ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE
TELECOMUNICACIÓN
UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Trabajo Fin de Máster

# Diseño y evaluación de algoritmos para VRP

# Design and evaluation of algorithms for VRP

AUTOR: LUIS CABALLERO ARNALDOS
DIRECTOR: JUAN JOSÉ ALCARAZ ESPÍN
CODIRECTOR: JOSÉ ANTONIO AYALA ROMERO
Septiembre / 2017

| | |
|---|---|
| **Autor** | Luis Caballero Arnaldos |
| **E-mail** | luiscaballeroarnaldos@gmail.com |
| **Director/es** | Juan José Alcaraz Espin |
| **E-mail directores** | Juan.alcaraz@upct.es |
| **Codirector/es** | José Antonio Ayala Romero |
| **E-mail del Codirector** | Jose_a_ayala@hotmail.es |
| **Titulo del TFM** | Diseño y evaluación de algoritmos para VRP |
| **Descriptores** | *NP-hard* ,VRP, *time-window, incompatible goods* |


**Resumen**

Los problemas VRP (Vehicle Routing Problems) son la generalización del problema del viajante (Travelling Salesman Problem), y consisten en encontrar un conjunto de rutas para K vehículos, tal que cada una de las ubicaciones sea visitada únicamente una vez, minimizando el coste total de las rutas. De este problema, ya de por sí computacionalmente complejo (NP-Hard), se han planteado diversas variantes. Entre ellas, algunas de las más comunes son el Capacitated VRP (CVRP), donde cada cliente demanda unas mercancías concretas y cada vehículo tiene una capacidad finita; el VRP con Time Windows (VRPTW).Además de ellas el proyecto intenta presentar una visión innovadora incorporando también restricciones de incompatibilidad de mercancías. Todo ello combinado con dataset reales de empresas del sector de la logística da lugar al también denominado R-VRP. Cada aplicación introduce características y parámetros específicos en la formulación del problema, dando lugar a nuevas restricciones, y requiriendo algoritmos desarrollados *ad-hoc* para cada caso. En concreto se va a utilizar un solver open source denominado Optaplanner, de cara a evaluar el rendimiento de la herramienta y comprobando su efectividad al resolver el problema en comparación con propuestas académicas concreta. Finalmente se observa como su rendimiento es realmente bueno aunque presenta ciertas restricciones a la hora de incorporar nuevas restricciones complejas al algoritmo como es el caso que nos atañe.


| | |
|---|---|
| **Titulación** | Máster Universitario en Ingeniería de Telecomunicación |
| **Departamento** | Tecnologías de la Información y las Comunicaciones |
| **Fecha de Presentación** | Septiembre de 2017 |

# Índice General

# Índice de Figuras y Tablas

# Capitulo 1. Introducción

## 1.1 Motivación del Proyecto

Empresas como Amazon o UPS, líderes mundiales en la logística y el transporte, basan gran parte de su modelo de negocio en algoritmos de optimización que les permiten reducir costes, mejorar el servicio y ser más competitivas. La computación de altas prestaciones, la algoritmia y el análisis de datos están resultando determinantes en cada vez más ámbitos económicos, y el transporte es un claro ejemplo de ello.

En concreto el sector de la logística y el transporte de mercancías es responsable aproximadamente del 3.1% del PIB en España según OTLE (Observatorio del transporte y la logística en España).En el último informe anual elaborado por dicho organismo (dependiente del Ministerio de Fomento), se destaca como esta contribución al PIB permanece estable en los años evaluados(2008-2014). También se destaca el crecimiento global del transporte de mercancías durante el año 2014 tanto en el ámbito internacional como en el nacional. La logística, según la definición del informe es "principal actividad de transporte". Pues bien, si estos datos macroeconómicos tienen gran repercusión en la economía global del país lo cierto es que su importancia aun es mayor en aquellas comunidades en las cuales se concibe este sector como estratégico. Es el caso de la Región de Murcia donde el sector de la logística y el transporte supone el 4.34% del empleo regional y aproximadamente un 4.14% del PIB regional. Por ello la Comunidad Autónoma posee gran interés en el desarrollo de "TICs compatibles con procesos vinculados al transporte y gestión de flotas" .Este último aspecto, se enmarca en uno de los ámbitos de negocio emergentes de la actividad prioritaria "lógica y transporte" definida en la "Estrategia de Investigación e Innovación para la Especialización Inteligente de la Región de Murcia", aprobada por el Consejo de Gobierno de la CARM en 2014.Sin duda alguna, el sector de la logística y el transporte de mercancías se concibe como uno de los motores principales de la economía regional.

Dentro del sector del transporte, una de las empresas más importantes a nivel regional, con una gran proyección nacional e internacional, es el Grupo Caliche. Poseen una flota superior a 100 vehículos, incluyendo vehículos propios y de terceros, con una facturación anual cercana a los 75 millones de euros. Debido a la propia envergadura de la empresa y a su tendencia expansionaría para los próximos años, plantean la posibilidad de optimizar el plan de rutas seguidas por los vehículos de la compañía. Por otro lado, una de las principales líneas de investigación en el campo de las ciencias de computación es el planteamiento y resolución de problemas de optimización. Se trata de resolver problemas clásicos conocidos cuyo interés ha ido en aumento en los últimos años debido al desarrollo de equipos de computación. Por ende este interés conjunto en resolver un mismo problema plantea este trabajo como un proyecto de I+D en cooperación público-privada.

Normalmente, desde el punto de vista de la investigación algorítmica, la mayoría de *surveys* y *papers* científicos no llegan a resolver problemas reales a los que se enfrentan empresas del sector. Los planteamientos más teóricos suelen abordar problemas que distan en gran medida de la complejidad de los problemas diarios que intentan resolver las empresas. Este hecho no suele estar relacionado con el desconocimiento o falta de información. En general suele estar relacionado con la intención de elaborar formulaciones genéricas que intenten resolver una gran cantidad de variantes del mismo problema de optimización. Eso, no siempre es posible. Dependerá del tipo de problema lo que implicará que sea más o menos complejo (teniendo un mayor o menor número de restricciones, formulando el problema de forma más o menos sencilla, etc.).

El hecho de disponer de datos reales, es un valor diferenciador de este proyecto una de las principales razones de ser. Algo que sin duda nos permitirán evaluar el problema de optimización aquí presente de una forma más concreta, precisa y por tanto intentando ofrecer un resultado más optimo.

## 1.2 Objetivos

Motivado por el hecho de trabajar con datos reales del sector de la logística, el trabajo presenta unos objetivos ambiciosos que pueden desglosarse en los siguientes:

.

1. Desarrollar algoritmos de búsqueda de soluciones eficientes del problema de enrutamiento de vehículos (VRP) en situaciones reales.

2. Abordar el problema VRP evaluando nuevas características respecto al problema clásico como:
    - La presencia de ventanas temporales tanto en los puntos de recogida como en los de entrega.
    - Las restricciones por capacidad de cada vehículo, que puede variar de un país a otro.
    - La variabilidad en el número de vehículos que pueden componer la flota cada día.
    - La incompatibilidad de algunas mercancías entre sí
    - La incertidumbre estadística en cuando a tiempos de desplazamiento, carga y descarga.
    - La presencia de eventos imprevistos, como atascos en carretera o averías, que puede implicar la modificación dinámica de las rutas.

3. Evaluar algoritmos y herramientas disponibles para resolución de problemas VRP así como sus variaciones.

4. Evaluar y comparar la eficiencia de los algoritmos propuestos en situaciones reales, recogiendo y analizando datos de rendimiento.
5.

## 1.3 Descripción del problema

El proyecto aquí presente aborda el reto de desarrollar un algoritmo que resuelva una variación del problema VRP, *vehicle routing problem* de forma eficiente y aportando un buen resultado. Se abordará una solución del problema clásico teniendo en cuenta restricciones de capacidad de vehículos, tiempos de espera y carga e incompatibilidad de mercancías entre otras. Podríamos decir que se tratará de solucionar una mejora de un problema tipo CVRPTW.

El problema VRP en su forma más simple consiste en una serie de clientes que demanda una cierta cantidad de un producto. Para abastecer a dichos clientes, se dispone de una flota de vehículos. Los vehículos disponibles deberán salir de un único punto o *depot*, proveer los bienes solicitados por los clientes y volver al punto de partida o deposito .Solamente un único vehículo podrá visitar a un cliente. Pues bien, el problema será encontrar el conjunto de rutas que pueden seguir los vehículos para atender a los clientes, minimizando los costes totales.

*VRP*, es uno de los problemas comúnmente conocidos, clasificado como *NP.hard* y con un espacio de soluciones de $n!$ para el mejor de los casos. Por ello una instancia grande del problema no puede ser resulta mediante un algoritmo exacto en un tiempo de computo finito razonable. Es necesaria la elaboración de los llamados algoritmos aproximados o heurísticos o bien una combinación de estos con otros algoritmos (metaheuristicos).

# Capitulo2. Fundamentos teóricos

## 2.1 Complejidad Algorítmica

Tan solo el hecho de abordar soluciones para el problema aquí planteado está directamente relacionado con la Teoría de la Complejidad. Normalmente tendemos a pensar que una solución exacta es sinónimo de eficiencia y de buenos resultados. Lamentablemente, esto no siempre es posible y por ello soluciones aproximadas a diversos problemas deben considerarse como buenas. Por otro lado, la posible apariencia similar entre un problema resuelto en tiempo polinomial con otro que no lo puede ser, puede llevarnos a una idea incorrecta acerca de la intratabilidad computacional de algunos problemas.

Es por ello que conviene realizar una breve reseña acerca de los tipos (clases) de problemas presentes en áreas del conocimiento como la optimización[1] pero también en otras como inteligencia artificial, combinatoria, lógica, etc. Esta información nos permitirá comprender en mejor medida el por qué de calificar nuestro problema VRP como *NP-hard* y las implicaciones que ello conlleva.

Previamente, es importante aclarar algunos aspectos referentes a la notación que se empleará para expresar la complejidad del algotimo que resuelve un determinado problema. Normalmente resulta de interés tener una estimación del comportamiento que puede presentar un determinado algoritmo ante *inputs* de gran tamaño similares a los de la vida real. Además sería útil poder hacerlo independientemente de la maquina que lo ejecute. Pues bien, para ello debemos observar que ocurre cuando la entrada del problema tiende a infinito, esto es, analizar su eficiencia asintótica. Para ello existen distintas notaciones ($O, \theta\ u\ \Omega$ entre otras) que permiten expresar cual es el comportamiento de las cotas asintóticas Para no realizar un extenso análisis exhaustivo al respecto ya que no es objeto del presente trabajo, únicamente ilustraremos la complejidad de un algoritmo a través de la notación $O$ también denominada big- $O$.Una definición formal de la citada notación podría ser:

Dada una función $g(n)$ , denotamos como $O\big(g(n)\big)$ al conjunto de funciones:

$$O\big(g(n)\big) = \left\{ \begin{matrix} f(n)\colon \exists_{C,n_0} > 0\ tal\ que, \\ 0 \le f(n) \le c \cdot g(n) \forall_n \ge n_0 \end{matrix} \right\} \quad \textbf{(1)}$$

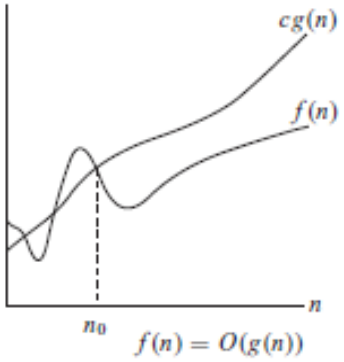



**Figura 1.** Ejemplo grafico notación $\boldsymbol{O}$

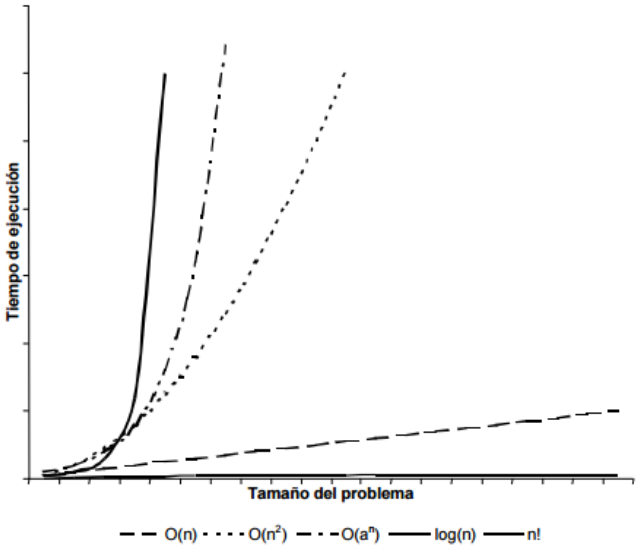
Esto quiere decir que para todo $n \ge n_0$, la función $f(n)$ presenta valores menores o iguales a $g(n)$ multiplicada por una constante $c$. La **Figura 1** ofrece una representación gráfica ilustrativa. Esta notación es por tanto una cota superior de la complejidad asintótica del algoritmo. En ocasiones $f(n)$ también suele considerarse como el número de instrucciones necesario para que un algoritmo resuelva problemas de tamaño $n$ . Dicha cantidad se establece asumiendo un modelo uniforme de coste en el cual se asume que todas las operaciones básicas necesarias para resolver el problema emplean un coste computacional constante. Naturalmente esto no es así pero resulta poco relevante a la hora de realizar medidas de la complejidad a grandes rasgos.
La **Tabla 1** indica, para distintos tamaños $n$ de un problema, el número de operaciones necesarias en

función de varias posibles complejidades típicas que puede presentar el algoritmo.

| $n$ | $\log_2 n$ | $\sqrt{n}$ | $n^2$ | $n^3$ | $2^n$ | $n!$ |
|---|---|---|---|---|---|---|
| **10** | 3.32 | 3.16 | $10^2$ | $10^3$ | $10^3$ | $3.6\ x\ 10^6$ |
| **100** | 6.64 | 10 | $10^4$ | $10^6$ | $1.27\ x\ 10^{30}$ | $9.3\ x\ 10^{157}$ |
| **1000** | 9.97 | 31.62 | $10^6$ | $10^9$ | $10^{301}$ | $4\ x\ 10^{2567}$ |
| **10000** | 13.29 | 100 | $10^8$ | $10^{12}$ | $10^{3010}$ | $2.8\ x\ 10^{35659}$ |

**Tabla 1.** Numero de operaciones para diferentes tipos de algoritmos y tamaños del problema

Para poner en valor estas cifras podemos compararlas con número estimado de átomos del universo ($\approx 10^{80}$) o edad estimada del universo[2].Una vez presentada la notación $O$ resulta interesante mostrar algunas de las complejidades de los algoritmos que tratan de resolver el problema VRP descrito de forma exacta. El representante del estado del arte en este aspecto es el algoritmo *Branch-and-Bound* El problema con restricciones de carga (*CVRP*, descrito en futuros capítulos), puede resolverse con algoritmos exactos del tipo *direct tree (branch-and-bound)* con simplificaciones AP *(Assignment Problem) o bien* con simplificaciones SST *(Shortest Spanning tree)* un orden de complejidad $O(n^3)$ [2.3]. Nuestro problema, incluirá más restricciones a parte de las de carga lo que implica algoritmos exactos más complejos. En efecto, el uso de algoritmos exactos resueltos con éxito se limita tamaños $n \approx 50$ .Problemas con *inputs* mayores solo pueden ser resueltos bajo ciertas condiciones. A parte de algoritmos tipo *direct tree* existen otros tipos como *dynamic programming (DP) o integer linear programming (ILP)*. [2.8] realiza un survey acerca de los algoritmos exactos aplicados a VRP.




Podemos observar como el hecho de necesitar algoritmos con orden de complejidad elevado (superior a $O(n^k)$) puede resultar inabordable desde el punto de vista computacional. La **Figura 2**, muestra cómo crece el tiempo de cómputo en función del tamaño del problema para distintos ordenes de complejidad, es decir, la tasa de crecimiento asintótica. Resulta evidente pensar en la necesidad de otra alternativa para resolver problemas como el que nos atañe. En este contexto aparecen los algoritmos heurísticos o aproximados, mencionados previamente y que serán detallados en profundidad en el siguiente epígrafe. Por otro lado, otra cifra representativa de la complejidad del problema que abordamos e íntimamente ligada al número de operaciones necesario para resolver el problema es, una estimación del espacio de soluciones potenciales del mismo. En **Tabla 2** mostramos espacio de soluciones de algunos benchmarks típicos sobre los cuales se prueban los algoritmos VRP.

| **Nombre** | ***Input*** | **Espacio de soluciones** |
|---|---|---|
| **A-n32-k5** | $n = 31, v = 5$ | $10^{46}$ |
| **A-n60-k9** | $n = 59, v = 9$ | $10^{104}$ |
| **Solomon_025_C101** | $n = 25, v = 25$ | $10^{34}$ |
| **Solomon_100_R201** | $n = 100, v = 25$ | $10^{200}$ |

**Tabla 2.**Espacio soluciones para distintos problemas VRP

*A-n-k* hace referencia al problema VRP con restricciones de carga. Por su parte el *benchmark* de

[2] La edad estimada del universo $\approx 10^{10}$años. Si un año tuviera 1000 días, 1 día 100horas, 1 hora 100 minutos, 1 minuto 100 segundos, un ordenador que ejecutase 1 operación cada nanosegundo, habría ejecutado $\approx 10^{28}$ operaciones desde que nació el Universo.

Solomon describe un problema VRP con restricciones temporales. En el Capitulo 7, aparecen descritos estos *datasets* de referencia que emplearemos más adelante para testar el algoritmo propuesto.

Pues bien, una vez aclarada la notación referente a la complejidad algorítmica, estamos en condiciones de proceder a la clasificación de nuestro problema de optimización. Esta categorización en clases se expresa en relación a la existencia de un algoritmo que se ejecute en una *maquina* y que sea capaz de resolver el problema con una complejidad $O(n^k)$ , $\forall k \in \mathbb{R}^+ : k \geq 0$.

Lo cierto es que hemos pasado de puntillas en torno a "maquina que es capaz de ejecutar el algoritmo". Dicha expresión engloba una gran complejidad ya que requiere definir formalmente el concepto de máquina. Pues bien, la teoría de la complejidad distingue entre dos tipos de maquinas: las maquinas deterministas y las no deterministas.
Este tipo de maquinas se basan en la definición de la conocida *maquina de Turing*. Entrar en detalle acerca del funcionamiento podría ser bastante extenso y no es objeto del proyecto. Autores como [1.11] hacen referencia al funcionamiento de la misma. En grandes rasgos podríamos decir que una maquina determinista puede asemejarse con los equipos de computación actuales, es decir, computadores basados en una memoria finita y un procesador con un conjunto de instrucciones capaz de determinar la secuencia de pasos a realizar a partir de unos datos de entrada. Los algoritmo que se ejecutan en este tipo de maquinas se denominan algoritmos deterministas. Se asume que el resultado de la ejecución de este tipo de algoritmos, es la terminación del mismo y además, produciendo la respuesta correcta. Por otra parte, las maquinas no-deterministas, son más complejas de definir. Una descripción simple podría ser la de considerarlas como una computadora que en cada paso puede ejecutar la siguiente instrucción de igual forma que una maquina determinista o bien ejecutar una instrucción de clonación para crear un numero finito de replicas de la computadora y su estado , cada una de las cuales saltara a distintas instrucciones del programa. Después de cada clonación cada replica puede repetir el proceso de clonación en futuros pasos pero no debe existir comunicación entre dos replicas. Evidentemente, las maquinas no-deterministas únicamente poseen interés desde el punto de vista teórico. Se trata de un concepto que no puede llevarse a cabo ya que no es posible construir una maquina con capacidades de ejecución en paralelo ilimitadas.

Pues bien, a partir de estas definiciones, introducimos las clases de complejidad: *polinomial,P* y *non-deterministic polinomial, NP*:

Decimos que un problema de decisión $p$ es de clase $P$ ($p \in P$) cuando existe un algoritmo determinista que resuelve el problema, cuya complejidad está acotada por un polinomio. Se suele referir a los problemas $P$ como aquellos que pueden *resolverse* en tiempo polinomial. Ejemplos de problemas que se encuentran en $P$ son: Determinar si un número es primo, *primality testing problem* ([1.13] resuelve el problema con un algoritmo de complejidad $O((\log n)^{12})$ ) o encontrar el máximo común divisor, *greatest common divisor* ([1.14] resuelve el problema con un algoritmo de complejidad $O(\frac{n}{\log n})$ )

Por otro lado, un problema de decisión $p$ es de clase $NP$ ($p \in NP$) cuando existe un algoritmo no-determinista capaz de resolverlo, cuya complejidad está acotada por un polinomio. Se suele referir a los problemas $NP$ como aquellos que pueden *verificarse* en tiempo polinomial. Algunos ejemplos de problemas que se encuentran en $NP$ son: Determinar si un grafo G contiene un camino hamiltoniano u estimar su número cromático.

Desde la década de los 70s, el estudio de los problemas $NP$ cobra un gran interés gracias en gran medida al famoso seminario publicado por Cook [1.2] donde se introduce el problema SAT. A raíz de ello se comienzan a hablar de nuevos tipos de problemas (clases): Los problemas $NP-hard$ y sobre todo, los problemas $NP-Completos$.En este entorno es importante el papel que juegan las llamadas reducciones polinomiales. Dichas transformaciones surgen como una herramienta para indicar relaciones $\leq$ y $\geq$ entre los problemas en función de su complejidad. Dados dos problemas $p$ y $p'$,

decimos que $p$ se puede reducir en $p'$cuando existe un algoritmo determinista que puede resolverse polinomialmente, el cual permite transformar el *input* de $p$ en *input* a $p'$, independientemente del *input*, de tal forma que la respuesta de $p$ y $p'$ a ese *input* sea igual. Esta relación se denota como $p \propto p'$.En [1.11] se aborda en gran profundidad el concepto incluyendo un ejemplo interesante al respecto del problema que nos describe como es la reducción entre el problema de encontrar un ciclo hamiltoniano (HC, *Hamiltonian Cycle*) y problema del viajante (TSP, *Travelling Salesman Problem)* siendo esta $HC \propto TSP$.

Dicho esto, introducimos las clases mencionadas anteriormente:

Decimos que un problema $p'$ es $NP-hard$ si cualquier problem en $NP$ puede ser transformado en él mediante una reducción en tiempo polinomial. Por ello, los problemas $NP-hard$ son al menos tan complejos como los problemas $NP$. No obstante, un problema $NP-hard$ no tienen que pertenecer necesariamente a la clase de problemas $NP$.

$$p' \in NP-hard \Leftrightarrow \forall_p \in NP, p \propto p' \quad \textbf{(2)}$$

Un problema $NP-hard$ que además se encuentra en $NP$ se denomina $NP-Completo$. Por tanto, estos problemas son los más complejos en $NP$. Formalmente:

$$p' \in NP-Completo \Leftrightarrow p' \in NP, \forall_p \in NP, p \propto p' \quad \textbf{(3)}$$

Como podemos observar, la definición $NP-Completo$ parece a priori, simple. El problema reside en encontrar el primer problema $NP-Completo$, de ahí la importancia de [1.2] mencionado anteriormente así como de todo el desarrollo de relaciones polinomiales.Pues bien, lo cierto es que todo este desarrollo parte de la asunción de $P \neq NP$.Se trata de una de las conjeturas básicas sobre los que se asienta la teoría de la complejidad que aun no ha sido demostrada. Siempre ha existido la cuestión acerca de si realmente cada problema verificable en tiempo polinomial también puede ser computable en tiempo polinomial y por tanto $P = NP$ o por el contrario $P \neq NP$, tal y como se cree mayoritariamente.

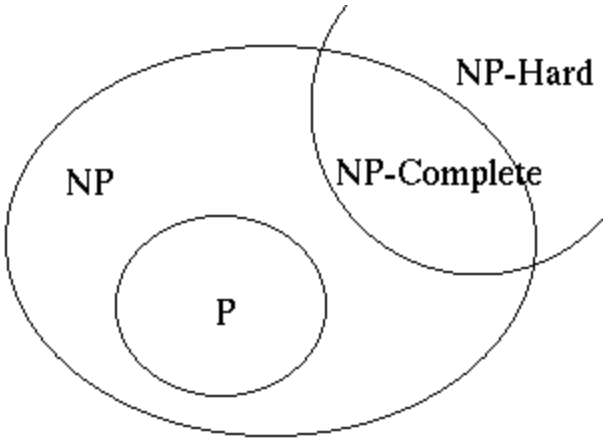



**Figura 3**. Dilema $\boldsymbol{P \ vs \ NP}$. Conjetura $\boldsymbol{P \neq NP}$

Pues bien, dejando a un lado el dilema, es importante que se perciban los principales retos de complejidad que se plantean al abordar el problema a tratar. Se trata de un problema $NP-hard$ , tal y como hemos visto, con un espacio de soluciones inmenso que hace que sea necesario nuevos métodos de resolución del problema distintos a los métodos exactos .Estas técnicas serán a bordadas a continuación.

## 2.2 Algoritmos Aproximados

Lo cierto es que una gran cantidad de problemas de optimización combinatoria [3](*VRP, TSP, QAP, NSP, Coloring Graph, etc.*), se pueden considerar *NP-hard*. Esto implica, tal y como hemos visto, que no es posible encontrar la mejor solución en un tiempo de computo finito. Por ello, y en parte también debido a la gran importancia práctica de estos problemas *CO* (Combinatorial Optimization) hemos asistido al desarrollo de procedimientos para encontrar buenas soluciones a los problemas (aun no siendo las más optimas), en un tiempo de computo razonable. Estos métodos o procedimientos en los cuales la rapidez del proceso es tan importante como la calidad de la solución obtenida, se denominan *heurísticos* o *aproximados*. Según [2.5], un heurístico puede definirse como:

*Un método heurístico es un procedimiento para resolver un problema de optimización bien definido mediante una aproximación intuitiva, en la que la estructura del problema se utiliza de forma inteligente para obtener una buena solución*

Podemos encontrar clasificaciones muy diversas en cuanto a heurísticos se refiere debido a la naturaleza diferente de estos. En nuestro caso emplearemos una de las más simples e intuitivas pero a la vez más utilizadas en la bibliografía. Se distinguen:

- Métodos *local search:* Se trata de algoritmos que siguen un proceso iterativo en el cual a partir de una solución inicial, el algoritmo reemplaza la solución actual por una solución vecina que mejore la solución (en términos de coste, rendimiento, etc.).Cuando no es posible mejorar la última solución, el algoritmo termina. La definición de qué es una solución vecina así como el criterio de elección entre varias de ellas cuando mejoran la solución actual, originan distintos tipos de algoritmos de búsqueda local.
- Métodos *constructivos:* Se trata de algoritmos que parten de una solución inicial vacía o parcialmente vacía y en cada iteración van tomando la mejor solución de tal forma que se va incrementando la solución actual. Este tipo de algoritmos se suele caracterizar porque una iteración no cambia partes de solución obtenidas en anteriores iteraciones.

En general, los métodos constructivos suelen ser más rápidos pero también los que proporcionan peores soluciones en comparación con los métodos local search. Dentro de los local search destaca. *hill-climbing*. Por su parte en los constructivos se destacan heurísticos tipo *greedy* (y sus variaciones).

Veamos el siguiente ejemplo de cara a ilustrar el funcionamiento de un heurístico. En concreto, se pretende resolver el problema de encontrar un ciclo hamiltoniano dado un grafo completo a través de un heurístico constructivo, en concreto un algoritmo *greedy*. El problema descrito se trata de un problema $NP-Completo$ y por tanto no es posible resolver de forma exacta en un tiempo de cómputo finito. No obstante se pretende resolver una instancia pequeña del problema con el objetivo de mostrar el funcionamiento de un heurístico. El algoritmo *greedy* funcionaria de la siguiente forma: i) Se elige un vértice con el que comenzar, y se visita el siguiente vértice conectado a través del enlace con el menor coste/peso .ii) Tras visitar el siguiente vértice se visita a través del enlace de menos coste/peso el siguiente vértice (que no haya sido visitado). Continua el proceso hasta visitar todos los vértices.iii) Regresamos al vértice inicial

[3] La optimización combinatoria es una rama de la optimización matemática cuyo dominio se compone de problemas de optimización donde el conjunto de posibles soluciones es discreto o se puede reducir a un conjunto discreto. En este tipo de problemas el objetivo consiste en encontrar la mejor solución posible o solución optima, minimizando una función de coste dada.

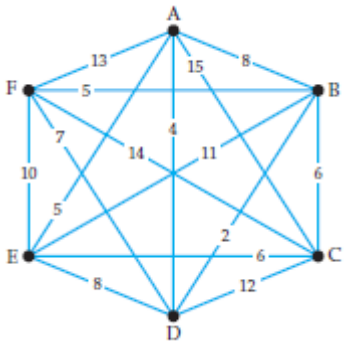



**Figura 4.**Grafo Completo

Empezando en el vértice $A$, el ciclo hamiltoniano seria: $A - D - B - F - E - C - A$ con un coste total de 42.

Al plantear este tipo de algoritmos es fundamental establecer una buena estrategia a la hora de ir permitiendo nuevas soluciones en el problema. El espacio de soluciones en los problemas en los cuales se aplican heurísticos suele ser muy grande y por tanto con gran cantidad de *mínimos locales*[4] Por ello se suelen emplear técnicas de intensificación o diversificación. Las técnicas de intensificación heurística suelen implicar que el algoritmo explore las regiones cercanas a las mejores soluciones. Un buen ejemplo del uso de esta técnica son los algoritmos *local search*. Por su parte la diversificación consiste en la idea contrapuesta. En este caso esta técnica se basa en evitar que el algoritmo quede buscando soluciones en una única región del espacio de soluciones. Por ello en ocasiones se permite explorar soluciones peores a la actual con el objetivo de encontrar soluciones no exploradas que mejoren la actual. Un claro ejemplo podrían ser los algoritmos basados en *random search*. Por otro lado, también es muy común emplear otras técnicas como permitir cambiar la definición de vecindario o agregar penalizaciones cuando el problema se desplaza a soluciones no factibles. Podemos observar una representación de cómo afecta la aplicación de cada una de estas técnicas en la **Figura 5**.Se trata de una solución constructiva $S_1$, Una solución con local search $S_4$ y la solución optima $S_9$ . Soluciones como $S_2$ o $S_5$ hacen referencia al cambio de vecindad. $S_6$ Y $S_{11}$ a la aplicación de técnicas de intensificación y diversificación. El resto de soluciones hacen referencia a la aplicación de penalizaciones que permitan en mayor o menor medida evitar mínimos locales.

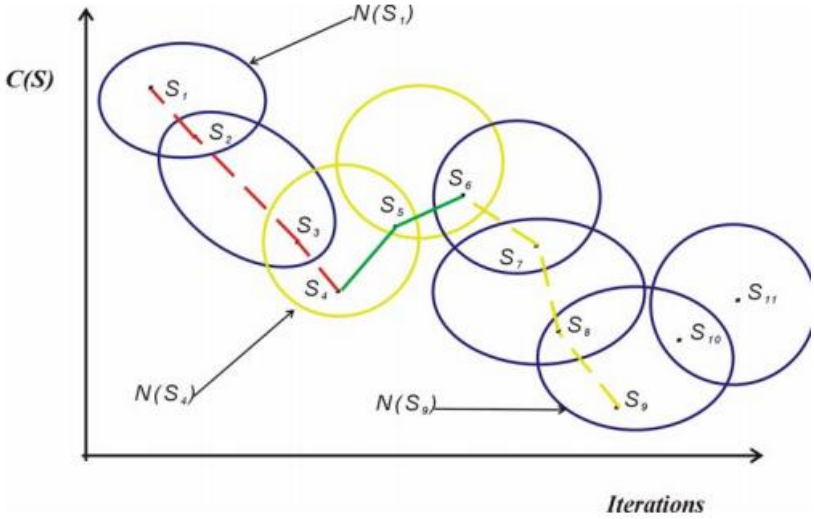



**Figura 5.** Solución método constructivo $S_1$, solución método local search $S_4$, solución optima $S_9$

[4] Un mínimo local en un problema de optimización es una solución factible $\bar{x}$ en la que la función objetivo es mayor o igual que sobre cualquier otra solución factible $\bar{y}$ que esté suficientemente próxima a $\bar{x}$, es decir, que cumpla $\|\bar{x} - \bar{y}\| < \varepsilon$ para un cierto $\varepsilon > 0$ .

Pues bien, hecha esta distinción entre los principales tipos de heurísticos así como las principales estrategias que suelen implementar, es importante destacar que el desarrollo de procedimientos o algoritmos para intentar solucionar los problemas de optimización *NP-hard*, no solo comprende el desarrollo de los algoritmos heurísticos tradicionales[5]. En muchos casos estos métodos heurísticos por si solos son ineficaces (grandes *inputs*) o bien proporcionan soluciones demasiado específicas sin posibilidad de generalizar. Por ello en los últimos 30 años se han intentado mejorar los resultados proporcionados por estos últimos mediante la combinación de varios heurísticos clásicos. De hecho la combinación de métodos heurísticos de construcción y de búsqueda local está considerada como una de las bases de los denominados algoritmos *metaheuristicos*. Se trata de algoritmos de complejidad superior a los heurísticos, superiores en cuanto a su nivel de abstracción y que surge bien de la combinación de varios heurísticos (por ejemplo: *local search* y *greedy*) o bien de la utilización de alguno de estos junto con técnicas u estrategias de distintas áreas del conocimiento (biología, industria, etc) Según [2.6]:

*Los algoritmos metaheurísticos son aquellos métodos empleados para resolver problemas complejos en los que los heurísticos clásicos no son efectivos. Los metheuristicos proporcionan un marco general para crear nuevos algoritmos híbridos combinando la inteligencia artificial, la evolución biológica y los mecanismos estadísticos*.

Se podría decir que son "filosofías" o plantillas a seguir a la hora de encontrar la solución y que pueden adaptarse a una gran cantidad de problemas. En general los metaheuristicos funcionan bien en numerosas ocasiones ya que tratan de emular aquellos procesos de la naturaleza capaces de resolver problemas complejos o de gran entidad. Existen diferentes formas de clasificar los métodos metahuristicos atendiendo a diferentes criterios: *Nature inspired vs non-nature inspired*, memory *usage vs memory-less algorithm*, etc. Pues bien, un análisis pormenorizado de todos los tipos de metahuristicos podría constituir otro trabajo final de máster y por tanto no es objeto del proyecto que aquí nos atañe. En concreto, nuestro interés reside en aquellos metahuristicos que mejor aplicación tienen en el problema VRP aquí descrito. Como *vehicle routing problem* es un problema clásico muy estudiado muchos son los autores que enumeran los metahuristicos con los mejores resultados. Es el caso de [2.3] quien establece como los principales tipos de metaheuristicos aplicados a VRP: *Simulated Annealing (SA), Deterministic Annealing (DA), Tabu Search (TS), Genetic Algorithm(GA) Ant System(AS)* y *Neural Networks(NN).*En nuestro caso, debido a las restricciones del problema (*time-window.capacity constraints o incompatible goods*) vamos a emplear los que mejores resultados han arrojado en relación al tiempo de cómputo o calidad de la solución a tenor de [3.5] de cara a no aumentar la complejidad del problema. Dejaremos para futuros trabajos la posibilidad de trabajar con algoritmos de tipo *Neural Network* o *GTS* cuyo rendimiento no ha sido explotado por completo sobre VRP y que tan esperanzadores resultados ha dado en [2.3]

### 2.2.1 Algoritmos de enfriamiento simulado (Simulated Annealing, SA)

Este tipo de algoritmos tienen su origen en [2.7].La metodología de los algoritmos de enfriamiento simulado (*Simulated Anneling*) es una analogía al proceso de enfriamiento que experimentan los sólidos (generalmente metales), con el objetivo de alcanzar un estado más estable. Con la intención de evitar estados químicos no estables, los metales se suelen enfriar lentamente de forma que tienen el tiempo suficiente para alcanzar estructuras finales fuertes, solidas y con mínima energía. Pues bien, esta analogía puede aplicarse a la optimización combinatoria [2.7], relacionando una solución factible con el estado del sólido, la función objetivo con la energía del sólido y la solución optima con el estado de mínima energía.

SA es un algoritmo basado en *local search*, en el que se intenta evitar mínimos locales permitiendo aceptar soluciones peores con una cierta probabilidad $P$ .El algoritmo empieza con una solución

[5] Los heurísticos arriba mencionados(local search) también se conocen como heurísticos tradicionales o clásicos

inicial $s$ y en cada iteración se elige una solución vecina $s' \in N(s)$ (normalmente de forma aleatoria , p ej. siguiendo una distribución uniforme). Si $s'$mejora a $s$ (mejora en términos de coste), es aceptada. En caso contrario, la solución podría seguir siendo aceptada (incluso con un coste peor) con una probabilidad $P$ que depende de la diferencia de costes entre ambas soluciones y el parámetro $T$, denominado temperatura. $P$ se define en referencia a una distribución Metrópolis [2.8]

$$P(s, s', T) = \begin{cases} 1, & Si\ C(s') < C(s) \\ e^{-\frac{C(s')-C(s)}{T}}, & cualquier\ otro\ caso \end{cases} \quad \textbf{(4)}$$

La temperatura $T$, es un variable global del algoritmo y tiene una importancia crítica. Cuando $T$ , toma valores elevados, la probabilidad de aceptar soluciones peores es alta ($T \to \infty \Rightarrow e^{-\frac{C(s')-C(s)}{T}} \to 1$) .En esta situación el algoritmo se comporta como un *random search*. Por el contrario cuando la temperatura es baja, la probabilidad de aceptar soluciones peores es muy pequeña($T \to 0 \Rightarrow e^{-\frac{C(s')-C(s)}{T}} \to 0$) . En este caso, SA, funciona de forma similar a un algoritmo *local search* Por ello se suele iniciar el parámetro $T$ con un valor alto, que va disminuyendo a medida que las iteraciones avanzan, de manera análoga al proceso de enfriamiento de los sólidos. **Figura 6**, muestra funcionamiento general de SA

```
s ← Generar Solución Inicial
Inicializar parametros Annealing(T, Condiciones parada, etc.)
s_best ← s
while condición parada algoritmo do
  while condición cambio T do
    s' ← Generar Solución Vecina
    s ← Aceptar Solución con P(s, s', T)
    if C(s') < C(s) then
      s_best ← s
    end if
  end while
  Actualizar temperatura, T
end while
return s_best
```

**Figura 6.** Pseudocódigo genérico para SA

### 2.2.2 Algoritmos de búsqueda tabú (Tabu Serch, TS)

Tienen su punto de partida en [2.9] al comenzar su aplicación a los problemas de optimización combinatoria. Este tipo de metaheurísticos son considerados algoritmos *local search* con procedimientos para diversificar la búsqueda de soluciones. En concreto, la característica principal para lograr esta diversificación es el empleo de la denominada "lista tabu". Con el objetivo de que en cada iteración no se vuelva sobre soluciones ya exploradas y sobre todo para evitar bucles, cada solución vecina visitada (o un atributo de la misma) se incluyen en la lista. De esta forma la lista tabu funciona como una memoria, "recordando" soluciones vecinas exploradas recientemente con el fin de no repetirlas, ya que el algoritmo prohíbe visitar aquellas soluciones de la lista tabu.

El tamaño de la lista tabu denominado *tabu tenure* tiene una importancia crucial en el algoritmo. Aunque quizás la opción más común es la de optar por tamaños de lista pequeña, es muy común enfrentarse a la decisión de elegir entre una propuesta *short memory vs long-term memory.* Un tamaño de lista pequeño provoca que el algoritmo se comporte como un *local search*, pudiendo estancar la búsqueda en un pequeño espacio de soluciones, sin una diversificación suficiente. Por otro lado, una lista tabu grande implica una mayor complejidad a la hora de almacenar información en la lista (P ej

es común que TS con long-term memory almacenen la frecuencia con la que aparece un determinado atributo). Este hecho trae consigo que puedan presentarse problemas de memoria, tiempo de cómputo y lo que es más importante puede prohibir arbitrariamente el probar soluciones de calidad.

El hecho de incluir en la lista tabu una solución vecina (en concreto un atributo de la misma) impide que se visiten un gran número de soluciones además las ultimas probadas. Esto en ocasiones puede acarrear que no se exploren soluciones que mejoran la actual. Para paliar este inconveniente se suele introducir algún tipo de excepción que permita saltar a soluciones que se encuentran en la lista tabu. Es el conocido *aspiration criterion (criterio de ambición)* .El criterio más común es el de permitir soluciones tabu si mejoran la solución actual. **Figura 7**, se puede apreciar el esquema de un algoritmo TS básico.

$s \leftarrow Generar\ Solución\ Inicial$
$Inicializar\ Lista\ tabu$
$s_{best} \leftarrow s$
**while** $condición\ parada\ algoritmo$ **do**
  $A \leftarrow Generar\ Soluciones\ Admisibles\ de\ s$
  $s \leftarrow Seleccionar\ Mejor\ Solución\ de\ A$
  $Actualizar\ Lista\ Tabu\ (añadir\ o\ eliminar\ elementos)$
  **if** $C(s') < C(s)$ **then**
    $s_{best} \leftarrow s$
  **end if**
**end while**
**return** $s_{best}$

**Figura 7**. Pseudocódigo genérico para TS

# Capitulo 3. *The Vehicle Routing Problem*

Pues bien, tal y como hemos comentado uno de los problemas $NP-hard$ clásicos y además, principal objeto de estudio de este trabajo es el *Vehicle Routing Problem*. Según [3.1] el problema VRP simétrico[6] puede definirse sobre un grafo completo no dirigido $G=(V,E)$.El conjunto $V=\{0,\dots,n\}$ es el conjunto de vértices. Cada vértice $i \in V\backslash\{0\}$ representa a un cliente, mientras que el vértice 0 se corresponde con el *depot*. Por otro lado cada enlace e $\in$ E $=\{(i,j): i,j \in \mathrm{V}, \mathrm{i}<j\}$ tiene asociado un coste $c_e$. Una flota de vehículos $k$ se encuentra disponible en el *depot*. Pues bien, el objetivo básico del problema reside en encontrar el conjunto de rutas con inicio y final en el *depot*, que minimicen el coste total (número de vehículos, coste enlace, etc.) recorriendo cada uno de los $i$ vértices, una única vez.

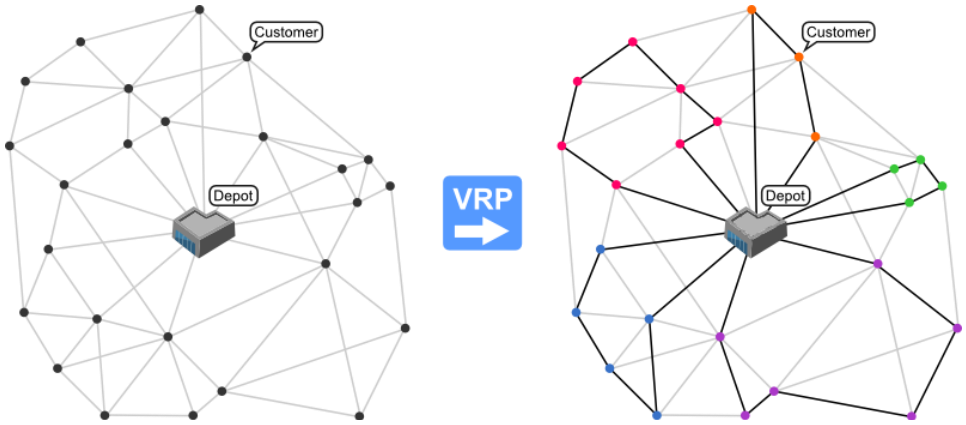



**Figura 8.** Resolución problema VRP

A la vista de la definición del mismo, podemos observar como este problema es una evolución del problema clásico del viajante (*Traveling Salesman Problem* o TSP), antes mencionado. Particularizando el problema con un único vehículo en nuestra flota, tendríamos una simplificación del problema VRP cuya resolución pasaría por resolver un problema TSP` .El problema VRP fue propuesto por primera vez por [3.2] y se ha caracterizado por presentar diversas variaciones (tipos) del mismo. En concreto nuestro interés reside en evaluar el problema en presencia de restricciones de capacidad del vehículo (CVRP, *capacity vehicle routing problem*), temporales (VRPTW, *vehicle routing problem with time windows*) en relación al tiempo de servicio y hora de descarga de mercancías y de incompatibilidad de cargas (*vehicle routing problem with incompatible goods*). Las dos primeras son dos variaciones clásicas del problema tratadas en multitud de artículos científicos empleando diversas técnicas de resolución (diferentes metaheuristicos, diferentes formas de codificar solución, etc.)Por ejemplo [3.3] Resuelve el problema VRPTW con metaheruristicos del tipo genéticos mientras que [3.4] lo hace con tabu search. En [3.5] podemos encontrar un barrido de los principales algoritmos metaherusiticos aplicados al problema VRPTW. Por otro lado, el último tipo de restricción es sin duda uno de los aspectos más novedosos de nuestro proyecto y es además un problema inherente al sector del transporte de mercancías. Es muy común que debido a cuestiones de perecimiento, dos cargas deban transportarse de la forma más independiente posible, llegando incluso a realizarlo en vehículos diferentes. Por ejemplo, algunos frutos desprenden gases (como el etileno) durante el proceso de maduración y ello puede acelerar el proceso de maduración de otras frutas, llegando incluso a dañarlas .Pues bien, nos centraremos en la descripción de los principales tipos de problema que trataremos de resolver posteriormente con la propuesta de nuestro algoritmo. Finalmente también describiremos brevemente otras variantes como VRP *with pick and delivery*, MDVRP, *multiple depot vehicle routing problem*, *stochastic* VRP, etc.

[6] Simétrico hace referencia a que se basa en un grafo no dirigido. En nuestro problema VRP no tiene sentido pensar en modelar el problema como un grafo dirigido.

## 3.1 CVRP, *Capacity Vehicle Routing Problem*

Es considerado la restricción más básica sobre el problema VRP clásico y fue por tanto uno de la primeros tipos de VRP en ser analizado .En el problema VRP con restricciones de capacidad (CVRP), $n$ clientes deben ser atendidos desde un *depot* normalmente identificado por el nodo $0$. Cada cliente $i$ demanda una cierta cantidad $m_i$ positiva ($m_i > 0$) de un mismo producto y para cada par de clientes existe una distancia $m_{ij}$ dada. Los clientes son atendidos por una flota de vehículos, cada uno de los cuales tiene una capacidad $C$.Pues bien el principal objetivo en CVRP es encontrar el conjunto de rutas que minimizan el tiempo total de viaje de forma que i) Un cliente es servido una sola vez por un único vehículo ii) La ruta seguida por cada uno de los vehículos empieza y termina en el *depot* .iii) El total de demandas cubiertas por cada vehículo no puede exceder su capacidad $C$.El problema CVRP es por tanto $NP - hard$ ya que contiene el problema TSP como subproblema. Por ello en la práctica resolver el problema VRP es más difícil ya que implica resolver dos problemas anidados. El primero es un problema de *bin-packing* donde se busca asignar cada uno de los clientes a un número de rutas a priori indeterminado. A continuación para cada ruta se busca el recorrido más corto recorriendo cada uno de los clientes asignados a la ruta, lo que implica resolver el problema TSP.

En este punto es importante resaltar algunas de las particularidades de nuestro problema en cuanto a capacidad se refiere. Uno de ellos es la capacidad $C$ que disponen cada uno de los vehículos de la flota. Pues bien, la unidad de medida de la capacidad del camión vendrá expresada en *pales*, entendiendo como tal el soporte de madera, plástico o metal que sirve para facilitar el transporte de mercancías o bienes, ya que impide su contacto directo con el suelo y facilita la mecanización de sus movimientos durante el transcurso de su transporte y almacenamiento. Existen diversos tipos de *pales* pero en nuestro caso, por convención se empleará el denominado EUR (europalé).Las medidas del mismo son: 120x80 cm con altura de 14.5 cm

Por otro lado, también existen diversos tipos de vehículos (camiones) para el transporte de mercancías. Cada uno de ellos presenta distinta capacidad de pales. Para nuestro problema asumiremos que nuestra flota de vehículos es homogénea y presenta camiones de dimensión estándar .Dicha dimensión permite cargar 33 eurpales de cargas no remontables[7] o bien 66 pales de cargas remontables. Por tanto nuestras restricciones de capacidad tratarán de llenar los vehículos para que su capacidad sea lo más próxima posible a estos 33 europales.

## 3.2 VRPTW, *Vehicle Routing Problem with Time-Windows*

Algunas de las extensiones más estudiadas del problema CVRP consiste en aplicar restricciones temporales al mismo, el llamado VRPTW. En esta versión del VRP además de las restricciones de capacidad antes mencionadas, cada cliente $i$ tiene una ventana temporal $[e_i, l_i]$ durante la cual debe recibir la mercancía solicitada. En este caso $e_i$ hace referencia a la fecha más temprano en el que se puede servir al cliente $i$ mientras que $l_f$ hace referencia al instante más tardío .Las ventanas temporales están asociadas con el *depot* .
Pues bien, el objetivo de VRPTW es diferente de CVRP. En concreto se persiguen dos hitos: i) Minimizar el número de vehículos empleados ii) Minimizar el tiempo total de viaje. Estos hitos u objetivos tienen cierto orden de prioridad o jerarquía, es decir siempre es preferible una solución que utilice un menor número de vehículos que una con un mayor número de vehículos que emplee un menor tiempo. Dos soluciones que utilicen el mismo número de vehículos serán comparadas en función del tiempo empleado y otros criterios como el margen temporal para decidir entre una u otra. Este margen temporal está relacionado con el tiempo sobrante de cada vehículo en un cliente determinado. Este tiempo es de gran importancia ya que influye en otras variables temporales como pueden ser los descansaos del chofer de un vehículo. También son importantes los posibles horarios

[7] La mercancía remontable hace referencia a aquella que puede apilarse sin que ello implique riesgo de daños en la mercancía. Por ejemplo mercancías como vino u aceite se consideran no remontables.

de apertura y cierre de los clientes ya que deben contemplarse a la hora de estimar tiempo total de viaje y de cara a respetar las ventanas temporales. No obstante, no ha sido posible obtener datos reales que modelen el tiempo de descanso de un conductor ni contemplar horarios de apertura de los clientes debido al sistema propuesto para resolver el problema que veremos en siguientes capítulos. Aun así estas variables deberán ser contempladas en futuras implementaciones de software específico.

Pues bien, centrándonos en el ejemplo podemos destacar algunas particularidades de las restricciones temporales .En principio nuestro problema parte de una fecha de carga de los vehículos inicial correspondiente al día 14 y una fecha de entrega máxima al cliente el día 16 a las 13 horas o 17 a las 6 horas, dependiendo del cliente. Estos datos marcaran nuestras time-Windows como podremos ver en el capítulo 5, con el input de entrada real de nuestro algoritmo. El principal escollo para considerar la solución factible no solo será cumplir las ventanas temporales sino también, el tiempo total de viaje. Este hecho provoca que en multitud de casos, muchas soluciones se descarten. Otro aspecto a tener en cuenta para futuras propuesta es el hecho de contemplar las variaciones que puedan presentar las ventanas temporales de cada cliente en función de si el cliente ofrece la posibilidad de concertad cita u no.

## 3.3 VRPIG, *Vehicle Routing With Incompatible Goods*

Se trata de la principal novedad de nuestro proyecto. En concreto las restricciones de incompatibilidad están estrechamente relacionadas con la capacidad del vehículo. Es muy común que debido a los gases que se desprenden durante el proceso de maduración muchas mercancías deben ir aisladas u bien transportadas en vehículos separados. En multitud de publicaciones se habla acerca de la diversidad en la conservación de los productos hortofrutícolas, principal mercancía que vamos a transportar en nuestro problema. Algunos frutos como manzana o pera desprenden compuestos químicos orgánicos como $C_2H_4$ (comúnmente conocido como etileno) durante el proceso de maduración. Dicha emisión puede afectar directamente en la conservación de otros alimentos, pudiendo acelerar el proceso de maduración de los mismos. Este proceso se encuentra ligado con el transporte de mercancías climatéricas. Las mercancías climatéricas son aquellas que pueden seguir madurando una vez recolectadas. En este proceso de maduración tiene una gran importancia el etileno tal y como hemos mencionado. Por otro lado, además de ciertas recomendaciones en cuanto al transporte, existe normativa que regulan el transporte de ciertas mercancías. Nos basaremos en [3.6] [3.7].Se trata de un recomendación elaborada por el Instituto del Frio, organismo integrado en el Instituto de Ciencia y Tecnología de Alimentos y Nutrición (ICTAN), dependiente del CSIC para realizar una clasificación de mercancías valida que respete incompatibilidades. **Tabla 3** muestra dicha clasificación

| Categoria | Tipo de Mercancia | Sintomas |
|---|---|---|
| C1 | Lechuga ,Apio,Brocoli,Pimiento,Calabacin | Sensibles etileno, temperatura de conservación baja |
| C2 | Tomate,Melon,Papaya ,Granada y Caqui | Productores etileno y temperatura de conservación alta |
| C3 | Cítricos (limon,naranja y mandarina), Ajo ,Cebolla y Castañas | Ni productores ni sensibles al etileno. Por lo general , no climatérico y con una temperatura de conservación media |

**Tabla 3**. Categorización de mercancías en función de incompatibilidad de mercancías

Pues bien, la clasificación se hace atendiendo a las mercancías empleadas en nuestro ejemplo. La primera categoría engloba la mayoría de productos "verdes" que son los que mayor sensibilidad plantean al etileno .Además necesitan de ambientes de conservación fríos. El segundo grupo hace referencia a los alimentos productores de etileno y que por lo general requieren de temperaturas elevadas. Por último en el grupo 3 se han agrupado aquellas mercancías no climatéricas, que tampoco

son sensibles ni productores de etileno y que pueden conservarse a una temperatura media..No obstante los costes son un aspecto vital en cualquier empresa y en este caso un cumplimiento riguroso de esta separación conllevaría el uso de multitud de vehículos. También implicaría una praxis poco común en el sector de la logística y el transporte de mercancías donde se suelen asumir riesgos en el transporte a cambio de reducir los costes. Por ello nuestra restricción de incompatibilidad de mercancías permitirá fusionar mercancías de la categoría C1 con C3 así como C2 con C3 asumiendo siempre un riesgo

## 3.4 Otros VRP

Además de los tipos de *vehicle routing problem* mencionados existen multitud de variaciones. La mayoría implican aumentar la complejidad del problema y en nuestro caso, debido al problema a resolver no ha sido necesario. Aun así, es conveniente conocer los principales problemas VRP alternativos, sus principales ventajas y desventajas, aplicaciones así como el por qué de no tener presentes estas restricciones en nuestro problema.

### 3.4.1 MDVRP, *multiple depot vehicle routing*

Se trata de una ampliación del problema clásico VRP en el cual se asume que existe un único punto de almacenaje y salida, *single-depot* VRP. En MDVRP existen múltiples depot y por tanto existe una mayor complejidad a la hora de resolver el problema ya que supone un paso más sobre el algoritmo. En concreto suele ser necesaria una primera etapa de clustering o reagrupamiento en la cual cada cliente suele ser asignado a uno de los *depot* disponible.

### 3.4.2 VRP with *backhauls*

La extensión con *backhuls* se presenta como una de las formas de maximizar la carga útil del vehículo durante el transcurso completo de la ruta. Comúnmente se suele dividir el conjunto de vértices (que representa a los clientes) en dos subconjuntos, *linehaul* y *backhauls*. El primero de ellos hace referencia a los clientes en los que se entrega mercancía mientras que el segundo hace referencia a aquellos clientes en los que se recoge mercancía (pudiendo también entregar mercancía). Las restricciones de capacidad son más fuertes que en el caso CVRP ya que dependen tanto de la mercancía demandada como de la que se recoge. Algunas aplicaciones de este tipo de *vehicle routing problem* son

### 3.4.3 VRP *with pickup and delivery*

Una de las variantes clásicas. En su versión más básica VRPPD, se basa en la idea de que cada cliente $i$ tiene asociadas dos cantidades $m_i$ $y$ $p_i$ las cuales representan la demanda del cliente $i$ y la cantidad de producto a recoger en el cliente $i$, respectivamente. En ocasiones se suele expresar la demanda como la diferencia entre ambas cantidades $m_i = m_i - p_i$. Normalmente se asume que el proceso de entrega se realiza antes que la recogida. Para cada cliente $i$, $O_i$ hace referencia al vértice origen de la demanda y $D_i$ al destino de la recogida. Es habitual que el origen de las demandas sea común (p.ej asociado con el *depot*). A este problema se le conoce como VRPSPD, *vehicle routing problem with simultaneus pickup and delivery.* Tanto VRPPD como VRPSPD se consideran $NP - hard$ ya que son ampliaciones del problema CVRP ($O_i = D_i = 0$ y $p_i = 0$ para cada $i \in V$ ).La variante *pickup and delivery* tiene varias aplicaciones prácticas como transporte de personas ancianas o impedidas [3.10], transporte de pasajeros y mercancías en contextos militares [3.14], entre otros.

### 3.4.4 SVRP, *stochastic vehicle routing problem*

Quizás una de las variantes menos conocidas aunque con multitud de aplicaciones. Normalmente el problema VRP descrito bien en su forma básica o aplicando alguna restricción sobre él, hace referencia al llamado VRP *deterministic.* Esta denominación se debe al carácter discreto o determinista del problema en relación a las distintas variables del problema (número de clientes, demandas de los mismos, tiempos de servicio).En SVRP muchas de estos componentes tienen un carácter azaroso (por ejemplo, un cliente $i$ esta presente con cierta probabilidad $p_i$, las demandas o tiempo de servicio del cliente $i$, $m_i$ $y$ $t_s$ son vriables aleatorias, etc.). Debido a ello las restricciones

no se suelen satisfacer para cada uno de los valores aleatorios de estas variables y por tanto se emplean nuevos conceptos en relación a saber si una solución en factible y óptima Es por ello que resolver el SVRP suele considerarse de mayor dificultad que la versión determinista. Algunos de las innumerables aplicaciones de SVRP podrían ser: Entrega de comidas a domicilio [3.11], recogida de depósitos de sucursales bancarias y entrega en oficina central [3.12], encaminamiento de montacargas en almacenes [3.13].

Pues bien, en lo que respecta a nuestro problema, las anteriores variaciones de VRP no son del todo aplicables. MDVRP no se adecua ya que nuestro proyecto trata de resolver el problema VRP aplicado al transporte de mercancías internacional. En concreto nuestro interés reside en la exportación internacional de mercancías. Simplificando las tereas internas de carga que puedan existir, podemos establecer un único *depot*, en nuestro caso será España. La extensión *Backhauls*, sin duda se emplea en innumerables ocasiones debido a cuestiones económicas. Su uso suele orientarse al viaje de regreso al *depot* una vez atendidos todos los clientes. A priori dicha opción no se contempla y se da prioridad a la resolución del VRP con restricciones de capacidad, tiempo e incompatibilidad de mercancías. Por otro lado, debido a las fuertes restricciones temporales así como el proceso de carga no empleamos una solución a VRPPD. Para optimizar el proceso de descarga en cada una de las ciudades, se suele llevar a cabo una carga de mercancías en orden inverso al de entrega. Por ello cualquier recogida u carga intermedia afectaría fuertemente los aspectos mencionados. Por último indicar que trataremos con datos reales en lo que respecta a tiempos de servicio, lugares de entrega , cantidades, etc. No obstante los conceptos del *stochastic vehicle routing* pueden tener cabida en futuras líneas del proyecto a través de simuladores, validadores, etc.

# Capitulo 4 Formulación del Problema

En este apartado vamos a exponer como se ha llevado a cabo la formulación de nuestro problema VRP con las restricciones mencionadas en el capitulo anterior (carga, tiempo e incompatibilidad de mercancías). Especificamos el problema ya que independientemente de que vaya a ser resulto a través de software libre, es muy importante contar con una especificación del mismo de cara a que pueda ser implementado en un futuro a través de software especifico.
Pues bien, el problema tal y como comentábamos en la introducción al capítulo anterior, se compone de un conjunto de $k$ vehículos idénticos y un conjunto de $n$ nodos siendo considerado como nodo 0, el *depot.* Cada enlace *e* entre cada uno de los vértices $i$ que representan a los clientes, indica la conexión entre dos nodos. Cada enlace tiene asociado un coste $c_{ij}$ y un tiempo $t_{ij}$.Dentro de este coste $c_{ij}$ debe incluirse una estimación monetizada del coste del camión. De esta forma no solo minimizamos la distancia recorrida (o coste total) sino también lo hacemos con el mínimo número de camiones. El número de rutas descritas debe ser igual al número de vehículos usados, es decir, los vehículos no pueden ser reutilizados ni pueden mezclar rutas. Cada uno de los clientes solo puede ser visitado en una ocasión por un único vehículo. Además, cada vehículo tiene una capacidad $C$ que es igual para cada vehiculo y que no puede ser superada por el conjunto de demandas $m_i$ de la ruta que recorra dicho vehículo. De esta forma un vehiculo no puede almacenar más carga de su capacidad. Es importante indicar que $m_i$ consiste en el aglomerado de demandas de acuerdo a las restricciones de incompatibilidad de mercancías. Es decir puede considerarse como la suma de las demandas que cumplan condición 1 y 3 o bien la suma de las demandas de 2 y 3. Simplemente realizamos este apunte para quede constancia de que internamente $m_i$ debe separarse de acuerdo a los criterios incompatibilidad de mercancías aunque no lo reflejaremos en la formulación del problema para no aumentar la complejidad del mismo. En cuanto a las restricciones temporales, están vienen especificadas de acuerdo a la ventana temporal marcada por el menor tiempo de llegada $t_{min}$ y el máximo tiempo de llegada $t_{max}$.Cuando un vehículo llega antes de $t_{min}$ , se produce un tiempo de espera. Además de estos tiempos también se considera un tiempo de servicio $f_i$ .Además cada uno de los vehículos tiene la condición impuesta de realizar la ruta en un tiempo total máximo que es en esencia el time Windows del *depot.*

Existen tres tipos principales de variables de decisión. La principal variable de decisión $x_{ijk}$ $(i, j \in [0,1,2,\dots,N]; k \in [1,2,\dots,K]; i \neq j)$ es 1 si el vehículo $k$ viaja desde vértice $i$ al vértice $j$ y 0 en caso contrario. La variable de decisión $t_i$ hace referencia al tiempo en el cual el vehículo llega al vértice $i$ y $w_i$ denota el tiempo de espera en el vértice $i$ . El objetivo es diseñar una red que satisfacen todas estas restricciones y minimizan el coste total. El modelo matemático se corresponde con:

Principales variables de decisión

$t_i$ $tiempo\ espera\ vertice\ i$
$w_i$ $tiempo\ espera\ vertice\ i$
$x_{ijk} \in [0,1], 0\ si\ no\ existe\ enlace\ e\ entre\ el\ vertice\ i\ y\ el\ j. 1\ en\ caso\ contrario\ i \neq j; i, j \in [0,1,2,\dots,N]$

Parámetros
$K$ $numero\ total\ de\ vehiculos$
$N$ $numero\ total\ de\ clientes$
$y_i$ $numero\ arbitrario\ real$
$d_i$ $distancia\ entre\ el\ vertice\ i\ y\ el\ j$
$c_{ij}$ $coste\ entre\ el\ vertice\ i\ y\ el\ j$
$t_{ij}$ $tiempo\ entre\ el\ vertice\ i\ y\ el\ j$
$m_i$ $demanda\ del\ nodo\ i$
$C$ $capacidad\ vehiculo\ k$
$e_i$ $tiempo\ más\ teprano\ de\ llegada\ al\ vertice\ i$

$l_i$ *tiempo más tardio de llegada al vertice i*
$f_i$ *tiempo de servicio en el vertice i*
$r_k$ *tiempo máximo para recorrer ruta para vehiculo k*

Minimizar

$$\sum_{I=0}^{N}\sum_{j=0,j\neq 1}^{N}\sum_{k=1}^{K} c_{ij}x_{ijk} \quad (1)$$

Sujeto a

$$\sum_{K=1}^{K}\sum_{J=1}^{N} x_{ijk} \leq K \quad i=0 \ (2)$$

$$\sum_{J=1}^{N} x_{ijk} = \sum_{J=1}^{N} x_{jik} \leq 1 \quad i=0; k \in [1,2,\dots,K] \ (3)$$

$$\sum_{K=1}^{K}\sum_{j=0,j\neq I}^{N} x_{ijk} = 1 \quad i \in [1,2,\dots,N] \ (4)$$

$$\sum_{K=1}^{K}\sum_{i=0,i\neq j}^{N} x_{ijk} = 1 \quad j \in [1,2,\dots,N] \ (5)$$

$$\sum_{i=1}^{N} m_i \sum_{J=0 j\neq 1}^{N} x_{ijk} \leq C \quad k \in [1,2,\dots,K] \ (6)$$

$$\sum_{i=0}^{N}\sum_{J=0,J\neq I}^{N} x_{ijk}(t_{ij}+f_i+w_i) \leq r_k \quad k \in [1,2,\dots,K] \ (7)$$

$$t_0 = w_0 = f_0 = 0 \ (8)$$

$$\sum_{k=0}^{K}\sum_{i=0,i\neq j}^{N} x_{ijk}(t_i+t_{ij}+f_i+w_i) \leq t_j \quad j \in [1,2,\dots,N] \ (9)$$

$$e_i \leq (t_i+w_i) \leq l_i \quad i \in [1,2,\dots,N] \ (10)$$

La expresión (1) hace referencia a la función objetivo o función de costes. (2) asegura que como máximo existen K rutas y (3) asegura que cada ruta empieza y termina en el *depot*. Por su parte (4) y (5) establecen que un cliente solo puede ser visitado una única vez por un único vehículo. La ecuación (6) está referida a la restricción de capacidad,(7) al tiempo máximo de viaje y las ecuaciones (8)-(10) definen la ventana temporal

# Capitulo 5. Diseño del Algoritmo. Propuesta

Pues bien, una vez especificado el problema VRP al que nos enfrentamos, vamos a tratar de explicar nuestra propuesta de resolución del mismo. En concreto, lo que se propone es un sistema integral que no solo incorpora un metaheuristco para resolver el problema sino que se incorporan una fase de pre-procesado y otra de post-procesado. Las tareas de pre-procesado están dirigidas a "preparar" el *input* de nuestro problema mientras que la fase de post procesado está orientada a la representación grafica de los resultados así como a la simulación de los resultados obtenidos a través de un modelado estadístico.
Cuando tratamos con problemas combinatorios reales, en multitud de ocasiones no suele ser posible la aplicación de un metaheristico "out of the box" para resolver el problema en su conjunto. Normalmente es necesaria la sincronización de diversos componentes de cara a lograr separar funcionalidades y de esta forma conseguir que cada una de las fases de nuestro sistema realice las tareas para las que está pensado.

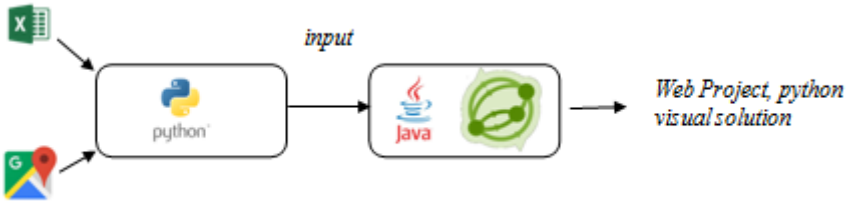



**Figura 9.** Sistemas propuesto

## 5.1 Fase Inicial

Trabajar con datos reales implica una mayor complejidad ya que nuestro algoritmo va a tratar con datos sin ningún tipo de "preparación". Por tanto, las demandas no siguen ninguna distribución concreta y los clientes no se reparten de forma predeterminada. Alguna de las particularidades de nuestro problema ya han sido mencionadas anteriormente: Una flota de camiones en principio "ilimitada" aunque independientemente nos interesa minimizar este número, capacidad de los vehículos de 33 europales siguiendo por tanto una flota heterogenia, restricciones temporales en función de la hora de entrega máxima, incompatibilidad de mercancías en función del tipo de producto (climatérico o no climatérico) y sus características (sensible o no al etileno).El conjunto de clientes o topología del problema queda recogida en la **Figura 10**

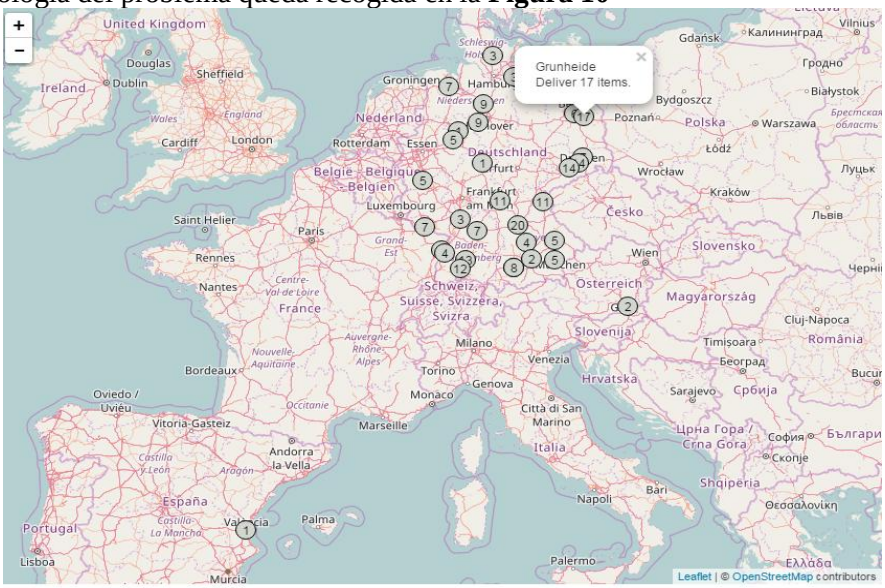



**Figura 10**.Topología problema VRP real

Pues bien, como se ha comentado el algoritmo tiene una fuerte dependencia con el *input* de entrada. En nuestro caso el *input*, se ha proporcionado a través de dos vías. Por un lado un documento de cálculo donde se indican las demandas de cada uno de los clientes .Por otro lado, una matriz de distancias entre localizaciones extraida con la ayuda de la herramienta Google Distance Matrix. Se trata de un servicio que permite obtener los valores de tiempo y distancia entre cada par de localizaciones. Para poder utilizar esta API, debemos respetar el siguiente formato: *https://maps.googleapis.com/maps/api/distancematrix/outputFormat?parameters?&key=YOUR_API_KEY*Donde *outputFormat* puede tomar el valor json o xml .Por su parte *parameters* hace referencia al conjunto de parámetros bien obligatorios u opcionales. Entre los obligatorios destacan origins, destinations y key haciendo referencia a las ciudades de origen, destino y la clave de API de la aplicación .En [5.1] se puede consultar información relacionada con resto de parámetros Todos estos datos son procesados a través de un script *python* con objeto de adaptarlos a un formato de entrada valido para *Optaplanner.* Dicho documento debe especificarse en formato .XML o .vrp

Por otro lado conviene resaltar el por qué de elegir *Python* como el lenguaje para el tratamiento de datos en lugar de otros como Matlab u R. Se ha optado por la elección del lenguaje interpretado *Python* ya que posee una gran flexibilidad, es ágil, sencillo y con una curva de aprendizaje muy corta. También se ha tenido muy en cuenta el hecho de que se trate de un lenguaje multiplataforma, *open-source* y con gran respaldo de la comunidad (aproximadamente 91000 repositorios en la plataforma de desarrollo colaborativo GitHub)

Es importante comentar también algunos aspectos a la hora de codificar valores sensibles del *input* . Entre ellos destacan el formato de las demandas así como la expresión de las ventanas temporales. Por un lado para poder trabajar con demandas que poseean cifras decimales es necesario multiplicar estas por algún factor de cara a no realizar aproximaciones y de esta forma perder precisión. Optaplanner trabaja con *inputs* en los cuales las demandas deben estar especificadas como enteros *(Integer)*.Por ello nuestras demandas aparecen con un factor *100. Posteriormente en la fase final son tratadas para expresar el valor decimal correctamente. Algo similar ocurre con las ventanas temporales. Estas también deben ser expresadas con algún factor para evitar errores. Además de ello existe una peculiaridad y es la de multipiclar las expresiones temporales *1000 para evitar errores de redondeo de coma flotante al comparar costes entre distintas soluciones. De esta forma, aunque trabajar con decimales es más lento que con tipos *double* o *float*, *Optaplanner* consigue eliminar posibles errores de estimación de la mejor solución. Cuando se calcula el tiempo empleado en recorrer la distancia entre dos ciudades de cara actualizar las ventanas temporales, se realiza de la siguiente manera: Por ejemplo la distancia entre Murcia y Almeria es de aproximadamente 224 km . Pues bien, en este caso *Optaplanner* estima que el tiempo necesario para recorrer dicha distancia es de 224000 u.t. En figura 5 puede observar el efecto del uso de tipos *doublé* o *float* frente a enteros.

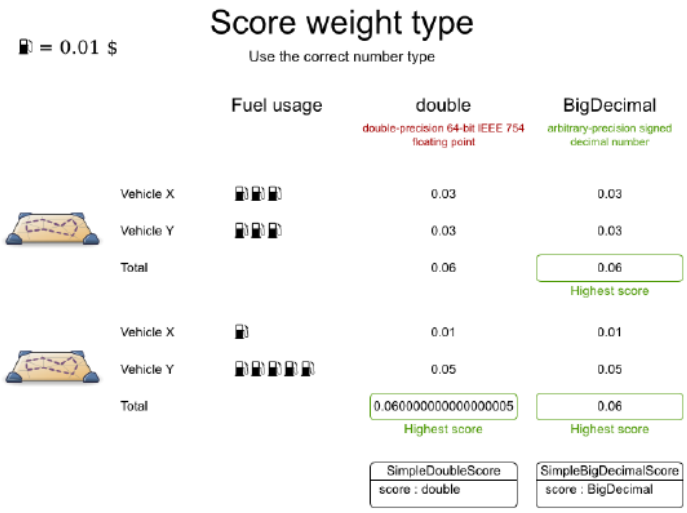



**Figura 11**. Errores en cálculos con tipos de datos *double*

Pues bien, en 5.2.3 aparece uno de los dataset de entrada empleado en el cual pueden distinguirse los aspectos arriba mencionados, resultados de esta primera etapa

## 5.2 Construcción del Metaheuristico: Vehicle Routing Problem con Optaplanner

Pues bien, la implementación del algoritmo de resolución del problema VRP es sin duda el núcleo de nuestro sistema. Para llevar a cabo dicha implementación existen numerosas opciones que pasan desde la realización de un software ad-hoc específico para el mismo hasta la utilización de herramientas de software que incluso resuelven el problema de forma gráfica ocultando la complejidad del mismo. En nuestro caso hemos optado por una solución intermedia. Antes de experimentar la elaboración de un software especifico para el problema que implica un mayor tiempo de desarrollo, se ha pensado en el empleo de software open source con capacidad de resolver problemas de optimización combinatoria. En este ámbito las herramientas más interesantes han sido *Optaplanner*, *Google Optimization Tools* y *Jsprit*. Entre todas ella, quizás la más popular y extendida es *Optaplanner* ya que permite resolver una gran cantidad de problemas no solo los relacionados con la teoría de grafos como en este caso. También el hecho de que permita construir diversos algoritmos metaheuristicos hacen que optemos por esta opción. No obstante, aunque finalmente no se han utilizado, es interesante explicar brevemente el funcionamiento de las otras herramientas.

*Optaplanner* puede definirse como un *solver,* de satisfacción de restricciones. Permite optimizar problemas que requieren planificar la provisión de determinados productos u servicios con un número limitado de recursos. Entre ellos se incluyen: Vehicle routing, Employee rostering, Cloud optimization, Bin packing, etc. Permite combinar distintos heurísticos y metaheuristicos como SAN o TS, descritos anteriormente y de esta forma conseguir un potente motor de búsqueda de soluciones factibles. Se trata de una herramienta ligera (≈140mb incluyendo ejemplos, *datasets* y documentación), embebida y desarrollada en Java™.Permite trabajar sobre cualquier JVM y es compatible con *Stándard Java*, *Enterprise Java*, y todos los lenguajes JVM. Su rendimiento está más que probado ya que incluso ha competido con investigadores obteniendo muy buenos resultado como en [4.1] y [4.2].

Pues bien, la herramienta escogida para implementar el algoritmo de resolución VRP, tal y como se menciono anteriormente, fue *Optaplanner.* Antes de describir como se lleva a cabo la resolución del problema VRP en este solver, es importante realizar una breve descripción de la estructura y funcionamiento general de la herramienta. Por ello, para resolver cualquier problema de optimización con *Optaplanner*, es necesario:

1. **Modelar el problema de optimización.** Se trata de modelar la clase Java que representará el problema de optimización así como la solución. El dataset que leemos con los datos de entrada del problema necesita ser encapsulado en una clase Java para poder resolverlo con la interfaz Solver. Pues bien, la clase Java que representan tanto la solución como el problema se referencia con la anotación @PlanningSolution
2. **Construir el solver**. A partir de una configuración determinada del solver (bien en XML o a través de una API Java) con la cual se configura la clase SolverFactory, consiste en construir una instancia de la interfaz Solver a partir de la clase SolverFactory .Por comodidad, trabajamos con ficheros de configuración en formato XML. Estos se estructuran en tres partes bien diferenciadas: modelo de datos, función objetivo y configuración del algoritmo de optimización.
3. **Cargar un dataset.** *Input* de datos del problema agregado a partir de un fichero XML u archivo con extensión .vrp.
4. **Resolver el problema.** A través del método .solve de la interfaz Solver , el cual devuelve la mejor solución encontrada para una instancia del problema de optimización

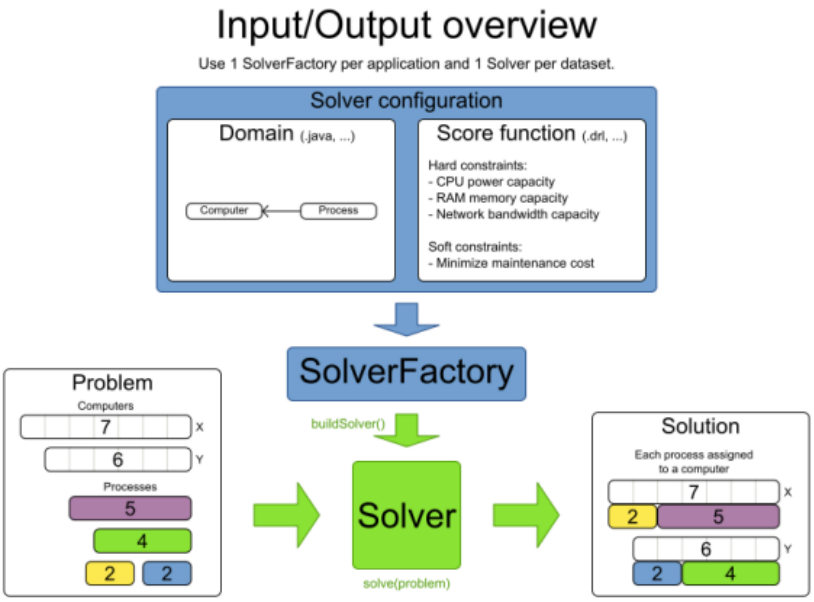



**Figura 12.** Arquitectura Optaplanner

### 5.2.1 Modelado del Problema

Cuando observamos el dataset del problema (*input*), se suele tener una intuición a cerca de las clases en las cuales podemos dividir el dominio de nuestro problema: Clientes, Vehículos, etc. **Figura 13** muestra el diagrama de clases UML (*Unified Modeling Language*) que describe el problema VRP en el solver Optaplanner. Antes de preguntarnos por los atributos de cada clase y su función, es conveniente entender la relación entre las clases. Por ejemplo tal y como se definió el problema en el **Capitulo 3**, sabemos que un cliente o punto de entrega no puede tener asignado dos ubicaciones distintas. También que un vehículo tiene asignado un único punto de partida y que el propio *depot* también tiene asignada una localización. Pues bien, todas estas asociaciones y otras más, que en la mayoría de los casos solo tiene sentido plantearlas en un sentido (unidireccional*)*, quedan plasmadas en **Figura 13 .**Pues bien, aparte de como se relacionan las distintas clases entres si, para entender como se ha realizado el modelado de datos es necesario definir algunos conceptos y extender otros ya mencionados. Esto es:

- *Planning Entity:* Se trata de un *JavaBean* que va cambiando durante la optimización del problema. Por ejemplo un cliente que requiere cierta demanda de un producto y que esta se deba entregar en un cierto intervalo de tiempo .Este tipo de clases deben ser anotadas con @PlanningEntity.. Normalmente suele existir una única clase de este tipo aunque pueden existir varias como en este caso (Customer y StandStill).
- *Problem Fact:* Se trata de un *JavaBean* que no cambia durante la optimización del problema. Por ejemplo, un vehículo (modelado a través de la clase Vehicle) tiene asignada una Capacidad y un punto de salida o *depot* que no varía tras ser asignado.
- *Planning Variable:* Es considerada como una propiedad de una *Planning Entity* que hace referencia a un valor que cambia durante la optimización. Existen dos tipos *Genuine* o *Shadow*
- *Planning Solution:* Tal y como comentamos, se trata de la clase que representa el *input* del problema y que contiene todas las *Planning Entities*. La clase que actúa de *Planning Solution* es *Vehicle Routing Solution* .Este tipo de clases deben anotarse con @PlanningSolution

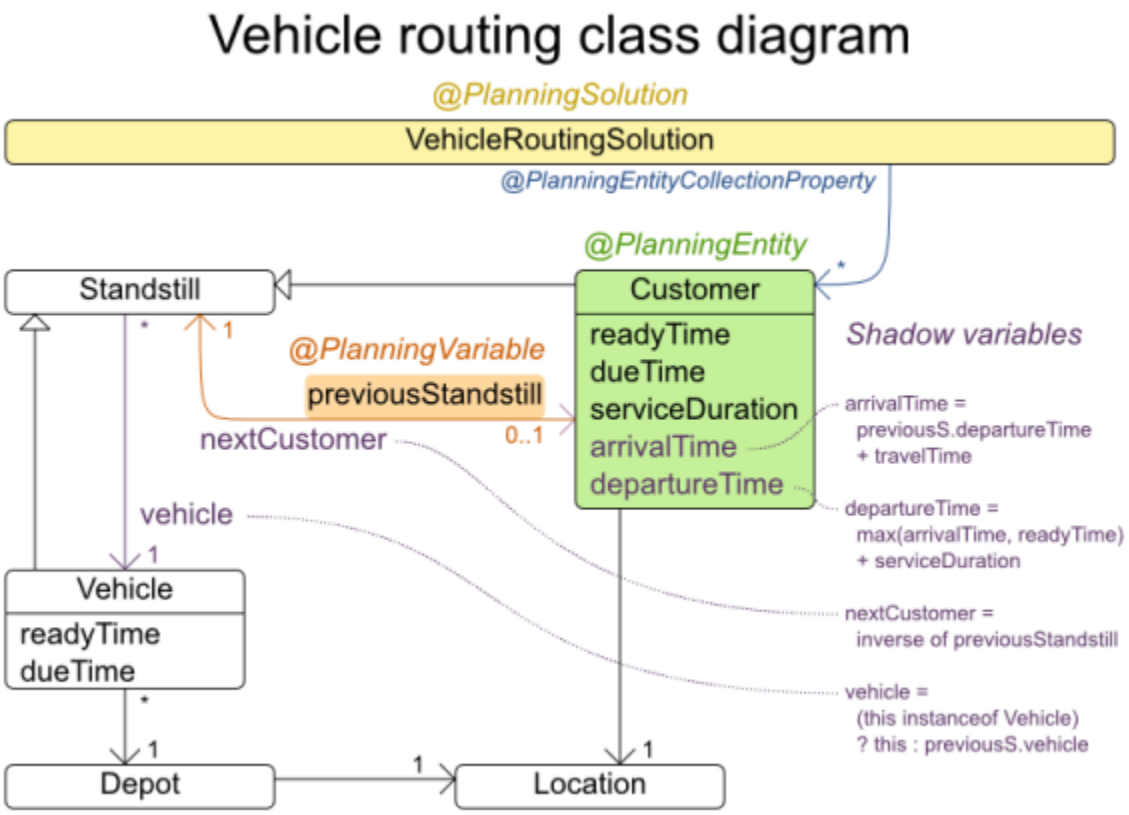



**Figura 13**. Diagrama de Clases UML para VRP

*Optaplanner* permite ayudar a los algoritmos a encontrar una mejor solución En ocasiones es posible saber que *planning entities* son más complejas (p ej.: las últimas ciudades recorridas por el vehículo). En estas situaciones es posible indicar en las *planning entities* cierta información que permitan al algoritmo aprovechar la información del modelado de datos.

```
@PlanningEntity(difficultyWeightFactoryClass = DepotAngleCustomerDifficultyWeightFactory.class)
@XStreamAlias("VrpCustomer")
@XStreamInclude({TimeWindowedCustomer.class})
public class Customer extends AbstractPersistable implements Standstill {
```

**Figura 14**. Planning entities difíciles

Otro aspecto importante, es la diferencia entre los tipos de *planning variable*. Se considera *Genuine* a la *planning variable* "estándar". Como podemos observar en **Figura 13** la *planning entity* Customer posee una *planning variable* denominada previousStandstill que hace referencia a la *planning entity* Standstill. Este tipo de propiedades se denominan Genuine. También puede ocurrir, y el problema que nos atañe es un claro ejemplo de ello que a partir del valor de una *planning variable* se puede deducir el valor de "otras". Estas otras *planning variables* indirectas, se denominan *Shadow* variable. En nuestro caso este hecho se produce al incorporar las restricciones temporales al problema. El tiempo en el que un vehículo llega a un cliente/destino puede calcularse en función del cliente previo visitado por ese vehículo y la distancia entre ambas localizaciones.

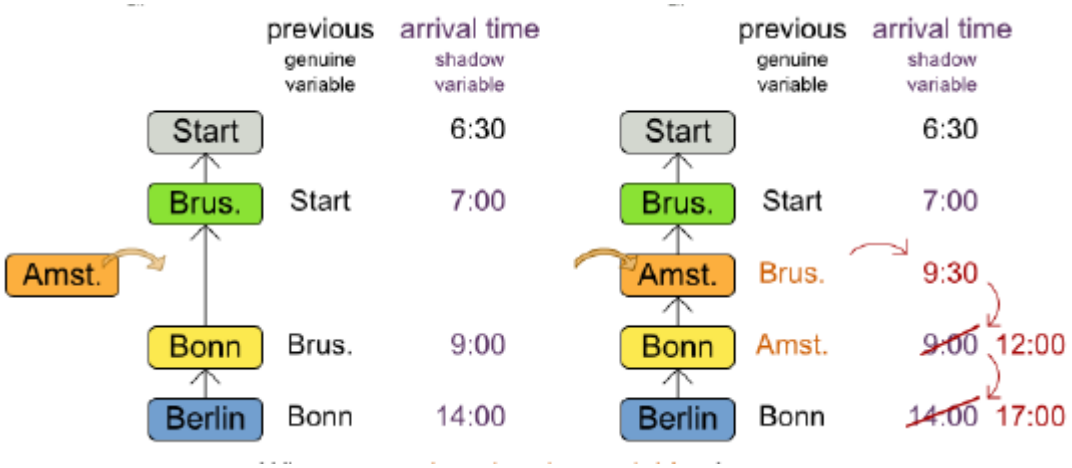



**Figura 15**. Funcionamiento *shadow variables*

Llegados a este punto es importante destacar la función de la interfaz Standstill. Pues bien, su uso está relacionado con la forma que tiene *Optaplanner* de implementar los problemas de encaminamiento en grafos como VRP o TSP. Con el objetivo de reducir de forma considerada el espacio de soluciones se propone una forma de relación entre *planning entities* encadenada **Figura 16** muestra el proceso encadenado .Dicho esquema se compone de varias *planning entities* relacionadas entre sí que finalizan en un *problem fact* denominado *anchor*. En nuestro caso el *anchor* se corresponde con la clase Vehicle mientras que el *plannning entity* hace referencia a la clase Customer como mencionábamos anteriormente. Pues bien , Standstiill es la interfaz implementada por Customer y Vehicle para poder mantener esta relación encadenada .De esta forma se consigue una cierta interacción entre los clientes de un mismo recorrido , facilitando el cálculo de los costes de una determinada ruta

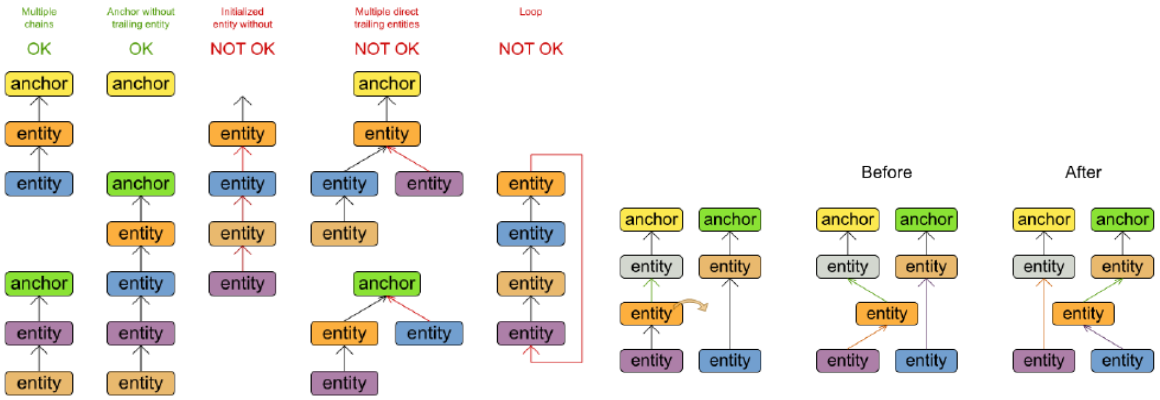



**Figura 16.**Proceso de encadenamiento entre entidades

Además del proceso de encadenamiento, podemos observar como el cambio de una *planning entity* puede implicar hasta 2 correcciones en la cadena.

### 5.2.2 Construcción del Solver

Pues bien, junto con el modelado de datos, el aspecto más importante en *Optaplanner* es la configuración del solver. Tal y como se menciono anteriormente, se ha optado por especificar la configuración del solver por medio de un fichero XML. **Figura 17** muestra el esquema típico de un fichero de configuración del solver.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<solver>
  <!-- Define the model -->
  <solutionClass>org.optaplanner.examples.nqueens.domain.NQueens</solutionClass>
  <entityClass>org.optaplanner.examples.nqueens.domain.Queen</entityClass>

  <!-- Define the score function -->
  <scoreDirectorFactory>
    <scoreDrl>org/optaplanner/examples/nqueens/solver/nQueensScoreRules.drl</scoreDrl>
  </scoreDirectorFactory>

  <!-- Configure the optimization algorithms (optional) -->
  <termination>
    ...
  </termination>
  <constructionHeuristic>
    ...
  </constructionHeuristic>
  <localSearch>
    ...
  </localSearch>
</solver>
```

**Figura 17.** Esqueleto fichero configuración XML

En él se pueden distinguir claramente tres partes: Definición del modelo, Definición de la función de costes y la configuración del heurístico .En la sección para la definición del modelo, se hace referencia al modelado de las clases. En concreto, se referencia *la Planning Solution* así como a las posibles *Planning Entities* del Problema.

En la segunda sección se hace referencia a la forma de estimar los costes de la solución en cada iteración. Dicha estimación se lleva a cabo mediante Drools. (.drl) .Se trata de un motor de reglas de negocio desarrollado por Red Hat. En apartado 5.2.2.1, describiremos en profundidad el mismo

Por último describiremos la parte fundamental de todo este proyecto, la configuración del heurístico. En concreto debido a que *Optaplanner* puede presentar una gran variedad de *settings*, nos centraremos en los aspectos de la mejor solución encontrada.

### *5.2.2.1 Drools*

La forma de discernir entre distintas soluciones es sin duda comparar su coste. Para ello *Optaplanner* ofrece varias opciones como pueden ser emplear clases Java u aplicar reglas para el cálculo. Esta última opción es la recomendada y se implementa a través del motor de reglas Drools .La principal ventaja es su flexibilidad ya que permite establecer las restricciones del problema de forma independiente y de esta forma facilita que puedan añadirse o eliminar restricciones del problema de una forma agil. Otro aspecto importante es que incorpora cálculos incrementales, lo cual permite conocer de una forma rápida si una solución mejora la anterior, simplemente comparando los deltas de mejora entre ambas soluciones en lugar de evaluar de nuevo el coste de toda la solución. Esta última cualidad es costosa de implementar directamente en JAVA ya que requiere gran cantidad de operaciones y una construcción del software muy escalable o modular, algo que en ocasiones resulta difícil. De ahí el hecho de utilizar Drools como método predeterminado. De existir alguna desventaja podríamos atribuirla a la curva de aprendizaje ya que esta puede ser alta.

Pues bien, para poder utilizar el motor de reglas Drools en Optaplanner basta con añadir un elemento scoreDrl en el fichero de configuración del solver tal y como aparece en la imagen anterior. Dicho elemento hace referencia al fichero .drl definido en el *classpath.* El contenido del dichero .drl es en el cual definimos las restricciones del problema. **Figura 18** muestra estructura típica de un archivo .drl.

```
rule "                    "
    when

    then

end
```

**Figura 18.** Estructura archivo .drl

En el podemos apreciar la distinción entre *hard constraints* y *Soft constraints* .Las primeras hacen referencia a las restricciones que deben cumplirse para que la solución sea factible. Es decir, en ningún caso la solución aportada puede romper este tipo de reglas. En nuestro caso las reglas más estrictas son las relacionadas con las restricciones de capacidad así como las restricciones de tiempo. Por su parte las soft constrainst son aquellas que pueden romperse en alguno momento de las iteraciones del algoritmo aunque se tiene a que sean minimizadas. Son las restricciones relacionadas con la distancia recorrida Debemos prestar atención especial a la variable *kcontext* y al método *scoreHolder*. La primera se trata de una variable global de Drools que permite acceder al contexto general de un objeto determinado o instancia lo cual puede ser empleado para obtener acceso al conjunto de variables. Por su parte score holder es empleado para dar un valor al coste de una solución. Es decir, se indica cuanto mejor son las soluciones aceptadas por medio de un valor de penalización. Como se puede observar se suele utilizar el símbolo $ para diferenciar las variables de las reglas de las variables JAVA del modelo de datos. En nuestro caso particular se hace uso también del elemento accumulate para realizar iteraciones en una lista de objetos de forma más ágil. Esta funcionalidad fue incoporada en las últimas versiones de Optaplanner. Pues bien, en la **Figura 19** podemos observar la configuración completa drools.

```
/*
 * Copyright 2012 Red Hat, Inc. and/or its affiliates.
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing,
software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied.
 * See the License for the specific language governing permissions
and
 * limitations under the License.
 */

package org.optaplanner.examples.vehiclerouting.solver;
    dialect "java"
```

```
import
org.optaplanner.core.api.score.buildin.hardsoftlong.HardSoftLongScor
eHolder;

import org.optaplanner.examples.vehiclerouting.domain.Depot;
import
org.optaplanner.examples.vehiclerouting.domain.location.Location;
import org.optaplanner.examples.vehiclerouting.domain.Vehicle;
import
org.optaplanner.examples.vehiclerouting.domain.VehicleRoutingSolutio
n;
import org.optaplanner.examples.vehiclerouting.domain.Customer;
import
org.optaplanner.examples.vehiclerouting.domain.timewindowed.TimeWind
owedDepot;
import
org.optaplanner.examples.vehiclerouting.domain.timewindowed.TimeWind
owedVehicleRoutingSolution;
import
org.optaplanner.examples.vehiclerouting.domain.timewindowed.TimeWind
owedCustomer
import org.optaplanner.examples.vehiclerouting.domain.Vehicle;

global HardSoftLongScoreHolder scoreHolder;

//
################################################################################
########
// Hard constraints
//
################################################################################
########

rule "vehicleCapacity"
    when
        $vehicle : Vehicle($capacity : capacity)
        accumulate(
            Customer(
                vehicle == $vehicle,
                $demand : demand);
            $demandTotal : sum($demand);
            $demandTotal > $capacity
        )
    then
        scoreHolder.addHardConstraintMatch(kcontext, $capacity -
$demandTotal);
end

//
################################################################################
########
// Soft constraints
//
################################################################################
```

```
########

rule "distanceToPreviousStandstill"
    when
        $customer : Customer(previousStandstill != null,
$distanceFromPreviousStandstill : distanceFromPreviousStandstill)
    then
        scoreHolder.addSoftConstraintMatch(kcontext, -
$distanceFromPreviousStandstill);
end

rule "distanceFromLastCustomerToDepot"
    when
        $customer : Customer(previousStandstill != null)
        not Customer(previousStandstill == $customer)
    then
        Vehicle vehicle = $customer.getVehicle();
        scoreHolder.addSoftConstraintMatch(kcontext, -
$customer.getDistanceTo(vehicle));
end


//
############################################################################
########
// TimeWindowed: additional hard constraints
//
############################################################################
########

rule "arrivalAfterDueTime"
    when
        TimeWindowedCustomer(dueTime < arrivalTime, $dueTime :
dueTime, $arrivalTime : arrivalTime)
    then
        scoreHolder.addHardConstraintMatch(kcontext, $dueTime -
$arrivalTime.longValue());
end

// Score constraint arrivalAfterDueTimeAtDepot is a built-in hard
constraint in VehicleRoutingImporter
```

**Figura 19.** Archivo de configuración .drl para vehicle routing problem

El formato en el que se suele expresar el coste es coste_hard/coste_soft. El primer valor, debe ser cero para que la solución sea factible.

### *5.2.2.2 Construcción del heurístico*

En este apartado se lleva a cabo la especificación del metaheuristico empleado en la resolución del problema así como las distintas configuraciones que este puede presentar. Para no extender demasiado el documento, únicamente vamos a describir la configuración del meta heurístico para tabu search ya que es el algoritmo que mejor resultados proporciona. En concreto, la configuración de este

metaheuristico que mejor datos a reportado ha sido:

```
<constructionHeuristic>
  <constructionHeuristicType>FIRST_FIT_DECREASING</constructionHeuristicType>
</constructionHeuristic>
<localSearch>
  <unionMoveSelector>
    <changeMoveSelector/>
    <swapMoveSelector/>
    <subChainChangeMoveSelector>
      <selectReversingMoveToo>true</selectReversingMoveToo>
    </subChainChangeMoveSelector>
    <subChainSwapMoveSelector>
      <selectReversingMoveToo>true</selectReversingMoveToo>
    </subChainSwapMoveSelector>
  </unionMoveSelector>
  <acceptor>
    <entityTabuSize>7</entityTabuSize>
  </acceptor>
  <forager>
    <acceptedCountLimit>1000</acceptedCountLimit>
  </forager>
</localSearch>
```

**Figura 20**.Mejor configuración para TS

### 5.2.3 Carga del dataset

Pues bien, como indicábamos en la introducción del capítulo, al igual que el modelado de clases y la configuración del solver es necesario utilizar un input valido para el problema. Como *Optaplanner* no permite trabajar con varios dataset simultáneos, indicamos los dataset empleados .Esto es diferenciando entre aquellos que incorporan incompatibilidad de mercancías con aquellos que no. De cara a no extender el documento, no aparece el apartado EDGE_WEIGHT_SECTION

Dataset incompatibilidad mercancías (asumiendo riesgo  entre C1y C3)

```
NAME: 5googleSISPERTtw-road-time-n33-k9
COMMENT: Generated for OptaPlanner Examples with Google Matrix API by Luis
Caballero.
TYPE: CVRPTW
DIMENSION: 34
EDGE_WEIGHT_TYPE: EXPLICIT
EDGE_WEIGHT_FORMAT: FULL_MATRIX
EDGE_WEIGHT_UNIT_OF_MEASUREMENT: km
CAPACITY: 3300
NODE_COORD_SECTION
0 37.9922399 -1.1306544000000258 Murcia
55 50.0035483 12.089981599999987 Marktredwitz1
110 49.3284557 11.024709799999982 Schwabach
165 52.4240211 13.820433600000001 Grunheide
220 50.9874835 13.184522499999957 Striegistal
275 48.2748206 8.849529800000028 Balingen
330 52.0302285 8.532470800000056 Bielefeld
385 46.87334   15.650290000000041 Gonitz
440 48.5788725 7.816082100000017 Kehl
495 53.4784568 12.421690000000012 Malchow
```

```
550 51.1276976 9.549567700000011 Melsungen
605 48.473451  7.949801699999966 Offenburg
715 48.0252373 12.555257500000039 Trostberg
770 53.5259098 10.84545300000002 Valluhn
825 48.2983793 11.62120630000004 Eching
880 48.810743  11.36910069999999 Gaimersheim
935 49.5077027 8.605912799999942 Heddesheim
990 54.0729431 9.984015799999952 Neumunster
1045 48.0272797 8.60376969999993 Tuningen
1100 51.130518  13.57792280000001 Coswig
1155 48.2769092 12.576325300000008 Erharting
1210 48.050783  10.87035149999997 Landsberg
1265 50.6258878 7.0319507000000385 Meckenheim
1320 48.8777333 12.580153800000062 Straubing
1375 51.7710340 8.317237699999964 Langenberg
1430 52.2740302 9.372417000000041 Lauenau
1485 53.2549953 8.117583500000023 Wiefelstede
1540 50.0218713 10.277437100000043 Gochsheim
1595 52.7650991 9.595746100000042 Hodenhagen
1650 49.2774456 7.111899499999936 StIngbert
1705 49.1459677 9.32068019999997 Ellhofen
1760 52.5200065 13.404953999999975 Berlin
1815 48.050783  10.87035149999997 Landsberg
1870 51.2974008 13.738835600000016 Thiendorf
EDGE_WEIGHT_SECTION
--------------------------------------
DEMAND_SECTION
0 0 0 2000000000 0
55 149 0 3700000 600000
110 2095 0 5400000 600000
165 729 0 5400000 600000
220 1356 0 5400000 600000
275 67 0 5400000 600000
330 95 0 3700000 600000
385 1 0 3700000 600000
440 200 0 3700000 600000
495 1 0 3700000 600000
550 1 0 3700000 600000
605 1 0 3700000 60000
715 147 0 3700000 600000
770 1 0 3700000 600000
825 447 0 3700000 600000
880 1 0 3700000 600000
935 240 0 3700000 600000
990 267 0 5400000 600000
1045 400 0 3700000 600000
1100 1 0 3700000 600000
1155 1200 0 5400000 600000
1210 100 0 3700000 600000
1265 500 0 3700000 600000
1320 1 0 3700000 600000
1375 914 0 5400000 600000
1430 180 0 5400000 600000
1485 960 0 5400000 600000
1540 57 0 5400000 600000
1595 700 0 3700000 600000
1650 1 0 3700000 600000
1705 516 0 5400000 600000
1760 212 0 3700000 600000
```

```
1815 346 0 5400000 600000
1870 1 0 5400000 600000
DEPOT_SECTION
0
-1
EOF
```

**Figura 21.** Dataset incompatibilidad mercancías (riesgo C1 y C3)

Dataset incompatibilidad mercancías (asumiendo riego entre C2 y C3)

```
NAME: googleSISPERTtw-road-time-n33-k9
COMMENT: Generated for OptaPlanner Examples with Google Matrix API by Luis
Caballero.
TYPE: CVRPTW
DIMENSION: 34
EDGE_WEIGHT_TYPE: EXPLICIT
EDGE_WEIGHT_FORMAT: FULL_MATRIX
EDGE_WEIGHT_UNIT_OF_MEASUREMENT: km
CAPACITY: 3300
NODE_COORD_SECTION
0 37.9922399 -1.1306544000000258 Murcia
55 50.0035483 12.089981599999987 Marktredwitz1
110 49.3284557 11.024709799999982 Schwabach
165 52.4240211 13.820433600000001 Grunheide
220 50.9874835 13.184522499999957 Striegistal
275 48.2748206 8.849529800000028 Balingen
330 52.0302285 8.532470800000056 Bielefeld
385 46.87334   15.650290000000041 Gonitz
440 48.5788725 7.816082100000017 Kehl
495 53.4784568 12.421690000000012 Malchow
550 51.1276976 9.549567700000011 Melsungen
605 48.473451  7.949801699999966 Offenburg
715 48.0252373 12.555257500000039 Trostberg
770 53.5259098 10.84545300000002 Valluhn
825 48.2983793 11.62120630000004 Eching
880 48.810743  11.36910069999999 Gaimersheim
935 49.5077027 8.605912799999942 Heddesheim
990 54.0729431 9.984015799999952 Neumunster
1045 48.0272797 8.60376969999993 Tuningen
1100 51.130518  13.57792280000001 Coswig
1155 48.2769092 12.576325300000008 Erharting
1210 48.050783  10.87035149999997 Landsberg
1265 50.6258878 7.0319507000000385 Meckenheim
1320 48.8777333 12.580153800000062 Straubing
1375 51.7710340 8.317237699999964 Langenberg
1430 52.2740302 9.372417000000041 Lauenau
1485 53.2549953 8.117583500000023 Wiefelstede
1540 50.0218713 10.277437100000043 Gochsheim
1595 52.7650991 9.595746100000042 Hodenhagen
1650 49.2774456 7.111899499999936 StIngbert
1705 49.1459677 9.32068019999997 Ellhofen
1760 52.5200065 13.404953999999975 Berlin
1815 48.050783  10.87035149999997 Landsberg
1870 51.2974008 13.738835600000016 Thiendorf
EDGE_WEIGHT_SECTION
--------------------------
DEMAND_SECTION
0 0 0 2000000000 0
55 975 0 3700000 600000
110 1 0 5400000 600000
```

```
165 1050 0 5400000 600000
220 50 0 5400000 600000
275 1278 0 5400000 600000
330 1 0 3700000 600000
385 200 0 3700000 600000
440 1 0 3700000 600000
495 328 0 3700000 600000
550 125 0 3700000 600000
605 458 0 3700000 60000
715 100 0 3700000 600000
770 250 0 3700000 600000
825 1 0 3700000 600000
880 373 0 3700000 600000
935 100 0 3700000 600000
990 1000 0 5400000 600000
1045 1 0 3700000 600000
1100 500 0 3700000 600000
1155 1 0 5400000 600000
1210 448 0 3700000 600000
1265 1 0 3700000 600000
1320 511 0 3700000 600000
1375 1 0 5400000 600000
1430 565 0 5400000 600000
1485 200 0 5400000 600000
1540 940 0 5400000 600000
1595 1 0 3700000 600000
1650 736 0 3700000 600000
1705 105 0 5400000 600000
1760 600 0 3700000 600000
1815 458 0 5400000 600000
1870 700 0 5400000 600000
DEPOT_SECTION
0
-1
EOF
```

**Figura 22**. Dataset incompatibilidad mercacias (riesgo C2 y C3)

Dataset compatibilidad mercancías

```
NAME: allgoogleSISPERTtw-road-time-n33-k9
COMMENT: Generated for OptaPlanner Examples with Google Matrix API by Luis
Caballero.
TYPE: CVRPTW
DIMENSION: 34
EDGE_WEIGHT_TYPE: EXPLICIT
EDGE_WEIGHT_FORMAT: FULL_MATRIX
EDGE_WEIGHT_UNIT_OF_MEASUREMENT: km
CAPACITY: 3300
NODE_COORD_SECTION
0 37.9922399 -1.1306544000000258 Murcia
55 50.0035483 12.089981599999987 Marktredwitz1
110 49.3284557 11.024709799999982 Schwabach
165 52.4240211 13.820433600000001 Grunheide
220 50.9874835 13.184522499999957 Striegistal
275 48.2748206 8.849529800000028 Balingen
330 52.0302285 8.532470800000056 Bielefeld
385 46.87334   15.650290000000041 Gonitz
440 48.5788725 7.816082100000017 Kehl
495 53.4784568 12.421690000000012 Malchow
```

```
550 51.1276976 9.549567700000011 Melsungen
605 48.473451  7.949801699999966 Offenburg
715 48.0252373 12.555257500000039 Trostberg
770 53.5259098 10.84545300000002 Valluhn
825 48.2983793 11.62120630000004 Eching
880 48.810743  11.36910069999999 Gaimersheim
935 49.5077027 8.605912799999942 Heddesheim
990 54.0729431 9.984015799999952 Neumunster
1045 48.0272797 8.60376969999993 Tuningen
1100 51.130518  13.57792280000001 Coswig
1155 48.2769092 12.576325300000008 Erharting
1210 48.050783  10.87035149999997 Landsberg
1265 50.6258878 7.0319507000000385 Meckenheim
1320 48.8777333 12.580153800000062 Straubing
1375 51.7710340 8.317237699999964 Langenberg
1430 52.2740302 9.372417000000041 Lauenau
1485 53.2549953 8.117583500000023 Wiefelstede
1540 50.0218713 10.277437100000043 Gochsheim
1595 52.7650991 9.595746100000042 Hodenhagen
1650 49.2774456 7.111899499999936 StIngbert
1705 49.1459677 9.32068019999997 Ellhofen
1760 52.5200065 13.404953999999975 Berlin
1815 48.050783  10.87035149999997 Landsberg
1870 51.2974008 13.738835600000016 Thiendorf
EDGE_WEIGHT_SECTION
-------------------
DEMAND_SECTION
0 0 0 2000000000 0
55 1124 0 3700000 600000
110 2095 0 5400000 600000
165 1779 0 5400000 600000
220 1406 0 5400000 600000
275 1345 0 5400000 600000
330 95 0 3700000 600000
385 200 0 3700000 600000
440 200 0 3700000 600000
495 328 0 3700000 600000
550 125 0 3700000 600000
605 467 0 3700000 60000
715 247 0 3700000 600000
770 250 0 3700000 600000
825 447 0 3700000 600000
880 373 0 3700000 600000
935 340 0 3700000 600000
990 1267 0 5400000 600000
1045 400 0 3700000 600000
1100 500 0 3700000 600000
1155 1200 0 5400000 600000
1210 548 0 3700000 600000
1265 500 0 3700000 600000
1320 511 0 3700000 600000
1375 914 0 5400000 600000
1430 745 0 5400000 600000
1485 1160 0 5400000 600000
1540 997 0 5400000 600000
1595 700 0 3700000 600000
1650 736 0 3700000 600000
1705 621 0 5400000 600000
1760 812 0 3700000 600000
```

```
1815 804 0 5400000 600000
1870 700 0 5400000 600000
DEPOT_SECTION
0
-1
EOF
```

**Figura 23**. Dataset compatibilidad mercancías

Como puede observarse se han seguido muchas de las recomendaciones de formato de *Optaplanner* como por ejemplo la utilización de factores de multiplicación a la hora de expresar la demanda de un cliente y los tiempos de la ventana temporal validos para cada localización

## 5.3 Etapa final

### 5.3.1 Representación de soluciones

En esta última etapa se realiza la representación grafica de las soluciones. Para ello se hace uso de un script python para visualizar las soluciones. De esta forma la solución emitida por *Optaplanner* en formato XML puede ser analizada y tratado para una mejor comprensión de los resultados. A continuación, se adjunta el código python empleado:

```
import numpy as np
import xml.etree.ElementTree as ET
#tree = ET.parse('C:\Users\Luis\Desktop\solve-sispert-cvrp-33customers.xml')
#tree = ET.parse('C:\\Users\\Luis\\Desktop\\belgium-road-time-n50-k10.xml')

#tree = ET.parse('C:\\Users\\Luis\\Desktop\\SISPERT-road-time-n33-k9_2.xml')

tree = ET.parse('C:\\Users\\Luis\\Desktop\\input\\solvedC1googleSISPERTtw-road-time-n33-k9.xml')
root = tree.getroot()
#Dictionary with xml references values
location2 = {3:'Murcia',
        5:'Trostberg',
        5:'Marktredwitz',
        7:'Schwabach',
        9:'Grunheide',
        11:'Striegistal',
        13:'Balingen',
        15:'Bielefeld',
        17:'Gonitz',
        19:'Kehl',
        21:'Malchow',
        23:'Melsungen',
        25:'Offenburg',
        27:'Trostberg',
        29:'Valluhn',
        31:'Eching',
        33:'Gaimersheim',
        35:'Heddesheim',
        37:'Neumunster',
        39:'Tuningen',
        41:'Coswig',
        43:'Erharting',
        45:'Landsberg',
        47:'Meckenheim',
        49:'Straubing',
        51:'Langenberg',
        53:'Lauenau',
        55:'Wiefelstede',
        57:'Gochsheim',
        59:'Hodenhagen',
        61:'St. Ingbert',
        63:'Ellhofen',
        65:'Berlin',
        67:'Landsberg2',
        69:'Thiendorf'
}

mat = np.chararray((9,100),itemsize=20)

aux = np.zeros((1000, 1000), dtype=np.int)
aux2 = np.chararray((10,100),itemsize=40)
aux3 = np.zeros((1000, 1000), dtype=np.int)

index=0
```

```
index_customer=0
index2=1


for index3 in range(10):
    for index4 in range (99):
        var = ""
        aux2[index3][index4]=var
#Set Depot Value in all routes
for index in range(10):
    var = "Murcia"
    aux2[index][0]=var

for vehicleList in root.findall('vehicleList'):
    for element in vehicleList.findall('VrpVehicle'):

        for customer in element.iter('nextCustomer'):
            for demand in customer.findall('demand') :
                aux3[index_customer][0]+=int(demand.text)

                print demand.text
            for location in customer.findall('location'):
                reference = location.attrib.get('reference');
                print reference
                aux[index_customer][index2]=reference
                if(aux[index_customer][index2] in location2):
                    aux2[index_customer][index2]=location2.get(aux[index_customer][index2])
                    print("test")
                    print location2.get(aux[index_customer][index2])
            index2 = index2+1
        index2=1
        index_customer = index_customer +1


index=0
for index in range(10):
    var = aux3[index][0]
    firstpart, secondpart = str(var)[:1],str(var)[2:]
    if secondpart=="":
        aux2[index][8]=firstpart+"/33"
    else:
        aux2[index][8]=firstpart+","+secondpart+"/33"

#Display routes
split1 = str(var).split()
print split1
#Debug
print location2.get(4)
print location2.get(5)
print aux[0][0],aux[0][1],aux[1][0],aux[1][1]
print mat[0][0]
print mat[1][0]
print mat[1][1]
```

**Figura 24.** Código Python representación de soluciones

Con él se consiguen visualizaciones de este tipo:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 0 | Murcia | Gaimersheim | Marktredwitz | Straubing | Erharting | Trostberg | | 33/33 |
| 1 | Murcia | Eching | Striegistal | Coswig | Thiendorf | Gonitz | | 32/33 |
| 2 | Murcia | Offenburg | Valluhn | Malchow | Berlin | Grunheide | | 33/33 |
| 3 | Murcia | Meckenheim | Langenberg | Lauenau | Melsungen | Gochsheim | Kehl | 33/33 |
| 4 | Murcia | | | | | | | 0/33 |
| 5 | Murcia | | | | | | | 0/33 |
| 6 | Murcia | Tuningen | Balingen | Ellhofen | Heddesheim | St. Ingbert | | 33/33 |
| 7 | Murcia | Schwabach | Landsberg | Landsberg2 | | | | 33/33 |
| 8 | Murcia | Bielefeld | Hodenhagen | Neumunster | Wiefelstede | | | 31/33 |

**Figura 25**. Visualización solución tras ejecución script python

También se ha decidido emplear un proyecto web para visualizar las soluciones propuestas. El proyecto web funciona siguiendo el siguiente esquema:

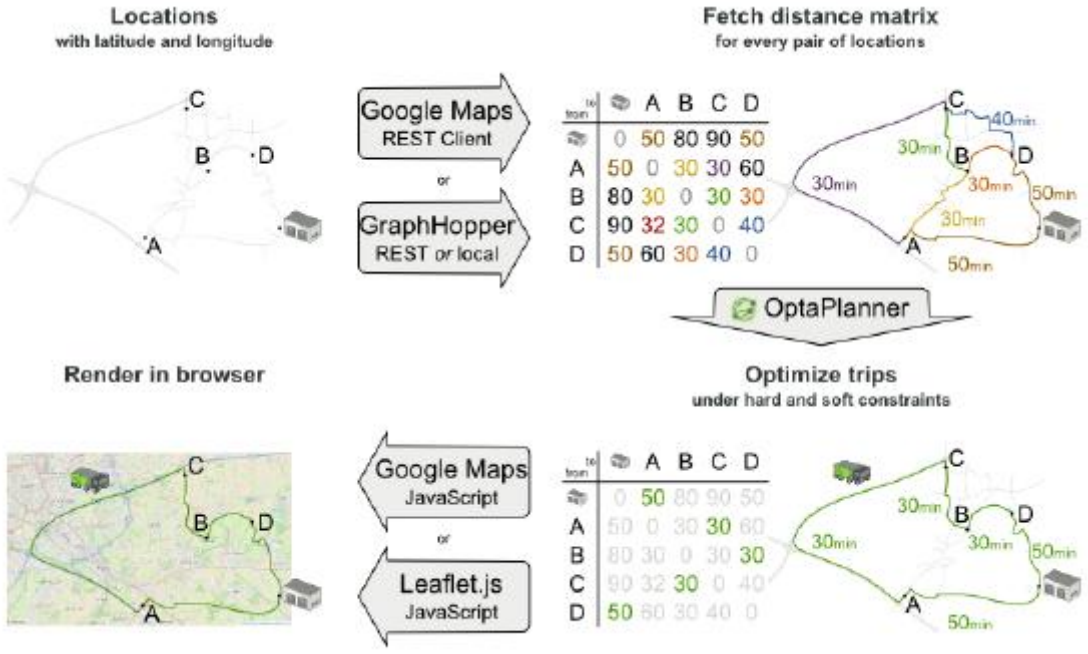



**Figura 26**.Principio de funcionamiento proyecto web en Optaplanner

En nuestro caso se ha decidido implementar un servidor JEE (Wildfly 9) que permite combinar la lógica de *Optaplanner* con tecnologías web de cara a obtener una mejor representación, logrando visualizar soluciones como:

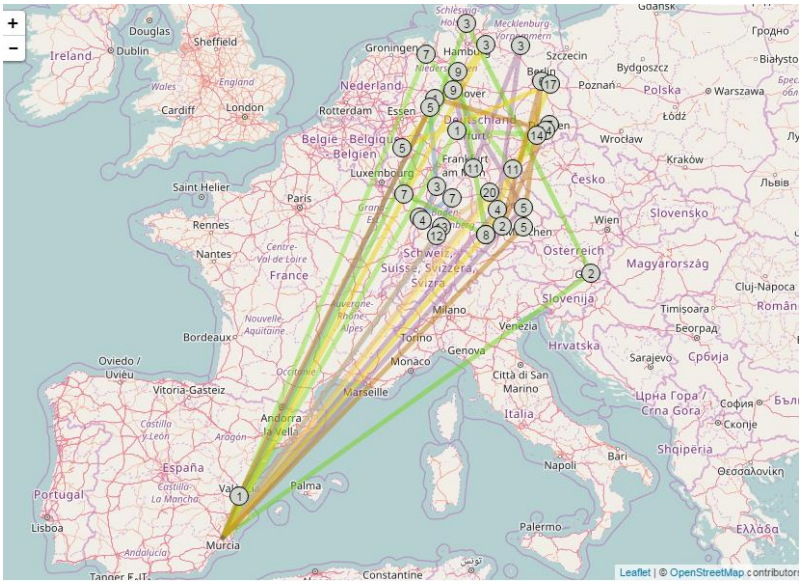



**Figura 27**. Representación web solución en Optaplanner

En anexo aparece descrito el procedimiento para instalar servidor JEE en eclipse y su posteiror integración con Optaplanner

# Capitulo 6. Resultados

## 6.1 Tabla comparativa resultados en función del metaheurístico empleado en Optaplanner

En esta primera etapa de evaluación de resultados vamos a mostrar el proceso seguido para evaluar el algoritmo que mejor se adapta a nuestro problema. Para ello se compararan distintas configuraciones de TS, SA y Late-Acceptance entre otros. En función del rendimiento (tiempo consumido, mejor solución) nos decantamos por utilizar *tabu search* tal y como se comento en el apartado anterior. Pues bien para llevar a cabo esta comprobación hemos empleado dos input distintos. Como *Optaplanner* no permite trabajar con varios dataset de forma simultánea, hemos realizado las pruebas primero con un input en el cual no contemplábamos incompatibilidad de cargas (es decir se asume un riesgo máximo) y otro en el que si, en concreto para mercancías del tipo 1 y 3 .Pues bien los resultados obtenidos han sido los siguientes:

Dataset sin compatibilidad de mercancías

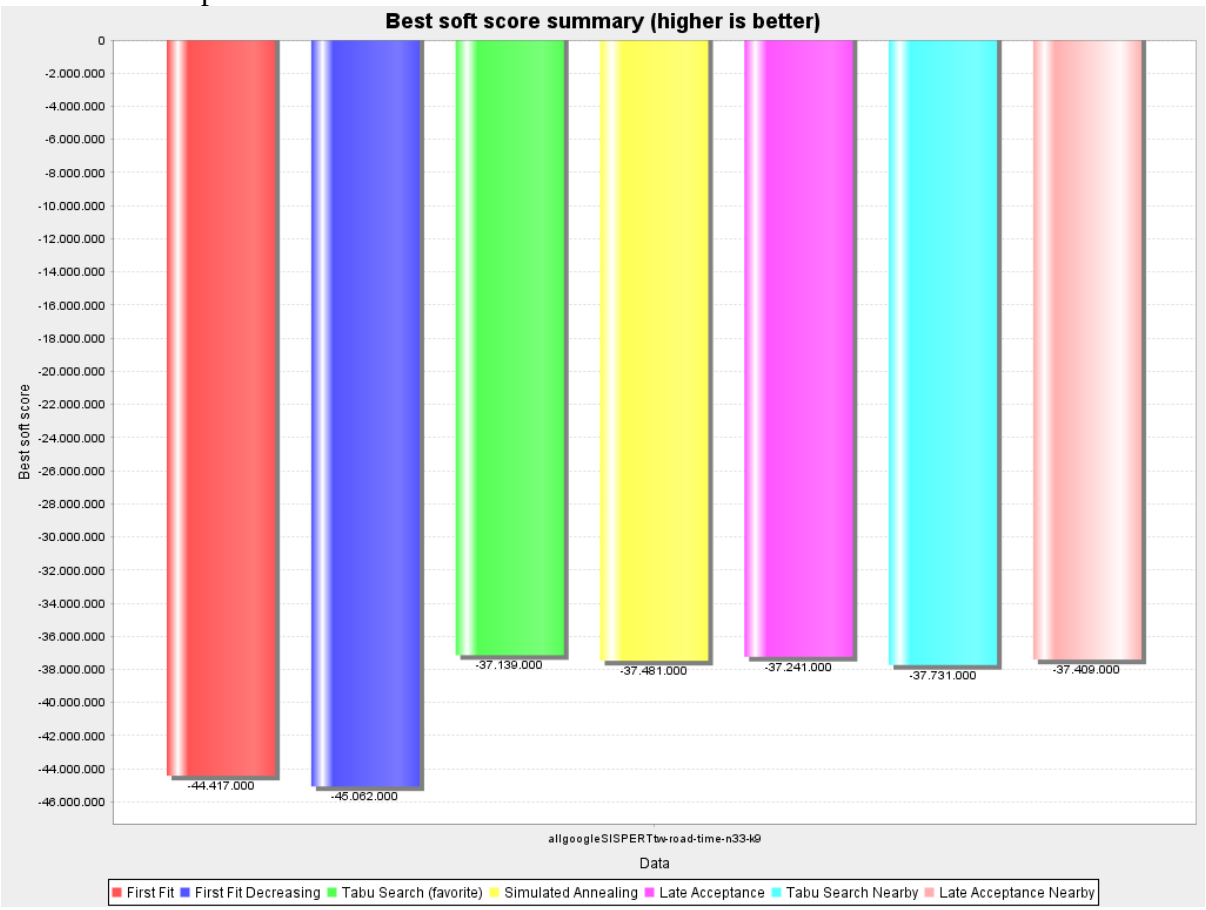



**Figura 28**. Puntuaciónes metaheuristicos utilizados sin compatibilidad de mercancias

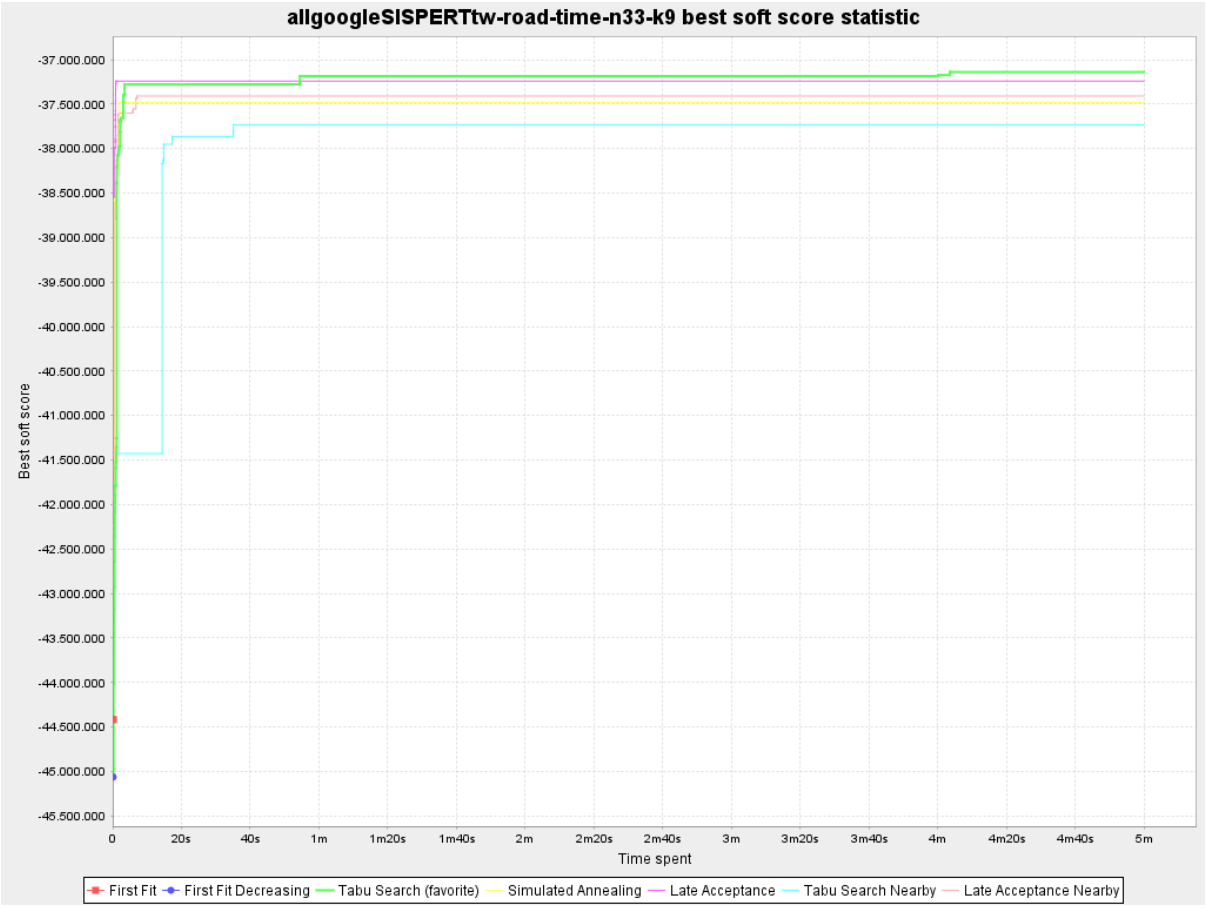



**Figura 29** Tiempo empleado metaheuristicos utilizados sin compatibilidad de mercancias

Dataset con compatibilidad de mercancías:

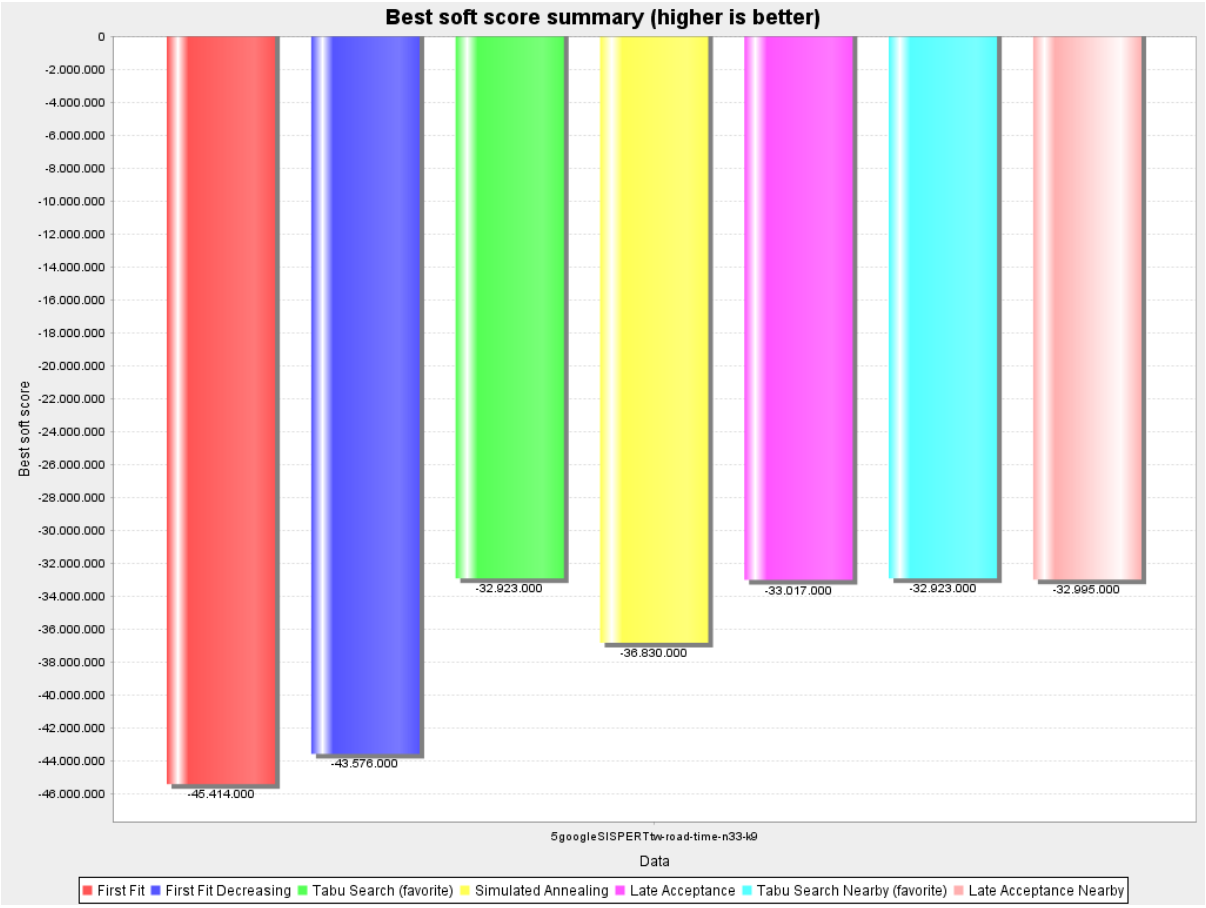



**Figura 30** Puntuaciónes metaheuristicos utilizados con compatibilidad de mercancias

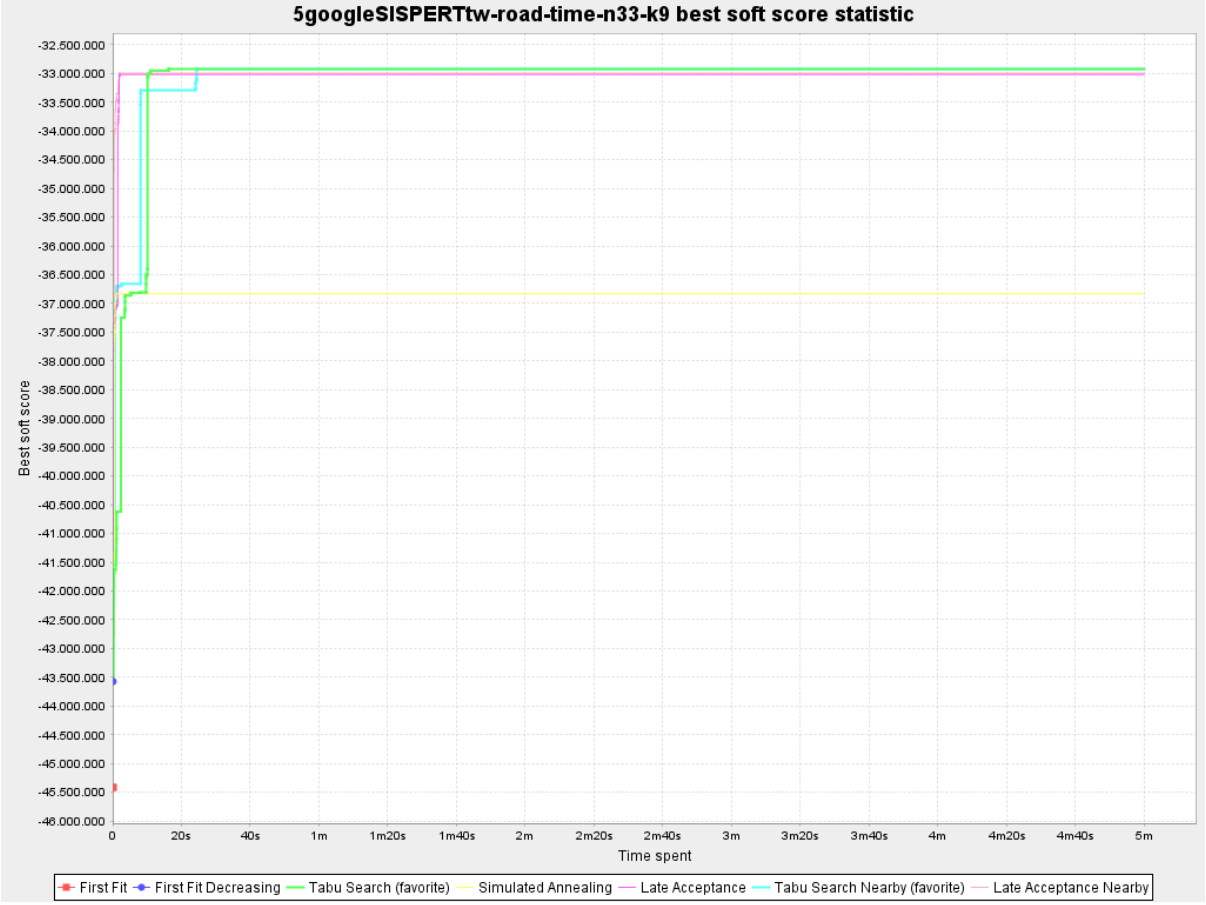



**Figura 31.** Tiempo empleado metaheuristicos utilizados con compatibilidad de mercancias

Como podemos observar en las distintas imágenes TS es el metaheuristico más efectivo, alcanzando el mejor resultado bajo las distintas entradas. Sin embargo, también es cierto que emplea un mayor tiempo en realizar los cálculos.SA, en comparación es más rápido.
Para poner en valor los resultados de nuestro algoritmo podemos pensar en monetizar algunos valores de cara a ilustrar el enorme beneficio que puede representan los resultados obtenidos. Pues bien, antes de aplicar algoritmos para VRP, la empresa, utilizaba un pequeño algoritmo de clustering y mucha experiencia para planificar rutas. Con ello conseguían emplear un mayor número de vehículos, que a la postre es decisivo ya que nuestra intención es minimizar el número de camiones. Pero no solo eso, sino que también recorrían una mayor distancia. Haciendo un cálculo a grandes rasgos, podemos pensar que con lo que cuesta el combustible y lo que consumen los vehículos estamos ante un gran ahorro.

## 6.2 Tabla comparativa mejor metaheuristico con instancias de Solomon (benchmark)

En este apartado vamos a comprobar cómo funciona el algoritmo empleado con otras topologías típicas .Es muy común recurrir a este tipo de *benchmarking* en los distitos papers y artículos científicos relacionados con *vehicle routing problem.* En concreto uno de los más utilizados son las 56 instancias de Solomon [6.1] . Se trata de inputs de aproximadamente 100 clientes categorizados en 6 clases denominadas C1, C2, R1, R2, RC1, RC2. Los problemas de tipo C hacen referencia a que los datos están agrupados o clusterizados ya sea geográficamente o en términos temporales. Por su parte en los problemas de la categoría R los datos se encuentran uniformemente distribuidos .Por último la clase RC se trata de un hibrido entre ambas categorías anteriores. Destacar también que los problemas

C1, R1 y RC1 presentan ventanas temporalas más estreñas en comparación con el resto. Pues bien, en nuestro caso hemos utilizado algunas de las instancias de Solomon, en concreto: C101, R201y RC101.Tabla, muestra comparativa de soluciones obtenidas para dichas instancias en comparación con [6.2] y [3.5] respectivamente.

| | **Publised Best** | **Article(best)** | **Diseño *Optaplanner*** |
|---|---|---|---|
| | NV/TD | NV/TD | NV/TD |
| **C101** | 10/829 | 10/828.937 | 10/828.94 |
| **R201** | 4/1354 | 5/1437,49 | 8/1201,32 |
| **RC101** | 14/1669 | 16/1734,17 | 17/1668.85 |

**Tabla 1.** Comparación rendimiento algoritmo con instancias de Solomon

Tal y como puede apreciarse el empleo de Optaplanner para resolver problamas de Optimización presenta resultados muy similares a algoritmos propios elaborados por investigadores.

# Capitulo 7. Conclusiones

Gran cantidad de las actividades comerciales que se realizan diariamente implican resolver complejos problemas de optimización $NP-hard$ , y su resolución de forma más o menos eficiente tiene una importante repercusión en los beneficios de las compañías. Este hecho es un aspecto clave en el sector de la logística y el transporte, uno de los principales motores económicos tanto a nivel nacional como autonómico. Pues bien, para poder llevar a cabo esta resolución eficiente en un tiempo finito es necesario el uso de algoritmos aproximados. En concreto metaheuristicos que sean capaces de resolver con garantías una versión concreta del *vehicle routing problem.* Una versión que incorpora restricciones de carga, de tiempo e incompatibilidad de mercancías. Para poder resolver este problema se ha hecho un barrido de distintos metaheruisticos con el objeto de detetectar aquel que mejor funcionaba ante una instancia del problema con datos reales.. En concreto, se ha optado un sistema cuyo núcleo se basa en el empleo de un metaheuristico *tabu-search,* implementado a través del solver *Optaplanner*.

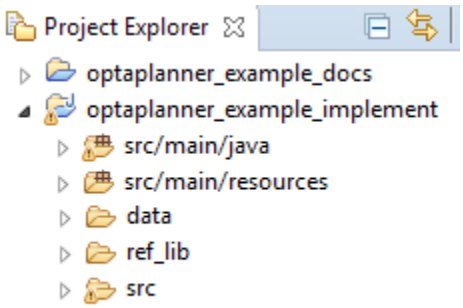
Existen numerosas herramientas *open source* como *Optaplanner*, que permite solucionar problemas de optimización combinatoria del tipo VRP de forma intuitiva, rápida y ágil. *Optaplanner*, además de ser la más potente y compleja, también es la más utilizada ya que puede emplearse para resolver otros problemas complejos cotidianos menos relacionados con la teoría de grafos pero a la vez frecuentes como: *bin packing*, *equipment scheduling* o *employeee rostering*. Entre las principales ventajas del solver *Optaplanner* destaca su corta curva de aprendizaje y su alto rendimiento, que permite obtener resultados muy similares a los desarrollos académicos en un corto periodo de tiempo. Por otro lado, también podemos enumerar ciertas limitaciones de la propia herramienta. Entre ellas destaca el reducido número de metaheuristicos a utilizar que impiden emplear algoritmos del tipo ACO o genéticos. Otro aspecto limitante del solver podría ser la complejidad para modelar ciertas restricciones reales o bien para modificar otras ya existentes. Es el caso que ocurre al intentar incorporar variables como legislación distinta entre países que puede afectar al transporte en ciertos días, horarios de apertura de centros de descarga, etc. Es por ello que queda como una futura línea de investigación el hecho de abordar el problema mediante una propuesta propia que permita modelar todo tipo de restricciones y que permita incorporar una amplia mayoría de meatheuristicos o incluso otras técnicas como por ejemplo redes neuronales o *reinforcement learning*. A pesar de las numerosas ventajas de la citada herramienta, plantea límites que en muchas ocasiones se exceden en los casos reales diarios a los que se enfrenan las empresas.

# Capitulo 8. Anexos

## 8.1 Instalación Optaplanner

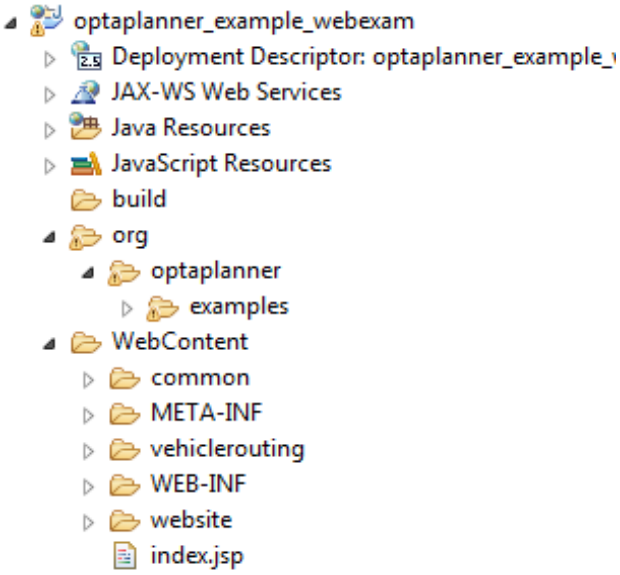
Existen distintas opciones a la hora de trabajar con *Optaplanner.* Cada una hace referencia a los distintos IDE JAVA (IntelliJ, Eclipse o Netbeans) entre otros. En nuestro caso hemos optado por la utilización del entorno eclipse. Para utilizar eclipse con *Optapanner* es necesario seguir una serie de pasos:

1. Descargar distribución *Optaplanner* de www.optaplanner.org (recomendable versión 7.0.0 en adelante)
2. Creamos un proyecto donde descomprimimos toda la distribución de cara a tener toda la información accesible desde el workbench. Podemos denominarlo optaplanner_example_docs
3. Creamos otro proyecto, el cual contendrá los ficheros fuentes necesarios para la implementación. El primer paso es definir un directorio para incluir las librerías necesarias. Las librerías necesarias son los .jar contenidos en el directorio binaries y en el examples/binaries a excepción del archivo examples/binaries/optaplanner-examples-*.jar. A continuación debemos añadir el código fuente de los directorios src/main/java y src/main/resources. También será necesario incluir el directorio data Renombramos el proyecto como optaplanner_example_implement.
4. Por último configuramos una *run configuration* con los siguientes parámetros :
    - Clase principal : org.optaplanner.examples.app.OptaPlannerExamplesApp
    - Parámetros VM : -Xmx512M –server




Por otro lado, si queremos ejecutar *Optaplanner* con mapas reales y obtener una representación gráfica más precisa debemos utilizar la tecnología JEE. En concreto debemos efectuar los siguientes pasos:

1. Descargar e instalar un servidor JEE real. En nuestro caso hemos empleado Wildfly 9.0.1 (con una versión anterior, es posible que no funcionen todas las librerías importadas)
2. A continuación debemos crear un proyecto web el cual servirá para generar el archivo.war con el que se hará el volcado web en el servidor. Puede ser útil crear el proyecto a partir del proyecto web bajo el directorio webexamples de la distribución *Optaplanner.*
3. Una vez configurado el proyecto web, debemos generar un archivo.war tal y como comentamos. A continuación debemos realizar un deploy del archivo.war en el directorio /standalone/deployments (en el caso de Wildfly)
4. Tras relizar el deploy, al arrancar el servidor JEE, ya tendremos disponible el proyecto web.Se encontrara por defecto en la dirección : http://localhost:8080/optaplanner-webexamples-*/ (sustituir símbolo* por versión de *Optaplanner*)




## 8.2 Benchmarks

```
C101

VEHICLE
NUMBER     CAPACITY
  25         200

CUSTOMER
CUST NO.  XCOORD.   YCOORD.    DEMAND   READY TIME  DUE DATE   SERVICE   TIME

    0      40         50          0          0       1236          0
    1      45         68         10        912        967         90
    2      45         70         30        825        870         90
    3      42         66         10         65        146         90
    4      42         68         10        727        782         90
    5      42         65         10         15         67         90
    6      40         69         20        621        702         90
    7      40         66         20        170        225         90
    8      38         68         20        255        324         90
    9      38         70         10        534        605         90
   10      35         66         10        357        410         90
   11      35         69         10        448        505         90
   12      25         85         20        652        721         90
   13      22         75         30         30         92         90
   14      22         85         10        567        620         90
   15      20         80         40        384        429         90
   16      20         85         40        475        528         90
   17      18         75         20         99        148         90
   18      15         75         20        179        254         90
   19      15         80         10        278        345         90
   20      30         50         10         10         73         90
   21      30         52         20        914        965         90
   22      28         52         20        812        883         90
   23      28         55         10        732        777         90
   24      25         50         10         65        144         90
   25      25         52         40        169        224         90
   26      25         55         10        622        701         90
   27      23         52         10        261        316         90
```

| | | | | | | |
|---|---|---|---|---|---|---|
| 28 | 23 | 55 | 20 | 546 | 593 | 90 |
| 29 | 20 | 50 | 10 | 358 | 405 | 90 |
| 30 | 20 | 55 | 10 | 449 | 504 | 90 |
| 31 | 10 | 35 | 20 | 200 | 237 | 90 |
| 32 | 10 | 40 | 30 | 31 | 100 | 90 |
| 33 | 8 | 40 | 40 | 87 | 158 | 90 |
| 34 | 8 | 45 | 20 | 751 | 816 | 90 |
| 35 | 5 | 35 | 10 | 283 | 344 | 90 |
| 36 | 5 | 45 | 10 | 665 | 716 | 90 |
| 37 | 2 | 40 | 20 | 383 | 434 | 90 |
| 38 | 0 | 40 | 30 | 479 | 522 | 90 |
| 39 | 0 | 45 | 20 | 567 | 624 | 90 |
| 40 | 35 | 30 | 10 | 264 | 321 | 90 |
| 41 | 35 | 32 | 10 | 166 | 235 | 90 |
| 42 | 33 | 32 | 20 | 68 | 149 | 90 |
| 43 | 33 | 35 | 10 | 16 | 80 | 90 |
| 44 | 32 | 30 | 10 | 359 | 412 | 90 |
| 45 | 30 | 30 | 10 | 541 | 600 | 90 |
| 46 | 30 | 32 | 30 | 448 | 509 | 90 |
| 47 | 30 | 35 | 10 | 1054 | 1127 | 90 |
| 48 | 28 | 30 | 10 | 632 | 693 | 90 |
| 49 | 28 | 35 | 10 | 1001 | 1066 | 90 |
| 50 | 26 | 32 | 10 | 815 | 880 | 90 |
| 51 | 25 | 30 | 10 | 725 | 786 | 90 |
| 52 | 25 | 35 | 10 | 912 | 969 | 90 |
| 53 | 44 | 5 | 20 | 286 | 347 | 90 |
| 54 | 42 | 10 | 40 | 186 | 257 | 90 |
| 55 | 42 | 15 | 10 | 95 | 158 | 90 |
| 56 | 40 | 5 | 30 | 385 | 436 | 90 |
| 57 | 40 | 15 | 40 | 35 | 87 | 90 |
| 58 | 38 | 5 | 30 | 471 | 534 | 90 |
| 59 | 38 | 15 | 10 | 651 | 740 | 90 |
| 60 | 35 | 5 | 20 | 562 | 629 | 90 |
| 61 | 50 | 30 | 10 | 531 | 610 | 90 |
| 62 | 50 | 35 | 20 | 262 | 317 | 90 |
| 63 | 50 | 40 | 50 | 171 | 218 | 90 |
| 64 | 48 | 30 | 10 | 632 | 693 | 90 |
| 65 | 48 | 40 | 10 | 76 | 129 | 90 |
| 66 | 47 | 35 | 10 | 826 | 875 | 90 |
| 67 | 47 | 40 | 10 | 12 | 77 | 90 |
| 68 | 45 | 30 | 10 | 734 | 777 | 90 |
| 69 | 45 | 35 | 10 | 916 | 969 | 90 |
| 70 | 95 | 30 | 30 | 387 | 456 | 90 |
| 71 | 95 | 35 | 20 | 293 | 360 | 90 |
| 72 | 53 | 30 | 10 | 450 | 505 | 90 |
| 73 | 92 | 30 | 10 | 478 | 551 | 90 |
| 74 | 53 | 35 | 50 | 353 | 412 | 90 |
| 75 | 45 | 65 | 20 | 997 | 1068 | 90 |
| 76 | 90 | 35 | 10 | 203 | 260 | 90 |
| 77 | 88 | 30 | 10 | 574 | 643 | 90 |
| 78 | 88 | 35 | 20 | 109 | 170 | 90 |
| 79 | 87 | 30 | 10 | 668 | 731 | 90 |
| 80 | 85 | 25 | 10 | 769 | 820 | 90 |
| 81 | 85 | 35 | 30 | 47 | 124 | 90 |
| 82 | 75 | 55 | 20 | 369 | 420 | 90 |
| 83 | 72 | 55 | 10 | 265 | 338 | 90 |
| 84 | 70 | 58 | 20 | 458 | 523 | 90 |
| 85 | 68 | 60 | 30 | 555 | 612 | 90 |
| 86 | 66 | 55 | 10 | 173 | 238 | 90 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 87 | 65 | 55 | 20 | 85 | 144 | 90 |
| 88 | 65 | 60 | 30 | 645 | 708 | 90 |
| 89 | 63 | 58 | 10 | 737 | 802 | 90 |
| 90 | 60 | 55 | 10 | 20 | 84 | 90 |
| 91 | 60 | 60 | 10 | 836 | 889 | 90 |
| 92 | 67 | 85 | 20 | 368 | 441 | 90 |
| 93 | 65 | 85 | 40 | 475 | 518 | 90 |
| 94 | 65 | 82 | 10 | 285 | 336 | 90 |
| 95 | 62 | 80 | 30 | 196 | 239 | 90 |
| 96 | 60 | 80 | 10 | 95 | 156 | 90 |
| 97 | 60 | 85 | 30 | 561 | 622 | 90 |
| 98 | 58 | 75 | 20 | 30 | 84 | 90 |
| 99 | 55 | 80 | 10 | 743 | 820 | 90 |
| 100 | 55 | 85 | 20 | 647 | 726 | 90 |

R201

VEHICLE

| NUMBER | CAPACITY |
|---|---|
| 25 | 1000 |

CUSTOMER

| CUST NO. | XCOORD. | YCOORD. | DEMAND | READY TIME | DUE DATE | SERVICE TIME |
|---|---|---|---|---|---|---|
| 0 | 35 | 35 | 0 | 0 | 1000 | 0 |
| 1 | 41 | 49 | 10 | 707 | 848 | 10 |
| 2 | 35 | 17 | 7 | 143 | 282 | 10 |
| 3 | 55 | 45 | 13 | 527 | 584 | 10 |
| 4 | 55 | 20 | 19 | 678 | 801 | 10 |
| 5 | 15 | 30 | 26 | 34 | 209 | 10 |
| 6 | 25 | 30 | 3 | 415 | 514 | 10 |
| 7 | 20 | 50 | 5 | 331 | 410 | 10 |
| 8 | 10 | 43 | 9 | 404 | 481 | 10 |
| 9 | 55 | 60 | 16 | 400 | 497 | 10 |
| 10 | 30 | 60 | 16 | 577 | 632 | 10 |
| 11 | 20 | 65 | 12 | 206 | 325 | 10 |
| 12 | 50 | 35 | 19 | 228 | 345 | 10 |
| 13 | 30 | 25 | 23 | 690 | 827 | 10 |
| 14 | 15 | 10 | 20 | 32 | 243 | 10 |
| 15 | 30 | 5 | 8 | 175 | 300 | 10 |
| 16 | 10 | 20 | 19 | 272 | 373 | 10 |
| 17 | 5 | 30 | 2 | 733 | 870 | 10 |
| 18 | 20 | 40 | 12 | 377 | 434 | 10 |
| 19 | 15 | 60 | 17 | 269 | 378 | 10 |
| 20 | 45 | 65 | 9 | 581 | 666 | 10 |
| 21 | 45 | 20 | 11 | 214 | 331 | 10 |
| 22 | 45 | 10 | 18 | 409 | 494 | 10 |
| 23 | 55 | 5 | 29 | 206 | 325 | 10 |
| 24 | 65 | 35 | 3 | 704 | 847 | 10 |
| 25 | 65 | 20 | 6 | 817 | 956 | 10 |
| 26 | 45 | 30 | 17 | 588 | 667 | 10 |
| 27 | 35 | 40 | 16 | 104 | 255 | 10 |
| 28 | 41 | 37 | 16 | 114 | 255 | 10 |
| 29 | 64 | 42 | 9 | 190 | 313 | 10 |
| 30 | 40 | 60 | 21 | 259 | 354 | 10 |
| 31 | 31 | 52 | 27 | 152 | 275 | 10 |
| 32 | 35 | 69 | 23 | 660 | 777 | 10 |
| 33 | 53 | 52 | 11 | 45 | 200 | 10 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 34 | 65 | 55 | 14 | 529 | 614 | 10 |
| 35 | 63 | 65 | 8 | 686 | 813 | 10 |
| 36 | 2 | 60 | 5 | 41 | 208 | 10 |
| 37 | 20 | 20 | 8 | 606 | 693 | 10 |
| 38 | 5 | 5 | 16 | 302 | 405 | 10 |
| 39 | 60 | 12 | 31 | 33 | 224 | 10 |
| 40 | 40 | 25 | 9 | 360 | 437 | 10 |
| 41 | 42 | 7 | 5 | 396 | 511 | 10 |
| 42 | 24 | 12 | 5 | 25 | 172 | 10 |
| 43 | 23 | 3 | 7 | 620 | 705 | 10 |
| 44 | 11 | 14 | 18 | 233 | 340 | 10 |
| 45 | 6 | 38 | 16 | 29 | 189 | 10 |
| 46 | 2 | 48 | 1 | 515 | 628 | 10 |
| 47 | 8 | 56 | 27 | 85 | 250 | 10 |
| 48 | 13 | 52 | 36 | 773 | 906 | 10 |
| 49 | 6 | 68 | 30 | 501 | 540 | 10 |
| 50 | 47 | 47 | 13 | 547 | 642 | 10 |
| 51 | 49 | 58 | 10 | 348 | 453 | 10 |
| 52 | 27 | 43 | 9 | 174 | 299 | 10 |
| 53 | 37 | 31 | 14 | 414 | 489 | 10 |
| 54 | 57 | 29 | 18 | 641 | 734 | 10 |
| 55 | 63 | 23 | 2 | 620 | 739 | 10 |
| 56 | 53 | 12 | 6 | 585 | 692 | 10 |
| 57 | 32 | 12 | 7 | 421 | 530 | 10 |
| 58 | 36 | 26 | 18 | 849 | 980 | 10 |
| 59 | 21 | 24 | 28 | 17 | 229 | 10 |
| 60 | 17 | 34 | 3 | 721 | 862 | 10 |
| 61 | 12 | 24 | 13 | 290 | 377 | 10 |
| 62 | 24 | 58 | 19 | 163 | 302 | 10 |
| 63 | 27 | 69 | 10 | 34 | 191 | 10 |
| 64 | 15 | 77 | 9 | 214 | 333 | 10 |
| 65 | 62 | 77 | 20 | 49 | 188 | 10 |
| 66 | 49 | 73 | 25 | 592 | 693 | 10 |
| 67 | 67 | 5 | 25 | 294 | 401 | 10 |
| 68 | 56 | 39 | 36 | 637 | 752 | 10 |
| 69 | 37 | 47 | 6 | 162 | 293 | 10 |
| 70 | 37 | 56 | 5 | 788 | 968 | 10 |
| 71 | 57 | 68 | 15 | 268 | 367 | 10 |
| 72 | 47 | 16 | 25 | 31 | 208 | 10 |
| 73 | 44 | 17 | 9 | 308 | 399 | 10 |
| 74 | 46 | 13 | 8 | 681 | 802 | 10 |
| 75 | 49 | 11 | 18 | 236 | 345 | 10 |
| 76 | 49 | 42 | 13 | 290 | 373 | 10 |
| 77 | 53 | 43 | 14 | 817 | 952 | 10 |
| 78 | 61 | 52 | 3 | 384 | 499 | 10 |
| 79 | 57 | 48 | 23 | 388 | 465 | 10 |
| 80 | 56 | 37 | 6 | 839 | 968 | 10 |
| 81 | 55 | 54 | 26 | 411 | 456 | 10 |
| 82 | 15 | 47 | 16 | 162 | 289 | 10 |
| 83 | 14 | 37 | 11 | 96 | 249 | 10 |
| 84 | 11 | 31 | 7 | 436 | 511 | 10 |
| 85 | 16 | 22 | 41 | 376 | 461 | 10 |
| 86 | 4 | 18 | 35 | 388 | 465 | 10 |
| 87 | 28 | 18 | 26 | 420 | 447 | 10 |
| 88 | 26 | 52 | 9 | 279 | 388 | 10 |
| 89 | 26 | 35 | 15 | 755 | 920 | 10 |
| 90 | 31 | 67 | 3 | 392 | 487 | 10 |
| 91 | 15 | 19 | 1 | 739 | 866 | 10 |
| 92 | 22 | 22 | 2 | 18 | 181 | 10 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 93 | 18 | 24 | 22 | 811 | 969 | 10 |
| 94 | 26 | 27 | 27 | 436 | 503 | 10 |
| 95 | 25 | 24 | 20 | 92 | 231 | 10 |
| 96 | 22 | 27 | 11 | 607 | 690 | 10 |
| 97 | 25 | 21 | 12 | 612 | 673 | 10 |
| 98 | 19 | 21 | 10 | 183 | 306 | 10 |
| 99 | 20 | 26 | 9 | 333 | 432 | 10 |
| 100 | 18 | 18 | 17 | 798 | 965 | 10 |

RC101

VEHICLE

| NUMBER | CAPACITY |
|---|---|
| 25 | 200 |

CUSTOMER

| CUST NO. | XCOORD. | YCOORD. | DEMAND | READY TIME | DUE DATE | SERVICE TIME |
|---|---|---|---|---|---|---|
| 0 | 40 | 50 | 0 | 0 | 240 | 0 |
| 1 | 25 | 85 | 20 | 145 | 175 | 10 |
| 2 | 22 | 75 | 30 | 50 | 80 | 10 |
| 3 | 22 | 85 | 10 | 109 | 139 | 10 |
| 4 | 20 | 80 | 40 | 141 | 171 | 10 |
| 5 | 20 | 85 | 20 | 41 | 71 | 10 |
| 6 | 18 | 75 | 20 | 95 | 125 | 10 |
| 7 | 15 | 75 | 20 | 79 | 109 | 10 |
| 8 | 15 | 80 | 10 | 91 | 121 | 10 |
| 9 | 10 | 35 | 20 | 91 | 121 | 10 |
| 10 | 10 | 40 | 30 | 119 | 149 | 10 |
| 11 | 8 | 40 | 40 | 59 | 89 | 10 |
| 12 | 8 | 45 | 20 | 64 | 94 | 10 |
| 13 | 5 | 35 | 10 | 142 | 172 | 10 |
| 14 | 5 | 45 | 10 | 35 | 65 | 10 |
| 15 | 2 | 40 | 20 | 58 | 88 | 10 |
| 16 | 0 | 40 | 20 | 72 | 102 | 10 |
| 17 | 0 | 45 | 20 | 149 | 179 | 10 |
| 18 | 44 | 5 | 20 | 87 | 117 | 10 |
| 19 | 42 | 10 | 40 | 72 | 102 | 10 |
| 20 | 42 | 15 | 10 | 122 | 152 | 10 |
| 21 | 40 | 5 | 10 | 67 | 97 | 10 |
| 22 | 40 | 15 | 40 | 92 | 122 | 10 |
| 23 | 38 | 5 | 30 | 65 | 95 | 10 |
| 24 | 38 | 15 | 10 | 148 | 178 | 10 |
| 25 | 35 | 5 | 20 | 154 | 184 | 10 |
| 26 | 95 | 30 | 30 | 115 | 145 | 10 |
| 27 | 95 | 35 | 20 | 62 | 92 | 10 |
| 28 | 92 | 30 | 10 | 62 | 92 | 10 |
| 29 | 90 | 35 | 10 | 67 | 97 | 10 |
| 30 | 88 | 30 | 10 | 74 | 104 | 10 |
| 31 | 88 | 35 | 20 | 61 | 91 | 10 |
| 32 | 87 | 30 | 10 | 131 | 161 | 10 |
| 33 | 85 | 25 | 10 | 51 | 81 | 10 |
| 34 | 85 | 35 | 30 | 111 | 141 | 10 |
| 35 | 67 | 85 | 20 | 139 | 169 | 10 |
| 36 | 65 | 85 | 40 | 43 | 73 | 10 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 37 | 65 | 82 | 10 | 124 | 154 | 10 |
| 38 | 62 | 80 | 30 | 75 | 105 | 10 |
| 39 | 60 | 80 | 10 | 37 | 67 | 10 |
| 40 | 60 | 85 | 30 | 85 | 115 | 10 |
| 41 | 58 | 75 | 20 | 92 | 122 | 10 |
| 42 | 55 | 80 | 10 | 33 | 63 | 10 |
| 43 | 55 | 85 | 20 | 128 | 158 | 10 |
| 44 | 55 | 82 | 10 | 64 | 94 | 10 |
| 45 | 20 | 82 | 10 | 37 | 67 | 10 |
| 46 | 18 | 80 | 10 | 113 | 143 | 10 |
| 47 | 2 | 45 | 10 | 45 | 75 | 10 |
| 48 | 42 | 5 | 10 | 151 | 181 | 10 |
| 49 | 42 | 12 | 10 | 104 | 134 | 10 |
| 50 | 72 | 35 | 30 | 116 | 146 | 10 |
| 51 | 55 | 20 | 19 | 83 | 113 | 10 |
| 52 | 25 | 30 | 3 | 52 | 82 | 10 |
| 53 | 20 | 50 | 5 | 91 | 121 | 10 |
| 54 | 55 | 60 | 16 | 139 | 169 | 10 |
| 55 | 30 | 60 | 16 | 140 | 170 | 10 |
| 56 | 50 | 35 | 19 | 130 | 160 | 10 |
| 57 | 30 | 25 | 23 | 96 | 126 | 10 |
| 58 | 15 | 10 | 20 | 152 | 182 | 10 |
| 59 | 10 | 20 | 19 | 42 | 72 | 10 |
| 60 | 15 | 60 | 17 | 155 | 185 | 10 |
| 61 | 45 | 65 | 9 | 66 | 96 | 10 |
| 62 | 65 | 35 | 3 | 52 | 82 | 10 |
| 63 | 65 | 20 | 6 | 39 | 69 | 10 |
| 64 | 45 | 30 | 17 | 53 | 83 | 10 |
| 65 | 35 | 40 | 16 | 11 | 41 | 10 |
| 66 | 41 | 37 | 16 | 133 | 163 | 10 |
| 67 | 64 | 42 | 9 | 70 | 100 | 10 |
| 68 | 40 | 60 | 21 | 144 | 174 | 10 |
| 69 | 31 | 52 | 27 | 41 | 71 | 10 |
| 70 | 35 | 69 | 23 | 180 | 210 | 10 |
| 71 | 65 | 55 | 14 | 65 | 95 | 10 |
| 72 | 63 | 65 | 8 | 30 | 60 | 10 |
| 73 | 2 | 60 | 5 | 77 | 107 | 10 |
| 74 | 20 | 20 | 8 | 141 | 171 | 10 |
| 75 | 5 | 5 | 16 | 74 | 104 | 10 |
| 76 | 60 | 12 | 31 | 75 | 105 | 10 |
| 77 | 23 | 3 | 7 | 150 | 180 | 10 |
| 78 | 8 | 56 | 27 | 90 | 120 | 10 |
| 79 | 6 | 68 | 30 | 89 | 119 | 10 |
| 80 | 47 | 47 | 13 | 192 | 222 | 10 |
| 81 | 49 | 58 | 10 | 86 | 116 | 10 |
| 82 | 27 | 43 | 9 | 42 | 72 | 10 |
| 83 | 37 | 31 | 14 | 35 | 65 | 10 |
| 84 | 57 | 29 | 18 | 96 | 126 | 10 |
| 85 | 63 | 23 | 2 | 87 | 117 | 10 |
| 86 | 21 | 24 | 28 | 87 | 117 | 10 |
| 87 | 12 | 24 | 13 | 90 | 120 | 10 |
| 88 | 24 | 58 | 19 | 67 | 97 | 10 |
| 89 | 67 | 5 | 25 | 144 | 174 | 10 |
| 90 | 37 | 47 | 6 | 86 | 116 | 10 |
| 91 | 49 | 42 | 13 | 167 | 197 | 10 |
| 92 | 53 | 43 | 14 | 14 | 44 | 10 |
| 93 | 61 | 52 | 3 | 178 | 208 | 10 |
| 94 | 57 | 48 | 23 | 95 | 125 | 10 |
| 95 | 56 | 37 | 6 | 34 | 64 | 10 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 96 | 55 | 54 | 26 | 132 | 162 | 10 |
| 97 | 4 | 18 | 35 | 120 | 150 | 10 |
| 98 | 26 | 52 | 9 | 46 | 76 | 10 |
| 99 | 26 | 35 | 15 | 77 | 107 | 10 |
| 100 | 31 | 67 | 3 | 180 | 210 | 10 |

# Capitulo 9. Referencias


*Capitulo 1*

[1.1] *Estrategia de Investigación e Innovación para la Especialización Inteligente de la Región de Murcia* (RIS3), Febrero 2014.

[1.2] *Informe Anual del Observatorio del Transporte y la Logística en España 2016*, Marzo 2017.

Capitulo *2*

[1.1]Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. Introduction to Algorithms, 3rd ed., The MIT Press, Cambridge, MA, 2009.(1.3 Growth of Functions)

[1.11] Michael R. Garey and David S. Johnson. 1990. *Computers and Intractability; a Guide to the Theory of Np-Completeness*. W. H. Freeman & Co., New York, NY, USA.

[1.12]Pavon

[1.13]M.Agrawal, N.Kayal, and N.Saxena.PRIMES is in P. *Annals of Mathematics*, 160:781-703, 2004

[1.14]B.Chor and O.Goldreich, An improved parallel algorithm for integer GCD, *Algorithmica*,5 (1990), 1-10

[1.2]S.A.Cook .*The Complexity of theorem-proving procedures*. Proceedings of the third annual ACM symposium on Theory of computing, 1971

[1.3] L. Fortnow. The Status of the P Versus NP Problem. *Communications of the ACM*, 52(9):78–86, 2009.

[1.4] P. P. Marino, Optimization of Computer Networks: Modeling and Algorithms: A Hands-On Approach. Wiley, 2016.(Appendix C : Complexity Theory)

[2.1]C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, 2003.

[2.2] Rafael Martí, *Procedimientos Metaheurísticos en Optimización Combinatoria*. Valencia: Universidad de Valencia, 2006.

[2.3] P. Toth, D. Vigo (Eds.) The Vehicle Routing Problem, SIAM Monographs on Discrete Mathematics, vol. 9, SIAM Philadelphia, 2002(Chapter 6)

[2.4]Osman, I. H. (2004). Metaheuristics: Models, design and analysis. In E. Kozan (Eds.), *Proceedings of the 5th Asia-Pacific industrial engineering and management systems conference* (pp. 1.2–1.16). Australia: Queensland University of Technology.

[2.5]Diaz,A.,Glover,F,.Ghazari,H.M.,Gonzalez,J.L.,Laguna,M,Moscato,P. y Tseng, F.T.(1996), *Optimización Heuristica y Redes Neuronales*, Paraninfo, Madrid.

[2.6]Osman,I.H.and Kelly,J.P.(eds.)(1996*),Meta-Heuristics:Theory and Applications,Ed.Kluwer Academic*, Boston

[2.7]S.Kirkpatrick, D.G.Jr., and M.P.Vecchi, "Optimization by simmulatd annealing" *sience*, vol.220, no.4598, pp.671-680, 1983.

[2.8] G. Laporte and Y. Nobert. Exact algorithms for the vehicle routing problem. *Annals of Discrete Mathematics*, 31:147-184, 1987.

*Capitulo 3*

[3.1]C. Barnhart, G. Laporte (Eds.), Transportation, Handbooks in Operations Research and Management Science, vol. 14, Elsevier, Amsterdam, The Netherlands (2007), pp. 367-428(Chapter 6)]

[3.2]Dantzig, G.B.,Ramser, J.M.(1959).The truck dispatching problem. *Management Science* 6,81–91

[3.3] Baker, B.M., Ayechew, M.A., 2003. A genetic algorithm for the vehicle routing problem. *Computers & Operations Research*, 30:787–800.

[3.4] Cordeau J-F, Gendreau M, Laporte G. A tabu search heuristic for periodic and multi-depot vehicle routing problems. Networks 1997; 30: 105–19

[3.5]Tan et al.,.Heuristics methods for vehicle routing problem with time windows. *Artificial Intelligence in Engineering*, 15 (3) (2001), pp. 281–295

[3.6] Domínguez M.; García C; Arias J.Mª. Recomendaciones para la Conservación y y transporte de alimentos perecederos. (2009)

[3.7] Domínguez M. ; Pinillos Jª M. ; García C**.** Problemas actuales y tendencias en la cadena del frio. Alimentacion nº 204 ,2005.85,91
[3.8] Surekha P et al., Solution to multi-depot vehicle routing problem using genetic algorithms *World Applied Programming*, 1 (3) (2011), pp. 118-131
[3.9] Pickup and delivery -> book (2001)
[3.10]B. Rekiek, A. Delchambre, H.A. Saleh. Handicapped person transportation: an application of the grouping genetic algorithm .*Engineering Applications of Artificial Intelligence*, 19 (5) (2006), pp. 511–520.
[3.11] 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), 1–8.
[3.12] Lambert, V., Laporte, G., Louveaux, F.V. (1993). Designing collection routes through bank branches. *Computers & Operations Research* 20, 783–791.
[3.13] Bertsimas, D.J. (1992). A vehicle routing problem with stochastic demand. *Operations Research* 40, 574–585.
[3.14] J.Desrosiers, Y. Dumas, M.M. Solomon, and F. Soumis. Time constrained routing and scheduling. In M.O. Ball, T.L. Magnanti, C.L. Monma, and G.L. Nemhauser, editor *Network Routing, Handbooks in Operations Research and Management Science* 8, North-Holland, Amsterdam, 1995, pp. 35-139

*Capitulo 4*
[4.1]Iconchallenge.insight-centre.org. (2017). *ICON Challenge on Algorithm Selection* [online] disponible en: http://iconchallenge.insight-centre.org/[Visitado 22 Sep. 2017].
[4.2]Cs.qub.ac.uk.(2017). *International Timetabling Competition*. [online] disponible en: http://www.cs.qub.ac.uk/itc2007/[Visitado 22 Sep. 2017].
[4.3]T.O. Team,, "Optaplanner user guide versión 7.3.0 final," Agosto 2017.

Capitulo 5
[5.1] https://developers.google.com/maps/documentation/distance-matrix/intro?hl=es-419

Capitulo 6
[6.1] Solomon MM. Algorithms for vehicle routing and scheduling problems with time window constraints.Oper Res 1987;35(2):254-66.
[6.2]http://dmawww.epfl.ch/~rochat data/Solomon.html