

Tarea 06 - Problema de rutas de vehículos



Matemática computacional

Universidad de Sevilla

Daniel de los Reyes Leal

Alejandro Sánchez Medina

Índice

[Introducción](#)

[Problema de rutas de vehículos](#)

[Variantes](#)

[Función objetivo](#)

[Tamaño de la flota](#)

[Tipo de flota](#)

[Número de almacenes](#)

[Tipo de demanda](#)

[Restricciones de coste](#)

[Ventanas de tiempo](#)

[Precedencia](#)

[Capacidad del vehículo](#)

[Grafo de ciudades](#)

[Métodos de resolución](#)

[Método exacto](#)

[Método heurístico](#)

[Local 1-shift en fase constructiva](#)

[Local 1-shift en fase de mejora](#)

[Local 2-opt en fase de mejora](#)

[Perturbation](#)

[Random Solution](#)

[Resolución práctica](#)

[Método exacto](#)

[Método heurístico](#)

[Experimentación](#)

[Tiempo](#)

[Calidad de la solución](#)

[Conclusiones](#)

[Integración](#)

[Cálculo de las rutas de hoy](#)

[Consulta de ruta de hoy](#)

[Conclusiones](#)

[Bibliografía](#)

[Anexo I: Carga de trabajo](#)

Introducción

Tras ver en la anterior tarea una primera aproximación a los problemas del viajante nos enfrentamos a todo un ámbito ampliado de este problema: Los problemas de rutas de vehículos. Su investigación supone una completa variación, en cada aspecto imaginable de los problemas de viajante típicos.

De esta variación surge la versatilidad y aplicaciones que estos problemas tienen en la vida real. Por ello, vamos a investigar sus variantes y soluciones. De entre todas las soluciones posibles que podríamos investigar, vamos a seleccionar un método exacto y un método heurístico.

El objetivo de estas implementaciones es el de poder añadir una nueva funcionalidad al proyecto Acme-Supermarket. Esta funcionalidad es la de calcular la ruta de mínimo coste para repartir productos a unos clientes, restringidos a unas ventanas de tiempo. Este ámbito de problemas se llaman los problemas del vendedor con ventanas de tiempo (TSPTW) el cual explicaremos en profundidad en este trabajo.

Por último documentaremos tanto la integración, como los resultados de nuestras pruebas de rendimiento de calidad y tiempo.

Con todo ello, cabe comentar que con este trabajo queremos optar al nivel A.

Problema de rutas de vehículos

El problema de rutas (o enrutado) de vehículos (en inglés Vehicle Routing Problem, VRP) es una generalización del anterior problema del viajante. Se basa en planificar un conjunto de rutas para una flota de vehículos que deben visitar a unos clientes de manera que las rutas sean óptimas.

Su aplicación es de gran utilidad en ámbitos de reparto de mercancías, rutas para vendedores, rutas de autobuses escolares, teledirección de drones, planificación de sistemas de satélites, planificación de programas de publicidad entre otros, recogida de basuras y en muchos sistemas de producción, donde suponen un ahorro de entre el 5% y el 20% del coste total del transporte [1].

El VRP es un nombre que engloba a muchas variantes de un problema estándar, donde se pretende encontrar unas rutas óptimas de reparto desde unos almacenes a un número de ciudades o clientes con una serie de restricciones añadidas. Entre uno de los problemas englobados, está el problema del vendedor (en inglés Travel Salesman Problem, TSP), en el que la ciudad de origen es un almacén concreto (llamado nodo 0) y las restricciones cambian.

Estas restricciones diferentes hacen que el VRP sea un TSP con múltiples vehículos, añadiéndoles capacidades e incluyendo un problema de empaquetamiento de contenedores (en inglés Bin Packing Problem, BPP). Este BPP es también otro problema NP-Completo, lo que resalta el verdadero reto del VRP, como unión de dos problemas complejos.

El VRP básico puede modelizarse de la siguiente manera, según el modelo de Laporte (1985) [2]. Teniendo:

- S objetos a repartir, con un peso asociado.
- Q capacidad de cada vehículo.

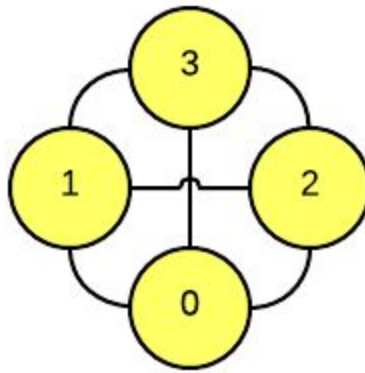
- m vehículos, previamente calculado como la mínima cantidad de vehículos necesarios para dar servicio a los clientes (BPP) como $r(S)$.

Debido a que BPP es un problema NP-Completo, $r(S)$ puede ser aproximado mediante una cota inferior mediante el cálculo:

$$m \approx \frac{\sum_{i \in S} q_i}{Q}$$

- Un grafo completo de V vértices (clientes) y A aristas (caminos).

X es una ruta para un vehículo m , donde cada posición es un nodo del grafo visitado, por orden. Por ejemplo, para un vehículo que recorrerá una ruta comenzando y finalizando en el mismo almacén, X sería:



0	1	2	3	0
---	---	---	---	---

Figura 1: Ejemplo de codificación de VRP básico.

C es el coste de la ruta de un vehículo, donde cada posición corresponde al coste asociado para viajar del nodo X_e a X_{e+1} . De esta manera, la longitud de C será inferior en 1 a X .

Para una función objetivo, minimizar el coste total de las rutas:

$$F(X) : \text{Min} \sum_{e \in A} C_e X_e$$

Añadimos las restricciones:

- Cada cliente debe ser visitado exactamente una vez.

$$\sum_{e \in \delta(i)} X_e = 2$$

- Cada vehículo (m) tendrá una ruta, pasando dos veces por el almacén.

$$\sum_{e \in \delta(0)} X_e = 2m$$

- El número de aristas necesario para cualquier subconjunto de clientes es mayor o igual que el doble de vehículos mínimos necesario para suplir la necesidad de ese subconjunto. Si esto no se cumpliera, algún vehículo tendría que pasar dos veces por un nodo.

$$\sum_{e \in \delta(S)} X_e \geq 2r(S) \text{ , } S \subseteq V \setminus \{0\}, S \neq \emptyset$$

- Las aristas que no pasen por el almacén pueden ser visitadas 0 o 1 vez.

$$X_e \in \{0, 1\}, e \notin \delta(0)$$

- Las aristas que pasen por el almacén pueden ser visitadas 0, 1 o 2 veces.

$$X_e \in \{0, 1, 2\}, e \in \delta(0)$$

Variantes

Como hemos introducido anteriormente, los VRP conforman una variedad de subproblemas derivados o similares al visto anteriormente. En esta sección tratamos de comentar estas variantes y cómo el modelo es alterado.

Función objetivo

La función objetivo es aquello que determina la valoración de una solución. Típicamente, la función objetivo del VRP es minimizar un coste, aunque qué representa este coste puede variar en función de las necesidades del sistema.

Para el planteamiento hemos considerado que la función objetivo era minimizar la distancia. En la vida real, ciertas carreteras o caminos del mapa (grafo) se recorren a

diferentes velocidades y con distintas densidades de tráfico, por lo que la minimización de distancia puede no ser un valor adecuado para minimizar.

En este caso, necesitaríamos una función que fuera capaz de traducir cada arista a un valor de coste temporal. Por ejemplo, una función que recoja la velocidad de tránsito de esa arista y su distancia y nos calcule el tiempo de viaje de su recorrido. Esta variante de la función objetivo no requiere un gran cambio en el modelo, tan solo la creación de esta estructura de datos intermedia.

En otras ocasiones, por necesidades del negocio, conviene disminuir drásticamente la cantidad de vehículos necesarios para realizar el transporte de los objetos. En este caso, el problema VRP se concreta en un problema BPP.

Ahora consideramos todas las aristas de peso 0, debido a que ahora las rutas estarán determinadas por los objetos contenidos en los vehículos. En un primer proceso para calcular el número de vehículos (m), resolvemos un BPP en el que se asignen los objetos de los clientes a un número indeterminado de camiones. De esta manera, la función objetivo del problema queda transformada a la función objetivo de este subproblema.

De este problema se obtendría como solución los nodos por los que tiene que pasar cada camión, independiente de un orden. Esto no afectaría en la solución para un grafo simétrico, pero si tuviéramos un grafo dirigido, deberíamos realizar un post-proceso de ordenación para obtener la ruta de menor coste (distancia o tiempo) en recorrer los nodos previamente definidos.

Tamaño de la flota

Al variar el tamaño de la flota, consideramos un número m diferente de vehículos. En este aspecto, podemos considerar una flota como una generalización del problema con $m = 1$ vehículos, que no requeriría de un cambio drástico en el modelo. De esta manera, el cálculo del BPP quedaría anulado y m quedaría predeterminado. Si el peso

total de los objetos supera la capacidad Q del vehículo, entonces debe dar más de un viaje y puede que las restricciones deban ajustarse.

El problema típico de VRP actúa como generalización en este aspecto al igual que el problema del vendedor múltiple (en inglés Multiple Travel Salesman Problem, MTSP) lo es para un TSP básico. En este caso, el VRP se considera una generalización del MTSP. Ambos problemas tienen mucho en común, excepto:

- El MTSP siempre comienza y finaliza en el mismo almacén (1 almacen), mientras que el VRP admite una configuración con varios almacenes.
- El VRP añade una restricción mediante capacidades a los vehículos, mientras que el MTSP asume que los vehículos tienen capacidades infinitas. Es decir, los motivos por los que ambos utilizar varios vehículos son diferentes:
 - MTSP considera varios vehículos para intentar disminuir el coste total en varios viajes simultáneos en vez de un solo vehículo recorriendo todos los nodos.
 - VRP considera varios vehículos porque, al estar limitados en capacidades, sería imposible para un solo camión repartir todos los objetos en solo un viaje.

Tipo de flota

El problema de rutas de vehículos con capacidades (en inglés Capacitated Vehicle Routing Problem, CVRP) considera un conjunto de m vehículos, cada uno de ellos con una capacidad Q igual en todos los casos. Esto simplifica en gran medida el problema subyacente BPP del VRP para el cual dividimos la carga entre los vehículos según como convenga para el coste de la ruta.

En la vida real, tenemos flotas con diferentes características (camiones, furgonetas...) que no permiten esta modelización. Además, estos vehículos tienen costes de seguridad, mantenimiento y operación diferentes. Debemos, por tanto, considerar una

nueva variante de los VRP: Los problemas de rutas de vehículos de flotas heterogéneas (en inglés Heterogeneous Fleet Vehicle Routing Problem, HFVRP) [3].

La modelización es similar a la vista para el VRP básico, con algunas diferencias:

- La flota está compuesta por m tipos de vehículos diferentes, cada uno con una capacidad distinta, tal que Q_u es la capacidad del tipo de vehículo u .
- Cada tipo de vehículo también va asociado con un coste fijo de puesta en marcha expresado mediante f_u .

El objetivo es el de determinar la mejor planificación de la flota, así como las rutas, para minimizar la suma de los costes fijos y los costes de viaje en una manera que:

- Cada ruta comienza y finaliza en un almacén y está asociada a un tipo de vehículo.
- Cada cliente pertenece a exactamente una ruta.
- La capacidad de los vehículos no es excedida.

El HFVRP es una generalización del VRP básico, en el que se considera como caso especial que todos los vehículos son idénticos (sólo existe un tipo de vehículo). Al igual que el VRP, el HFVRP es un problema NP-Completo

Número de almacenes

Típicamente en el VRP básico consideramos un nodo 0 o de almacén, que sirve de punto de partida y meta para los vehículos del problema. De esto es derivado, que las aristas (carreteras) que conectan con este vértice serán las únicas donde los vehículos pueden pasar más de una vez.

En el HFVRP vimos un comienzo de lo que suponía crear varios almacenes para diferentes tipos de vehículos, aunque no entramos en profundidad en este aspecto. Las variantes del VRP que tratan este tipo de cuestiones son los problemas de rutas de vehículos con múltiples almacenes (en inglés Multi-depot Vehicle Routing Problem, MDVRP) [4].

Este problema consta de varios almacenes. Una flota de vehículos parte de cada almacén. Cada vehículo sale de un almacén, sirve a los clientes asignados y vuelve al mismo almacén. El resto de restricciones y función objetivo es idéntico al VRP.

En estos problemas, primeramente se recomienda agrupar los clientes en almacenes. De esta manera, la división de los clientes en rutas asignadas a almacenes (vehículos) es más sencillo y queda reducido a subproblemas menores [5].

En este modelo consideramos que tenemos V nodos o clientes. A estos nodos, añadimos otros V_0 nodos almacenes. $V = \{v_1, \dots, v_n\} \cup V_0$ donde $V_0 = \{v_{01}, \dots, v_{0d}\}$ son los vértices representando los almacenes. Ahora, una ruta i queda definida como $R_i = \{d, v_1, \dots, v_m, d\}$ con $d \in V_0$

Tipo de demanda

Comúnmente, podemos pensar que todos los valores de distancias, capacidades y otras características de los problemas de rutas de vehículos son algo determinista, es decir, tienen un valor determinado y conocido.

Existe una variante de los VRP que afronta situaciones donde ciertos valores del problema toman valores aleatorios con cierta probabilidad: Los problemas de rutas de vehículos estocásticos (en inglés Stochastic Vehicle Routing Problem, SVRP).

Una demanda estocástica supone que la demanda de un cliente es una variable aleatoria. Esta variable aleatoria puede tomar ciertos valores esperados en un rango. En ciertos casos, tenemos información sobre la probabilidad con la que se dan estos valores aunque en otros no. El problema que nos plantea este nuevo paradigma es que nuestra planificación puede no ajustarse perfectamente al grafo debido a estas demandas cambiantes, por lo que debemos de hacer una planificación consciente de los riesgos.

Todo ello sin contar en que estas variaciones pueden provocar no solo una peor valoración de la solución aportada, sino la violación de algunas de las restricciones y por tanto la no factibilidad de la solución.

Las aplicaciones de los SVRP aumentan en nuevos dominios de mercado que antes parecían inabarcables para los VRP básicos: restaurantes sobre ruedas, reparto de gas butano entre otros [6][7].

Los SVRP pueden ser formulados mediante un problema de programación estocástica. En este paradigma, destacan la programación de oportunidades con restricciones (en inglés Chance Constrained Programming, CCP) y la programación estocástica con recurso (en inglés Stochastic Programming with Recourse, SPR).

El enfoque de ambos es el de calcular una primera solución a priori. Más tarde, los valores de las variables aleatorias (demanda) son reveladas. En una segunda etapa, una acción correctiva es aplicada a la primera solución. Esta corrección generalmente causa una variación del coste que debe ser tenido en cuenta para la solución final.

Como ejemplo de CCP, consideremos un vehículo con una ruta planificada mediante el primer paso a priori. Debido a que las demandas son estocásticas, las variaciones inesperadas de la carga del vehículo provocaría que éste tuviera un exceso o falta de carga, con un grave retraso de su tarea. Por ello es necesario tomar una maniobra preventiva. Debe poder planear un conjunto de alternativas para poder reducir lo mínimo el rendimiento de su ruta. Estas alternativas pueden ser:

- Retornos preventivos al almacén para reponer mercancía.
- Posponer la visita a ese cliente y pasar antes por clientes con demanda esperada menor.

Es importante también saber con qué antelación tenemos la información sobre los valores estocásticos. Por ejemplo, la información sobre la demanda de un cliente puede estar solo disponible al llegar a su nodo o al cliente anterior. A medida que avanza el tiempo, las alternativas posibles van reduciéndose.

El SPR se basa en una programación en dos fases ya descrita con dos conjuntos de variables. El primer conjunto caracterizan la solución a priori dada en la primera generación, antes de la revelación de las variables aleatorias. Tras ver la factibilidad de la solución a priori se toma una corrección. El coste de la solución queda definido como la suma del coste de la solución primera y el coste del recurso de la segunda fase.

Restricciones de coste

Hasta ahora hemos considerado que un vehículo tiene la capacidad de recorrer una infinita distancia en el grafo, aunque no conformen una solución óptima. Las restricciones de coste pretenden limitar la distancia o tiempo en la que se lleva a cabo una ruta.

Esto se aplica de manera realista a situaciones de entrega, donde un vehículo tiene una capacidad de un depósito de gasolina y por tanto, no puede recorrer una distancia superior a un número determinado. O en otro caso, un conductor tiene una jornada laboral de un número de horas y no se puede superar ese tiempo para recorrer toda la ruta.

Esta variante del VRP son los problemas de rutas de vehículos con restricciones de coste (en inglés Distance-constrained Vehicle Routing Problems, DVRP) [8].

Esta variante no presenta grandes modificaciones de la modelización respecto a un problema básico VRP. No obstante, debemos añadir una restricción de coste máximo de la ruta.

$$\sum_{i \in V, i \neq j} c_{ij} \times x_{ij} \leq L_r$$

donde:

- V es el conjunto de nodos del grafo.
- c_{ij} es el coste (tiempo, distancia...) del camino que va desde el cliente i a j .

- x_{ij} es el número de veces que recorremos el camino que va desde el cliente i a j .
- L es la cota máxima para restricción de coste para una ruta r . Comúnmente se usa un valor de L fijo para todos los vehículos, aunque consideramos interesante modelarlo de esta forma pues tendría fácil integración con los VRP de flotas heterogéneas, donde vehículos con características diferentes pueden tener capacidades de recorrido diferentes.

En el ámbito de los TSP, añadir esta restricción es fácil aunque puede no tener sentido. El TSP se aplica para un sólo vehículo y busca una solución con mejor función valoración de coste. Si encuentra una solución mejor que la anterior, ya está siendo desechada. Es más interesante aplicar esta restricción de coste cuando hay múltiples vehículos.

Ventanas de tiempo

La inclusión de las ventanas de tiempo es uno de los recursos más estudiados en la resolución de los VRP y TSP. Parten de la consideración que cada cliente tiene un horario limitado al que puede ser servido y las rutas de los vehículos deben adaptarse a estos horarios, reduciendo el coste de la ruta. Tiene aplicaciones en rutas de vehículos escolares, planificación de líneas aéreas, rutas turísticas, etc. [9] Esta variante son los problemas de rutas de vehículos con ventanas de tiempo (en inglés Vehicle Routing Problems with Time Windows, VRPTW) [10].

El VRPTW tiene la misma modelización que un VRP, pero añadiendo una restricción donde cada cliente $i \in V$ tiene un intervalo $[a_i, b_i]$ en el cual puede ser visitado. Se considera para este modelo un grafo dirigido. Se modeliza formalmente de la siguiente manera. Siendo una ruta:

$$R_i = \{v_0, v_1, \dots, v_m, v_{m+1}\}$$

donde v_0 y v_{m+1} son los almacenes de comienzo y fin. Y siendo el tiempo para visitar a un cliente β_i , el tiempo de servicio en un nodo s_i .

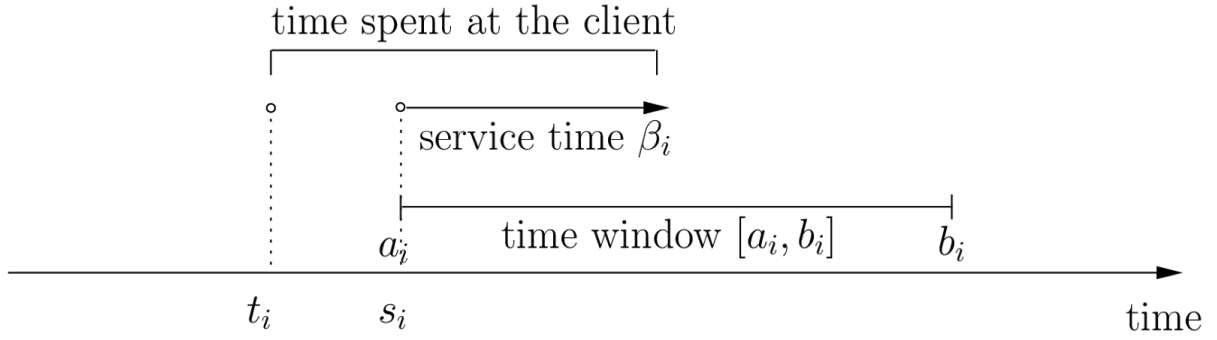


Figura 2: Esquema de variables de tiempo en VRPTW (Fuente: [11])

Las restricciones son:

- Una solución no es factible si algún cliente es visitado después de su fin de horario b_i .
- Un vehículo que visita a un cliente antes del comienzo de su horario a_i produce una espera adicional en la ruta.
- Cada ruta debe comenzar y finalizar dentro de la ventana de tiempo asociada al almacén. Esto es, el horario de trabajo del conductor.

$$a_i \leq \beta_i \leq b_i \quad (1, y \ 2)$$

$$1 \leq i \leq m$$

Se pueden calcular los horarios de los almacenes de manera automática como [12]:

$a_0 = \min_{i \in V} \{a_i - t_{0i}\}$	$b_0 = \max_{i \in V} \{b_i - t_{0i}\}$
$a_{m+1} = \min_{i \in V} \{a_i + \beta_i + t_{i,m+1}\}$	$b_{m+1} = \max_{i \in V} \{b_i + \beta_i + t_{i,m+1}\}$

donde:

- t_{ij} es el tiempo que se tarda en recorrer desde nodo i hasta el nodo j .
- $i = 0$ es el nodo correspondiente al almacén de inicio, y $i = m + 1$ es el nodo correspondiente al almacén de fin.

Cabe comentar que al definir un valor concreto para el horario del almacén (hora de inicio y de recogida) estamos añadiendo una restricción similar a la vista en la sección

Restricciones de coste, donde limitamos que las rutas tengan una distancia superior a una cota. En esta ocasión estaríamos limitando todas las rutas que finalicen en este almacén a acabar su ruta en un tiempo determinado.

Para mantener el valor de β_i para los nodos consistente, se añade la restricción:

$$x_{ij}^k(\beta_i^k + s_i + t_{ij} - \beta_j^k) \leq 0, \quad k \in K$$

donde:

- K es el conjunto de vehículos.
- s_i es el tiempo de servicio que se da en el nodo i .

$$w_j^k \geq w_i^k + s_i + t_{ij} - M_{ij}(1 - x_{ij}^k)$$

donde M es una constante tal que

$$M_{ij} = \max\{0, b_i + s_i + t_{ij} - a_j\}$$

Tal y como se propone en [13], pueden añadirse restricciones para ajustar los valores de β para que aparezcan en los rangos de horarios.

$$w_i^k \geq a_i + \sum_{j \in V, i \neq j} \max\{0, a_j - a_i + s_j + t_{ij}\} \times x_{ji}^k$$

$$w_i^k \leq b_i - \sum_{j \in V, i \neq j} \max\{0, b_i - b_j + s_i + t_{ij}\} \times x_{ij}^k$$

Al igual que en los casos del VRP básico, la función objetivo es la de minimizar el costes de las rutas, aunque debería considerarse las variaciones sobre tiempo o distancia propuestas en la sección [Variantes: Función objetivo](#).

En el ámbito de los TSPTW, solo se utiliza un vehículo por lo que la formulación es más sencilla, aunque no provoca grandes cambios en el modelo.

Precedencia

Hay casos en los que se puede encontrar un problema de rutas de vehículos en las que ciertos nodos deban ser visitados obligatoriamente antes que otros. Este caso se conoce como VRP con restricciones de precedencia.

Es difícil encontrar en la literatura referencias de soluciones y formulaciones para el VRP con restricciones de precedencia, ya que siempre se encuentra junto al de ventanas de tiempo. Es por ello que una formulación propuesta para el VRP con restricciones de precedencia puede ser similar a la de Ventanas de tiempo, pero ajustando unas ventanas “artificiales” que sean las que establezcan las relaciones de precedencia. Por ejemplo, si se tienen dos clientes distintos, llamados A y B, y el cliente A debe ser visitado antes que el cliente B, en cualquier caso, se podría establecer un VRP con ventanas de tiempo tal que así:

Cliente	Ventana
A	4:00-5:00
B	5:00-6:00

Con éste modelo, no se podría dar el caso de que el cliente B fuese visitado antes que el cliente A, porque caeríamos en una solución inválida.

El único problema que presenta esta formulación es que es incompatible con el VRP con ventanas de tiempo. En caso de querer modelar un VRP con ventanas de tiempo y con restricciones de precedencia, sería necesario usar otra formulación.

Una posible formulación alternativa podría ser el modelo llamado flujo de dos mercancías [14]. Este modelo propone pasar el problema a un problema de flujo en redes. Se crean unas mercancías ficticias p y q , que se recogen y dejan del almacén respectivamente. Mientras que la p disminuye en una unidad cada vez que se visita un nodo, la q aumenta. Con esto se pueden modelar las restricciones de precedencia de

forma que, tomando de nuevo el ejemplo de A y B, el número de unidades de p en el nodo A debe ser una unidad menor que en el nodo B, y a la inversa con la q.

Este modelado, sin embargo, es genérico, y necesita ser adaptado a cada problema concreto, con las dificultades que eso conlleva.

En el TSP no se contemplan restricciones de precedencia pero, en caso de necesitar incluirlas, el modelado es igualmente aplicable, ya que no afecta a las restricciones existentes, sino que se deben añadir nuevas.

Capacidad del vehículo

En este caso lo que se impone es que la capacidad del vehículo sea, o no, infinita. En caso de encontrarnos ante la situación de que la capacidad es infinita, estaremos modelando un TSP en lugar de un VRP. Si la capacidad es finita, el único cambio necesario es añadir una restricción adicional como la siguiente:

Sea v_i el volumen del objeto i incluido en el camión, sea O el conjunto de objetos incluidos para el reparto y sea C la capacidad total del camión, se tiene que:

$$\sum_{i \in O} v_i \leq C$$

Grafo de ciudades

En cuanto al grafo de ciudades nos podemos encontrar en dos situaciones, que sea simétrico o asimétrico. En caso de que sea asimétrico lo que se tendrá es que $c_{ij} \neq c_{ji} \forall i, j \in V, i \neq j$ siendo V el conjunto de nodos y c_{ij} el coste de ir del nodo i al nodo j .

Si se considera que el grafo siempre es completo (todos los nodos están conectados con todos), no hay ningún cambio que hacer, simplemente el vector de costes incluirá de manera diferenciada el coste de ir de i a j y el de ir de j a i .

En el caso del TSP, el cambio es el mismo, ninguno. Ya que lo único que varía es la matriz de costes entre nodos que pasa de ser una matriz simétrica a una matriz asimétrica.

Métodos de resolución

En esta sección se introducen los planos teóricos de las rutinas de cálculo de soluciones para nuestro problema TSPTW. Tal y como se especifican en los requisitos del trabajo, se han desarrollado unos enfoques de algoritmos exactos y algoritmos heurísticos.

Como ya vimos en temas anteriores, los algoritmos exactos calculan la solución óptima aunque tienen una demora considerable en tiempo debido a la complejidad del problema. Los algoritmos heurísticos nos dan una solución que consideremos como suficiente buena. Estos algoritmos se suelen aplicar a cantidades de datos suficiente grandes como para ser inabarcables con enfoques exactos y que no requieran una precisión milimétrica en la solución. Como ventaja, obtenemos una solución en tiempos reducidos.

Método exacto

El método exacto considerado para la resolución de este problema es una simplificación del planteado en [15]. Este método se basa en una resolución de programación dinámica. La simplificación define el coste de una arista como el tiempo que se tarda en recorrer el camino de nodo a nodo. Esta simplificación hace que no podamos calcular la función objetivo como la minimización de distancia u otra función de coste. Aunque aplicado a nuestro problema, con ello basta.

Consideremos la red grafo del mapa $G = (N, A)$ donde $N = \{1, \dots, n\}$ son el conjunto de nodos (clientes) del grafo, todos ellos unidos por aristas. Se considera G un grafo completo y dirigido, donde un nodo tiene $n-1$ nodos de salida y $n-1$ nodos de entrada. En caso de que dos aristas no deban estar conectadas explícitamente, puede asignársele un peso ∞ .

El nodo $1 \in N$ y el nodo $n \in N$ son los nodos almacén (en este caso, el mismo almacén virtualmente). Los nodos almacén están conectados a todos los demás nodos del grafo

con igual distancia (son el mismo nodo), aunque no están conectados entre sí, para evitar soluciones no factibles del problema.

Un vehículo partiendo de un nodo i a un nodo j no solo tendrá que tener en cuenta el tiempo de viaje t_{ij} entre ambos, sino también las ventanas de tiempo del destino $[a_j, b_j]$ las cuales marcan los momentos a partir del cual el cliente puede ser visitado y a partir del cual no puede visitarse más, y s_i como el tiempo de servicio en el que permanecemos en el nodo i .

Queremos encontrar una solución en la que se visite cada cliente exactamente una vez, además de cumplir las restricciones de las ventanas de tiempo de cada cliente y minimizando el tiempo en el que el vehículo visite a todos los clientes.

Sea definido N' como los nodos $N - \{n\}$, se considera $F(S, i)$ como el mínimo tiempo en el que podemos visitar a i en un camino que comienza en 1, pasando por cada nodo del subconjunto $S \subseteq N'$ exactamente una vez y terminando en el nodo $i \in S$.

Se considera entonces la llamada recursiva como:

$$F(S, j) = \min\{F(S - \{j\}, i) + t_{ij} \mid \text{restricciones válidas}\}$$

donde

- i y j son respectivamente los nodos origen y destino.
- las restricciones que se aplican son las descritas a continuación.

Para cada par de nodos ij en los que se calcule un viaje, el tiempo t_j debe depender, evidentemente, del tiempo t_i en el que visitamos el nodo anterior. Además, también debemos tener en cuenta de que si el vehículo llega al destino antes del inicio de su ventana de tiempo, debe esperar. En este caso:

$$t_j = \max\{t_i + s_i + t_{ij}, a_j\}$$

Dado t_i como el tiempo en el que el vehículo visita el nodo i y. Las restricciones son:

$$a_i \leq t_i \leq b_i$$

$$t_j \geq t_i + s_i + t_{ij}$$

El caso base ocurre cuando S es un subconjunto de dos elementos. Estos dos elementos serán el nodo de inicio (almacén) y otro nodo. Queda denotado como:

- $F(\{1, j\}, j) = a_1 + s_1 + t_{1j}$ cuando existe una arista $(1, j) \in A$.
- $F(\{1, j\}, j) = \infty$ en otros casos.

Finalmente, la llamada inicial al método recursivo será $F(N, n)$.

El método de Dumas supone una mejora frente a un enfoque de fuerza bruta. Revisemos la fórmula de recursividad:

$$F(S, j) = \min\{F(S - \{j\}, i) + t_{ij} \mid \text{restricciones válidas}\}$$

El cálculo de una solución se basa en el cálculo de sus subsoluciones mediante recursividad, pero sólo en aquellos casos en que no violen sus restricciones. Es decir, mediante esta comprobación de restricciones, podemos ir acortando el espacio de búsqueda a medida que vamos reconstruyendo las soluciones. El método Dumas devuelve siempre la solución óptima.

La expresión en pseudocódigo es la siguiente:

```
F (S , j):
    si |S| = 2:
        <- caso_base(S, j)
    si no:
        s = S - {j}
        soluciones = []
        por cada i en s:
            sol = F(S,i)+t_ij
            si es factible(sol):
```

```
soluciones <- sol
<- min (soluciones)
```

No obstante, la complejidad de este problema hace que sea NP-Completo. Es decir, no es un método de resolución escalable. En la sección [Resolución práctica: Experimentación](#) veremos con detalle este aspecto.

Método heurístico

El método exacto implementado se ha basado en un artículo que propone un algoritmo heurístico en 2 fases [16]. El método trata de construir una primera solución inicial al problema, que no tiene por qué ser óptima (este problema ya es NP-Completo) y luego intentar mejorarla durante un número de iteraciones para intentar llegar a la óptima.

En el problema lo que tendremos son un conjunto de clientes a visitar, cada uno de los cuales tendrá una ventana de tiempo disponible en la que podrá atender al viajante. Por ello, cada uno de los clientes deberá ser visitado dentro de su ventana de tiempo.

Primero se presentará la estructura general del algoritmo para posteriormente ir profundizando en cada una de las partes del mismo.

Algoritmo 1: Two Phase Heuristic

```
1 Input: IterMax
2 Output: X*
3 iter <- 0
4 while iter < IterMax do
5   X <- BuildFeasibleSolution()
6   X <- GVNS(X)
7   X* <- Better(X, X*)
8   iter++
9 end
```

Como se muestra en el código Algoritmo 1, el proceso general recibe como entrada el número máximo de iteraciones del algoritmo y devuelve una solución del problema.

En la línea 3 se inicializa el número de iteraciones a 0 y entre las líneas 4 y 9 se itera sobre los procedimientos del algoritmo para mejorar la solución, hasta que se alcance el número de iteraciones máximo.

Ahora se entrará en detalle dentro del proceso de la línea 5 *BuildFeasibleSolution*. Este proceso intenta generar una solución factible del problema, a partir de una generada aleatoriamente.

El proceso se muestra en detalle en el código Algoritmo 2.

Algoritmo 2: Constructive Phase

```
1 Output: X
2 repeat
3   level <- 1
4   X <- RandomSolution()
5   X <- Local1Shift(X)
6   while X is infeasible and level < levelMax
7     X' <- Perturbation(X, level)
8     X' <- Local1Shift(X')
9     X <- better(X, X')
10    if X is equal X' then
11      level <- 1
12    else
13      level++
14  end while
15 until X is feasible
```

En el código Algoritmo 2 se presenta la fase constructiva la cual comienza, como de costumbre, iniciando el nivel a 1. En la línea 4 se genera una solución aleatoria que, posiblemente, será inválida. En la línea 5 se hace una primera búsqueda entre todos los vecinos de hacer un movimiento 1-shift. Luego, entre las líneas 6 y 14 se itera mientras

la solución sea inválida y no se haya alcanzado el nivel máximo de aleatoriedad. Durante ese proceso se van aplicando perturbaciones y búsquedas 1-shift sobre la solución, siempre quedándonos con las mejores soluciones que se van obteniendo.

A continuación se mostrará la subrutina llamada GVNS, que se corresponde con la fase de optimización.

Algoritmo 3: GVNS

```
1 Input: levelMax
2 Output: X
3 level <- 1
4 X <- VND(X)
5 while level <= levelMax
6   X' <- Perturbation(X, level)
7   X' <- VND(X')
8   X <- better(X, X')
9   if X equal X' then
10     level <- 1
11   else then
12     level++
13 end while
```

El código mostrado en Algoritmo 3 comienza de nuevo con el nivel a 1, se aplica el proceso VND en la línea 4 a la solución actual (ya factible) y, mientras no se alcance el nivel máximo, se repite el proceso de la línea 6 a la 12. En dicho proceso se van aplicando perturbaciones y VND a la solución obtenida para ir mejorándola.

Por último, se presenta el proceso VND, que se usa en la fase constructiva para ir moviendo la solución hacia el mínimo, para intentar llegar a la óptima. Dicho código se muestra en Algoritmo 4.

Algoritmo 4: VND


```

1 Input: X
2 Output: X
3 X' <- null
4 repeat
5     X <- Local1Shift(X)
6     X' <- X
7     X <- Local2Opt(X)
8 until X = X'

```

El proceso VND se limita a buscar las mejores soluciones en el vecindario resultante de aplicar todos los posibles 1-shift para, posteriormente variar ese vecindario tomando la mejor solución posible haciendo los movimientos 2-opt de la solución actual. Este proceso se repite hasta que las dos búsquedas den la misma solución, es decir, cuando la solución X ya no pueda mejorar.

Una vez que se tienen las partes del algoritmo, se pueden estudiar y detallar los métodos concretos. En primer lugar se va a introducir lo que es el 1-shift.

Este movimiento se basa en tomar un cliente y adelantarlo o atrasarlo a otra posición. El resultado es que los clientes a partir de esa posición quedarán desplazados. Un ejemplo de este movimiento se muestra en la siguiente Figura.

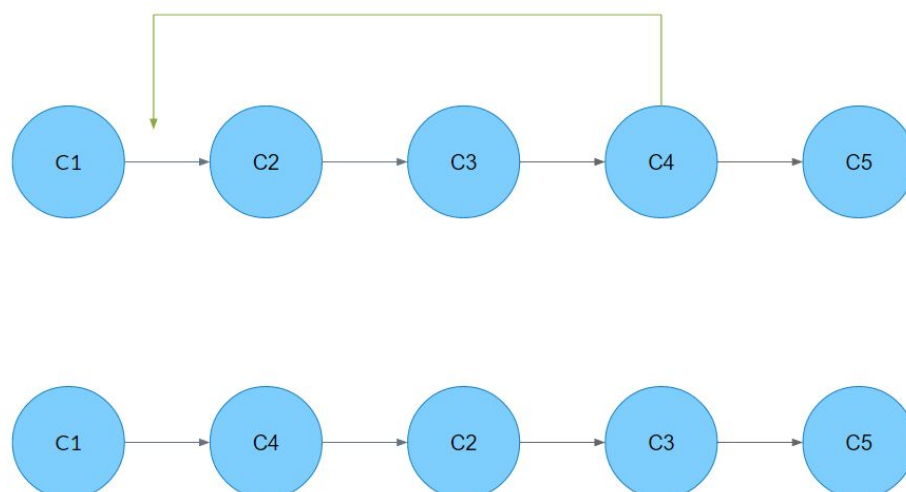


Figura 3: Movimiento 1-shift

Se observa que el cliente 4 ha sido movido a la posición 2, siendo los clientes a partir de dicha posición desplazados.

La búsqueda Local1Shift se basa en hacer todos los posibles movimientos de este tipo de todos los clientes para así quedarse con aquel movimiento que de la mejor solución. Esta búsqueda se usa tanto en la fase de construcción como en la fase de mejora. En función de la fase en la que se use, el proceso de búsqueda variará ligeramente.

El otro movimiento que se usará durante el algoritmo es el 2-opt. Este movimiento se basa en tomar el ciclo actual, dividirlo en 2 y reconectar una de las partes de forma inversa. En la siguiente figura se muestra un ejemplo de dicho movimiento.

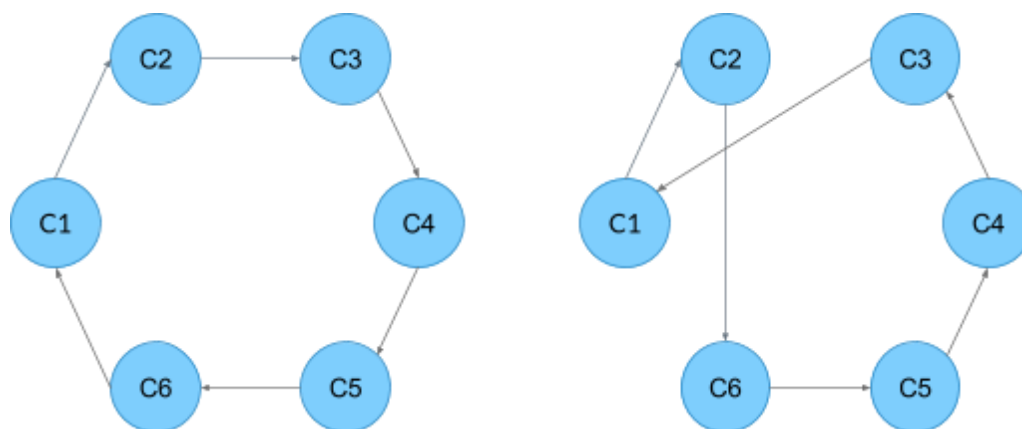


Figura 4: Movimiento 2-opt

Como se observa, se tiene el ciclo de la derecha C1-C2-C3-C4-C5-C6. Este ciclo se divide en dos distintos, el que contiene a C1 y C2 y el que contiene a todos los demás. Una vez se tiene esta división se reconectan, pero de forma inversa, C2 con C6 y C3 con C1. Debido a que el orden del ciclo C3-C6 ha cambiado, es necesario invertir las direcciones del mismo. Es importante remarcar que el cliente C3 ha pasado de estar en la 3era posición a estar en una más adelantada. De esto nos serviremos más adelante para acortar la búsqueda.

La búsqueda 2-opt intenta realizar todos los cambios posibles de todos los clientes para quedarse con aquel cambio que produzca la mejor solución.

Local 1-shift en fase constructiva

Durante la fase constructiva, la función objetivo que debe cumplir la solución será ligeramente diferente a la de la fase de mejora. Debido a que en la fase constructiva buscamos una solución factible, sin importar su coste, se debería poder cuantificar lo inválida que es una solución para que, si eso llega 0, la solución sea válida.

Para ello se toma, por cada cliente, la diferencia entre el tiempo en el que se visita a dicho cliente y el final de su ventana de tiempo, si ese resultado es positivo, el tiempo en el que se visita es mayor a cuando termina su ventana y, por tanto, ese cliente es inválido. Por ello, la nueva función objetivo será:

$$\text{minimize } \sum_{i \in C} \max(0, \beta_i - b_i)$$

Siendo β_i el tiempo en el que se visita al cliente i y b_i el final de la ventana del cliente i .

Como buscamos acercarnos a una solución factible, en cuanto encontremos una solución que mejore a la actual, esa será la que devolvamos, esto se hace para evaluar el mínimo número de soluciones vecinas, en la búsqueda de 1-shift. Por ello se comenzarán a explorar esas soluciones que tengan más posibilidades de mejorar a la actual.

Los movimientos 1-shift se pueden dividir en 2 grupos, movimientos hacia adelante y movimientos hacia atrás. Además, los clientes de la solución se pueden dividir a su vez en dos subconjuntos, clientes válidos y clientes inválidos. Los clientes inválidos serán aquellos cuyo momento de visita sea posterior a su final de ventana. Por tanto, los movimientos se podrán dividir en 4 grupos:

- Movimientos hacia atrás de clientes inválidos.
- Movimientos hacia delante de clientes válidos.
- Movimientos hacia atrás de clientes válidos.

● Movimientos hacia delante de clientes inválidos.

Cada uno de estos movimientos tendrá un efecto distinto sobre el cliente movido. En general se puede afirmar que cualquier movimiento de un cliente hacia adelante puede hacer que sea inválido si era válido o dejarlo siendo válido, y si ya era inválido lo seguirá siendo. Un movimiento hacia atrás, sin embargo, dejará a los clientes válidos como tal y a los inválidos podrá pasarlos a válidos o dejarlos igual.

Atendiendo a esto el primer vecindario que se explorará por ser más propenso a contener soluciones mejores es el de movimientos hacia atrás de clientes inválidos. Después de este los movimientos hacia delante de clientes válidos seguido de movimientos hacia atrás de clientes inválidos y, por último, movimientos hacia delante de clientes inválidos.

Local 1-shift en fase de mejora

En este caso la solución que tenemos ya es factible, por lo que en este caso la función objetivo ya es la clásica, en la que se minimiza el coste de la ruta. Ahora no se aceptarán soluciones que sean inválidas, aunque mejoren el coste de la ruta. El coste de la ruta puede reevaluarse con complejidad $\Theta(1)$ ya que consiste en eliminar 3 arcos y añadir otros 3. Es por ello que primero se evaluará el coste de la nueva ruta y, sólo en el caso de que mejore a la actual, se recalculará la validez de la misma, ya que la complejidad de esto es mayor.

Todos los clientes entre la posición actual y la nueva posición del cliente afectado por el movimiento pueden sufrir cambios en el momento en el que son visitados, por lo que habrá que recalcularlo. Además, los clientes situados después de los clientes afectados por el movimiento, también podrán sufrir cambios en sus tiempos de visita, por lo que también deberán ser recalculados. Sin embargo, todos los demás clientes no sufrirán cambios en su momento de visita y, como la solución del a que se parte es factible, no hay que volver a calcularlos.

Por último, de entre los clientes que se deben reevaluar, si se llega a uno cuyo momento de visita es el mismo que en la solución anterior, todos los siguientes no deben ser reevaluados, ya que seguirán siendo válidos y su momento de visita no cambiará. Es importante remarcar que este “atajo” puede tomarse siempre y cuando el cliente que se visite en el mismo momento que en la solución anterior se encuentre en una posición posterior a la del cliente movido, y a que en caso contrario se estaría asumiendo que el cliente movido se visita en el mismo momento que en la solución anterior, lo cual no tiene sentido.

Local 2-opt en fase de mejora

De nuevo este movimiento se puede hacer con complejidad $\Theta(1)$, debido a que hay que eliminar dos aristas e introducir otras dos. Esto en el caso de que se esté tratando el problema simétrico y que el coste de ir de i a j sea el mismo que el coste de ir de j a i . Si se toma el problema asimétrico, como era nuestro caso, este movimiento no tiene complejidad constante, ya que es necesario recalcular el coste del ciclo que se invierte. En cualquier caso, recalcular el coste será igual o menos costoso que recalcular la validez. Es por ello que, al igual que en la búsqueda local 1-shift, primero se evalúa el coste y luego la validez.

Como se dijo anteriormente, en el movimiento 2-opt se puede interpretar que uno de los clientes avanza un número de posiciones. Si en un movimiento 2-opt un cliente pasa a ser inválido, no tiene sentido seguir probando movimientos en los que dicho cliente se vaya a posiciones aún más posteriores, pues seguirá siendo inválido, es por ello que se puede cortar la búsqueda en ese punto y ahorrar operaciones.

Para el cálculo de la validez se asumen los mismos casos que en el movimiento 1-shift para optimizar las operaciones.

Perturbation

La perturbación tiene como objetivo salir de los mínimos locales. Por ello, este método se limitará a hacer tantos movimientos 1-shift aleatorios como indique la variable level que recibe.

Random Solution

En el artículo se establece que, aunque buscar una primera solución factible ya puede considerarse un problema NP-Completo, si se ordenan los clientes acorde al comienzo o final de sus ventanas de tiempo, el procedimiento es capaz de encontrar rápidamente una solución factible. Sin embargo, el hecho de introducir una solución inicial aleatoria se hace para que haya cierta diversidad en la solución del algoritmo, pudiendo así llegar a no una sola solución en todos los casos, sino que se pueda llegar a distintos puntos del espacio de soluciones.

Resolución práctica

Hemos implementado al completo los dos algoritmos propuestos en la investigación de métodos de resolución. El objetivo es el que los dos métodos puedan ser completamente integrados en el sistema Acme-Supermarket, alternando entre ellos según las circunstancias lo requieran.

Método exacto

La implementación del método exacto está fundamentada en tres ficheros, principalmente. Estos ficheros son

- Dumas.py: Gestiona las llamadas recursivas y el flujo del algoritmo de Dumas.
- Graph.py: Gestiona las utilidades de grafos, vértices y aristas. Son operaciones como, por ejemplo, hallar la distancia entre dos vértices, hallar el tiempo de servicio de un vértice, etc.
- Solution.py: Gestiona un estado o solución del problema de Dumas. Una solución está definido como un objeto que contiene una lista de nodos a visitar (comenzando por el nodo almacén) y una lista de tiempos en los que se visita cada nodo. Además dispone de métodos de utilidad para hallar el coste total o comprobar si la solución cumple las restricciones del problema.

Existen otros ficheros en el proyecto, más relacionados con la lectura de datos desde diferentes formatos que no serán explicados con detalle en esta sección.

Gracias a esta separación en métodos de utilidad, el aspecto de la clase Dumas es muy simple. Vamos a explicar con detalle los métodos más importantes.

```
def f(self, S, j)
    if len(S)==2:
        return self.base_case(S, j) #Devuelve caso base
    else:
        s = list(S)
        s.remove(j)
        feasible_solutions = []
        for i in s:
            if i!=self.graph.start: # Para todos aquellos i!=almacen
```

```

        solution = self.f(s, i)    # llamada recursiva
        if solution is None;# Si no hay solucion:
            continue                # Pasa al siguiente
        new_solution = self.join_solution(solution, i)
        if new_solution.is_feasible():
            feasible_solutions.append(new_solution)
    return self.choose_sol(feasible_solutions) #Devuelve la mejor

```

El método *f()* se desempeña la función del pseudo código expuesto en la sección [Métodos de resolución: Método exacto](#). No hay más que comentar.

```

def join_solution(self, solution, vertex):
    t_i = solution.last_time()
    t_ij = self.graph.time_edge(solution.last_vertex(), vertex)
    a_j = vertex.time_window[0]
    t_j = max([t_i + t_ij, a_j])    # cálculo del nuevo  $t_j$ 
    solution.add_vertex(vertex, t_j) # Añadir [nodo, tiempo] a la solución
    return solution

```

El método *join_solution()* se encarga de calcular el tiempo de llegada del nuevo nodo *j* y añadir este par de nodo y tiempo a una nueva solución. Lo siguiente será comprobar que esta nueva solución construida es factible.

```

def choose_sol(self, solutions):
    if len(solutions)==0:                # Si no hay solución, NULL
        return None
    else:
        return min(solutions, key=attrgetter('cost')) # En caso contrario,
mínimo coste

```

El método *choose_sol()* se encarga de elegir la mejor solución de un conjunto de soluciones factibles. La mejor solución se considera aquella de mínimo coste. En caso de que no haya soluciones factibles, se devuelve Nulo. Luego este valor Nulo es tratado de forma especial en la función recursiva.

Si desea probar esta ejecución hemos preparado un archivo capaz de calcula rutas para pequeños ejemplos obtenidos en internet¹. Para ejecutar un ejemplo seguir los pasos siguientes:

- En consola, en la carpeta raiz, ejecutar:

¹ <http://lopez-ibanez.eu/tsptw-instances> (Fecha de consulta: 6 de Mayo de 2016).

- `python3 main_exact.py -s example`
- Especificar el número de clientes cuando la rutina lo pregunte.

Método heurístico

Para el método heurístico se han hecho una serie de clases que modelan las diferentes partes del problema. La idea es que con una sola llamada a la clase que modela el problema se ejecute todo el algoritmo devolviendo la mejor solución encontrada.

Los archivos empleados para modelar el problema son los siguientes:

- `Customer.py`: Clase que modela a un cliente. Cada cliente tiene una ID, la columna de la matriz de distancias con la que se corresponde, el comienzo y fin de su ventana y el momento en el que se visita.
- `Graph.py`: Clase que modela el grafo del problema. Contiene la matriz de tiempos y distancias (en función de lo que se quiera optimizar) y una serie de métodos auxiliares para acceder a los costes entre dos clientes.
- `Solution.py`: Clase que modela la solución del problema. Tiene una lista de clientes en orden de visita, una lista que indica para cada cliente si es válido o no, y los costes asociados a la solución.
- `TPH.py`: Clase que modela al algoritmo en sí. Posee los métodos para ejecutar el algoritmo y los parámetros del mismo, como el nivel máximo, el número de iteraciones, el grafo de clientes...
- `Constructive.py`: Aquí están contenidos todos los métodos necesarios para la fase constructiva, véanse `Perturbation` y `Local1Shift` (en su versión para la fase constructiva).
- `VND.py`: Aquí se encuentra el proceso VND, es decir, el `Local1Shift` de la fase de mejora y el método `Local2Opt`.

Cada uno de los métodos implementados es una transcripción del pseudocódigo mostrado en la sección [Métodos de resolución: Método heurístico](#) a código python, por lo que es innecesario volver a mostrarlo aquí. Cualquier detalle sobre la

implementación se encuentra en los ficheros de código fuente aportados junto a este documento.

Experimentación

En esta sección hemos puesto a prueba las implementaciones de los dos métodos investigados. Hemos realizado pruebas con los tiempos de cálculo de ruta y la calidad de las rutas calculadas.

Tiempo

En cuestiones de tiempo, se han ejecutado los dos métodos para los mismos problemas con los mismos tamaños (mismo número de clientes). Estos problemas han adquirido sus datos de la base de datos de Acme-Supermarket, cuya estructura explicamos en la sección [Integración](#).

Las pruebas han arrojado los siguientes resultados:

Tamaño del problema	Tiempo exacto Dumas	Tiempo heurístico TPH
5	0.003 segundos	1.5 segundos
6	0.022 segundos	2.14 segundos
7	0.15 segundos	3.44 segundos
8	1.4 segundos	4.14 segundos
9	12.1 segundos	5.41 segundos
10	1 min 56 segundos	6.16 segundos
11	22 min 33 segundos	8.74 segundos
12	No calculado	10.27 segundos
13	No calculado	12.06 segundos
14	No calculado	13.82 segundos
15	No calculado	16.86 segundos

Tabla 1: Comparación de las pruebas de tiempos para los dos métodos.

Los tiempos para tamaños de problema más grandes en el algoritmo de Dumas no han sido calculados debido a que se esperaban unos tiempos de espera demasiado grandes para este trabajo.

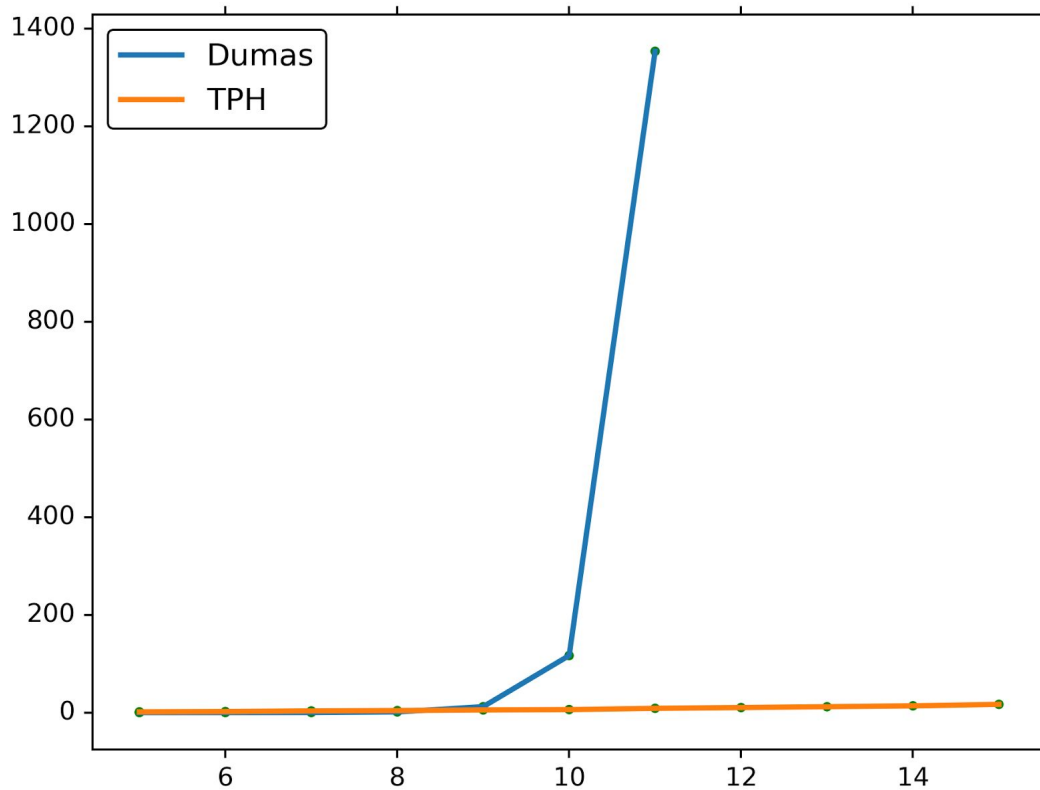


Figura 5: Comparación de las pruebas de tiempos para los dos métodos (x: Tamaño del problema, y: Tiempo en segundos).

Por la tabla y la figura, apreciamos fácilmente cómo el tiempo de cálculo del algoritmo de Dumas escala muy rápidamente con el tiempo. Esto es un resultado esperado, puesto que al ser un método exacto, la complejidad resulta NP-Completa. El tiempo del algoritmo heurístico también aumenta pero a un ritmo mucho menor, de forma lineal.

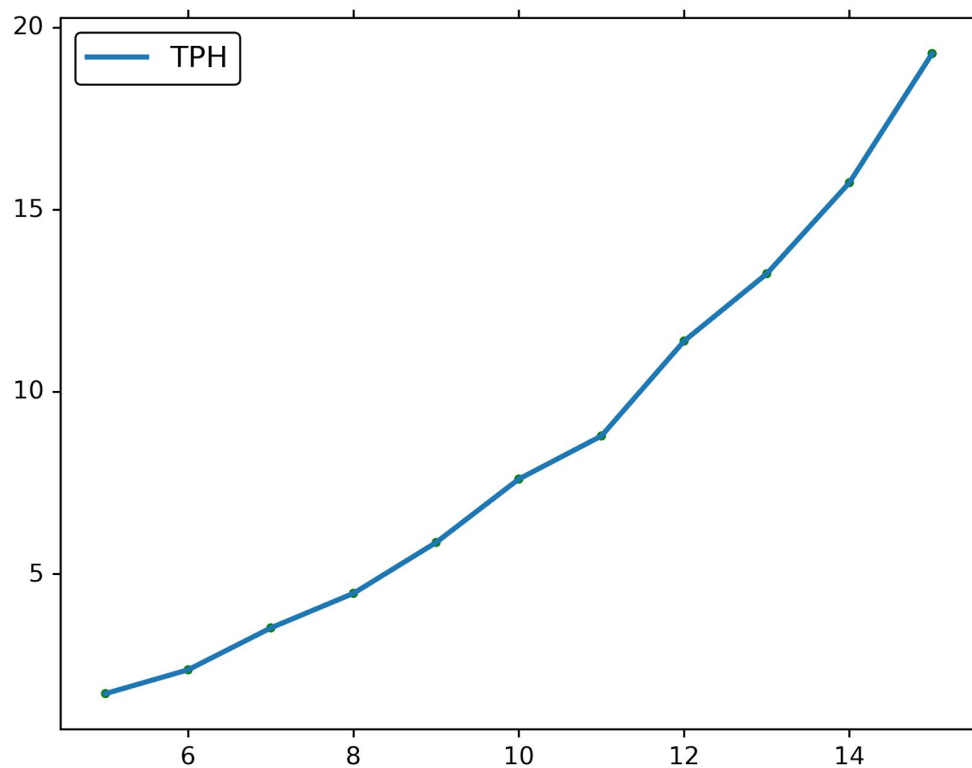


Figura 6: Tiempos de cálculo de ruta de algoritmo heurístico (x: Tamaño del problema, y: Tiempo en segundos).

Calidad de la solución

De igual manera que con el tiempo, hemos enfrentado los resultados de ambos métodos para los mismos problemas.

Clientes	Heurístico (coste)	Exacto (coste)	Calidad heurístico(%)
5	51233	51233	100
6	52192	51454	98
7	52920	52034	98
8	53878	52033	96
9	54535	52033	95
10	54535	52033	95

11	55492	52033	93
12	56516	No calculado	
13	56516	No calculado	
14	58276	No calculado	
15	59022	No calculado	

Tabla 2: Comparación de calidad de la solución de ambos métodos

La calidad de la solución de esta tabla está expresada como el coste total de la solución final aportada por cada algoritmo para cada tamaño de problema. Vemos cómo el algoritmo exacto siempre da una solución igual o mejor que la aportada por el heurístico. A medida que el tamaño de problema aumenta, la solución aportada por el heurístico va empeorando. Sin embargo, el orden de calidad para unos valores cercanos a los que calcularemos en Acme-Supermarket son excelentes, del orden de un 90%. Por tanto, el método heurístico supone una buena alternativa al método exacto.

Conclusiones

Como resultado de la experimentación nos resulta interesante utilizar ambos métodos de resolución según el tamaño del problema. Hemos decidido que para problemas con 8 clientes o menos, usaremos el método exacto Dumas; y para mayores problemas usaremos el heurístico TPH.

La razón se debe a que, en pequeños problemas, los tiempos del método Dumas son mejores y la solución es óptima. A partir de un tamaño de 9, los tiempos del método Dumas son inaceptables para un sistema de estas características, por lo que sacrificamos calidad de la solución por un tiempo adecuado con el método heurístico TPH.

Integración

La integración con Acme-Supermarket ha sido realizada mediante las tecnologías NodeJS, AngularJS y MongoDB.

De forma similar a como ya realizamos la integración en la tarea 03: Decodificación de códigos de barras, vamos a integrar estas nuevas funcionalidades desde un nuevo servicio que adherimos a nuestro sistema. Esta separación en servicios está pensada para que el impacto del cálculo de estas operaciones no tengan consecuencias graves en la estabilidad del sistema principal. Además, esto ayuda a que este servicio pueda ser desplegado en una máquina diferente que el resto de servicios.

Este nuevo servidor será llamado “routes-server” y estará montado mediante Javascript y NodeJS. Los accesos a los datos de la base de datos se realizará mediante el módulo Mongoose de NodeJS para MongoDB. Este servicio debe ser capaz de comunicarse entre el lenguaje Javascript y Python (en el que está implementada la tarea del cálculo de rutas). Para ello, hemos utilizado el módulo `child_process` de *NodeJS*. Este módulo permite crear procesos hijos de manera que se creen canales de comunicación entre el padre (*NodeJS*) y el proceso hijo (*Python*), pudiéndose comunicar mediante una transmisión de datos de manera no bloqueante.

Este nuevo servidor tiene tareas que no son triviales. Se diferencian dos tareas principales, expuestas en las siguientes subsecciones.

Cálculo de las rutas de hoy

Todos los días a las 01.00 AM comienza la ejecución de una tarea programada. Hemos implementado la programación de tareas mediante el módulo “node-schedule” de NodeJS.

Esta tarea busca en la base de datos todas las compras cuya fecha de entrega tuviera lugar el presente día. Para todas estas compras, extraemos los clientes que están implicados en dichas compras. Los clientes del sistema Acme-Supermarket tienen una serie de informaciones que se hallan guardadas en nuestra base de datos. Entre ellas se encuentran sus coordenadas geográficas. Podemos conseguir las coordenadas de todos los clientes a los que hay que repartir mercancías hoy.

Aquí entra en juego Google Maps. Google Maps provee de unos métodos de API para ofrecer funcionalidades y utilidades varias a desarrolladores. Nosotros usaremos Google Maps Distance Matrix API. Este método provee de unas matrices de distancias y tiempos entre coordenadas. Cada una de estas matrices están organizadas de la misma forma. Cada columna y fila corresponden a una localización concreta del mapa, de manera que cada celda corresponde a la distancia (o tiempo) recomendada que existe desde la localización i (fila) a la localización j (columna).

Para poder acceder a estas funcionalidades antes debemos registrar nuestra aplicación en el servicio de Google APIs, de manera que la empresa proveedora tenga constancia del uso que estamos dando de sus servicios.

Para registrar nuestra aplicación en Google APIs hay que seguir los siguientes pasos:

- Acceder a la consola de desarrolladores de Google².
- Pulsar en “Crear proyecto”. Introducir nombre del proyecto y pulsar “Crear”.
- Se nos redirigirá al administrador de la nueva aplicación. Entre nuestras opciones de API, pulsamos en “Google Maps Distance Matrix API”.
- Pulsamos en “Habilitar”.
- Pulsamos en “Credenciales” > “Crear credenciales” > “Clave de API” > “Clave de servidor” > “Crear” > “Aceptar”.
- A continuación, recibimos nuestra clave de API la cual será necesario usar en el desarrollo de nuestra aplicación.

A la hora de la verdad, no usamos directamente la API JSON de Google Maps, sino que usamos un preprocesador implementado para Python³. Este preprocesador nos permite obtener con sencillas funciones los resultados de las llamadas a las API de Google Maps.

Google Maps ofrece unas limitaciones para el uso de su API en su versión gratis:

- No se admiten más de 250.000 peticiones diarias.

² <https://console.developers.google.com/project> (Fecha de consulta: 4 de Mayo de 2016).

³ <https://github.com/googlemaps/google-maps-services-python> (Fecha de consulta: 4 de Mayo de 2016).

- No se pueden devolver unas matrices de distancias de más de 10x10 elementos.
- Debe haber una espera entre petición y petición de 10 segundos.

Estas limitaciones provocan que para aquellos casos que calculemos la ruta para más de 9 clientes (más un nodo almacén) necesitaremos realizar una estrategia diferente. La estrategia propuesta es la de realizar varias peticiones a pequeñas submatrices de la matriz de distancias que necesitamos. De esta manera, mediante varias peticiones podríamos completar una matriz de más de 100 elementos. Sin embargo, la limitación del tiempo entre peticiones sigue ahí, por lo que este proceso puede ralentizar varios segundos, incluso minutos. la obtención de los datos. En un caso real de Acme-Supermarket se propondría adquirir la versión premium de uso de esta API, pudiendo acceder a matrices grandes sin límites de cuota.

También son requeridas en estos problemas las ventanas de tiempo, rangos de momentos en los que un cliente puede ser visitado. Esto está implementado en Acme-Supermarket mediante los periodos de entrega. Al registrarse, los usuarios seleccionan cuando prefieren que les lleguen sus compras a casa. Existen tres opciones:

1. Turno de mañana (de 08.00AM a 14.00PM).
2. Turno de tarde (de 14.00PM a 20.00PM).
3. Ambos (de 08.00AM a 20.00PM).

De cara a una representación real de Acme-Supermarket, se han considerado todas las coordenadas de los clientes, como coordenadas del área de Sevilla y alrededores. El almacén es el nodo de inicio y fin de la ruta. Se ha considerado que el almacén será la Escuela Técnica Superior de Ingeniería Informática de la Universidad de Sevilla. Este almacén tiene una ventana de tiempo mayor (de 00.00AM a 23:59PM).

Una vez obtenidos los clientes y las distancias entre ellos. Se hace una valoración de la envergadura del problema. Normalmente, se usa un algoritmo exacto para obtener la solución óptima. Si la cantidad de clientes supera un umbral preestablecido, se considera que un algoritmo exacto resulta inadecuado, debido a que sobrecargaría el

servidor. En estos casos, se utiliza un algoritmo heurístico. Estos algoritmos exacto y heurístico, son los documentados anteriormente.

En algunos casos, puede ocurrir que el problema no tenga solución, es decir, que dadas las condiciones de ventanas de tiempo de los clientes y la cantidad de clientes para visitar, no encontremos ninguna configuración válida de ruta. En estos casos, debemos alterar el planteamiento de clientes que se nos plantea. Nosotros proponemos eliminar al último cliente que solicitó una compra para hoy. Este cliente es eliminado de los clientes a visitar y sus compras son pospuestas para repartir al día siguiente.

Una vez se calcula la solución final de estos algoritmos, se crea un objeto “ruta”. Este objeto almacena la fecha de cálculo, la ruta de clientes ordenador, y los momentos en que se visita cada cliente. Este objeto ruta es guardado en la base de datos de Acme-Supermarket.

El esquema final de la estructura de todo este sistema es el siguiente:

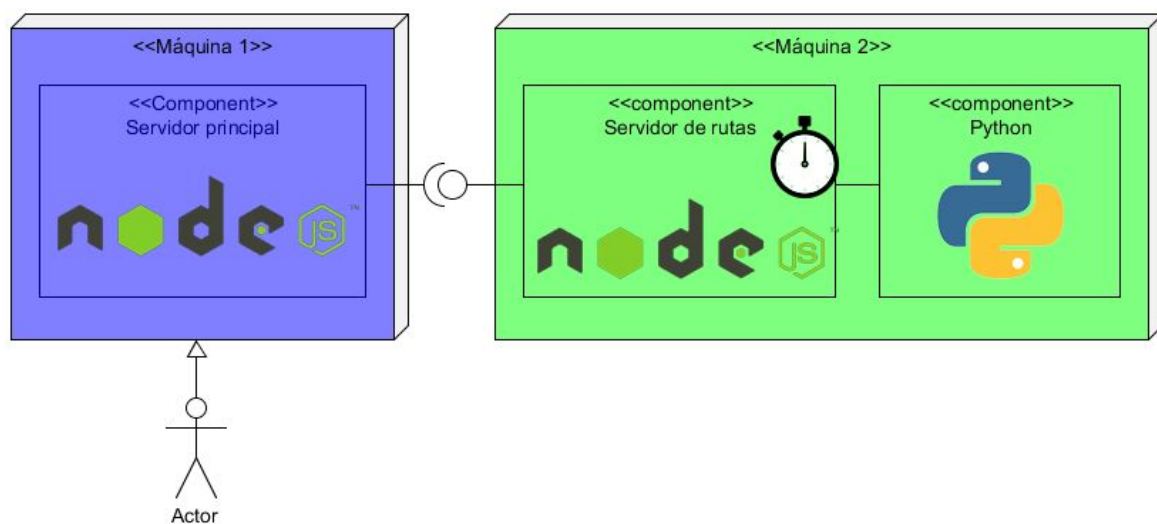


Figura 7: Esquema de estructura de los componentes del sistema.

Consulta de ruta de hoy

Esta es la parte visible al usuario de todas las funcionalidades implementadas en esta tarea. Cuando un administrador del sistema lo solicite, se le mostrará una vista detallada de la ruta de reparto para las compras de hoy.

El sistema solicita a la base de datos la ruta del día actual (precalculada) y es mostrada al usuario. Se desglosan qué clientes y pedidos hay que repartir y a qué hora. Estos datos se muestran ordenados según el orden de visita.

ACME

Inicio

Productos

Administración

Medios sociales

Business Intelligence

Mi cuenta


Inicio / Reparto de hoy

REPARTO DE HOY

Fecha de entrega

05/05/2016

ID Cliente	Cliente	Hora de entrega	Periodo de entrega
Almacen	--	--	--
7	German Vega	10:00	MAÑANA (9-14AM)
2	Sergio Gomez	10:10	AMBOS
6	Magdalena Nuñez	10:14	AMBOS
4	Consuelo Soler	10:16	AMBOS
5	Isabel Herrera	10:19	AMBOS
8	Luis Gomez	10:31	AMBOS
1	Daniel Diaz	16:00	TARDE (15-22PM)
3	Belen Carrasco	16:08	TARDE (15-22PM)
9	Margarita Medina	16:14	TARDE (15-22PM)
Almacen	--	--	--

 ENVÍOS RÁPIDOS DIRECTOS A TÍ


 FÁCIL CONTACTO CON PROVEEDORES

Figura 8: Vista de pantalla de la funcionalidad “Consultar la ruta de reparto de hoy”.

Conclusiones

Como conclusiones generales se puede decir que variantes del TSP hay tantas como se quieran inventar. No hay límites en cuanto a variantes del problema y siempre se le puede dar una vuelta de tuerca más al problema planteado.

Por otro lado, las soluciones y los algoritmos propuestos para resolver estos problemas son tanto o más complicadas que los propios problemas. Aunque las soluciones tengan bastante tiempo desde que se desarrollaron, sigue siendo un reto implementarlas y que den los resultados esperados tanto en calidad como en tiempo.

Debido a los tiempos de ejecución de este tipo de algoritmos, es una muy buena práctica disponer de dos algoritmos diferentes que respondan de manera distinta ante distintos números de clientes. Es por ello que el algoritmo exacto se ha decidido usar hasta un número de clientes en el que responde en un tiempo razonable y, a partir de dicho número de clientes, se pasa al heurístico.

Por último, el problema tratado es tan amplio y tan extendido que realmente hay empresas que pueden mantenerse de solucionar el problema de cálculo de rutas de reparto a empresas de logística.

Bibliografía

1. Toth, P., Vigo, D. (2002). The Vehicle Routing Problem. SIAM Monographs on Discrete Mathematics and Applications.
2. Laporte, G., Nobert, Y., Desrochers, M. (1985) Optimal routing under capacity and distance restrictions. Operations Research 33, pp. 1050-1073
3. Euchi, J. and Chabchoub, H. (2010). A hybrid tabu search to solve the heterogeneous fixed fleet vehicle routing problem. Logist. Res., 2(1), pp.3-11.
4. Kuo, Y. and Wang, C. (2012). A variable neighborhood search for the multi-depot vehicle routing problem with loading cost. Expert Systems with Applications, 39(8), pp.6949-6954.
5. Herrera, J.,Reissig. Comparing assignment algorithms for the Multi-Depot VRP. Dpto. Investigación Operativa, Instituto de Computación, Facultad de Ingeniería, UDELAR. [online-PDF] Disponible en <http://www.fing.edu.uy/inco/pedeciba/bibliote/reptec/TR0108.pdf> (Fecha de consulta: 19 de Abril de 2016).
6. Bartholdi, J.J., Platzman, L.K., Collins, R.L., Warden, W.H. (1983). A minimal technology routing system for meals on wheels. Interfaces 13 (3), pp. 1-8.
7. Dror, M., Ball, M.O., Golden, B.L. (1985). A computational comparison of algorithms for the inventory routing problem. Annals of Operations Research 4, pp. 3-23.
8. Kek, A., Cheu, R. and Meng, Q. (2008). Distance-constrained capacitated vehicle routing problems with flexible assignment of start and end depots. Mathematical and Computer Modelling, 47(1-2), pp.140-152.
9. Bektas, T. (2006). The multiple traveling salesman problem: an overview of formulations and solution procedures. Omega, 34(3), pp.209-219.
10. NEO (Networking and Emerging Optimization). (2016). VRP with Time Windows. Vehicle Routing Problem. [online] Disponible en <http://neo.lcc.uma.es/vrp/vrp-flavors/vrp-with-time-windows/> (Fecha de consulta: 21 de Abril de 2016).
11. The Travelling Salesman Problem with Time Windows (TSPTW) [online]. Disponible en

https://or-tools.googlecode.com/svn/trunk/documentation/user_manual/manual/tsp/tsptw.html (Fecha de consulta: 21 de Abril de 2016).

12. Handbook in OR & MS C. Barnhart and G. Laporte(Eds.),Vol. 14. Chapter 6, Vehicle Routing (2007)

13. Desrochers, M., Laporte, G. (1991). Improvements and extensions to the Miller–Tucker–Zemlin sub-tour elimination constraints. *Operations Research Letters* 10, 27–36.

14. Moon, C., Kim, J., Choi, G. and Seo, Y. (2002). An efficient genetic algorithm for the traveling salesman problem with precedence constraints. *European Journal of Operational Research*, 140(3), pp.606-617.

15. Yvan Dumas, Jacques Desrosiers, Eric Gelinas, Marius M. Solomon, (1995) An Optimal Algorithm for the Traveling Salesman Problem with Time Windows. *Operations Research* 43(2):367-371. <http://dx.doi.org/10.1287/opre.43.2.367> (Fecha de consulta: 4 de Mayo de 2016).

16. da Silva, R. and Urrutia, S. (2010). A General VNS heuristic for the traveling salesman problem with time windows. *Discrete Optimization*, 7(4), pp.203-211.

Anexo I: Carga de trabajo

Los informes detallados de las horas, tiempos y tareas de este proyecto pueden consultarse en los ficheros PDF adjuntos.

Miembro	Horas
Daniel de los Reyes	57h 38 minutos
Alejandro Sánchez	45h 18 minutos