-1]][truckRoutes[i]]
            ttruck.append(a)
    return
```

- ***Inicialización:***

```
decision=[]
#Cprime: Clientes visitables por el dron.
Cprime = [45,5,12,33,36,11,15,0,16,25,10,44,7,23,14,30,40,27,29,47]
ttruck= #Lista de tiempos de llegada a cada nodo
maxsavings=0
savings=0
sr=2 #Valor adoptado para simular el coste temporal de recoger el dron y prepararlo para su despegue por parte del transportista.
e=150 #Valor adoptado como autonomía del dron
n=#Número de drones
inicio_subrutasUAV=[]
clientes_UAV=[]
for x in range(0,n):
    inicio_subrutasUAV.append([])
for x in range(0,n):
    clientes_UAV.append([])
Cprimecopia= Cprime[:]
print("Clientes a los que puede ir el dron",Cprimecopia)
#Debido a que tenemos diferentes drones con distintas subrutas, tenemos que crear un diccionario de dimensión igual al número de drones, donde iremos sumando las distintas subrutas generadas en función del dron
subrutas={}
for x in range(0,n):
    subrutas[x]=[truckRoutes[:]]
```

- ***Código principal:***

```
for j in [45,5,12,33,36,11,15,0,16,25,10,44,7,23,14,30,40,27,29,47]:
  print ("Cliente a estudiar su visita con el dron",j)
  #Aplico la función calcSavings para calcular los ahorros
  calcSavings(j)
  savings=calcSavings(j)
  if (savings>0):
    #Por cada dron, evalúo si puedo servir al cliente con el dron o si puedo cambiarlo de posición en la ruta del camión.
    #Posteriormente, me quedo con el cambio del dron que suponga mayor ahorro
    for k in subrutas.keys():
      for x in range(0,len(subrutas.get(k))):
        #Para cada subruta, suponiendo que los ahorros son positivos, si la subruta tiene un dron asociado y el cliente "j" no está en la subruta
        if (subrutas.get(k)[x][0] in inicio_subrutasUAV[k]) and j not in subrutas.get(k)[x]:
          calCostTruck(j,subrutas.get(k)[x])
          maxsavings_if=calCostTruck(j,subrutas.get(k)[x])[0]
          iprima_if=calCostTruck(j,subrutas.get(k)[x])[2]
          kprima_if=calCostTruck(j,subrutas.get(k)[x])[3]
          servedbyUAV_if=calCostTruck(j,subrutas.get(k)[x])[4]
      #Si la subruta no tiene un dron asociado
        elif subrutas.get(k)[x][0] not in inicio_subrutasUAV[k]:
          calCostUAV(j,subrutas.get(k)[x])
          maxsavings_else=calCostUAV(j,subrutas.get(k)[x])[0]
          iprima_else=calCostUAV(j,subrutas.get(k)[x])[2]
          kprima_else=calCostUAV(j,subrutas.get(k)[x])[3]
          servedbyUAV_else=calCostUAV(j,subrutas.get(k)[x])[4]

  #Para varias subrutas tengo que comprobar qué opción me da mayores ganancias
      if (maxsavings_if>maxsavings_else) and savings>0:
        maxsavings=maxsavings_if
        iprima=iprima_if
        kprima=kprima_if
        servedbyUAV= servedbyUAV_if
```

```
            print("Se ha cambiado al cliente",j,"de posición")
            decision.append([maxsavings,iprima,kprima,servedbyUAV])
        elif savings>0:
            maxsavings=maxsavings_else
            iprima=iprima_else
            kprima=kprima_else
            servedbyUAV=servedbyUAV_else
            decision.append([maxsavings,iprima,kprima,servedbyUAV])
    maximo=0
    #En la lista decision he ido incluyendo sublistas cuyo primer elemento es los ahorros asociados al movimiento del dron del índice en cuestión
    #Tomo el de mayor ahorro
    for f in range(0,len(decision)):
        if decision[f][0]>maximo:
            indicef=f
            maximo=decision[f][0]
            maxsavings=decision[indicef][0]
            iprima=decision[indicef][1]
            kprima=decision[indicef][2]
            servedbyUAV=decision[indicef][3]
    if (maxsavings>0):
        performUpdate(j, kprima, iprima, servedbyUAV, inicio_subrutasUAV,indicef)
        maxsavings=0
        maxsavings_else=0
        maxsavings_if=0
        for k in subrutas.keys():
            for x in range(0,len(subrutas.get(k))):
                if subrutas.get(k)[x]==[]:
                    subrutas.get(k).remove(subrutas.get(k)[x])
        print ("Clientes que quedan como posibles visitas del dron",Cprimecopia)
        print("Nueva ruta del camión:",truckRoutes)
        print("Nuevos tiempos de llegada a cada nodo del camión",ttruck)
        print("Nodos de inicios de las subrutas en función del dron",inicio_subrutasUAV)
        print("Clientes visitados por drones",clientes_UAV)
        #Aquí ordeno las subrutas de modo que sigan la misma secuencia de la ruta del camión original
```

```
            lista=[]
            for x in range(0,len(subrutas.get(indicef))):
                i_sub=truckRoutes.index(subrutas.get(indicef)[x][0])
                lista.append([i_sub,subrutas.get(indicef)[x]])
            lista.sort()
            for z in range (0,len(lista)):
                subrutas.get(indicef).remove(lista[z][1])
                subrutas.get(indicef).insert(z, lista[z][1])
        decision.clear()
    print("Las nuevas subrutas en función de los drones",subrutas,"\n")
```