Technical Report
IIIA 2009 - 04

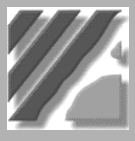
Març 2009

Estudio de técnicas de Inteligencia Artificial aplicadas a una plataforma de servicios móviles para la planificación de recursos móviles

José Manuel Moyano-Hidalgo
Ferran Ollé-Pla
José Antonio Rodríguez-Barranco
Eric Teruel-Casajuana

Institut d'Investigació en Intel·ligència Artificial
Consell Superior d'Investigacions Científiques

Campus de la Universitat Autònoma de Barcelona
08193 Bellaterra, Barcelona



ESTUDIO DE TÉCNICAS DE INTELIGENCIA ARTIFICIAL APLICADAS A UNA PLATAFORMA DE SERVICIOS MÓVILES PARA LA PLANIFICACIÓN DE RECURSOS MÓVILES

PROYECTO AVANZA I+D

TSI-020302-2008-13

Moyano-Hidalgo, José Manuel

Ollé-Pla, Ferran

Rodríguez-Barranco, José Antonio

Teruel-Casajuana, Eric

20 de Abril de 2009

Historia del Documento

Versión: **1.0, 1.1** *Descripción*: **Primera versión completa.**

Elaborado por:	*Fecha*:
Jose Manuel Moyano Hidalgo	**19/03/2009, 30/03/2009**
Ferran Ollé Pla	
Jose Antonio Rodríguez Barranco	
Eric Teruel Casajuana	

Revisado por:	*Fecha*:
Josep Puyol Gruart	**26/03/2009, 30/03/2009**
Lluís Godo Lacasa	
Enric Plaza Cervera	
Pere Garcia Calvés	
David Sierra Hernández	

Versión: **1.2** *Descripción*: **Versión completa revisada.**

Elaborado por:	*Fecha*:
Jose Manuel Moyano Hidalgo	**20/04/2009**
Ferran Ollé Pla	
Jose Antonio Rodríguez Barranco	
Eric Teruel Casajuana	

Revisado por:	*Fecha*:
Josep Puyol Gruart	**20/04/2009**
Lluís Godo Lacasa	
Enric Plaza Cervera	
Pere Garcia Calvés	
David Sierra Hernández	

Aprobado por:	*Fecha*:
Josep Puyol Gruart (IP)	**20/04/2009**
Firmado:	

Índice

Índice de figuras

1. Introducción

La revolución de Internet ha provocado cambios en el mundo empresarial dando la posibilidad de acercar las empresas a sus clientes. En los últimos tiempos, ofrecer este tipo de servicios a los clientes se ha convertido en una exigencia más que en valor añadido.

El caso de la logística no es ajeno a esta nueva forma de concebir el mundo. Hoy en día existen una multitud de empresas que se dedican a ofrecer servicios remotos de control de flotas, GPS, GSM, GPRS, localizadores, mapas, cartografía, servicios de localización, gestión de flotas, planificación de rutas, optimizadores y optimización de rutas; por lo tanto la manera de marcar la diferencia, que tienen las empresas que se dedican a ello, pasa por encontrar sistemas que den mejores prestaciones y servicios que los de la competencia abaratando costes y mejorando la producción.

La mayor parte de los problemas que se dan en logística son problemas de optimización combinatoria, ya que se trata de un espacio de soluciones discreto y factorialmente grande. Parece un campo idóneo para aplicar técnicas de Inteligencia Artificial [Robus05].

El conocimiento que el Instituto de Investigación en Inteligencia Artificial posee en esta materia lo hace único para poder abordar con éxito un reto de tal magnitud. La Unidad de Desarrollo Tecnológico en Inteligencia Artificial (UDT-IA), desde su posición privilegiada dentro del Instituto de Investigación en Inteligencia Artificial, afronta la realización del presente estudio cumpliendo así su principal objetivo: la transferencia de tecnología a la sociedad.

Este estudio se enmarca dentro de un proyecto de desarrollo de una *suite* de aplicaciones informáticas complementarias y necesarias para construir una **Plataforma de Servicios Móviles** para dar soporte a la

- **planificación**,
- control y
- gestión

de recursos móviles, así como las actividades diarias del sector empresarial.

El objetivo de la Unidad de Desarrollo Tecnológico en Inteligencia Artificial es estudiar las tecnologías, basadas en Inteligencia Artificial, adecuadas para solucionar el problema de la **planificación** de las rutas que los servicios móviles deben seguir.

Estos servicios móviles pueden hacer referencia a vehículos dedicados a:

- transporte de viajeros: transporte público.
- distribución de mercancías: general, nacional, internacional.
- servicios de mantenimiento y servicio técnico: residuos, emergencias.

Lo cual sugiere que estamos ante un problema de optimización en la cobertura de nodos.

1.1. Problema a resolver

1.1.1. Proyecto

Para solucionar de manera óptima los problemas de planificación se debe explorar todo el espacio de soluciones, que puede ser extremadamente amplio, debido a su naturaleza combinatoria. Ello sugiere estudiar diversas técnicas basadas en Inteligencia Artificial que ofrezcan resultados subóptimos pero satisfactorios al problema tanto en la solución propuesta como en el tiempo de obtención de la misma.

El objetivo, entonces, es crear un planificador subóptimo de servicios móviles.

Aunque Speedcomm Telematic Systems dispone de un software comercial similar llamado SpeedPLAN, este producto parte de algunas limitaciones que hay que solventar poniendo gran parte del esfuerzo en:

- Mejorar el tiempo de respuesta: Actualmente, según Speedcomm, una planificación en la que intervienen 100 recursos (vehículos) y 500 puntos (servicios) puede llegar a tardar muchas horas. El reto consiste al poder realizar una planificación online en cuestión de segundos.
- Añadir capacidad para replanificar: Actualmente una variación en el servicio (p.ej. avería de un vehículo) implica una nueva planificación con el pertinente problema de tiempo de respuesta.

1.1.2. Limitaciones

Para la definición del problema se parte, única y exclusivamente, de la documentación aportada por Speedcomm [Anexo 1] ya que desde la Unidad de Desarrollo Tecnológico:

- No se ha tenido acceso a la aplicación SpeedPLAN.
- No se ha tenido acceso a datos reales: ficheros de entrada (ficheros con listado de servicios y censo de vehículos disponibles), ficheros de resultados o datos referentes a la parametrización del sistema SpeedPLAN.
- No se ha tenido acceso a datos reales (o simulados) de los socios participantes en el proyecto.

1.1.3. Suposiciones

Debido las circunstancias excepcionales de falta de información, la Unidad de Desarrollo Tecnológico marca las siguientes líneas de estudio o suposiciones:

- La UDT-IA entiende que se trata de un problema de cobertura de nodos mediante rutas, donde los nodos se refieren a servicios a realizar y los arcos representan las rutas que recorrerán vehículos de transporte (de carácter general) para visitar los nodos.

- La UDT-IA entiende que las entradas, salidas y parámetros son los proporcionados en el Anexo 1, que han de ser adaptadas si procede la realización de un prototipo en el futuro.

1.1.4. Planteamiento

El problema del diseño de rutas de repartos es un problema bien conocido desde el punto de vista de la investigación. Se trata de un **problema de cobertura de nodos**. Estos nodos son unidos mediante rutas que parten y finalizan en nodos especiales llamados depósitos. A este problema se le conoce como VRP (Vehicle Routing Problem).

Por lo tanto, el problema se centra en encontrar la ruta de coste mínimo para cubrir un conjunto de nodos añadiendo ciertas restricciones. Se debe pasar una sola vez por cada nodo, desde un nodo origen que a su vez será destino generando así una ruta cerrada.

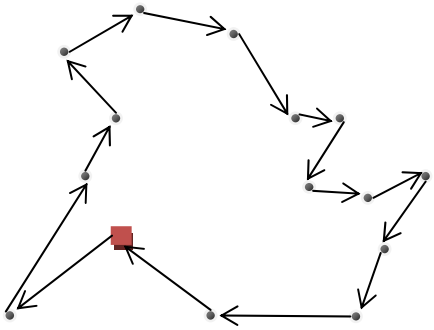


Figura 1: Ruta simple. Los vehículos cubren los nodos partiendo y finalizando en un depósito.

Dado que el problema de enrutamiento de vehículos VRP es de dificultad combinatoria (*NP-complete*), se deben explorar otras técnicas, basadas en Inteligencia Artificial, con la finalidad de obtener soluciones subóbtimas en tiempos aceptables. La realidad es que no conocemos en el mercado programas informáticos que puedan ofrecer una solución óptima, sin limitaciones, en un tiempo aceptable.

Los problemas de cobertura de nodos, a los que hacen referencia el presente estudio, basados en [Robus05] se pueden clasificar en:

- TSP (Traveling Salesman Problem)
    - m-TSP (Traveling Salesman Problem con *m* vehículos)
- VRP (Vehicle Routing Problem)
    - CVRP (Capacitated Vehicle Routing Problem)
    - VRPTW (Vehicle Routing Problem with Time Window)
    - CVRPTW (Capacitated Vehicle Routing Problem with Time Window)

La clasificación anterior responde a diferentes evoluciones del problema de cobertura de nodos o diseño de rutas de reparto. Estas evoluciones añaden complejidad al problema:

- Rutas para *m* vehículos, *m*>1.

- Rutas multi-depósito.
- Rutas con restricciones de tiempo o ventanas de tiempo.
- Rutas con restricciones de capacidad en los vehículos.

Los problemas de cobertura de nodos se suelen representar mediante el uso de grafos. En nuestro caso el grafo vendría definido de la siguiente manera:

- Sea un grafo G= (V, A) donde V = (v_0, v_1 , … , v_n) el conjunto de nodos por los que se ha de pasar, donde v_0 es el almacén y A = $((v_i , v_j): v_i, v_j \in V,\ i \neq j)$ un conjunto de arco. Una matriz no negativa (distancia, tiempo/coste) D= (d_{ij}) se define en A.

1.1.4.1. TSP - Traveling Salesman Problem

El problema del viajante de comercio o TSP (Traveling Salesman Problem), es uno de los problemas de optimización combinatoria NP-completo más ampliamente estudiado.

Un viajante o vehículo tiene que visitar *m* ciudades, o nodos, una sola vez y volver a la ciudad de origen.

La función a optimizar, u objetivo, es encontrar la secuencia de visitas que minimice la distancia total recorrida, esto es, el largo de ruta.

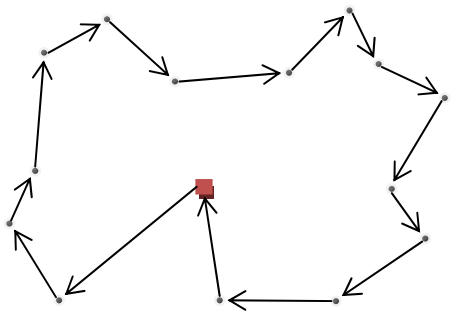


Figura 2: Ruta TSP

Traveling Salesman Problem
Rutas/vehículos: 1
Nodos a cubrir: N
Origen/Depósito: 1
Optimizar: largo de ruta
Función: $min \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} x_{ij}$ (minimizar la distancia total)

Restricciones:

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

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

(Solo puede haber un arco de entrada y de salida para cada nodo)

$x_{ij} \in \{0,1\}$ (variable binaria que dice si existe el arco entre “i” y “j”)

El TSP es una primera aproximación, la más básica, a nuestro problema concreto de reparto. A partir de aquí iremos añadiendo complejidad a la definición del problema hasta alcanzar la definición que más se acerque a nuestras necesidades.

1.1.4.2. M-TSP - Multi Traveling Salesman Problem

El problema de los *m*-Viajantes de Comercio (*m*-TSP) es una variante del TSP. En este problema, se trata de construir *m* rutas, una para cada viajante de comercio o vehículo.

Al igual que el TSP, cada ciudad debe ser visitada exactamente una vez. Las rutas de los vehículos deben comenzar y finalizar en el mismo lugar.

La función a optimizar es la distancia total de las *m* rutas.

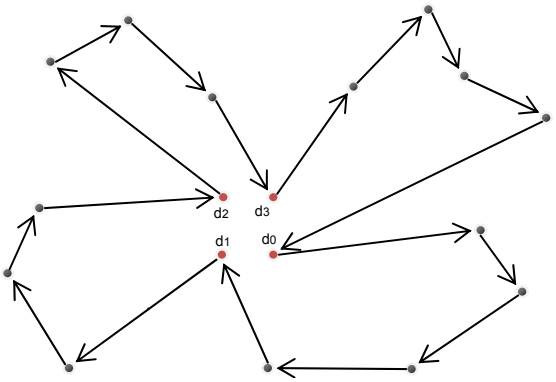


Figura 3: Ruta m-TSP

Multi Traveling Salesman Problem

Rutas/vehículos: M

Nodos a cubrir: N

Origen/Depósito: 1 (clonado, es decir, se trata del mismo punto cuadruplicado para convertir el problema en un TSP)

Optimizar: $min \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} x_{ij}$

Restricciones:

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

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

(Solo puede haber un arco de entrada y de salida para cada nodo)

$$\sum_{i=0}^{n} x_{i0} = \sum_{j=0}^{n} x_{0j} = m$$

(Salen del nodo 0 “m” vendedores y llegan al final “m” vendedores de nuevo al nodo 0)

$$\sum_{i \in S} \sum_{j \in S} x_{ij} \leq |S| - 1, \forall S \subseteq V \backslash \{V_0\}, \quad S \neq \emptyset$$

(Esta fórmula nos indica que no hay subrutas de cardinalidad S que no incluyan el almacén (nodo 0)).

$x_{ij} \in \{0,1\}$ (variable binaria que dice si existe el arco entre “i” y “j”)

Esta aproximación al problema, añade algo más de complejidad y se acerca más a la realidad ya que intervendrán m vehículos en el cálculo, en vez de uno como es el caso del TSP.

1.1.4.3. VRP - Vehicle Routing Problem

La necesidad de transportar determinadas mercancías de un lugar a otro, hace que existan empresas que se dediquen a prestar servicios de transporte, teniendo éstas un importante proceso de logística en cuanto a la programación de las diferentes rutas a realizar.

El VRP (Vehicle Routing Problem), heredero del TSP y m-TSP, así como sus variantes representan los diferentes problemas con los que se pueden encontrar dichas empresas.

El VRP básico consiste en diseñar el conjunto de rutas para que m vehículos visiten cada uno de los n nodos (o clientes) cumpliendo con las restricciones del sistema.

El VRP se puede clasificar según la cantidad de depósitos de vehículos que intervienen:

- VRP con depósito simple: Existe un único depósito común. Todos los vehículos parten y finalizan en un único depósito común.

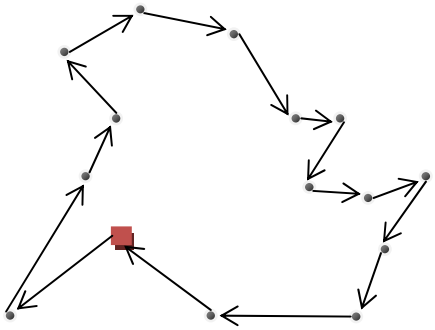


Figura 4: VRP depósito simple

- VRP multidepósito: Existen varios depósitos. Los vehículos parten y finalizan en el mismo depósito.

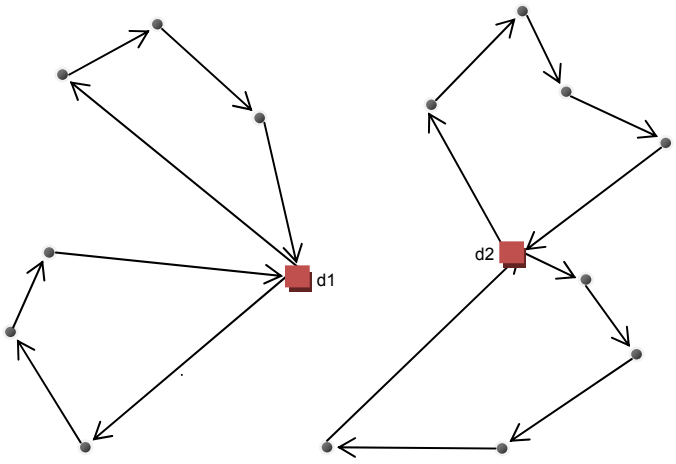


Figura 5: VRP multidepósito

Formalmente las restricciones que impone un VRP son:

1. Cada nodo (o cliente) debe ser visitado una única vez (excepto depósito/s).
2. Cada nodo (o cliente) debe ser visitado por un sólo vehículo (excepto depósito/s).
3. Un vehículo no puede quedarse en el nodo (o cliente), debe continuar con su ruta a otros nodos o regresa al depósito.
4. No pueden existir ciclos internos en las rutas.

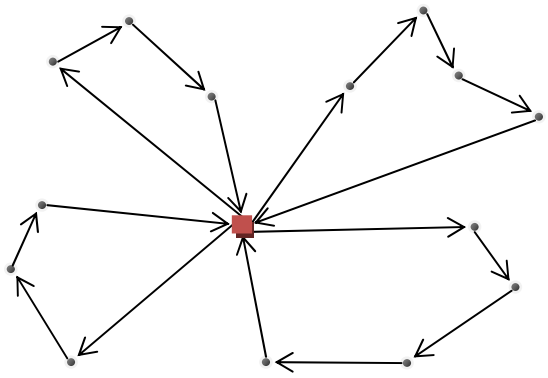


Figura 6: Ruta VRP

Vehicle Routing Problem

Rutas/vehículos: M

Nodos a cubrir: N

Origen/Depósito: 1 y multi

Optimizar: tiempo, capacidades, etc..

Función: $min \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} x_{ij}$

Restricciones:

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

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

(Solo puede haber un arco de entrada y de salida para cada nodo)

$$\sum_{i=0}^{n} x_{i0} = \sum_{j=0}^{n} x_{0j} = m$$

(Salen del nodo 0 “m” vendedores y llegan al final “m” vendedores de nuevo al nodo 0)

$$\sum_{i \in S} \sum_{j \in S} x_{ij} \leq |S| - 1, \forall S \subseteq V \backslash \{V_0\}, \quad S \neq \emptyset$$

(Esta fórmula nos indica que no hay subrutas de cardinalidad S que no incluyan el almacén (nodo 0)).

$x_{ij} \in \{0,1\}$ (variable binaria que dice si existe el arco entre “i” y “j”)

1.1.4.4. CVRP - Capacited Vehicle Routing Problem

El VRP aplicado al mundo real, ha de contar con las restricciones lógicas de capacidad de los vehículos. Un vehículo de transporte no cuenta con capacidad ilimitada, pudiendo suceder que al dar servicio a sus primeros clientes agote la capacidad de recogida de mercancías y no pueda ofrecer el servicio al resto de clientes durante la ruta planificada.

Formalmente estas restricciones son:

1. Cada nodo (o cliente) debe ser visitado una única vez (excepto depósito/s).
2. Cada nodo (o cliente) debe ser visitado por un sólo vehículo (excepto depósito/s).
3. Un vehículo no puede quedarse en el nodo (o cliente), debe continuar con su ruta a otros nodos (o clientes), o regresa al depósito.
4. No pueden existir ciclos internos en las rutas.
5. La capacidad del vehículo no puede ser excedida en ninguna de las rutas.

Se asume que la capacidad de todos los vehículos es la misma. La definición del grafo es la siguiente:

- Sea un grafo G= (V, A) donde V = (v_0, v_1 , … , v_n) el conjunto de nodos por los que se ha de pasar, donde v_0 es el almacén y A = $((v_i , v_j): v_i, v_j \in V,\ i \neq j)$ un conjunto de arco. Una matriz no negativa (distancia, tiempo/coste) D= (d_{ij}) se define en A. Un peso no negativo q_i asociado a cada vértice que representa la demanda del cliente a v_{ij} , y naturalmente la demanda total de una ruta no puede exceder la capacidad de cada camión Q_v.

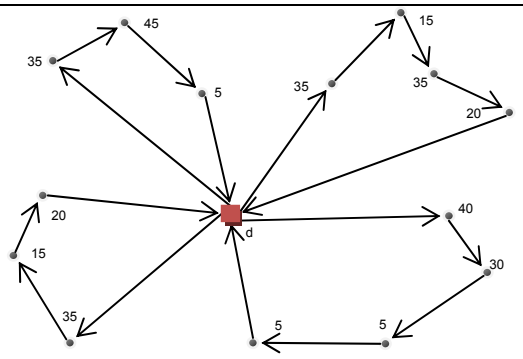


Figura 7: Ruta CVRP

Capacited Vehicle Routing Problem
Rutas/vehículos: M
Nodos a cubrir: N
Origen/Depósito: 1 y multi
Optimizar: ruta.
Función: $min \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} x_{ij}$

Restricciones:

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

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

(Solo puede haber un arco de entrada y de salida para cada nodo)

$\sum_{i=0}^{n} x_{i0} = \sum_{j=0}^{n} x_{0j} = m$ (Salen del nodo 0 “m” vendedores y llegan al final “m” vendedores de nuevo al nodo 0)

$$\sum_{i \in S} \sum_{j \in S} x_{ij} \leq |S| - 1, \forall S \subseteq V \backslash \{V_0\}, \quad S \neq \emptyset$$

(Esta fórmula nos indica que no hay subrutas de cardinalidad S que no incluyan el almacén (nodo 0)).

$$\sum_{i=1}^{n} q_i \left(\sum_{j=1}^{n} x_{ij}^{v} \right) \leq Q_v, \forall v$$

(Restricción de las capacidades de los camiones)

$x_{ij} \in \{0,1\}$ (variable binaria que dice si existe el arco entre “i” y “j”)

1.1.4.5. VRPTW - Vehicle Routing Problem with Time Window

Pueden aparecer nuevas restricciones que añadan complejidad al VRP. En este caso nos referimos a restricciones de tiempo. Un cliente seguramente exigirá ser servido en un intervalo de tiempo determinado. A este aspecto se le conoce formalmente como *time window* o ventana de tiempo.

Por lo tanto, las restricciones en este caso quedan:

1. Cada nodo (o cliente) debe ser visitado una única vez (excepto depósito/s).
2. Cada nodo (o cliente) debe ser visitado por un sólo vehículo (excepto depósito/s).
3. Un vehículo no puede quedarse en el nodo (o cliente), debe continuar con su ruta a otros nodos (o clientes), o regresa al depósito.
4. No pueden existir ciclos internos en las rutas.
5. Hay que cumplir las restricciones de ventanas de tiempo. Dos posibilidades:
    - Dura: Si un vehículo llega antes de lo pactado a un nodo (o cliente) deberá esperar a alcanzar la hora. Sin embargo, bajo ningún concepto, se le permite llegar fuera de tiempo.
    - Blanda: En este caso las restricciones temporales pueden ser violadas a cambio de una penalización en el coste.

La definición del grafo es la siguiente:

- Sea un grafo G= (V, A) donde V = (v_0, v_1 , … , v_n) el conjunto de nodos por los que se ha de pasar, donde v_0 es el almacén y A = $((v_i, v_j): v_i, v_j \in V,\ i \neq j)$ un conjunto de arco. Una matriz no negativa (distancia, tiempo/coste) D= (d_{ij}) se define en A. Sea T la matriz no negativa que contiene los tiempos que se tarda en viajar de un nodo a otro. Sea [e_i, l_i] la ventana de tiempo en la que el cliente “i” tiene que ser servido.

1.1.4.6. CVRPTW - Capacity Vehicle Routing Problem with Time Window

Esta definición del problema de enrutamiento de vehículos es la que, posiblemente, más se acerca a nuestras necesidades de planificación. No es más que el problema de enrutamiento de vehículos con restricciones de capacidad y tiempos a la vez.

Formalmente estas restricciones son:

1. Cada nodo (o cliente) debe ser visitado una única vez (excepto depósito/s).
2. Cada nodo (o cliente) debe ser visitado por un sólo vehículo (excepto depósito/s).
3. Un vehículo no puede quedarse en el nodo (o cliente), debe continuar con su ruta a otros nodos (o clientes), o regresa al depósito.
4. No pueden existir ciclos internos en las rutas.
5. La capacidad del vehículo no puede ser excedida.
6. Hay que cumplir la ventana de tiempo obligatoriamente o por el contrario fijar un coste por incumplimiento.

La definición del grafo es la siguiente:

- Sea un grafo G= (V, A) donde V = (v_0, v_1 , … , v_n) el conjunto de nodos por los que se ha de pasar, donde v_0 es el almacén y A = $((v_i, v_j): v_i, v_j \in V,\ i \neq j)$ un conjunto de arco. Una matriz no negativa (distancia, tiempo/coste) D= (d_{ij}) se define en A. Sea T la matriz no negativa que contiene los tiempos que se tarda en viajar de un nodo a otro. Sea [e_i, l_i] la ventana de tiempo en la que el cliente "i" tiene que ser servido. Un peso no negativo q_i asociado a cada vértice que representa la demanda del cliente a v_{ij} , y naturalmente la demanda total de una ruta no puede exceder la capacidad de cada camión Q_v.

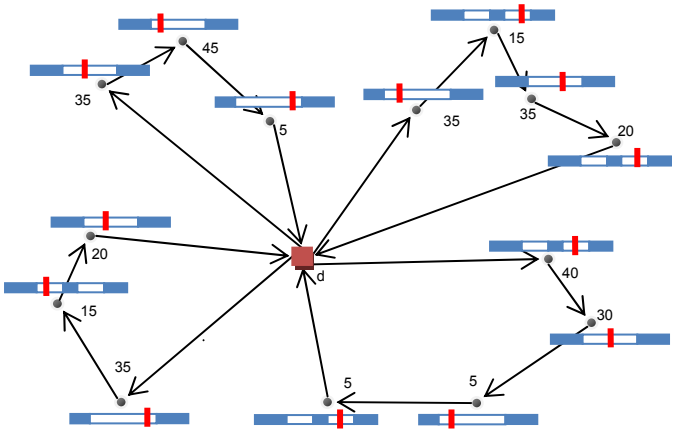


Figura 8: Ruta CVRPTW

Capacity Vehicle Routing Problem with Time Window

Rutas/vehículos: M

Nodos a cubrir: N

Origen/Depósito: 1 y multi

Optimizar: ruta

Función: $min \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} x_{ij}$

Restricciones:

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

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

(Solo puede haber un arco de entrada y de salida para cada nodo)

$\sum_{i=0}^{n} x_{i0} = \sum_{j=0}^{n} x_{0j} = m$ (Salen del nodo 0 “m” vendedores y llegan al final “m” vendedores de nuevo al nodo 0)

$$\sum_{i \in S} \sum_{j \in S} x_{ij} \leq |S| - 1, \forall S \subseteq V\backslash\{0\}, \quad S \neq \emptyset$$

(Esta fórmula nos indica que no hay subrutas de cardinalidad S que no incluyan el almacén (nodo 0)).

$$\sum_{i=1}^{n} q_i \left(\sum_{j=1}^{n} x_{ij}^{v} \right) \leq Q_v, \forall v$$

(Restricción de las capacidades de los camiones)

$$e_i = \max(e_0 + t_{0i}, e_i)$$

(Inicio de la primera ventana de tiempo)

$$l_i = \min(l_0 - t_{i0}, l_i)$$

(Fin de la primera ventana de tiempo)

$$si\ x_{ij} = 1 \Rightarrow p_i + s_i + t_{ij} \leq p_j, \forall (i,j) \in A$$

(donde p_i indica el comienzo del serivicio *i*, s_i el tiempode servicio en el nodo *i* y p_j el cominezo del servicio j)

$$e_i \leq p_i \leq l_i, \forall i \in V \setminus (V_0)$$

(Estas dos últimas ecuaciones son las restricciones temporales)

$x_{ij} \in \{0,1\}$ (variable binaria que dice si existe el arco entre “i” y “j”)

1.2. Objetivos generales

De los planteamientos anteriores el más cercano a la problemática planteada a la UDT-IA es el VRP, posiblemente con restricciones de capacidad y tiempo [Anexo 1].

Nuestro objetivo es realizar un estudio con el fin de comparar diversas tecnologías de Inteligencia Artificial aplicadas al problema de cobertura de nodos planteado.

Entre nuestras intenciones está la de encontrar algoritmos capaces de ofrecer soluciones subóptimas en tiempos aceptables. Para ello contamos con el estudio de diferentes tecnologías heurísticas, metaheurísticas y sistemas basados en el conocimiento para intentar:

- Mejorar el tiempo de respuesta: Actualmente una planificación en la que intervienen 100 recursos (vehículos) y 500 puntos (servicios) puede llegar a tardar muchas horas. El reto consiste al poder realizar una planificación online en cuestión de segundos.
- Añadir capacidad para replanificar: Actualmente una variación en el servicio (p.ej. avería de camión) implica una nueva planificación con el pertinente problema de tiempo de respuesta.

Los datos, procesos y conclusiones del presente estudio serán comunicados a los interesados mediante el presente documento.

Escenario Ficticio

- Clientes: 9
- Vehículos: 3
- Capacidad de los vehículos: 100
- Restricciones temporales y de capacidad de los clientes:

Cliente	Ventana de Tiempo	Demanda
1	[7,13]	30
2	[8,19]	50
3	[10,16]	10
4	[15,18]	5
5	[7,13] [16,22]	20
6	[10,13] [15,19]	80
7	[7,13] [16,22]	10
8	[10,16]	60
9	[13,16]	20

Una posible representación gráfica del problema sería la siguiente:

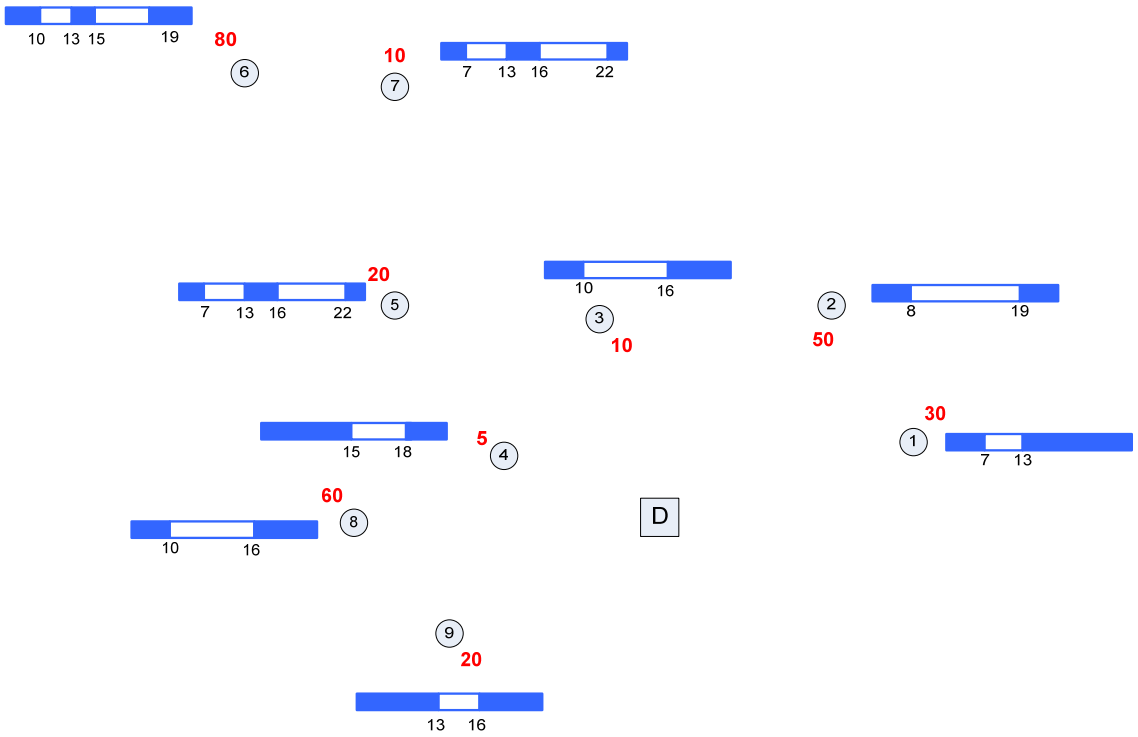


Figura 9: Escenario CVRPTW

Donde cada cliente es representado por un nodo con sus restricciones de tiempo y capacidad.

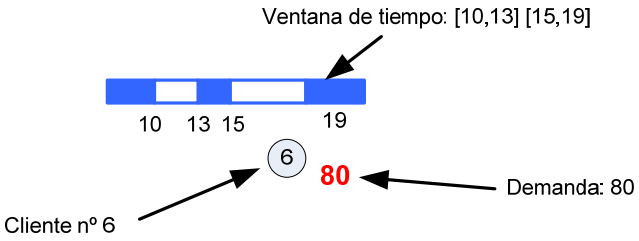


Figura 10: Representación cliente

El depósito estaría representado por:

D

Se supone que todos los vehículos están ubicados inicialmente en el depósito y al terminar los servicios volverían otra vez allí.

Una vez aplicado uno de los métodos que se presentan en los siguientes capítulos del documento obtendriamos una solución como la siguiente:

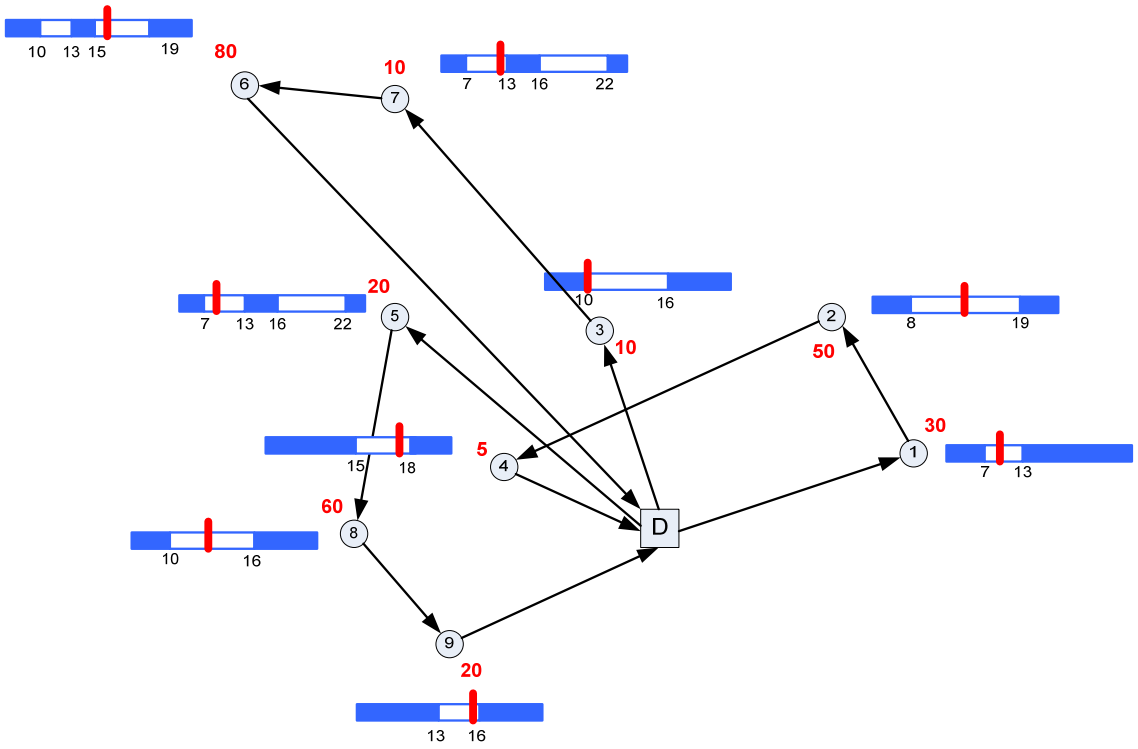


Figura 11: Posible solución al escenario CVRPTW

1.3. Estructura del Documento

El informe está dividido en seis capítulos y dos anexos. Este primer capítulo de introducción, donde se ha explicado el problema que se intenta resolver y posibles variantes que existen en la literatura, cinco más y dos anexos, el contenido de los cuales es el siguiente:

- **Capítulo 2:** En el siguiente capítulo se estudiarán diferentes técnicas existentes para resolver este tipo de problemas, ya sean específicas para el VRP o algoritmos más generales y su adaptación al problema específico del enrutamiento de vehículos.

- **Capítulo 3:** Una vez estudiadas las diferentes técnicas que se aplican en el mundo académico y que parece que utilizan algunos productos comerciales según sus propias páginas web, se realizan diferentes experimentos con implementaciones de dichas técnicas para comprobar su rendimiento. Además se realizan varios desarrollos por parte de la UDT-IA para poder adaptar los algoritmos a diferentes tests y combinaciones entre ellos.

- **Capítulo 4:** Uno de los objetivos de este estudio consistía en la adición de un mecanismo de replanificación. En este capítulo se estudian las diferentes posibilidades para realizar replanificación que existen en el mundo académico y sistemas que se puedan adaptar a cambios y eventos que ocurran sin estar previamente planificados.

- **Capítulo 5:** Tras estudiar las diferentes técnicas que se pueden utilizar para resolver el problema del enrutamiento de vehículos y probar diferentes soluciones, se muestra la arquitectura propuesta por la UDT-IA.

- **Capítulo 6:** Por último se comentan las conclusiones que se derivan del estudio realizado y los objetivos que se querían lograr. Además se muestran algunas futuras características que debería tener el sistema para poder adaptarlo a las necesidades reales de las empresas y no quedarse en el mundo académico.
- **Anexo 1:** En el anexo 1 se encuentra la documentación aportada por la empresa SpeedComm.
- **Anexo 2:** En el anexo 2 se indica el contenido del CD que acompaña a este estudio.

2. Estudio de la tecnología actual aplicada al problema

En esta sección exponemos los diferentes tipos de algoritmos, clasificados según la tecnología subyacente, que se pueden utilizar para solucionar el problema del enrutamiento de vehículos.

2.1. Exactos/Óptimos

Los algoritmos exactos u óptimos son aquellos que buscan en el espacio de búsqueda todas las posibles soluciones al problema que se está resolviendo y acaban dando la mejor que existe. Esto nos asegura que siempre vamos a obtener la solución óptima, pero el coste de estos algoritmos suele ser demasiado elevado. Hay algunos algoritmos que intentan reducir esta complejidad como el Brach and Bound, comentado a continuación.

2.1.1. Branch and bound

Tal como se describe en [Bara06], la técnica de Branch and Bound consiste en ir construyendo un árbol con todas las posibles soluciones pero en el momento que una rama ya no sea la mejor, se deja de construir el árbol por esa rama, para ahorrar recursos computacionales, de esta manera se puede llegar a la solución óptima sin necesidad de explorar todas y cada una de las posibles soluciones.

Para el caso del VRP se debe tener una solución factible inicial con una distancia total recorrida asociada y así realizar el árbol cortando las ramas que superen esta distancia.

Vamos a ver un ejemplo VRP con los siguientes datos:

- n = 4 clientes
- el nodo 0 es el deposito
- Matriz de distancias = { $c_{i,j}$ } =

(i,j)	0	1	2	3	4
0	0	9	8	6	13
1	9	0	8	11	13
2	8	8	0	5	6
3	6	11	5	0	8
4	13	13	6	8	0

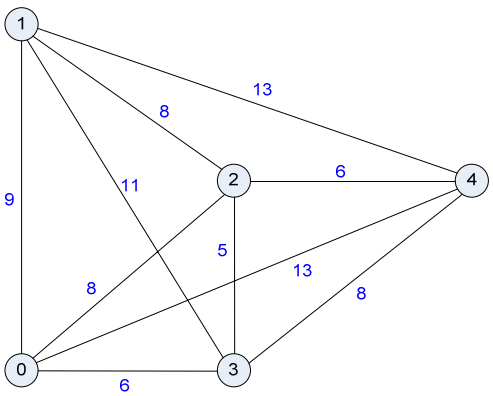


Figura 12: Grafo B&B

Dibujando un grafo con todas las posibles soluciones podríamos encontrar la solución mas corta recorriendo todas las ramas.

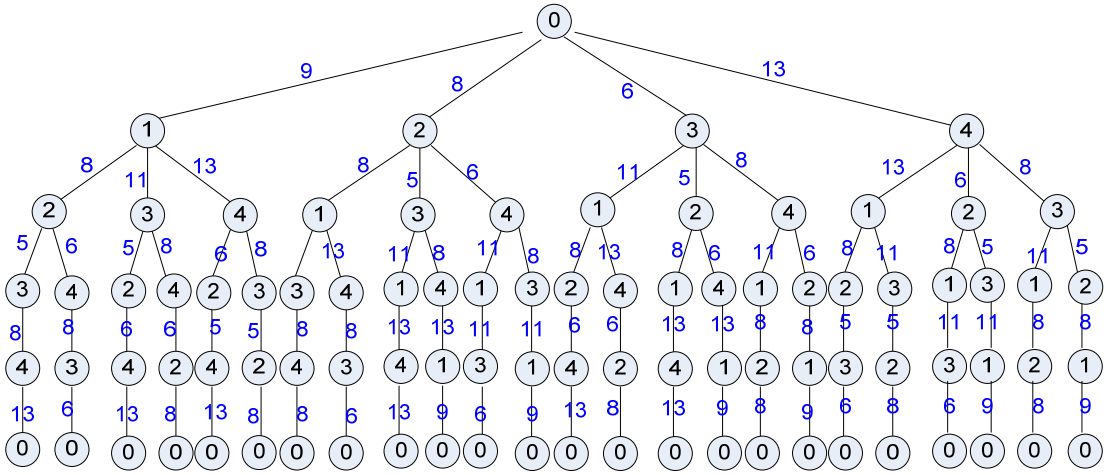


Figura 13: Árbol para B&B

Usando la técnica del Branch and Bound podemos ahorrarnos el recorrido por todas las ramas del árbol teniendo una solución factible inicial.

Suponemos que se nos da una solución factible inicial de 40.

En la Figura 14 aparecen en rojo todas las rutas que superan la solución inicial de 40 y que ya no serian exploradas por el agoritmo.

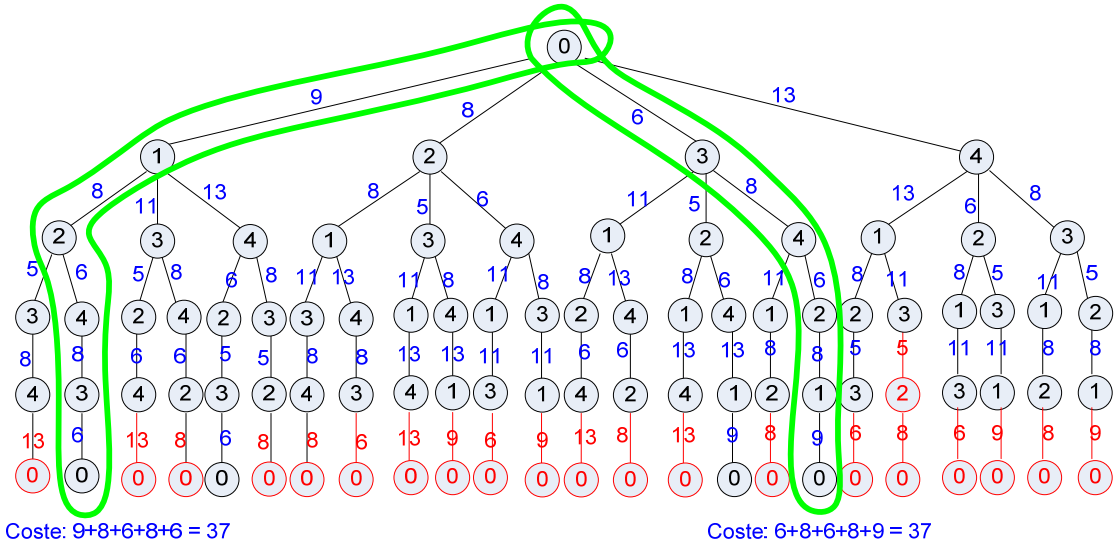


Figura 14: Solución B&B

Asi pues solo tenemos dos posibles rutas :

0 → 1 → 2 → 4 → 3 → 0 o 0 → 3 → 4 → 2 → 1 → 0 ::: Coste = 37

0 → 1 → 4 → 2 → 3 → 0 o 0 → 3 → 2 → 4 → 1 → 0 ::: Coste = 39

Nos quedamos con la ruta mas corta de coste 37 representada en la Figura 15.

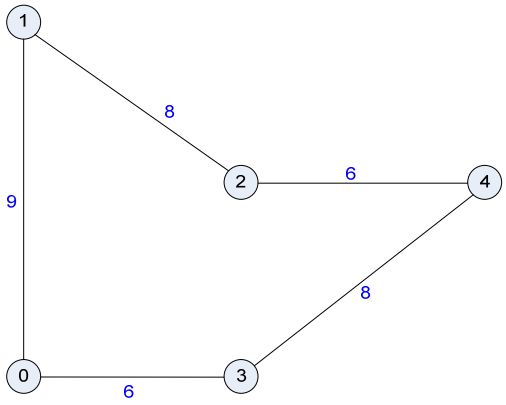


Figura 15: Ruta B&B

2.2. Algoritmos heurísticos

Las técnicas heurísticas son algoritmos que encuentran soluciones de buena calidad para problemas combinatorios complejos explotando el conocimiento del dominio de aplicación. Los algoritmos heurísticos son fáciles de implementar y encuentran buenas soluciones con esfuerzos computacionales relativamente pequeños; sin embargo, renuncian (desde el punto de vista teórico) a encontrar la solución óptima global de un problema. En problemas de gran tamaño rara vez un algoritmo heurístico encuentra la solución óptima global.

Dentro de los métodos heurísticos existen tres grupos; los constructivos, los de dos fases y los métodos de mejora.

2.2.1. Heurísticos constructivos

Estos métodos parten de una solución inicial vacía y construyen de forma gradual una solución factible cumpliendo las restricciones del problema y tratando de obtener la mejor solución posible.

2.2.1.1. Nearest Neighbor (vecino más cercano)

En este método la ruta se construye de la manera siguiente:

- Se elige el cliente más próximo al almacén.
- Después se selecciona el cliente más próximo que cumpla todas las restricciones y que todavía no haya sido visitado y así sucesivamente hasta que se han visitado todos los clientes.

Se suele utilizar para encontrar la primera solución para otros tipos de algoritmos, tales como busca tabú, hormigas, etc.

Vamos a ver un ejemplo VRP con los siguientes datos:

- n = 5 clientes

- el nodo 0 es el depósito
- Matriz de distancias = { $c_{i,j}$ }

(i,j)	0	1	2	3	4	5
0	0	2	4	6	6	5
1	2	0	2	3	4	4
2	4	2	0	4	5	6
3	6	3	4	0	2	6
4	6	4	5	2	0	5
5	5	4	6	6	5	0

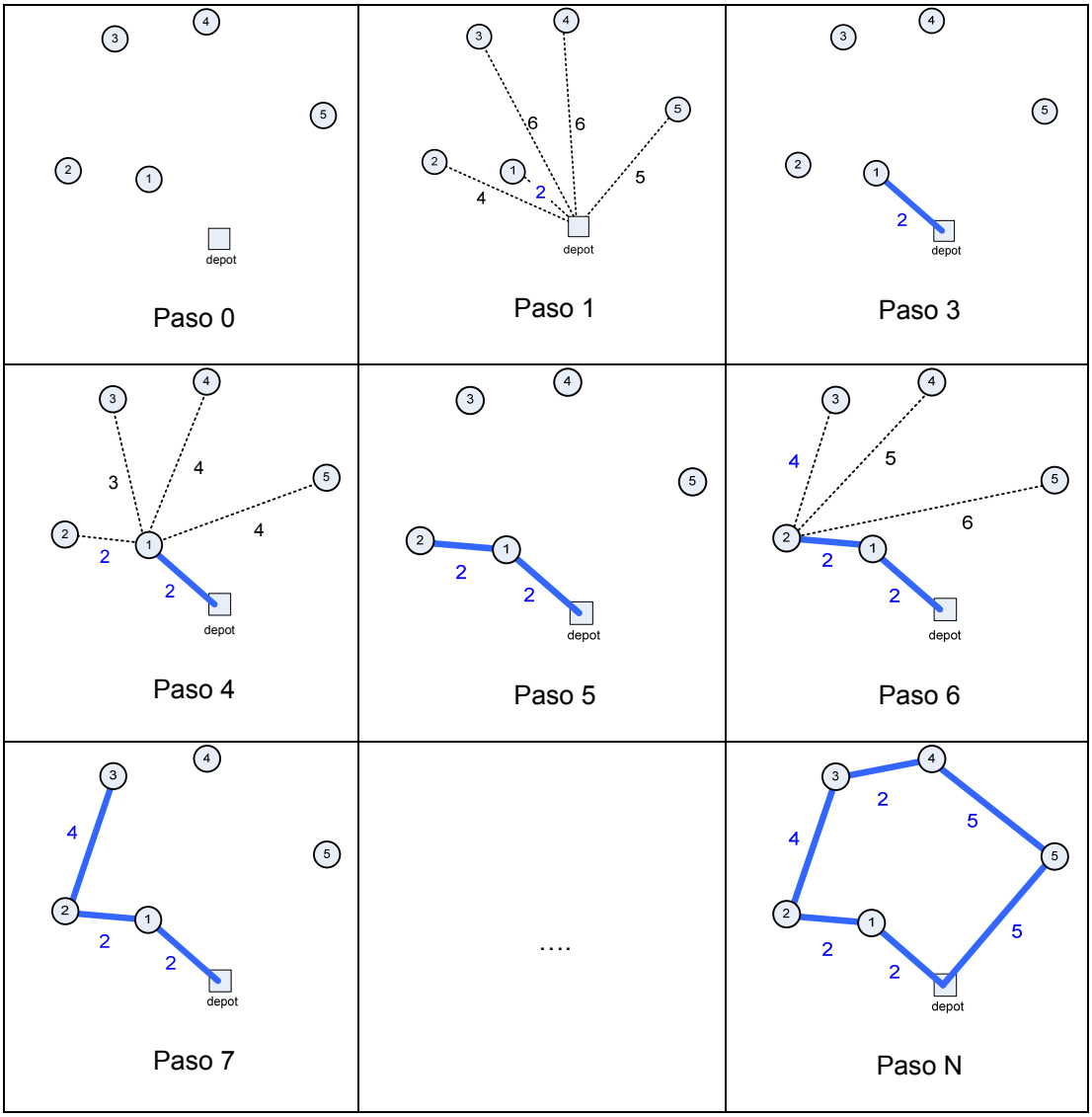


Figura 16: Ejemplo de Nearest Neighbor

Solución:

Coste total = 2 + 2 + 4 + 2 + 5 + 5 = 20

Secuencia = 0 1 2 3 4 5 0

Una extensión del *cliente más cercano* es el Nearest Addition (Adición más próxima) y consiste en seleccionar uno de los clientes que aún no han sido asignados a ninguna ruta y añadirlo a una ruta, pero en vez de hacerlo al final como en el *cliente más cercano*, hacerlo entre dos clientes de la ruta.

2.2.1.2. Clarke and Wright (método de los ahorros)

Este algoritmo también es conocido como método de los "ahorros" [Robus05]. Puede resolver el CVRP donde el número de recursos a utilizar es libre.

El método comienza con rutas que sólo están formadas por el depósito y un solo nodo. A cada paso del algoritmo se unen dos rutas si se genera un ahorro en tiempo/distancia

El algoritmo es el siguiente:

```
1. Calcular los ahorros s(i,j) = c(D,i) + c(D,j) - c(i,j), para todas
   las parejas de clientes y, j. En este caso D representa el almacén.
2. Ordenar los ahorros en orden decreciente.
3. Repetir
   3.1.  Empezando por el de mayor ahorro (fila superior del listado)
         buscar el primer arco viable (que cumpla todas las restricciones)
         que puede utilizarse para expandir la ruta en construcción.
   3.2.  Si la ruta no se puede expandir más (el vehículo ha llegado a
         su capacidad máxima), terminarla y empezar una nueva.
4. Hasta que no hayan más arcos disponibles.
5. Devolver la mejor solución encontrada.
```

Algoritmo Clarke and Wright

Un problema que tiene este algoritmo es que puede dejar clientes sin asignar a una ruta particular, o bien producir rutas circulares, y se ha comprobado su ineficiencia para algunos casos concretos, como por ejemplo, cuando los clientes ocupan posiciones equidistantes en los vértices de una red cuadrada.

Vamos a ver un ejemplo CVRP con los siguientes datos:

- n = 4 clientes
- el nodo 0 es el depósito
- di = (0, 2, 7, 6, 4) demanda

- Matriz de distancias = { $c_{i,j}$ } =

(i,j)	0	1	2	3	4
0	0	3	6	9	7
1	3	0	4	12	10
2	6	4	0	15	13
3	9	12	15	0	5
4	7	10	13	5	0

- Q = 10 capacidad de los vehículos

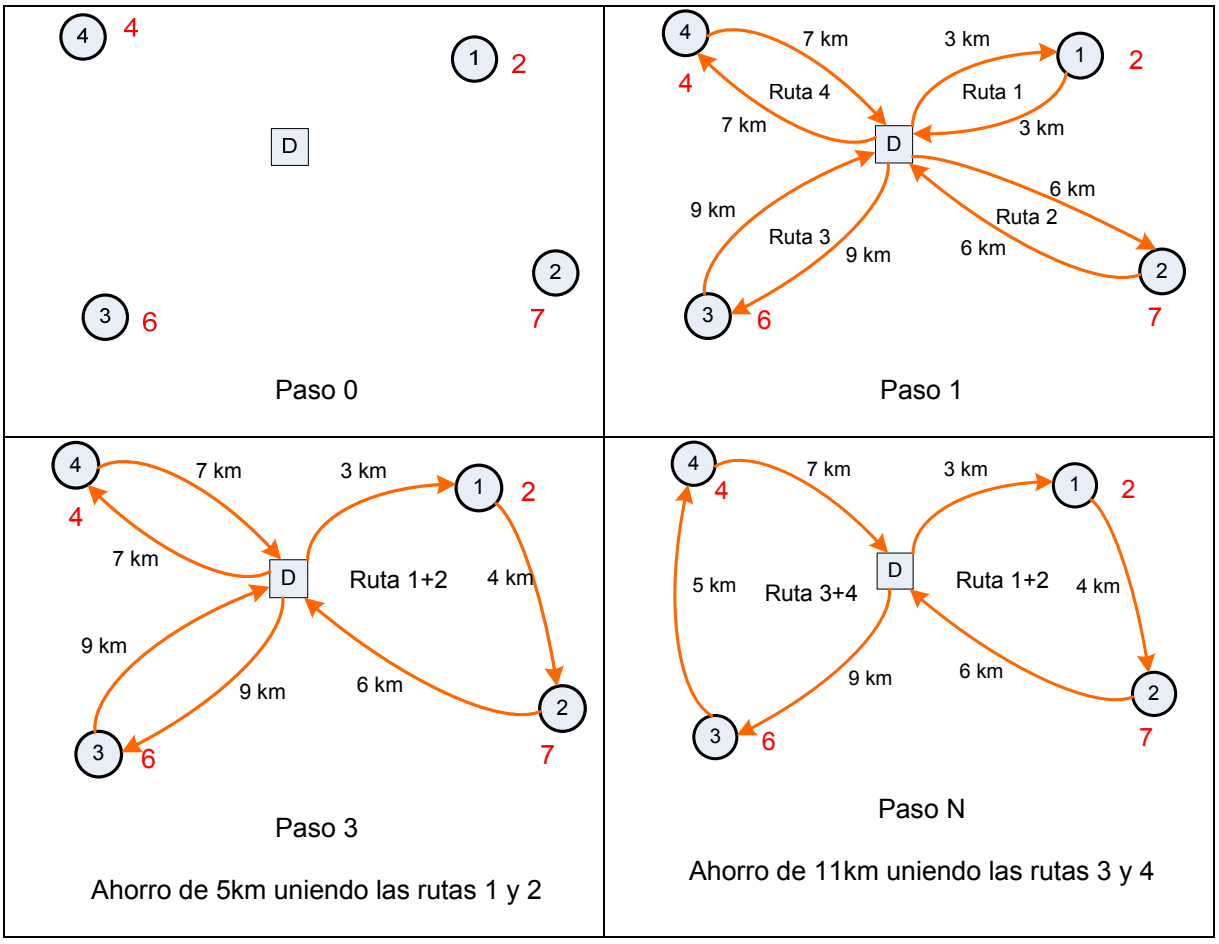


Figura 17: Ejemplo de Clarke and Wright

Solución:

Coste total = 34 km

Rutas:

- Coste = 3 + 4 + 6 = 13 km

  Demanda = 7 + 2 = 9

  Secuencia = 0 1 2 0

- Coste = 9 + 5 + 7 = 21 km

  Demanda = 4 + 6 = 10

  Secuencia = 0 3 4 0

2.2.2. Heurísticos de dos fases

Estos algorismos se pueden desarrollar de dos formas [Robus05]:

- *Cluster first, route second.*

Primero se divide la ciudad o región en distritos (o grupo de clientes) y luego se diseñan las rutas simples dentro de cada una de ellos. Usualmente este orden produce mejores resultados que realizar las tareas en orden inverso.

- *Route first, cluster second.*

Primero se diseña una gran ruta óptima para toda la región y luego se subdivide la ruta en un número de subrutas que serán cubiertas por diferentes vehículos.

2.2.2.1. Algoritmo de Gillet y Miller (método de barrido)

Este algoritmo asume que son conocidas las coordenadas polares (r_i, θ_i) de todos los puntos que deben ser visitados y que el depósito puede ser adoptado como origen de estas (r_i=0). La ruta se inicia con la unión del depósito a un punto arbitrario y los restantes puntos de ésta son determinados en términos de incrementos de ángulo de barrido (sentido horario y antihorario).

La dirección de la unión de los dos primeros se puede considerar com el ángulo θ_i =0. La ruta cubre tantos puntos de demanda como la capacidad del vehículo (C_k) permita, por lo tanto, el primer punto que no pueda ser incluido será el comienzo de la próxima ruta.

El proceso se completa cuando todos los puntos del sistema han sido barridos, o sea, que pertenecen a alguna ruta. Conocido el grupo de puntos que forman parte de una ruta, se puede usar un algoritmo de TSP para realizar el orden de visita de los puntos.

1. Tomar un vehículo ***k*** que aún no haya sido utilizado.
2. Empezando por el cliente ***i*** que aún no haya sido asignado a ninguna ruta y posea menor ángulo ***i***, incluir los siguientes ***i+1***, ***i+2***, etc., en la ruta ***k*** mientras no se supere la capacidad ***Ck*** del vehículo.
3. Si se han cubierto todos los puntos o bien se han utilizado todos los vehículos, proceder con el 4° paso. En caso contrario, volver al 1er paso.
4. Conectar los puntos dentro de cada sector circular empezando y finalizando en el almacén.

Vemos un ejemplo de aplicación CVRP del algorismo:

- n = 9 clientes
- el nodo 0 es el depósito

- di = (0, 3, 3, 2, 4, 2, 2, 2, 6, 6) demanda
- *Capacidad de los vehículos Ck* = 6

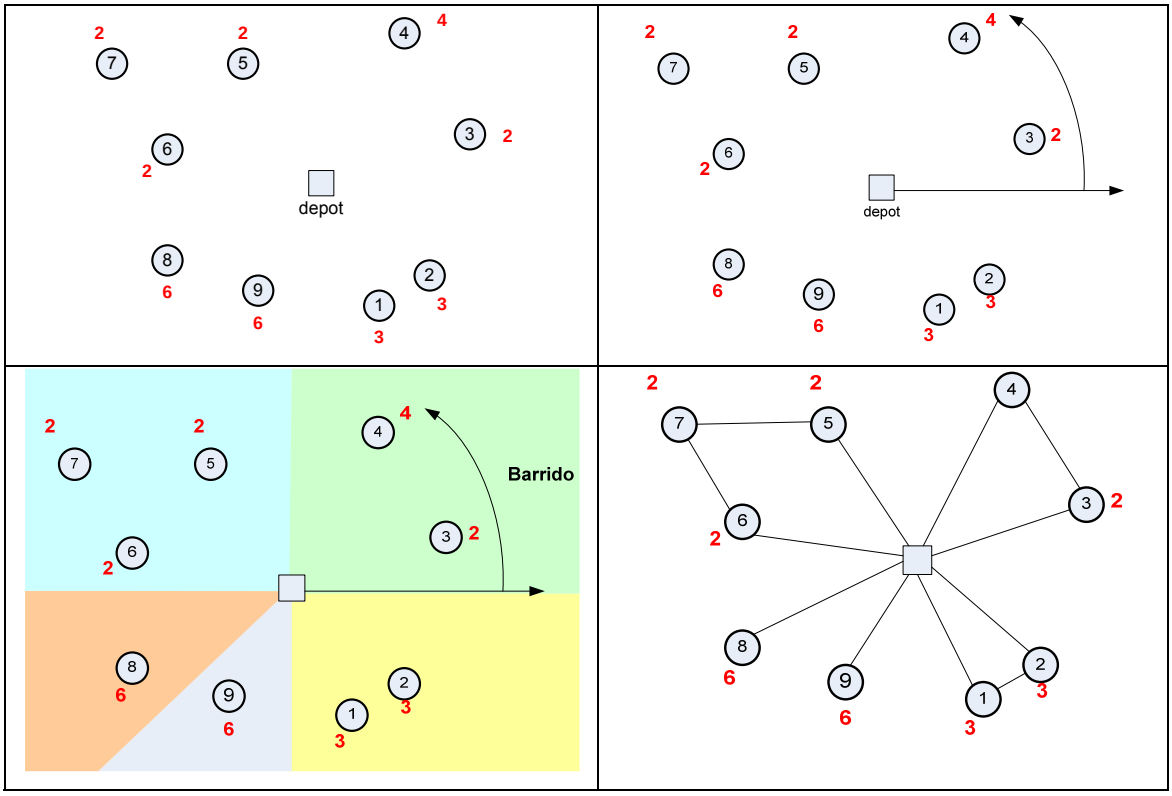


Figura 18: Ejemplo barrido

La mayor desventaja de este algoritmo es que genera las rutas a partir de los nodos determinados por el barrido. Esto puede traer problemas en los casos en que las rutas tengan restricción de tiempo y/o longitud, ya que muchas veces el grupo de nodos elegidos por el barrido lo superan.

La solución proporcionada por este algorimo no es óptima y depende enormemente del cliente escogido para empezar a barrer y del orden en el cual se asignan los vehículos. De hecho, en ciertos casos la aplicación de este algoritmo no es ni siquiera recomendable.

2.2.3. Heurísticos de mejora o de búsqueda local

Partiendo de una solución inicial, estos métodos intentan mejorarla haciendo pequeños cambios sucesivamente.

Según [Caric08] el proceso de mejora o búsqueda local parte de una solución inicial y mediante la iteración de ciertos “movimientos” se mueve de la solución actual a una solución vecina en el espacio de búsqueda en el que cada punto tiene un número relativamente pequeño de posibles vecinos. Los operadores de búsqueda local se pueden dividir en dos grupos:

- **Operadores Intra-Ruta**: estos operadores trabajan con los nodos pertenecientes a una única ruta. El objetivo de éstos es disminuir la distancia/tiempo total de la ruta.

- **Operadores Inter-Ruta**: estos operadores trabajan con nodos pertenecientes a diferentes rutas. El objetivo de éstos es disminuir la distancia/tiempo total de la ruta y en algunos casos también reducir el número de vehículos utilizados.

Estos operadores se pueden aplicar a problemas VRP, CVRP y CVRPTW.

Relocate intra-ruta: Modifica la posición del cliente a1 de su posición entre a0 y a2 por la posición entre b0 y b1. Este movimiento sólo se hará si se reduce la longitud total de la ruta.

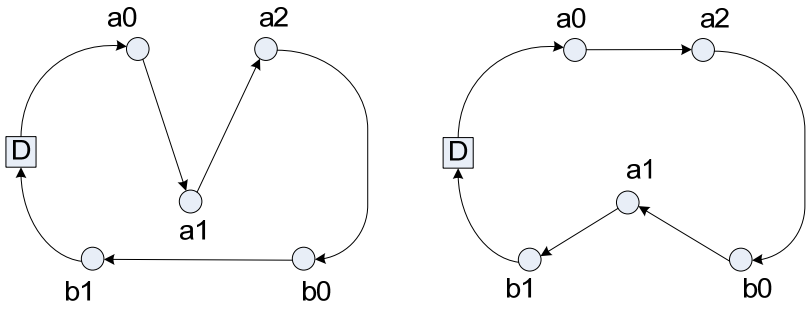


Figura 19: Relocate intra-ruta

Exchange intra ruta: Intercambia la posición de los clientes a1 y b1. Este movimiento sólo se hará si se reduce la longitud total de la ruta.

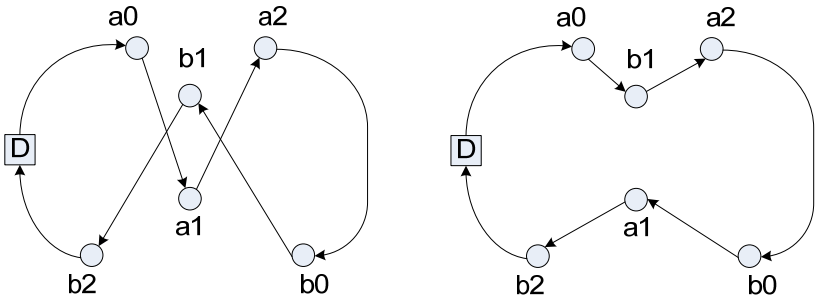


Figura 20: Exchange intra-ruta

2-Opt intra-ruta: Transforma las intersecciones de los arcos si se reduce la distancia después de haber cambiado la dirección de los arcos entre a0 y b1 y haber borrado y añadido los arcos correspondientes. Además se invierte el orden del tramo que queda enmarcado entre los arcos eliminados. En el siguiente ejemplo, se invertiría el orden de (b0,a1).

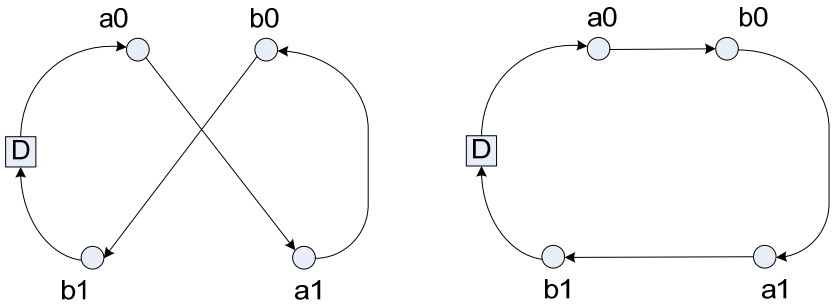


Figura 21: 2-Opt intra-ruta

Or-Opt intra-ruta: Si existe una reducción de distancias, transforma la intersección de los arcos reordenando a los clientes de una ruta. Es muy rápido.

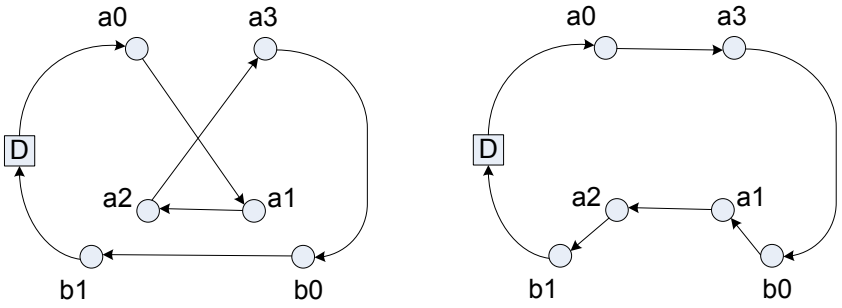


Figura 22: Or-Opt intra-ruta

Relocate inter-ruta: Mueve al cliente a1 de una ruta a otra si existe un ahorro de distancia recorrida.

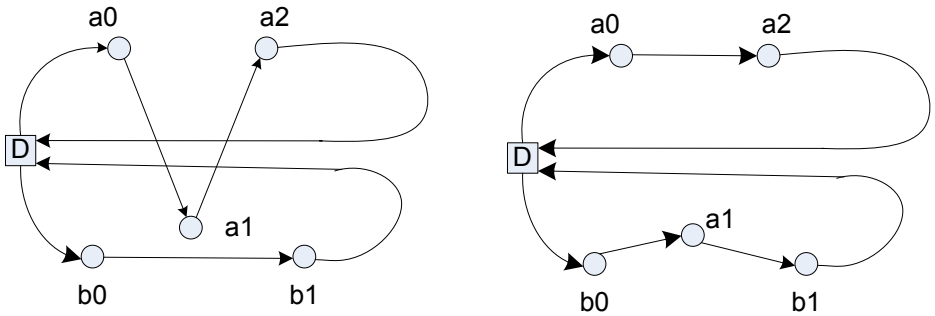


Figura 23: Relocate inter-ruta

Exchange inter-ruta: Intercambia a dos clientes a1 y b1 de diferentes rutas si existe un ahorro.

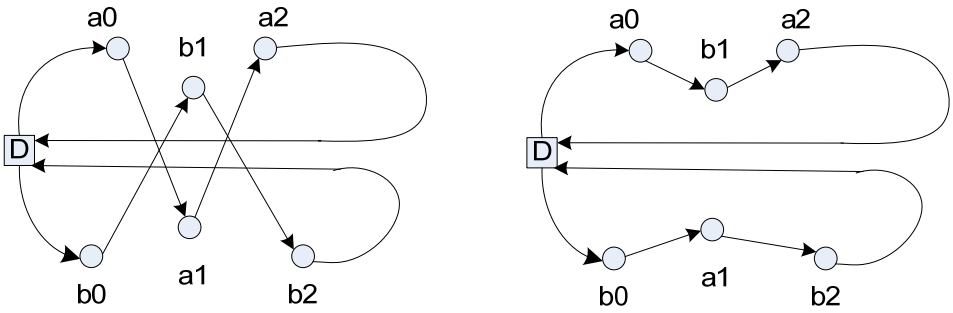


Figura 24: Exchange inter-ruta

Cross-Exchange inter-ruta: Intercambia dos grupos de clientes entre dos rutas. El grupo está formado entre 1 y 5 clientes si existe un ahorro.

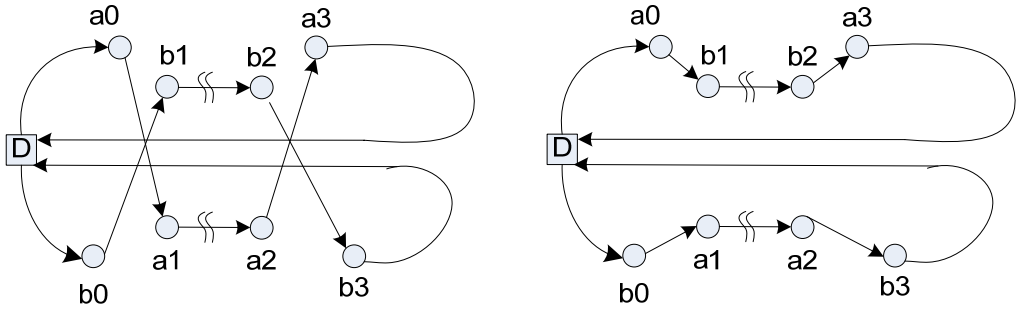


Figura 25: Cross-Exchange inter-ruta

ICross-Exchange inter-ruta: Intercambia dos grupos de clientes como el caso anterior, pero además intercambia la orden de visita de los clientes.

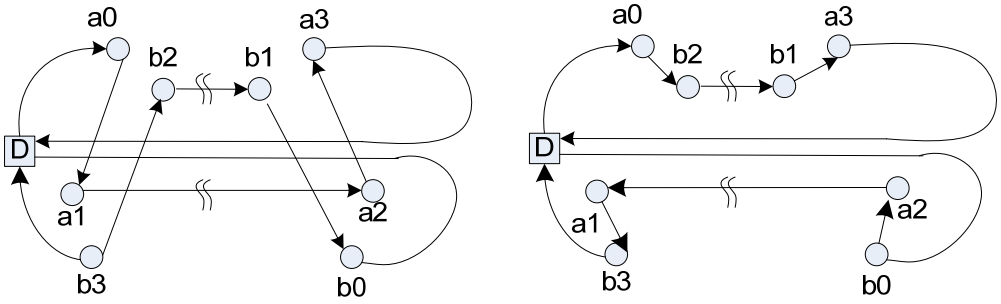


Figura 26: ICross-Exchange inter-ruta

2-Opt* inter-ruta: Es como un Cross-Echange pero en donde b2 y a2 son almacenes.

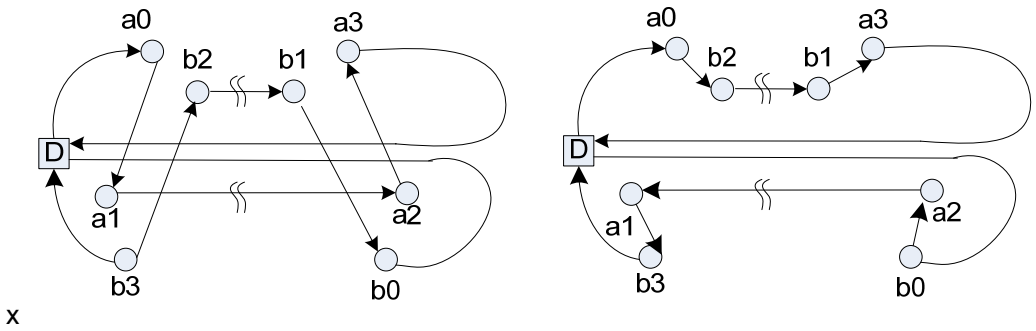


Figura 27: 2-Opt* inter-ruta

Vamos a ver un ejemplo de aplicación de diferentes operadores heurísticos intra-ruta y inter-ruta.

Partiendo de una solución inicial como la siguiente:

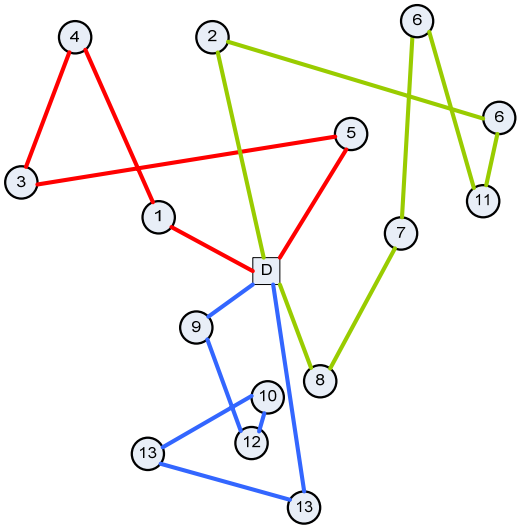


Figura 28: Solución inicial

Aplicamos primero algunos operadores Intra-Ruta.

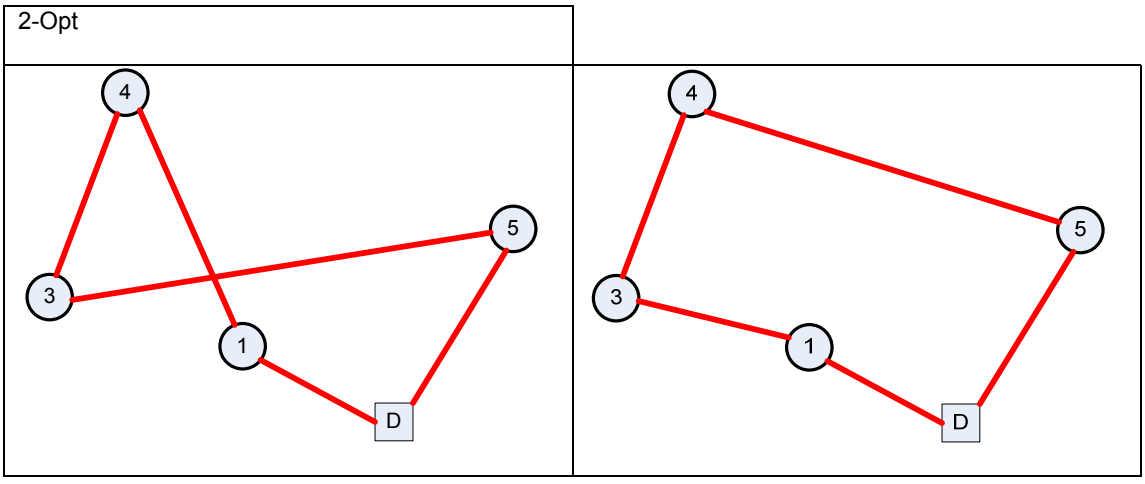


Figura 29: Solución 2-Opt

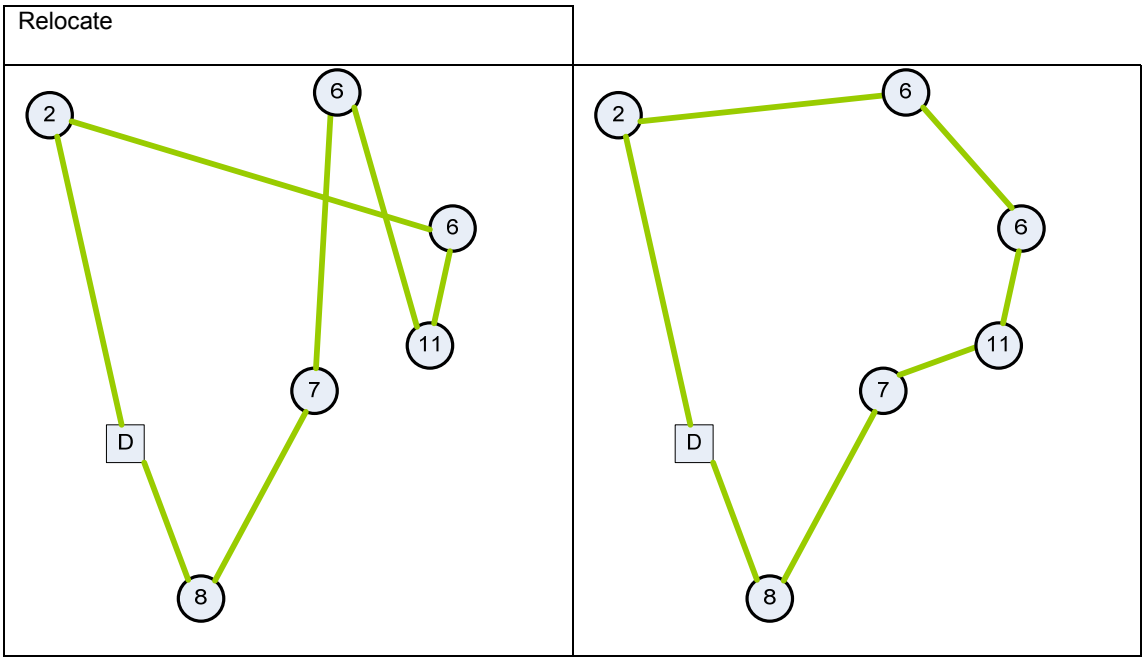


Figura 30: Solución Relocate

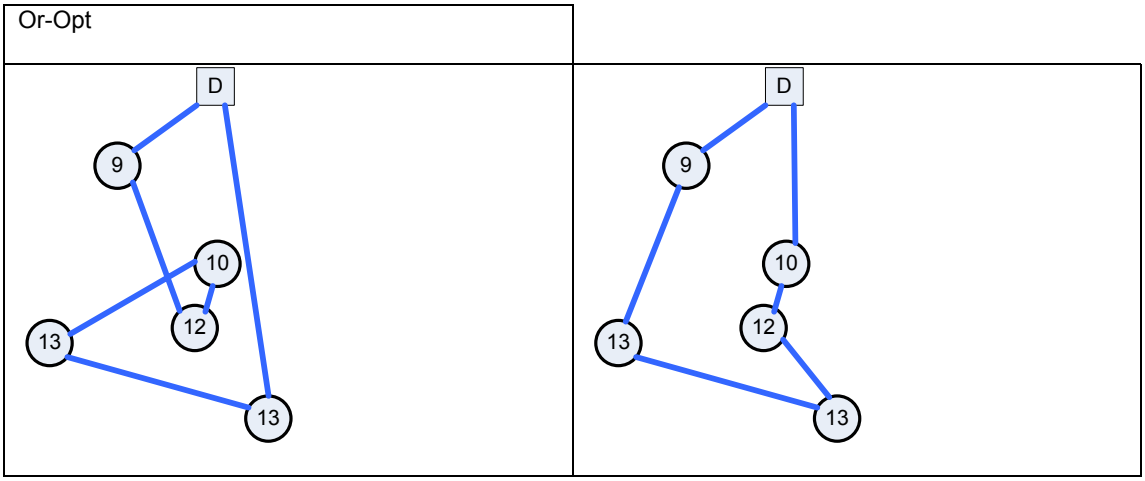


Figura 31: Solución Or-Opt

Después de la aplicación de los operadores nos quedaría la siguiente solución.

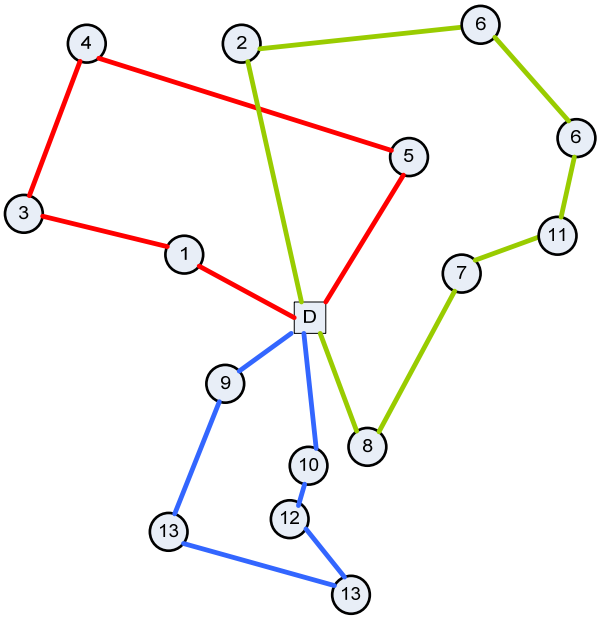


Figura 32: Solución conjunta

Ahora aplicaremos operadores Inter-Ruta

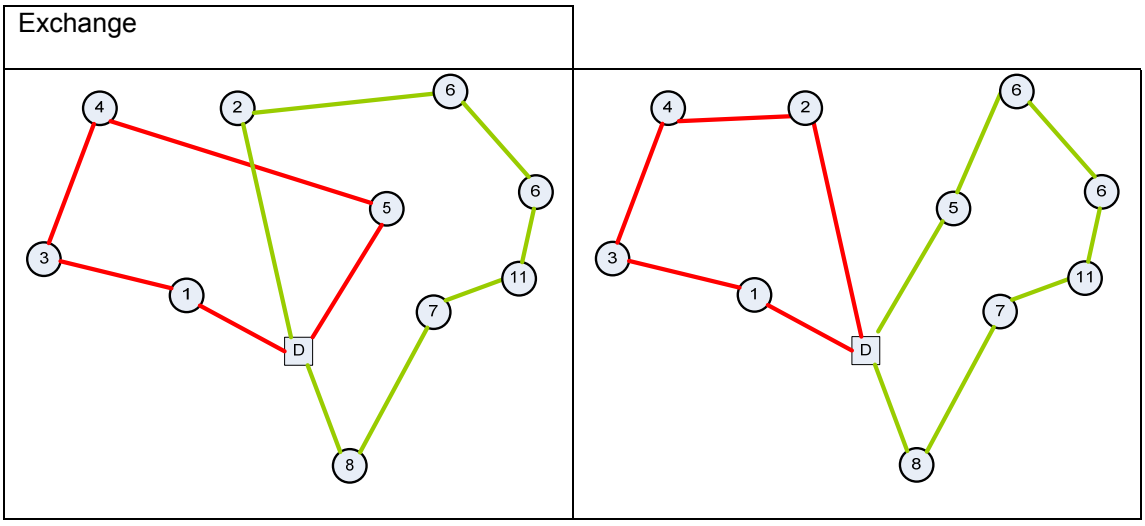


Figura 33: Solución Exchange

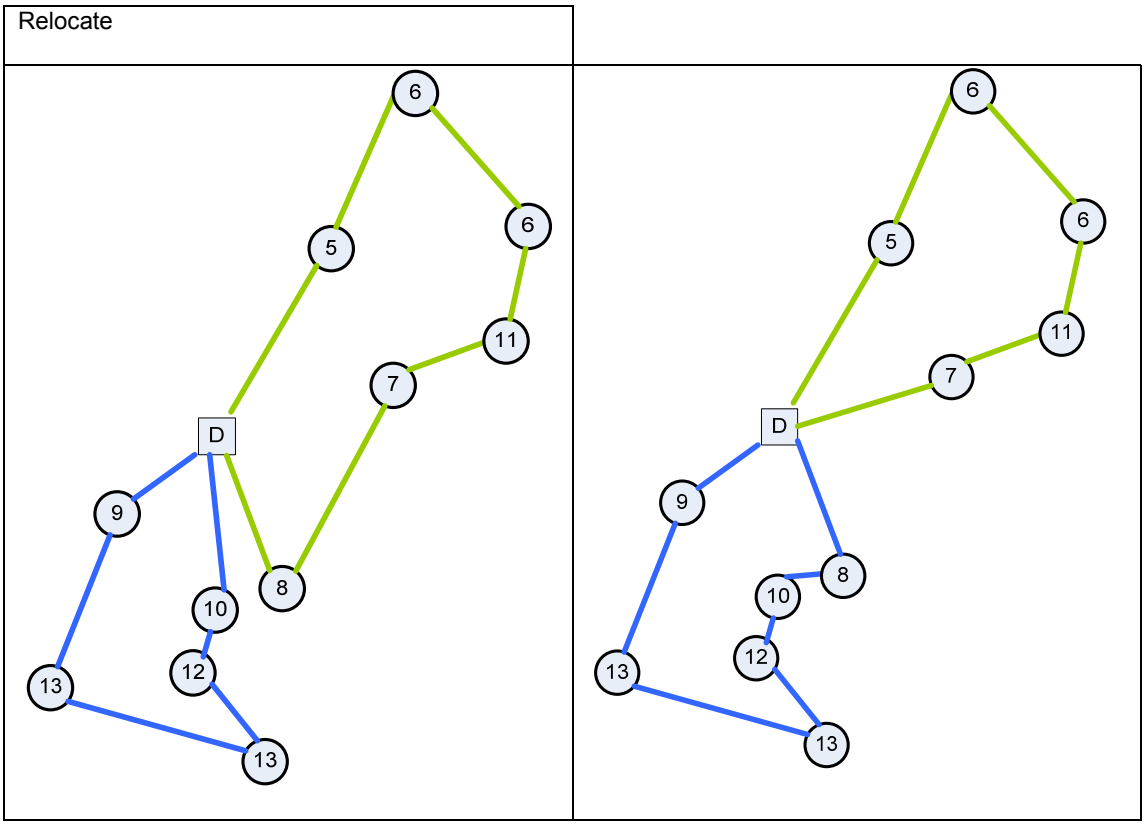


Figura 34: Solución Relocate

Finalmente tendríamos la siguiente solución que mejora sustancialmente el coste de la solución inicial.

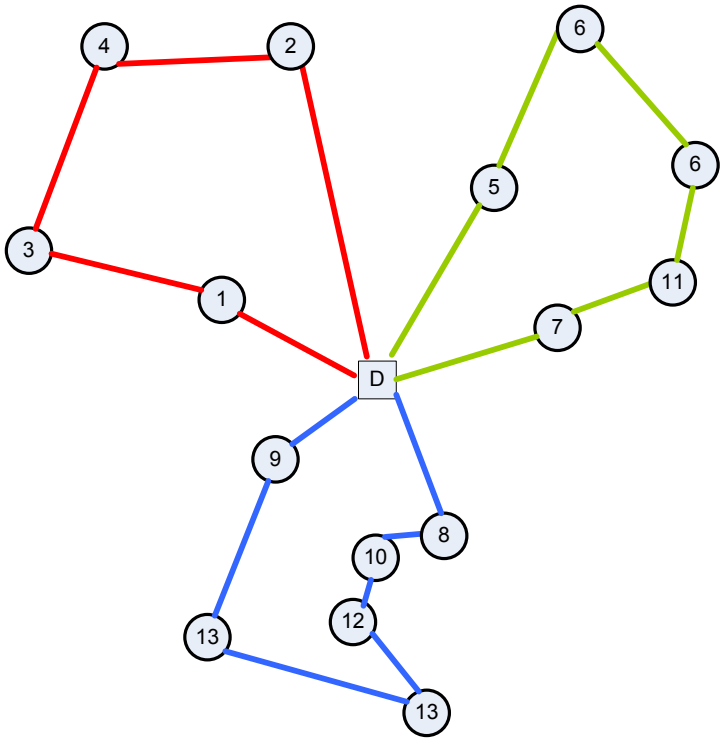


Figura 35: Solución conjunta

Como hemos podido ver más arriba, existe una gran cantidad de operadores de búsqueda local, que trabajan tanto dentro de una ruta, como entre diferentes rutas. Actualmente, la gran mayoría, sino todos los estudios y propuestas científicas, así como algunos productos comerciales, utilizan uno o varios de estos operadores para realizar fases de mejora a una o más de las soluciones encontradas mediante otros métodos. De esta manera se intenta llegar a mejores soluciones en menor tiempo, ya que la búsqueda local suele ser bastante rápida en términos de tiempo de cálculo.

2.3. Algoritmos metaheurísticos

Los algoritmos metaheurísticos son algoritmos de propósito general, que no dependen del problema, y que ofrecen buenos resultados pero que normalmente no acaban ofreciendo la solución óptima sino soluciones *sub*óptimas. Se acostumbran a utilizar para aquellos problemas en que no existe un algoritmo o heurística específicos que los resuelva, o bien cuando no es práctico implementar dichos métodos.

Existen multitud de implementaciones e híbridos, los más importantes:

- GA: Genetic Algorithms (Algoritmos Genéticos)
- TS: Tabu Search (Búsqueda Tabú)
- SA: Simulated Annealing (Recocido Simulado)
- ACO: Ant Colony Optimization (Optimización con Colonia de Hormigas)

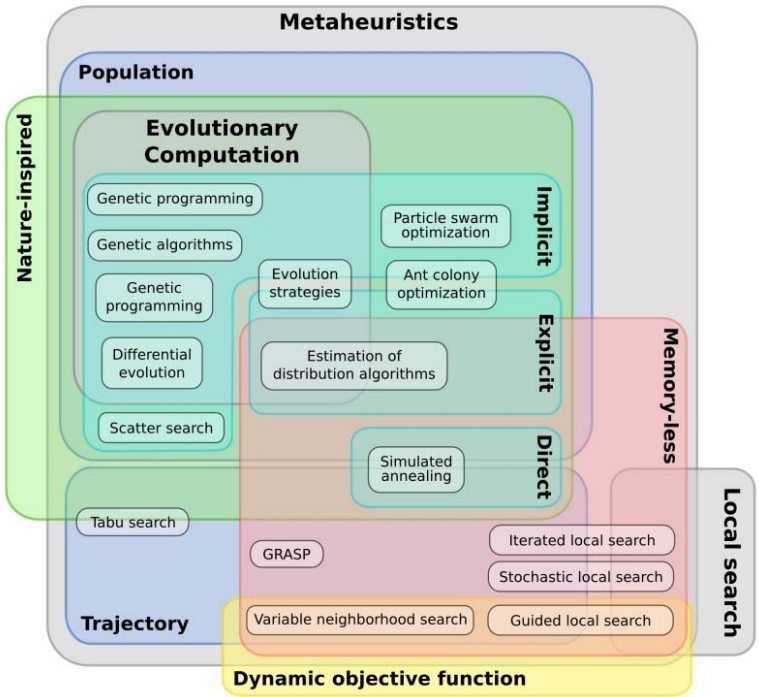


Figura 36: Clasificación de las Metaheurísticas

2.3.1. Algoritmos Genéticos

Un algoritmo genético (AG o GA) es una técnica de programación que imita la evolución biológica como estrategia para resolver problemas. Dado un problema a resolver, la entrada del GA es un conjunto de soluciones potenciales o candidatas a este problema, clasificadas de alguna manera, y una métrica llamada función de aptitud (fitness) que permite evaluar a cada candidata. Estas candidatas (normalmente generadas de manera aleatoria) pueden ser soluciones que ya se sabe que funcionan, con el objetivo de que el GA las mejore.

Una vez tenemos un conjunto de soluciones iniciales o población inicial, el GA evalúa a cada candidata con la función de aptitud. De todas estas, muchas pueden ser inservibles, pero unas pocas pueden ser muy interesantes o buenas.

Estas buenas candidatas se conservan y se les permite que se reproduzcan. Esto hace que se hagan cruces entre estas y tengan descendencia. Esta descendencia puede recibir una mutación y cambiar alguno de sus valores. Una vez tenemos una nueva población, volvemos a evaluar, crear descendencia, mutar, etc.

El factor clave de estos algoritmos es encontrar una buena representación de los datos, las probabilidades de cruce y las probabilidades de mutación.

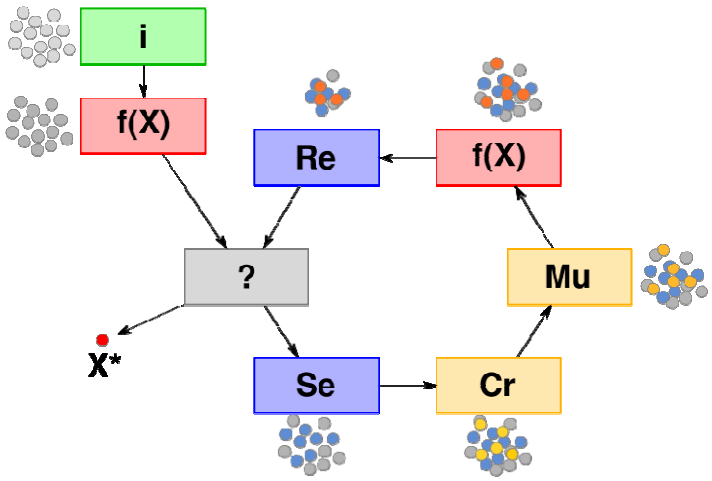


Figura 37: Estructura del algoritmo genético

En la figura 14 podemos ver cual es la estructura de un algoritmo genético y cual es el flujo de funcionamiento. A continuación mostramos su pseudocódigo:

```
1. Crear población inicial (Cuadro i de la figura)
2. Calcular la función de fitness (f(x)) de todos los elementos de la
   población
3. Mientras que no se cumpla la condición de parada (Cuadro ? de la figura)
   3.1.      Seleccionar los elementos para el cruce (Cuadro Se de la figura)
   3.2.      Realizar el cruce de los elementos seleccionados (Cuadro Cr de
```

```
      la figura)
   3.3.    Realizar la mutación de algunos elementos de la población
      (Cuadro Mu de la figura)
   3.4.    Calcular la función de fitness (f(x)) de todos los elementos de
      la nueva población
   3.5.    Reemplazar la antigua población por la nueva
4. Retornar la mejor solución hallada.
```

Algoritmo genético

El uso de este tipo de algoritmos se está extendiendo en los últimos tiempos en este tipo de problemas, tanto a nivel académico como a nivel comercial, donde muchos productos utilizan, o eso afirman en sus respectivas páginas web, como núcleo un algoritmo genético, aunque luego pueda ser ampliado mediante otras técnicas.

Estos algoritmos tienen un buen nivel de diversificación que permite explorar de una manera adecuada todo el espacio de soluciones y, gracias a conceptos como el elitismo, también proporciona un amplio grado de intensificación para poder explotar determinadas zonas del espacio de búsqueda y encontrar mejores soluciones.

Además, el operador de mutación, puede hacer que salgamos de mínimos locales que nos están ocultando otros mínimos que son mejores que el actual.

Por último, estos algoritmos, como se comenta en [Tan01], son un buen compromiso entre la calidad de soluciones que dan y el tiempo de cómputo para llegar a esas soluciones.

El uso de este tipo de algoritmos se está extendiendo en los últimos tiempos en este tipo de problemas, tanto a nivel académico como a nivel comercial, donde muchos productos utilizan, o eso afirman en sus respectivas páginas web, como núcleo un algoritmo genético, aunque luego pueda ser ampliado mediante otras técnicas.

Este tipo de algoritmos tienen un buen nivel de diversificación que permite explorar de una manera adecuada todo el espacio de soluciones y, gracias a conceptos como el elitismo, también proporciona un amplio grado de intensificación para poder explotar determinadas zonas del espacio de búsqueda y encontrar mejores soluciones.

Además, el operador de mutación, puede hacer que salgamos de mínimos locales que nos están ocultando otros mínimos que son mejores que el actual.

Por último, estos algoritmos, como se comenta en [Tan01], son un buen compromiso entre la calidad de soluciones que dan y el tiempo de cómputo para llegar a esas soluciones.

2.3.1.1. Experiencia encontrada

En [Alba06] nos presentan un método para resolver el problema de CVRP basado en algoritmos genéticos celulares (cGA) y un método especializado de búsqueda local. Estos algoritmos son una subclase de los algoritmos genéticos en los que la población es estructurada con una topología

determinada (normalmente una red) y las operaciones genéticas sólo se utilizan en una pequeña vecindad de cada individuo.

En este algoritmo, llamado JCell2oli, y en el algoritmo [Dorro07] (que es una variación de este) las soluciones (cromosomas) serán representadas por permutación de enteros. Cada permutación contendrá tanto los nodos a visitar como unos separadores de rutas para los diferentes camiones.

La población tiene una topología de tipo 2D toroidal. Esto quiere decir que, como población tenemos una matriz en la que los lados están conectados entre sí, es decir, el elemento de arriba a la izquierda de la matriz está conectado con el de su derecha, el de abajo, el último de la fila y el último de la columna. De esta forma, todo elemento tiene siempre 4 vecinos.

El algoritmo es el siguiente:

```
1. proc Steps_Up(cga) // Los parámetros del algoritmo están en cga.
2.   while not Termination_Condition() do
3.     for x = 1 to WIDTH do // Recorremos todos los elementos de la matriz.
4.       for y = 1 to HEIGHT do
              // GENÉTICO
5.         n_list = Get_Neighb(cga, x, y); // (x,y) (Norte, Sur, Este, Oeste)
6.         parent1 = Binary_Tournament(n_list);
7.         parent2 = Individual_At(cga, x, y);
8.         offspring = ERX(cga, Pc, parent1, parent2); // recombinación  vértices
9.         offspring = Combined_Mut(cga, Pm, offspring); // Utilizamos 3 métodos de
                                                         // mutación equiprobables:
                                                         // Insertion, Swap e
                                                         // Inversion.
             // BÚSQUEDA LOCAL
10.           sol_Opt = 2-Opt(offspring); // heurístico 2-Opt
11.           sol_Int = 1-Interchange(offspring); // 1-Interchange a la solución
12.           offspring = Best(offspring, sol_Opt, sol_Int); // Escogemos el mejor
                                                             // cromosoma entre la
                                                             // salida del genético,
                                                             // 2-Opt y 1-Interchange.
13.           Insert_If_Better(offspring, cga, aux_pop, x, y);
14.         end_for
15.       end_for
16.       cga.pop = aux.pop; // Substituimos la población actual por la nueva.
17.       Update_Statistics(cga); // Actualizamos estadísticas como el máximo, mínimo
                                  // y valor medio de fitness ...
18.     end_while
19.   end_proc Steps_Up;
```

La población inicial se crea de forma aleatoria. Se pueden aplicar métodos para corregir las soluciones que no sean válidas. En [Dorro07] también se crea la población inicial de manera aleatoria, pero si las soluciones no son válidas debido a que un camión está situado en una ruta con más capacidad de la que puede llevar, se aplica el siguiente método: se divide esta ruta en subrutas y un nuevo vehículo se añade a la solución. Si es necesario, se crean nuevas subrutas hasta que se tenga una solución válida.

El operador de cruce utilizado aquí es el ERO o ERX (Edge Recombination Operator). Su algoritmo es el siguiente:

```
1. Guardamos todas las conexiones de los nodos de los dos padres. Por
   ejemplo si los padres son [A,B,C,D,E,F] y [B,D,C,A,E,F] la tabla de
   conexiones sería la siguiente:
```

```
2. A: B F C E          D: C E B
3. B: A C D F          E: D F A
4. C: B D A            F: A E B

5. Escogemos una ciudad inicial de uno de los dos parientes (esta ciudad
   puede ser escogida al azar o bien siguiendo el criterio del paso 5).
6. Eliminamos las ocurrencias de la ciudad escogida en el resto de vecinos.
7. Si la ciudad tiene nodos en su lista de vecinos, entonces pasamos al
   paso 5: sino, vamos al paso 6.
8. Determinamos qué ciudad en la lista de vecinos de la ciudad actual tiene
   menos vecinos en la su lista. Esta ciudad pasa a ser la ciudad actual.
   En caso de empate se escoge al azar. Ir al paso 3.
9. Si no hay ciudades sin visitar, entonces, PARAR. De otra forma, escoger
   al azar una ciudad sin visitar e ir al paso 3.
```

En [Dorro07], [Perei02] también se presenta un algoritmo similar a este pero con ciertas diferencias. Aquí utilizan el segundo recombinador, que promueve la diversidad de la población:

```
// Sean l1 y l2 los padres a recombinar
1. Escoger una subruta SR={a1,...,an} de I2
2. Encontrar la ciudad "c" que no pertenezca a SR y que esté más cerca
   geográficamente
3. Borrar todas las ciudades de I1 que estén incluidas en SR
4. Insertar SR en I1 de manera que a1 estará justo después de "c"
```

Los distintos tipos de **mutación** utilizados, son los siguientes:

- **Swap**: Intercambia la posición de dos nodos/servicios aleatoriamente, independientemente de si pertenecen a la misma ruta o no.
- **Inversion**: Intercambia el orden de visita de un subgrupo de nodos de la misma ruta.
- **Insertion**: Se selecciona un gen (independientemente de que sea un nodo o un separador de ruta) y se inserta en un lugar aleatorio del mismo cromosoma.

En [Dorro07], además de estos tres, proponemos otro método para la mutación:

- **Dispersion**: Es similar a la inserción pero aquí se aplica otra subruta en lugar de sólo un gen. Esto puede mover esta subruta dentro de una misma ruta o a otra ruta.

Los operadores de **búsqueda local** son:

- **2-Opt:** Este método siempre trabaja sobre una misma ruta. Elimina dos conexiones entre nodos y conecta los nodos de otra forma cambiando el orden de visita de la nueva ruta. Por ejemplo, si tenemos la ruta R={1,2,3,4,5,6,7,8,9,10}, quitamos la conexión {3,4} y la conexión {8,9}. Ahora reconectamos el nodo 3 al 8 y el 4 al 9 y cambiamos el orden de la subruta (5,6,7). La nueva ruta, R1 quedaria R1={1,2,3,8,7,6,5,4,9,10}.

- **1-Interchange**: Este método consiste en intercambiar un nodo/servicio de una ruta a otra o bien insertar un nodo de una ruta en otra.

En [Alba06] no se indica cuantas veces se aplican estos operadores sobre un cromosoma. En cambio en [Dorro07] indican que se apliquen 50 pasos de 1-Interchange y 50 pasos de 2-Opt.

En [Fal08] nos proponen un algoritmo memético (algoritmo genético más búsqueda local para intensificar la búsqueda) donde se reparten m productos distintos a n clientes y donde cada camión tiene una capacidad determinada para cada producto.

La codificación de los cromosomas consiste en un vector o lista de los clientes sin separadores de ruta.

Para crear rutas a partir de los cromosomas, utilizan un método (splitting procedure) basado en un grafo auxiliar H=(X,A,Z). X contiene todos los clientes, más un nodo ficticio. El conjunto de arcos A contiene un arco (i,j) si la ruta para visitar S(i+1) en S(j) incluido, es una ruta viable, por ejemplo, si la capacidad del camión y la distancia máxima de la ruta se respetan. El peso z(i,j) representa el coste de la ruta. Aquí, una partición en rutas óptimas corresponde a encontrar el camino de coste mínimo desde el nodo 0 hasta el nodo n en H.

La población inicial se crea realizando la mitad de la población de manera aleatoria y la otra mitad de forma constructiva. Este método constructivo consiste en separar el problema en m problemas clásicos de VRP, uno para cada producto. Estos problemas se resuelven con el algoritmo de Clarke and Wright. Después de esto se hacen intercambios aleatorios a las rutas. Después se van fusionando las distintas soluciones de los productos para hacer la ruta global.

Para aceptar una nueva solución, esta nueva debe ser mejor que la anterior y esta diferencia debe ser mayor que un límite dado.

Cada iteración empieza por escoger aleatoriamente dos soluciones P1 y P2, utilizando el método del torneo binario. Se cruzan los padres para obtener un hijo mediante el operador comentado más adelante. Este hijo se somete a un proceso de búsqueda local con una cierta probabilidad. Si este hijo mejora la población actual y esta diferencia es más grande que el límite fijado, esta nueva solución reemplaza el cromosoma que está situado a la posición “k” donde “k” pertenece a [longitud_población / 2, longitud_población]. Después se vuelve a ordenar la población en orden creciente del coste.

Debido a que no hay limitadores de ruta, el cruce consiste en un cruce clásico de TSP.

Para calcular el coste, se debe aplicar a cada cromosoma el método de splitting procedure para crear las rutas y poderlas evaluar.

NO se utiliza ninguna función de mutación. En su lugar se utiliza la búsqueda local.

La búsqueda local se aplica a soluciones completas, es decir, soluciones con los limitadores de rutas, no a cromosomas. Los mecanismos de búsqueda local son:

- **2-Opt inter ruta:** Se aplica el 2-Opt a dos vértices (i, succ(i)) y (j, succ(j)) que pertenecen a distintas rutas. Se deben evaluar dos casos: Los dos vértices se reemplazan por (i,j) y (succ(i), succ(j)) o bien por (i, succ(j)) y (j, succ(i)).
- **Reordenación**: Sacar un nodo de la ruta actual y asignarlo a otra ruta.
- **1-Interchange**: Intercambia dos nodos de dos rutas distintas.

En cada iteración de la búsqueda local, los vecinos definidos por las operaciones 1 (2-Opt), 2 (Reordenación) y 3 (1-Interchange) son numeradas en este orden. La enumeración se para cuando encontramos una solución mejor que la actual. Se ejecuta este movimiento y el orden de ejecución de las operaciones se mueve cíclicamente hacia la izquierda para la siguiente iteración. Por ejemplo en la segunda iteración los movimientos se harán en orden 2-3-1, en la tercera 3-1-2, etc. El proceso de búsqueda local global acaba cuando no hay mejoras.

2.3.2. Algoritmos basados en Hormigas

Muchas sociedades de insectos, a pesar de la simplicidad de los individuos que la componen, poseen una complicada estructura social que les permiten realizar tareas de extrema complejidad, que un individuo solo sería incapaz de realizar.

Entre las diferentes sociedades insectívoras, la ciencia de la tecnología le ha dedicado un especial interés a las colonias de hormigas, sobretodo en como optimizan los caminos hacia fuentes de comida una vez éstas han sido descubiertas.

Para investigar sobre el tema, se realizaron muchos y variados experimentos, pero quizá el más conocido sea el *Double Bridge*. Dicho experimento consiste en elaborar dos caminos desde el nido de las hormigas hasta una fuente de comida, siendo uno de los dos camnios más largo que el otro.

Inicialmente las hormigas recorrían de manera aleatoria culquiera de los dos caminos en busca de comida. Al cabo de un tiempo la gran mayoría de hormigas escogían el camino más corto. Esto se debe a que al recorrer un camino, las hormigas van soltando un rastro de feromonas para avisar al resto de las hormigas por donde han pasado, para que puedan seguirla. A seleccionar el camino más corto, las hormigas iban y venían más rápido del nido a la comida y viceversa, provocando una inundación de feromonas, cosa que hacía que las hormigas se decantaran por ese camino. En la figura siguiente podemos ver este comportamiento:

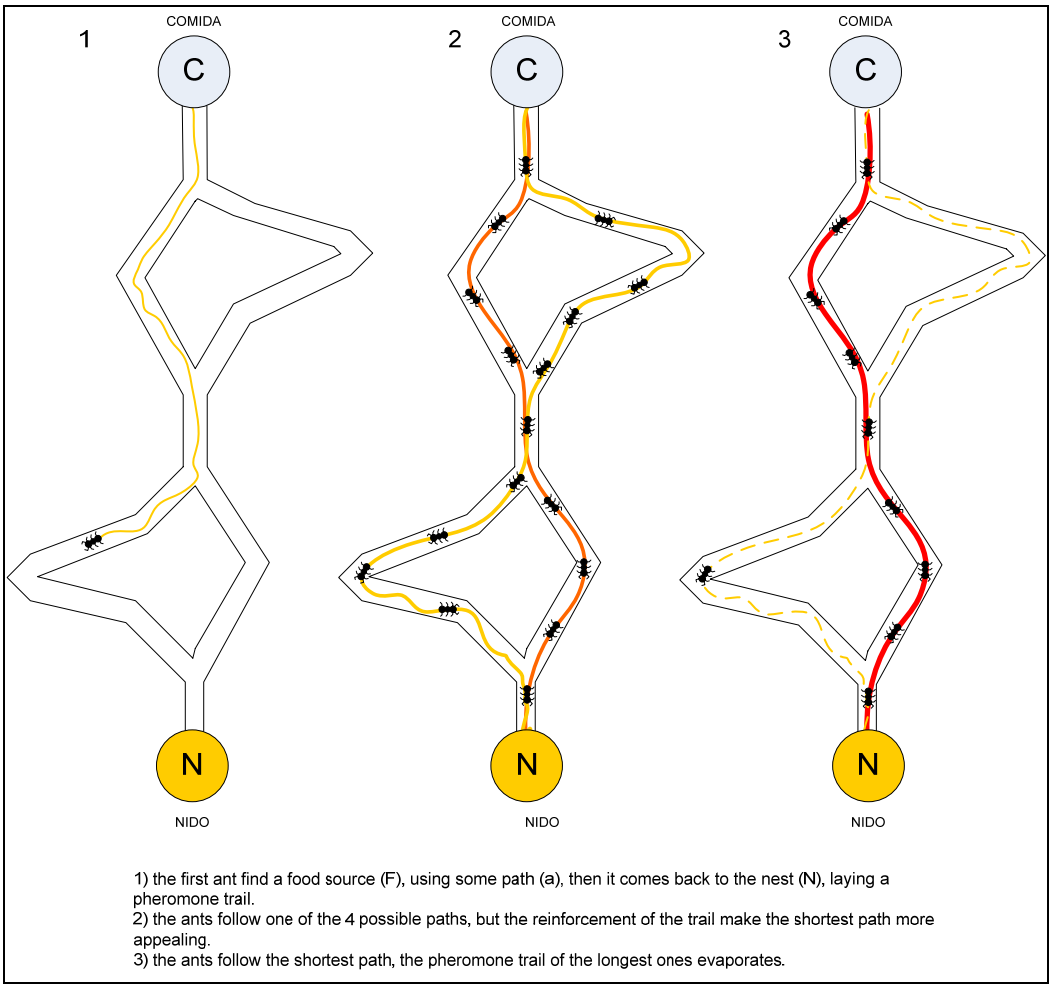


Figura 38: Ejemplo de comportamiento de hormigas

Otro factor que se tiene que tener en cuenta es la evaporación de las feromonas. Estas sustancias químicas no tienen una existencia eterna, sino que se van evaporando a medida que pasa el tiempo. Esto provoca que en los caminos más largos se evaporen antes las feromonas y se vuelvan menos atractivos para las hormigas.

Basándose en las observaciones y experimentos realizados se puede sacar un modelo computacional para recrear el comportamiento de las colonias de hormigas en problemas de optimización, sobretodo si se pueden definir estos problemas mediante grafos.

En este modelo, se crea una colonia de hormigas artificiales donde se definen parámetros como los niveles de feromonas, probabilidades de que las hormigas elijan un camino con mayor número de feromonas, etc. En [Dorigo96] y [Hur97] podemos ver más detalles sobre las colonias de hormigas. En [Hur97] podemos ver un algoritmo general para la colonia y para las hormigas. Aquí podemos verlo:

Algoritmo de Colonia de Hormigas

```
1.  Sea n el número de hormigas a utilizar.

2.  Para i=1 hasta n hacer:


    2.1.      Crear una nueva hormiga.
    2.2.      Establecer nodo de inicio para la hormiga recién creada.
    2.3.      Lanzar hormiga.
```

```
3. Mientras no se cumpla criterio de finalización:

   3.1.      Esperar a que las hormigas hayan completado su recorrido.
   3.2.      Llamar a la regla de Actualización global de feromonas.
   3.3.      Incrementar número de iteraciones.

4. Devolver mejor solución encontrada.
```

El algoritmo para cada hormiga es:

Algoritmo de Hormiga

```
1. Repetir

   1.1.      Invocar regla de transición de estado para obtener el siguiente
       nodo.
   1.2.      Actualizar valor del recorrido añadiendo la longitud del arco al
       nuevo nodo.
   1.3.      Invocar la regla de actualización local para actualizar el nivel
       de feromona para el nuevo arco.

2. Hasta que la actividad termine.

3. Llamar a la función de Comparación para comparar el resultado actual con el
   mejor resultado encontrado hasta el momento.


   3.1.      Si la solución actual es mejor, entonces actualizar el mejor
       valor con el actual.
```

En el algoritmo de las hormigas, vemos que hay una actualización local de las feromonas. Este proceso se realiza para evitar que las hormigas repitan soluciones que se están creando en la actualizadad y exploren el espacio en busca de otras soluciones.

Mediante el proceso de actualización global, iremos ajustando las feromonas y así las probabilidades de que las hormigas, en futuras ejecuciones, se vayan dirigiendo hacia el camino más corto (mejor solución).

2.3.2.1. Experiencia encontrada

En [Gamb99], los autores proponen un sistema llamado *MACS-VRPTW* para resolver el problema de enrutamiento de vehículos con ventanas de tiempo (VRPTW).

La regla utilizada para la selección del siguiente nodo utilizado en el proceso de construcción de las rutas, es uno de los que más se suele utilizar en estos problemas y es el siguiente:

- Con probabilidad $q_0$se elige el nodo con el mayor $\tau_{ij}[\eta_{ij}]^{\beta},\ j \in \mathrm{N}_i^k$ (explotación)
- Con probabilidad (1 – q_0) se elije el nodo j con probabilidad p_{ij}proporcional a $\tau_{ij}[\eta_{ij}]^{\beta},\ j \in \mathrm{N}_i^k$ (exploración). Esta probabilidad se calcula según la siguiente ecuación:
- $$p_{ij} = \begin{cases} \frac{\tau_{ij}[\eta_{ij}]^{\beta}}{\sum_{l \in N_i^k} \tau_{il}[\eta_{il}]^{\beta}}, si\ j \in \mathrm{N}_i^k \\ 0,\ sino \end{cases}$$

donde β y q_0 són parámetros: β representa la importancia del valor heurístico, mientras que q_0 ($0 \leq q_0 \leq 1$) representa la importancia de la explotación respecto la exploración: cuanto más pequeño sea q_0

mayor será la probabilidad de hacer exploración y viceversa. τ_{ij} indica el nivel de feromonas del arco (i,j), η_{ij} es la cercanía entre los nodos *i* y *j* (inverso de la distancia) y N_i^k es el conjunto de nodos posibles que pueden ser visitados por la hormiga *k* desde el nodo *i*.

El proceso de actualización global de las feromonas solo puede ser ejecutado por la mejor solución encontrada, y se realiza de la forma siguiente:

- $\tau_{ij} = (1-\rho)\tau_{ij} + \rho/J_\Psi^{gb}, \forall(i,j) \in \Psi^{gb}$, donde ρ (0 ≤ρ≤ 1) es un parámetro y J_Ψ^{gb} es la longitud de Ψ^{gb}, el camino más corto encontrado por las hormigas desde el comienzo del algoritmo.

La optimización local, se realiza siempre que una hormiga se mueve de un nodo *i* a un nodo *j*. Mediante esta fórmula, se decrementa la cantidad de feromonas del arco (i,j) para así evitar que las hormigas repitan soluciones:

- $\tau_{ij} = (1-\rho)\tau_{ij} + \rho\tau_0$, donde $\tau_0$es el valor inicial de las feromonas.

MACS-VRPWT utiliza dos colonias de hormigas. Una colonia de hormigas se utiliza para minimizar el número de vehículos utilizados (ACS-VEI) y otra colonia de hormigas para minimizar el tiempo total de las rutas (ACS-TIME), teniendo en cuenta que el número de vehículos tiene más prioridad que el tiempo total.

La solución inicial se crea mediante el algoritmo del vecino más cercano. A partir de esta solución, se aplica el siguiente algoritmo:

Algoritmo MACS-VRPTW

```
1. Ψgb = Solución inicial encontrada mediante vecino más cercano
2. Repetir
   2.1.      v = número de vehículos de Ψgb
   2.2.      Activar ACS-VEI (v – 1)
   2.3.      Activar ACS-TIME (v)
   2.4.      Mientras ACS-VEI y ACS-TIME están activas
      2.4.1. Esperar una solución mejorada Ψ de ACS-VEI o ACS-TIME
      2.4.2. Ψgb = Ψ
      2.4.3. Si número de vehículos de Ψgb< v => matar ACS-TIME y ACS-VEI
3. Hasta que se cumpla el criterio de finalización
4. Devolver la mejor solución encontrada
```

Creamos las colonias de hormigas y lanzamos la colonia de minimización de vehículoscon un vehículo menos que la mejor solución encontrada para intentar reducir el número de vehículos utilizados y lanzamos la colonia para minimizar el tiempo con el número de vehículos igual al de la mejor solución.

Al encontrar una nueva mejor solución, miramos si hemos conseguido reducir el número de vehículos utilizados. Si es así, eliminamos las colonias activas y comenzamos una nueva iteración del MACS-VRPTW. Como hemos comentado arriba, damos más importancia a minimizar el número de vehículos y esto se plasma en el algoritmo de esta manera.

También cabe recalcar que cada una de las dos colonias de hormigas utilzadas en este algoritmo, tiene su propia matriz de feromonas, no hay una matriz global a la que accedan dichas colonias.

Por último, comentar que ACS-TIME implementa un método de búsqueda local para mejorar la calidad de las soluciones encontradas, basado en el intercambio de sub-rutas.

En [Dona08] se adapta el MACS-VRPTW para trabajar con funciones de distribución de la velocidad y funciones de distribución de tiempo de viaje dependiendo de la hora del día en la que se encuentre el vehículo durante la ruta y además añaden más procedimientos de búsqueda local, bastantes de lso cuales se encuentran en [Caric08].

Los autores de [Zabala05] proponen un sistema de Colonias de Hormigas al que añaden un proceso de búsqueda local para solucionar el problema de Enrutamiento de Vehículos con Capacidad y Ventanas de Tiempo (CVRPTW).

Adicionalmente, se utiliza una heurística de inserción para completar las rutas en caso de que algún cliente quede sin ser visitado por las hormigas. Esto suele suceder cuando las restricciones de capacidad y tiempo son muy fuertes.

La función objetivo a minimizar es la distancia total de las rutas.

La solución inicial también la realizan con la heurística del Vecino más Cercano.

Cuando las hormigas acaban de hacer su ruta, puede suceder que queden vecinos sin visitar. Para solucionar este problema, se completa la solución mediante una heurística de inserción en la que primero se intentan ir añadiendo los nodos que faltan por insertar entre otros nodos de la ruta y, para la posición en la que el coste de la inserción sea menor para ese nodo, comprobar que es factible la nueva ruta.

No todas las soluciones generadas por las hormigas pasan por el proceso de búsqueda local. Sólo aquellas que se encuentren dentro de un porcentaje por encima de la mejor solución encontrada hasta el momento, por ejemplo 8% - 20% por encima de la mejor solución, se someten al proceso de explotación realizado por el operador de búsqueda local CROSS-exchange, uno de los operadores utilizados en[Caric08].

En [Bianchi07], [Qi08] vemos como, una vez más, se hacen métodos híbridos de colonias de hormigas con operadores de búsqueda local para mejorar las soluciones encontradas por las hormigas.

Además de usar operadores de búsqueda local, en [Zhang08] se utiliza un algoritmo metaheurístico llamado Scatter search (SS). Este algoritmo meta-heurístico combina soluciones de un conjunto de referencia para crear soluciones mejores. El conjunto de referencia es un conjunto de posibles soluciones válidas y normalmente se actualiza combinando estas soluciones para generar nuevas soluciones. Scatter Search mantiene algunas soluciones de calidada durante el proceso de búsqueda y asegura la diversidada para poder explorar otras regiones y salir de mínimos locales.

Otra posible hibridación, es utilizar listas tabú junto con hormigas, como se puede ver en [Bouh08] y [Qiang08]. En [Zhen08], además de usar una lista tabú, aplican primero la heurística de sweep para separar en grupos los clientes (clusterización) y después aplicar la colonia de hormigas con lista tabú para encontrar nuevas soluciones.

2.3.3. Búsqueda Tabú

La búsqueda tabú o TS (Tabu Search) es un algoritmo metaheurístico que puede utilizarse para resolver problemas de optimización combinatoria.

Este método explora el espacio de soluciones moviéndose en cada iteración de una solución *s* a la mejor solución de un subconjunto de vecinos *N(s)*. El TS funciona según la suposición de que no hay ninguna razón para aceptar una solución a no ser que evite un camino que ya se ha investigado. Esto nos asegura que se investigarán nuevas regiones del espacio con el objetivo de evitar quedar atrapados en mínimos locales.

Para crear las nuevas soluciones del vecindario se suelen utilizar algoritmos de búsqueda local u operadores parecidos a los utilizados en algoritmos genéticos para la mutación.

Para evitar las regiones que ya se han visitado con anterioridad, se utiliza una estructura de datos que nos determina qué soluciones son admisibles o no. A esta estructura se la conoce como *lista tabú.* Esta lista contiene, temporalmente, soluciones que ya se han visitado antes o ciertas características que no queremos que se repitan durante un cierto tiempo.

La búsqueda, cuando encuentra una nueva solución, comprueba si esta está dentro de la lista tabú o no. Si está, entonces se considera que esta solución es una solución tabú y no se tiene en consideración. Si no se encuentra en la lista, se coge ésta como mejor solución y se añade a la lista tabú.

Esta definición de la lista puede ser más efectiva para determinados dominios, pero presenta un nuevo problema. Cuando un atributo o característica es marcada como tabú implica que más de una solución se vuelven tabú. Alguna de estas soluciones podrían ser mejores que la mejor solución encontrada hasta ahora, pero se puede perder debido a que contiene características que son tabú. Este problema se puede intentar evitar mediante el uso de criterios de aspiración. Estos criterios suavizan la prohibición de que no se puedan seleccionar elementos con características tabú, dependiendo de si cumplen o no el criterio de aspiración elegido. Un criterio muy extendido es permitir soluciones mejores que la mejor solución encontrada hasta el momento.

Además de todo esto, se tienen que tener en cuenta los conceptos de intensificación y de diversificación. El primero consiste en intensificar la búsqueda de soluciones con determinadas características. Por otro lado, la diversificación es interesante si el algoritmo se queda en algún mínimo local o no consigue mejorar el resultado actual.

La estructura del algoritmo es la siguiente:

```
1. Obtener una solución inicial x.

2. Mientras no se verifique el criterio de finalización.

   2.1.      Seleccionar una solución x1 próxima a x.

   2.2.      Si (x1 no es tabú) o (x1 es tabú y satisface el criterio de
       aspiración).

      2.2.1.  x = x1;

      2.2.2. Actualizar lista tabú con la nueva solución.
```

```
      2.2.3. Actualizar criterio de aspiración.
   2.3.      Sino volver al punto 2.1.
   2.4.      Ejecutar estrategia de diversificación o intensificación.
3. Retornar la mejor solución hallada.
```

Algortimo de búsqueda tabú

2.3.3.1. Experiencia encontrada

Los autores de [Barba99] presentan un algoritmo llamado DETABA que utiliza una búsqueda tabú para resolver el CVRP, pero donde proponen no utilizar el procedimiento de diversificación y explican un nuevo esquema de intensificación.

El objetivo es tratar de minimizar la distancia/tiempo total de las rutas de los vehículos.

En cada iteración del algoritmo de búsqueda, conocido como SENE, utiliza dos funciones, TANE y TANEC, para encontrar posibles soluciones en el vecindario y una vez encontradas, escoge la mejor (la que tenga un coste menor).

Para la creación de las soluciones, los dos procedimientos utilizan combinaciones de movimientos de búsqueda local, tales como el 2-Opt, la reasignación de nodos entre diferentes rutas y el intercambio de nodos entre rutas.

La diferencia entre TANE y TANEC es que la segunda tiene en cuenta la relativa proximidad entre los nodos mediante la creación de *clusters* mientras que TANE no tiene en cuenta dicha característica.

```
1. Crear una solución inicial.
2. Repetir N veces
   2.1.      Escoger dos rutas al azar (R1 y R2).
   2.2.      Escoger una subruta de R1, SR1.
   2.3.      Escoger una subruta de R2, SR2.
   2.4.      Comprobar que el intercambio de nodos entre rutas no va a
             provocar una solución no válida. En caso de que sea así, volver al
             punto 2.2.
   2.5.      Insertar SR2 en R1 usando la función de inserción.
   2.6.      Insertar SR1 en R2 usando la función de inserción.
   2.7.      Aplicar 2-Opt a R1 y R2 independientemente k veces.
   2.8.      Calcular el valor de la función de coste.
3. Retornar la solución que minimice la función de coste.
```

Algoritmo TANE

A veces los diferentes servicios a realizar por los recursos móviles pueden tener una topología en la que se vea que los nodos se podrían agrupar y de esta manera poder aprovechar esta característica para decidir a donde enviar cada vehículo, por ejemplo, un vehículo a cada agrupación.

Normalmente este factor no se tiene en cuenta, pero aquí los autores sí que lo contemplan mediante la función TANEC. La idea principal es, primero, encontrar el centroide de cada una de las rutas R1 y R2. Una vez tenemos estos centroides, calculamos la relación entre la distancia de cada nodo de R1 con respecto al centroide de R1 (d1) y de R2 (d2) y repetimos lo mismo con los nodos de R2. Una vez tenemos esta ratio, ordenamos los nodos de R1 y R2 según este valor.

Por último se cogen n elementos de R1 y m elementos de R2 de la lista de nodos ordenados y se intercambian de ruta, se aplica 2-Opt a cada ruta y se devuelve la mejor solución encontrada.

La restricción para saber si una solución es tabú o no es la siguiente:

1. Si en la iteración *k* los vértices v1 perteneciente a la ruta R1 y el vértice v2 de R2 son intercambiados, está prohibido volver a poner v1 en R1 y v2 en R2 durante *t* iteraciones, donde *t* es un número escogido al azar entre (t_min, t_max).

Si en una iteración encontramos que todas las soluciones encontradas son tabú y además no cumplen el criterio de aspiración, entonces se retorna el último mejor resultado encontrado que no fuera tabú.

Para encontrar la solución inicial, se hace primero un conjunto de $\sqrt{n}$ posibles soluciones, donde n es el número de nodos. Para construirlas, se hacen dos subconjuntos de $\sqrt{n}/2$ elementos, donde cada conjunto subconjunto se construye con uno de los métodos siguientes:

- Generación aleatoria de rutas.
- Mediante un método constructivo. A partir de un nodo elegido al azar, se busca el vértice que minimice el coste de insertarlo en la ruta. A partir de aquí, se van añadiendo nodos en la posición donde se minimice el coste de añadirlo. Cuando no se puedan añadir más nodos a una ruta, se crea una ruta nueva con un nodo al azar.

El procedimiento de inserción utilizado tanto en TANE como en TANEC es el mismo que para la creación de la solución inicial, pero en vez de para todos los nodos, solo para un subconjunto y se añaden todos a la misma ruta. Si no se pueden añadir todos, entonces no es una ruta válida.

Como se ha comentado más arriba, en este algoritmo no se realiza un procedimiento de diversificación, pero sí de intensificación. Este proceso sirve para localizar zonas de búsqueda donde podrían haber mínimos globales que quedan enmascarados por mínimos locales. Durante la fase de mejora, las soluciones que indican una zona donde podría haber un mínimo local que el encontrado actualmente, se identifican y se incluyen en un conjunto llamado INS.

El uso de este conjunto INS es el siguiente:

- Supongamos que en la iteración k tenemos la mejor solución, S(k), con un coste C(k) y en la iteración k+1 tenemos la solución S(k+1) y coste C(k+1). Se calcula D(k) = C(k+1) – C(k). Si D(k-1) ≤ 0 y D(k) > 0, entonces S(k) se identifica como un indicador de una región donde se puede profundizar más la explotación y se inserta en el conjunto INS. Las soluciones que se encuentran en el conjunto INS se guardan de forma ordenada en función del coste.

Finalmente el algoritmo general queda:

1. Inicialización: Generar $\sqrt{n}$elementos usando los métodos comentados más arriba.
2. Primera mejora: Aplicar SENE a cada solución inicial.
3. Mejora de la solución: Comenzando con la mejor solución encontrada en el paso anterior, aplicar otra vez SENE con otros parámetros.
4. Intensificación: Aplicar SENE a algunas soluciones determinadas en el paso 3 y que estén dentro del conjunto INS.

Algoritmo general DETABA

Los autores de [Lau03] nos proponen un algoritmo para resolver el problema VRPTW con un número limitado de recursos, m camiones, utilizando la búsqueda tabú.

Estos autores, en lugar de utilizar una función de coste compuesta, es decir, una función en la que el coste es la suma de varios factores, proponen una función de coste jerárquica en orden decreciente de la prioridad de los objetivos siguientes:

- Maximizar el número de clientes servidos.
- Minimizar el número de clientes servidos tarde.
- Minimizar el tiempo total.
- Minimizar el número de vehículos.
- Minimizar la distancia total.

La manera de comparar soluciones con funciones de coste jerárquicas es la siguiente:

Se compara el coste en la categoría más importante de las dos soluciones. Si una de las soluciones tiene menor coste que la otra, ya está. Se escoge ésta y se devuelve. Si las dos tienen el mismo valor, entonces se calcula el valor de la siguiente categoría y se compara. Se va repitiendo hasta que se elige una. En caso de que las soluciones sean iguales, se escoge una de las dos al azar.

El trabajar con estas funciones, nos permite no tener que calcular siempre toda la función de coste, sino que sólo se calcula lo necesario.

El algoritmo propuesto está basado en un proceso iterativo de búsqueda local más lista tabú.

El algoritmo empieza con un número determinado de camiones menor que el límite m y se va aplicando búsqueda local para maximizar el número de clientes que pueden ser insertados en las rutas de los vehículos actuales. En una lista auxiliar tenemos todos los clientes que todavía no hemos insertado en la iteración actual.

Las posibles operaciones de búsqueda local son:

- Mover un nodo de la lista auxiliar a la solución que estamos creando.
- Mover un nodo de la solución actual a la lista auxiliar.
- Intercambiar un nodo de la lista con otro de la solución.

Para pasar de una iteración a otra y así aumentar el número de vehículos que podemos utilizar, esperamos un cierto número de operaciones de búsqueda local sin que se encuentre ninguna nueva solución mejor que la actual.

El algoritmo acaba cuando la lista de clientes (nodos) por visitar está vacía o cuando se llega al máximo número de recursos permitidos, *m*.

```
1  numVeh = valor inicial de vehículos.
2  Repetir
   2.1   contador = 0
   2.2   Mientras contador ≤ máximo
      2.2.1    Hacer búsqueda tabú con numVeh vehículos.
      2.2.2    Si se encuentra una mejor solución => contador = 0;
      2.2.3    sino contador ++;
   2.3   Aumentar el número de vehículos
3  hasta que la lista auxiliar está vacía o numVeh = m.
```

Algoritmo A

Por otro lado, la lista tabú contiene los clientes que han sufrido un movimiento en las últimas k iteraciones, donde k es la longitud de la lista tabú. Se considera que un movimiento (creación de una solución) es tabú si todos sus clientes están en la lista tabú y además la nueva solución no es mejor que la actual en base a la función de coste (criterio de aspiración).

En [Fal08] también proponen un algoritmo de búsqueda tabú, además de un memético.

Para la creación de las soluciones del vecindario, aplican a la solución actual operadores de búsqueda local como el 2-Opt, la inserción de nodos de una ruta en otra (Relocate) y el intercambio de un nodo entre dos rutas (1-interchange)

Para la solución inicial utilizan el algoritmo de Clarke and Wright o de los ahorros.

Por último, la lista tabú consiste en una matriz donde cada elemento tiene la iteración en la que el vértice (i,j) se ha eliminado de la solución. Hasta que no haya pasado un cierto número de iteraciones, no puede reinsertarse ese vértice en la solución.

Además, los autores no aplican ningún criterio de aspiración, ya que dicen que no aporta un aumento significativo de la optimización.

Otro documento interesante [Bray02]. Aquí se realiza un estudio de múltiples técnicas utilizando Búsqueda Tabú, comparando los resultados experimentales. Cabe destacar que todos los algoritmos presentados en este artículo utilizan, como operadores de vecindario, los operadores comentados en el apartado de Heurísticas de mejora o de búsqueda local, en particular 2-opt, Relocate, Or-Opt, CROSS, y alguno más.

2.3.4. Simulated Annealing

La técnica del recocido simulado está basada en la técnica de la metalurgia que consiste en el tratamiento físico de los metales (recocido) donde un metal es llevado a altas temperaturas alcanzando altos niveles de energía llegando al punto de fusión para después irlo enfriando gradualmente, en un proceso por fases donde el sólido puede alcanzar el punto de equilibrio térmico n cada fase, hasta

obtener un nivel energético mínimo definido previamente [Bara06]. Este proceso se usa para ablandar el acero, regenerar la estructura de aceros sobrecalentados, etc.

El recocido simulado o *Simulated Annealing (SA)* es un método probabilístico que construye soluciones aleatoriamente y las somete a unas determinadas reglas de probabilidad, en función de un parámetro llamado *temperatura*, para tratar de evitar o salir de mínimos locales. Esto se consigue aceptando con cierta probabilidad, soluciones que son peores que la actual, para poder escalar y saltar el mínimo local.

El objetivo es encontrar soluciones próximas al mínimo global.

El SA se basa en "calentar" a alta temperatura el sistema que se quiere optimizar y después ir disminuyendo la temperatura hasta que no hayan cambios. La variación de temperatura se hace de forma escalonada.

Este algoritmo permite explorar ampliamente las regiones de búsqueda en el comienzo del proceso iterativo y mientras se va enfriando el sistema se fomenta la explotación de ciertas regiones.

A continuación se muestra el algoritmo, donde *C* es la constante de Boltzman y *r* es el escalón de temperatura, que puede ser dinámico o estático:

Simulated Annealing

1 Dada una solución inicial *x* y una temperatura inicial *t*

2 Mientras no se verifique la condición de finalización

2.1 Seleccionar una solución perturbada *x1* próxima a *x*

2.2 Clacular *dc = coste (x1) - coste(x)*

2.3 Si $dc \leq 0$ =>*x = x1*

2.4 Sino =>*x = x1* con probabilidad $e^{-(\frac{dc}{tC})}$

2.5 *t = r * t*

3 Devolver la mejor solución encontrada

Para realizar la perturbación de la solución actual (paso 2.1 del algoritmo), se pueden aplicar técnicas como las de mutación de los algoritmos genéticos u operadores de búsqueda local heurísticos.

2.3.4.1. Experiencia encontrada

En este apartado vamos a explicar diversas soluciones encontradas en la literatura mediante la utilización del Recocido Simulado.

Una primera evolución del SA la podemos ver en [Czar02], donde se plantea un algoritmo paralelo para resolver el VRPTW. Este algoritmo consiste en tener cierto número de procesadores p, y enviar una solución inicial a cada procesador donde se ejecuta el SA y cada cierto número de pasos del proceo, se envían todas las soluciones encontradas hasta el momento para comprobar si se ha encontrado una nueva mejor solución. El algoritmo paralelo acaba cuando se llega a un equilibrio, o lo

que es lo mismo, cuando se ejecutan un cierto número de iteraciones sin que cambie la mejor solución encontrada.

Otra posibilidad la tenemos en [Tavak07], donde se presenta un algoritmo para resolver el CVRP. Para modificar las soluciones, utilizan los operadores heurísticos 1-Opt y 2-Opt. A partir de ahí, aplican el algoritmo clásico de SA. Lo interesante de este artículo es que modifican el CVRP para asumir que se tiene una flota heterogenia de vehículos y adaptan el SA para este tipo de problemas. La función objetivo consiste en minimizar el número de vehículos y maximizar el uso de la capacidad de los camiones.

Podemos ver que en [Lin09], [Bran06], [Tan01] y otros muchos, también es muy importante el uso de operadores de búsqueda local para la creación de soluciones vecinas a la actual. En este caso tenemos múltiples operadores entre los que se escoge uno de forma aleatoria para aplicar a la solución actual y encontrar una nueva solución. Además de utilizar operadores de búsqueda local, en [Lin09b] se utiliza la unión del SA con la búsqueda Tabú.

2.3.5. Razonamiento Basado en Casos

El Razonamiento Basado en Casos (CBR, por sus siglas en inglés) es una técnica de Inteligencia Artificial que se basa en la utilización de experiencias previas para resolver nuevos problemas mediante la hipótesis *problemas similares tienen soluciones similares*.

Dado un problema a resolver, el CBR busca, en una base de datos llamada Base de Casos, problemas similares que anteriormente se hayan resuelto con éxito, llamados casos, y adapta las soluciones para dar una solución al problema actual. Este mecanismo de razonamiento es utilizado por los humanos en múltiples problemas y permite que sea un sistema de fácil comprensión.

El CBR involucra toda una metodología con un ciclo de actividades que además de solucionar nuevos problemas nos permita aprender de las buenas soluciones obtenidas por los nuevos problemas:

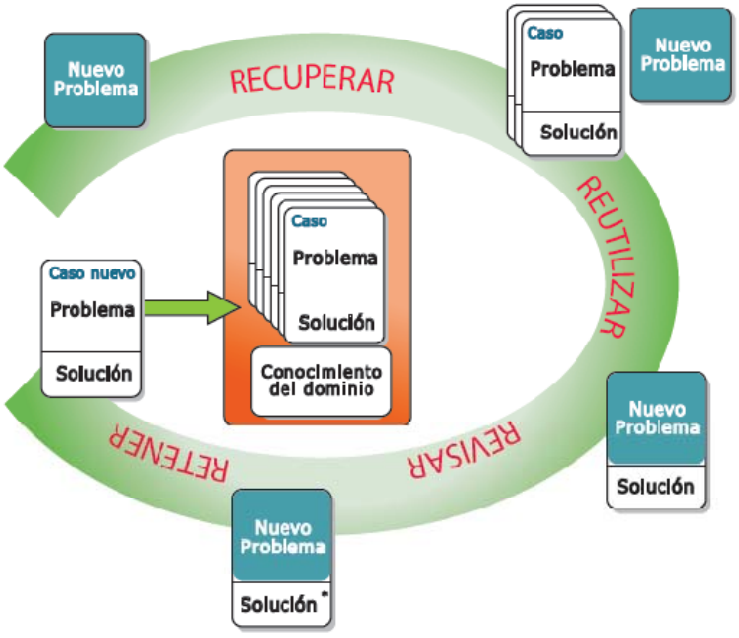


Figura 39: CBR

- **Recuperar**: Dado un problema, se recuperan los casos más similares de la Base de Casos. Un caso es un problema anterior con su solución.
- **Reutilizar**: Extraer la solución del caso seleccionado para utilizarla. Esto puede implicar adaptar la solución a la nueva situación.
- **Revisar**: Se debe analizar si la nueva solución es aceptable y si es necesario revisarla.
- **Retener**: Después de haber aplicado las solución con éxito, se debe almacenar la experiencia como un nuevo caso en la Bases de Casos.

Por lo tanto, para utilizar CBR es conveniente disponer de casos de éxito para diferentes problemas y conocer los diferentes factores que influyen en la solución. Luego será necesario tener un conocimiento sobre el dominio que nos permita evaluar y mejorar las soluciones propuestas.

La idea de reutilizar componentes de planificaciones que han funcionado bien en el pasado nos lleva a explorar la tecnología del Razonamiento Basado en Casos aplicada a la planificación. [Cunn97] El principal problema de aplicar CBR a la planificación es:

- la complejidad de los requerimientos
- la cantidad de requerimientos

Aunque la información encontrada sobre CBR se refiere básicamente al problema TSP básico, y no al VRP, es interesante comentar esta posibilidad para abrir posibles líneas de investigación.

Una de las posibilidades que se abren es la de utilizar métodos CBR de reutilización, como los expuestos en los algoritmos que presentamos a continuación, con la finalidad de insertar nodos en las rutas anteriormente planificadas minimizando el impacto (en coste) del cambio.

Como veremos, es posible solucionar subóptimamente el TSP, reutilizando y adaptando soluciones anteriores completas mediante el método T-CBR.

El problema se vuelve complica a medida que se añaden restricciones (tiempo, capacidad,..), por ello las investigaciones que existen hasta este momento se refieren al problema.

El hecho de que, en problemas de optimización, las soluciones puedan ser reutilizables sugiere que el CBR pueda ser aplicado.

La solución CBR es excelente desde el punto de vista de la velocidad pero la calidad de las soluciones está fuertemente subordinada a la calidad de los casos de la base de casos.

La idea tras la utilización de CBR en el TSP pasa por desplazar la complejidad desde el domino del tiempo al dominio del espacio, esto es, almacenando una cantidad importante de casos, se espera una mejora significativa en la solución del problema. El principal inconveniente se debe a la naturaleza combinatoria del problema, la cual obliga a disponer de una gran base de casos.

Solución mediante generación de esqueleto

Una posibilidad de aplicación del CBR pasa por crear una solución esqueleto a partir de una única ruta anterior recuperada de la Base de Casos. Recuperando rutas similares que han funcionado bien en el pasado podemos obtener una ruta esqueleto y más tarde adaptarla a nuestras necesidades.

Un mundo de N ciudades y un objetivo potencial de ruta de 20 ciudades:

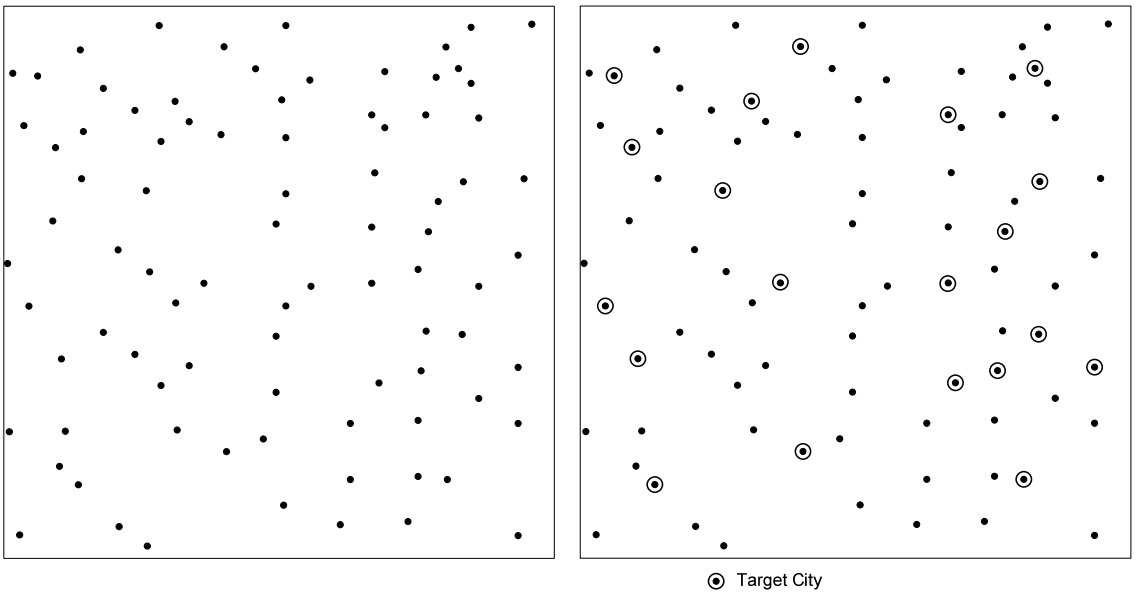


Figura 40: Mapa ciudades

Para buscar en la Base de Casos podemos usar como medida de similitud el número de ciudades coincidentes entre nuestro objetivo y los casos almacenados, aunque existen muchas otras técnicas para decidir qué caso es más similar a nuestro problema. El mecanismo de recuperación se encarga de encontrar casos similares al problema en la Base de Casos.

En la Base de Casos encontramos una ruta que pasa por 10 de las ciudades objetivo. Al ser la ruta con mayor similaridad la recuperamos como ruta esqueleto. Por lo tanto, el primer paso será producir una solución esqueleto a partir de las ciudades coincidentes.

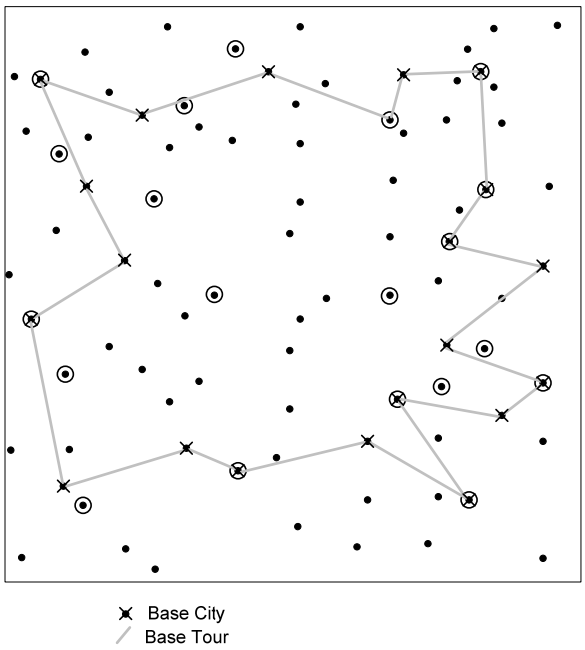


Figura 41: Solución sobre mapa

Como el problema inicial es ligeramente diferente al caso recuperado será necesario adaptar la solución recuperada, y entramos por lo tanto en la actividad de reutilizar. Por lo tanto, tendremos dos características importantes que afectaran a la viabilidad de la aplicación de esta técnica a problemas

TSP: la estructura y complejidad de las soluciones y el número de casos que hemos utilizado para componer la nueva solución.

Siguiendo con el ejemplo es necesario 'reparar' la solución para añadir las ciudades restantes a la ruta generada y obtener la ruta deseada:

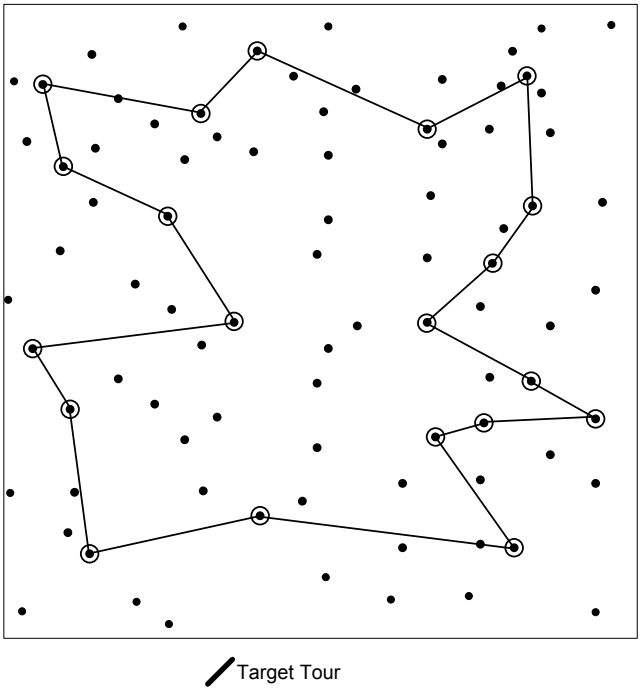


Figura 42: Solución reparada

El algoritmo aplicado para la **recuperación** es el siguiente:

```
function Retrieve-Skeleton(target-jobs, case-base)
{
  base-cases <- Base-Filter(target-jobs, case-base)
  best-case <- First(base-cases)
  best-score<- Skeletal-Quality(best-case)
  For each base-case in Rest(base-cases)
  {
    base-score<- Skeletal-Quality(target-jobs, base-case)
    if base-score> best-score
    {
      best-score<- base-score
      best-case<- base-case
    }
  }
}
```

El pseudocódigo anterior describe cómo se explora la Base de Casos buscando el mejor caso esqueleto. En el primer paso `Base-Filter` Filtra los casos a los que sobreponen alguna ciudad objetivo.

La calidad de la solución esqueleto propuesta debe ser medida, esto es lo que hace la función `Skeletal-Quality`.

El algoritmo aplicado para la **reutilización** es el siguiente:

```
function Add-cities(tour, rem-cities)
{
  if rem-cities
  {
    best-city <-Select-best-city(tour, rem-cities)
    rem-cities <-Remove-city(best-city, rem-cities)
    tour <-Insert-city(tour, best-city)
    Add-cities(tour, rem-cities)
  }
}
```

Donde `Select-best-city` selecciona las ciudades-objetivo restantes, que no entran en la ruta recuperada, y las inserta en el lugar de mínimo coste.

Este algoritmo puede ser utilizado con otras finalidades. Para nuestro caso, utilizado adecuadamente, podría ser una solución al problema de la replanificación de rutas. Dada una ruta, este algoritmo es capaz de insertar ciudades inicialmente no contempladas en la ruta, haciendo frente así a posibles cambios *a posteriori*.

La **revisión** se puede hacer de forma supervisada. Alguien debe decidir si la ruta es válida o no y así poder **retenerla** en la Base de Casos.

Solución mediante construcción de bloques

Otra posibilidad de aplicación del CBR pasa por recuperar algunas rutas anteriores con la finalidad de construir una solución única. Recuperando subsecciones de rutas similares que han funcionado bien en el pasado podemos obtener una subsecciones de la ruta objetivo.

La idea es segmentar los casos en secuencias de bloques que puedan ser reutilizables. Estos trozos, de 3 o más ciudades como mínimo, se tienen en cuenta para componer la solución final. **Recuperar**:

```
function Retrieve-Building-Blocks(target-jobs, case-base)
{
  bits <-{}
  base-cases <-Base-Filter(target-jobs, case-base)
  For each base-case in base-cases
  {
    bits <-bits + Extract-Bits(target-jobs, base-case)
  }
  Return (bits)
}
```

La función `Extract-Bits` extrae trozos que contengan 3 o más ciudades coincidentes. De esta forma se puede construir una solución completa:

```
function Extract-Bits(target-jobs, base-case)
{
  bits <-{}
  unwanted-jobs <-Set-Difference(base-case, target-jobs)
  bit-ends <-Positions(unwanted-jobs, base-case)
  bit-start <-First(bit-ends)
  For each bit-end in Rest(bit-ends)
  {
    bits <-bits + Extract-Bit(base-case, bit-start, bit-end)
    bit-start <-bit-end
  }
  Return(Bits)
}
```

Los trozos recuperados pueden tener continuidad o no, en todo caso será necesario concatenarlos de alguna forma. Este es el proceso de **reutilización**:

```
schedule <- starting job
function Add-bits(schedule, bits-pool)
{
  if bits-pool
  {
    best-bit <- Select-best-bit(schedule-end, bits-pool)
    if Close-to-schedule(best-bit)
    {
      schedule <-Concatenate(schedule, best-bit)
      bits-pool <-Remove-unusable-bits(schedule, bits-pool)
      Add-bits(schedule, bits-pool)
    }
  }
}
```

Donde la función `Select-best-bit` en primera instancia busca el trozo más largo que empieza en el último punto de la ruta incompleta actual, si existe. Si no existe, se selecciona el trozo más cercano. A cada paso, los trozos utilizados se eliminan del *bits-pool*.

*La **revisión** se puede hacer de forma supervisada. Alguien debe decidir si la ruta es válida o no y así poder **retenerla** en la Base de Casos.*

3. Experimentación

En un estudio completo y en profundidad como el que ha llevado a cabo la UDT-IA para este proyecto, se debe trabajar también desde un enfoque práctico, realista y de desarrollo del estado del arte de las soluciones existentes.

Es por esta razón que en el capítulo que nos compete, se muestran y estudian tanto las soluciones existentes más remarcables del mercado, como las soluciones fruto de la investigación y el posterior desarrollo, trabajadas desde la UDT-IA.

3.1. Test

Con el objetivo de conocer la situación actual de las soluciones para el problema tratado en este estudio, así como su enfoque dada la evolución en este campo los últimos años, la UDT-IA ha buscado y probado tanto software privativo (demostraciones) como libre.

En este aspecto, las principales soluciones libres quedan aún bastante limitadas, respecto a lo que un problema de estas características precisa. Por otro lado, se han encontrado estándares de *facto* en lo que a la resolución de problemas VRP y TSP se refiere, cosa que ha ayudado a la comparación de distintos software, contrastando su calidad.

A continuación podemos ver distintas metodologías y las mejores soluciones encontradas, algunas de ellas extraídas del ámbito académico.

3.1.1. Búsqueda Tabú

3.1.1.1. OpenTS

OpenTS [OpenTS09] es un *framework* Java de búsqueda tabú que ayuda a implementar la popular metaheurística búsqueda tabú, en un diseño orientado a objeto bien definido. Ventajas claves de este *framework*:

- Funciona para cualquier tipo de problema. Puede resolver VRPs, problemas de asignación, o cualquier otro problema de embalajes. *OpenTS* no hace suposiciones sobre el tipo de problema que deberías resolver.
- Descarga de repetitivas funciones de búsqueda tabú. Se puede definir un problema y la búsqueda local ideal, es entonces cuando *OpenTS* se encarga del trabajo pesado.
- Organiza lógicamente la búsqueda tabú. Se puede entender y explicar mejor la búsqueda tabú cuando se sigue el diseño bien estructurado del framework *OpenTS*.

- Explota la tecnología de computación moderna. Se puede aprovechar la programación orientada a objeto y el lenguaje multiplataforma Java, incluyendo la habilidad de usar máquinas multiprocesador y aplicaciones distribuidas a través de Internet.
- Integra la programación por metas. Se pueden evaluar soluciones en más de una dimensión y comparar soluciones, primero comparando el valor más importante, luego el segundo, el tercero, etc.

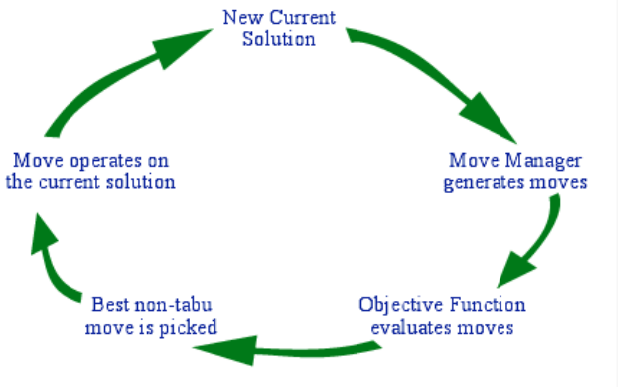


Figura 43: Una Iteración en OpenTS

3.1.2. Algoritmos Genéticos

3.1.2.1. JOpt - Advanced Vehicle Routing Components

JOpt [JOpt09] es un conjunto de librerías pensadas y desarrolladas para la resolución de todo tipo de problemas en cuanto al transporte por carretera – sus rutas – se refiere.

JOpt es desarrollado y distribuido por la empresa DNA-Evolutions, la cual nos ofrece su software de diversas formas, según las necesidades técnicas del equipo que vaya a usarlo. Así podemos optar entre JOpt.NET (plataforma .NET), JOpt.SDK (Java/J2SE), JOpt.ASP (SOAP *interface* para ASP.NET) y JOpt.J2EE (SOAP *interface* para Java).

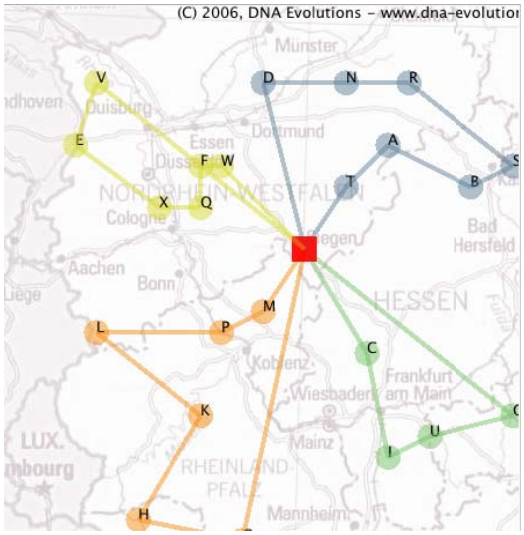


Figura 44 - JOpt Route Planning Demonstrator

Desde la web podemos descargar una demo (JOpt.SDK), que nos permite probar su librería, con restricción para diez ciudades y dos camiones. Si bien la impresión que da es de eficiencia y velocidad, al tratarse de una demo los resultados son en modo texto, no visual.

La conclusión que podemos extraer de esta potente herramienta es que, si estamos dispuestos a pagar entre 500 y 20.000 euros (según necesidad), tendremos solucionado el "motor" del sistema para las rutas de una empresa de transporte. El problema aparente es que no existe posibilidad de modificación/adaptación a la empresa en concreto, así como la replanificación de rutas.

3.1.2.2. TSPGA

El *Traveling Salesman Problem (TSP) Genetic Algorithm (GA)* [TSPGA09] es una implementación en Matlab, en forma de demo, que nos ofrece ver gráficamente el resultado de un TSP, utilizando los algoritmos genéticos como motor de solución.

Como se puede ver en la imagen siguiente (parte inferior izquierda), la solución consiste en pasar por todos los puntos marcados como objetivo, sin repetirlos, creando una ruta circular.

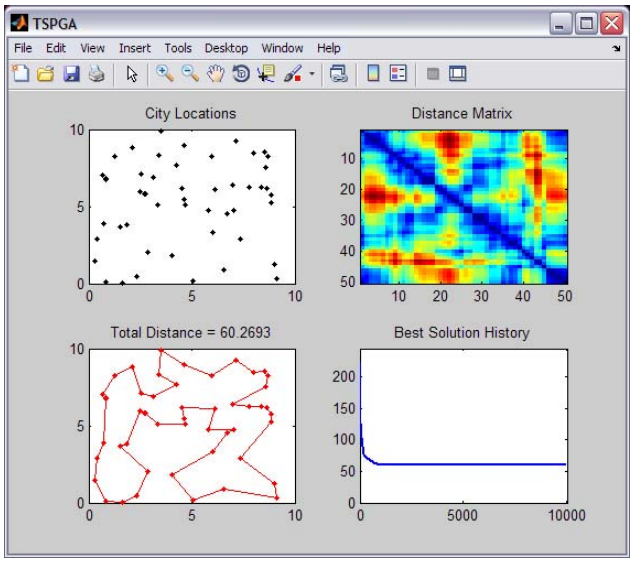


Figura 45 - Resultado ofrecido por la aplicación

Si bien es un ejemplo muy ilustrativo, no nos sirve para nada más, dada su poca flexibilidad en lo que a la personalización de los datos se refiere.

3.1.3. Hormigas

3.1.3.1. AntSim

AntSim (v1.1) [AntSim09] es una aplicación desarrollada en VisualBasic para la visualización del comportamiento de una colonia de hormigas.

Nos permite dibujar el entorno y observar como las hormigas avanzan hasta la fuente de comida volviendo al nido intentando encontrar el camino mas corto.

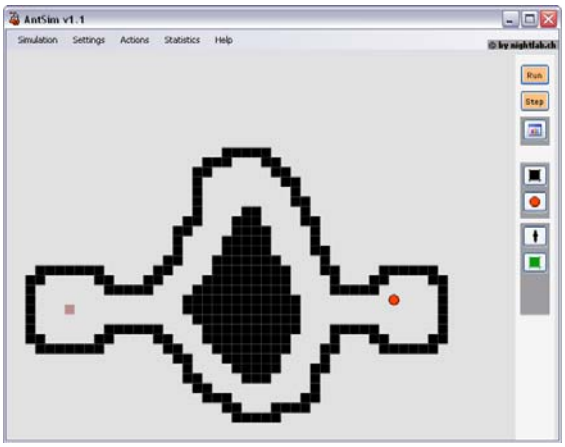


Figura 46: AntSim (1)

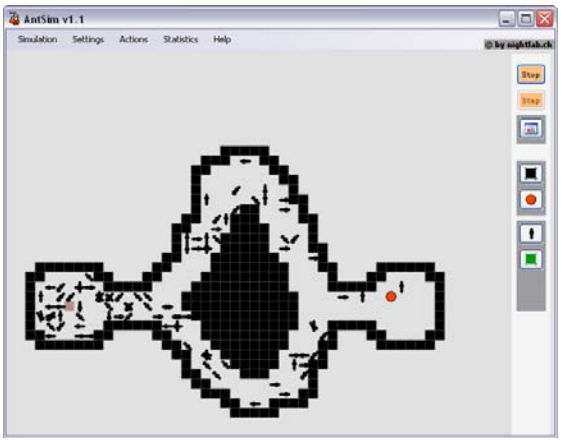


Figura 47: AntSim (2)

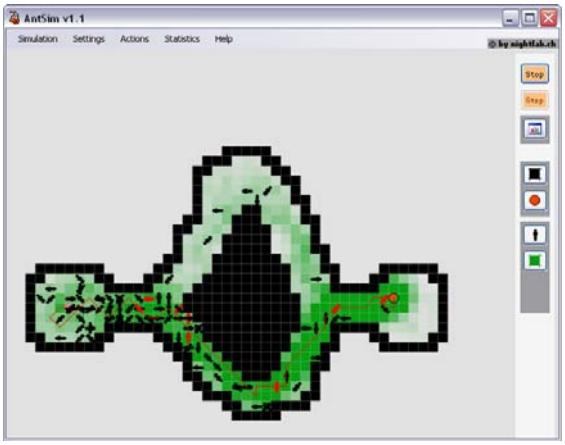


Figura 48: AntSim (3)

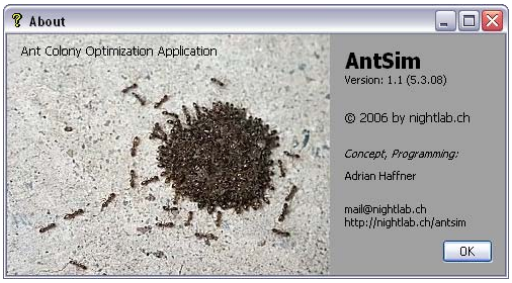


Figura 49: AntSim (4)

3.1.3.2. Ant Colony Optimization

ACODemo (v1.2) [ACODemo09] es una aplicación desarrollada en Java para demostrar como actúa una colonia de hormigas para solucionar el problema del viajante de comercio (TSP).

Las ciudades se muestran como círculos rojos, la feromona en las conexiones entre ellos (grafico completo) por líneas de color gris. El gris mas oscuro indica mas cantidad de feromona. Durante la optimización se va mostrando la ruta encontrada de color rojo.

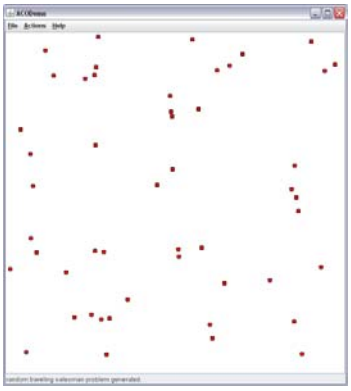


Figura 50: ACO (1)

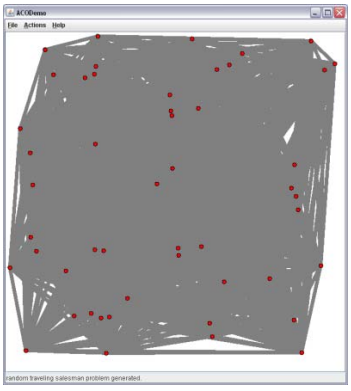


Figura 51: ACO (2)

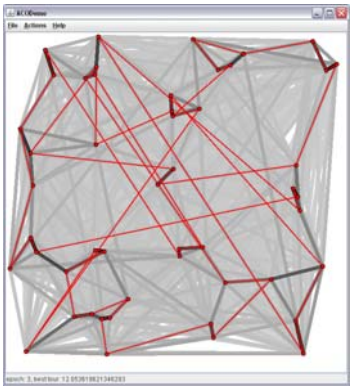


Figura 52: ACO (3)

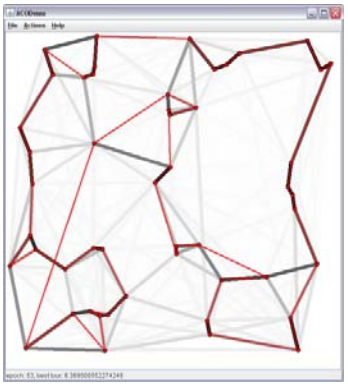


Figura 53: ACO (4)

Figura 54: ACO (5)

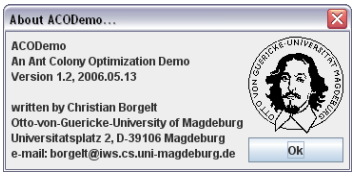


Figura 55: ACO (6)

3.1.4. Heurísticos

3.1.4.1. Concorde TSP Solver

Concorde TSP Solver [Concorde09] se trata de una suite de algoritmos heurísticos para resolver el problema del viajante de comercio (TSP). Esta programado en lenguaje ANSI C y se encuentra disponible para la investigación académica.

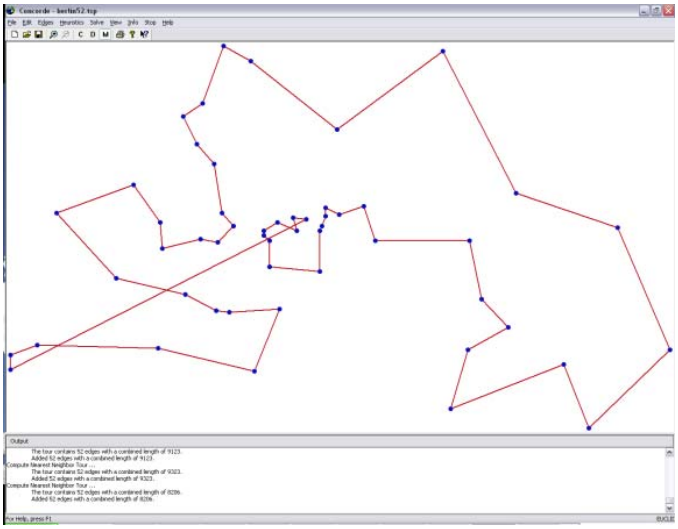


Figura 56: Concorde (1)

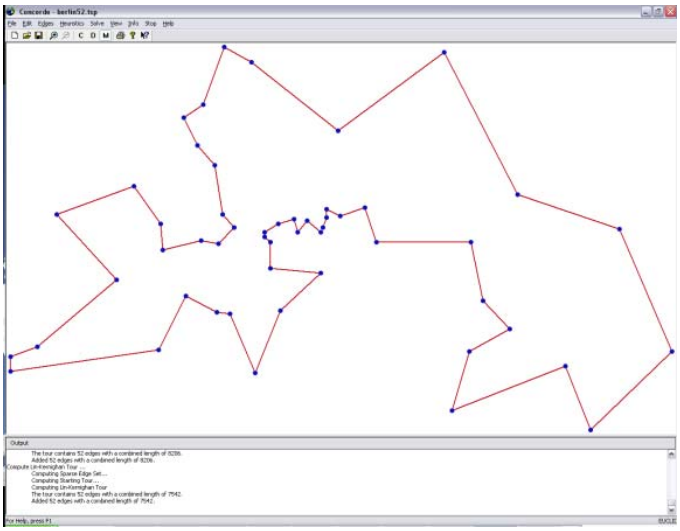


Figura 57: Concorde (2)

Esta aplicación en concreto nos ha servido para comparar los resultados de nuestros desarrollos para problemas del tipo TSP.

3.2. Desarrollado

El estudio del estado del arte y de las tecnologías referentes al problema del ruteo, realizado por la UDT-IA, llevó al desarrollo de distintas soluciones, buscando una visión más pragmática del problema, programando en el lenguaje de programación Java.

De esta forma, se optó por dos soluciones distintas, por una parte la heurística constructiva del Vecino Más Cercano (VMC) y, por la otra, la metaheurística de los Algoritmos Genéticos (GA).

En el primer caso (VMC), la facilidad y rapidez de implementación fueron rasgos que marcaron su elección. En el segundo caso, la muestra reiterada tanto en literatura como en soluciones software, que los GA son, hoy por hoy, la solución a los problemas de ruteo – con aportes de otros algoritmos complementarios – fue clave para elegir un algoritmo tan moldeable y potente.

Veamos pués, los resultados que se pueden extraer de tal experimentación.

3.2.1. Vecino Más Cercano

Se han hecho varias pruebas con la implementación por parte del equipo de la UDT-IA del algoritmo del vecino más cercano aplicado al CVRP.

Para el cálculo de distancias se ha utilizado la distancia euclídea de las longitudes y latitudes de los puntos porque no se dispone de una matriz de distancias real.

A continuación mostramos dos ejemplos del resultado obtenido por el algoritmo del vecino más cercano aplicado a dos zonas distintas de la geografía española. El primer ejemplo se realiza en la zona del País Vasco y el segundo ejemplo en la zona de Cataluña.

Ejemplo 1:

En este ejemplo, tenemos 8 servicios que atender y para ello contamos con 4 vehículos. En este ejemplo, todos los vehículos están ubicados en un depósito situado en el pueblo de Mondragón.

Entrada

- Recursos:

Nombre	Camión 1
Latitud	43.066466
Longitud	-2.489145
Capacidad	30

Nombre	Camión 2
Latitud	43.066466
Longitud	-2.489145
Capacidad	30

Nombre	Camión 3
Latitud	43.066466
Longitud	-2.489145
Capacidad	40

Nombre	Camión 4
Latitud	43.066466
Longitud	-2.489145
Capacidad	20

- Servicios:

Nombre	Bilbo
Latitud	43.256963
Longitud	-2.923441
Demanda	10

Nombre	Gasteiz
Latitud	42.846718
Longitud	-2.671635
Demanda	10

Nombre	Llodio
Latitud	43.143332
Longitud	-2.962985
Demanda	10

Nombre	Donostia
Latitud	43.320725
Longitud	-1.984449
Demanda	20

Nombre	Azpeitia
Latitud	43.1839
Longitud	-2.265466
Demanda	10

Nombre	Beasain
Latitud	43.047076
Longitud	-2.204605
Demanda	20

Nombre	Etxarri-Aranatz
Latitud	42.907739
Longitud	-2.064756
Demanda	20

Nombre	Aramaio
Latitud	43.052914
Longitud	-2.564295
Demanda	10

Salida

Ruta 1 - Color Verde

Recurso	Camión 1 (30)
Itinerario	Depósito → Aramaio(10) → Gasteiz(10) → Llodio(10) → Depósito
Ocupación	100%

Ruta 2 - Color Amarillo

Recurso	Camión 2 (30)
Itinerario	Depósito → Azpeitia(10) → Beasain(20) → Depósito
Ocupación	100%

Ruta 3 - Color Azul

Recurso	Camión 3 (40)
Itinerario	Depósito → Etxarri-Aranatz(20) → Donostia(20) → Depósito
Ocupación	100%

Ruta 4 - Color Rojo

Recurso	Camión 4 (20)
Itinerario	Depósito → Bilbo(10) → Depósito
Ocupación	50%

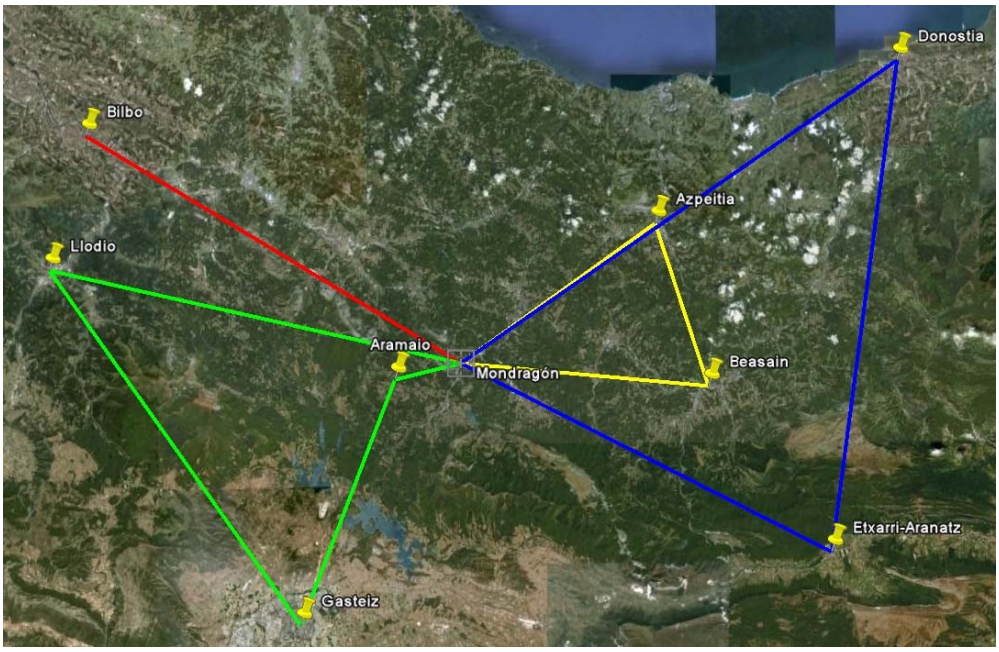


Figura 58: Solución ejemplo 1 - NN en Google Maps

```
### Lets start the NNA ###

Fitness: 2.475036699427754

XML Instance Document saved at : src\es\csic\iiia\udt\plan\test\euskadi.kml
Your CSV file has been written at: src\es\csic\iiia\udt\plan\test\euskadi.csv

### The NNA ends (1015ms) ###
```

Las conclusiones que podemos sacar de este ejemplo la alta velocidad de ejecución del algoritmo (1015ms). Las rutas obtenidas (Figura 58) aparentemente parecen correctas, en todo caso como solución inicial al problema es perfectamente válida, aunque seguramente se podría mejorar aplicando operadores de búsqueda local.

Ejemplo 2:

En este ejemplo, tenemos 12 servicios que atender y para ello disponemos de 4 vehículos. En este ejemplo, todos los vehículos están ubicados en un depósito situado en la ciudad de Barcelona.

Entrada

- Recursos:

Nombre	Camión 1
Latitud	43.066466
Longitud	-2.489145
Capacidad	70

Nombre	Camión 2
Latitud	43.066466
Longitud	-2.489145
Capacidad	20

Nombre	Camión 3
Latitud	43.066466
Longitud	-2.489145
Capacidad	40

Nombre	Camión 4
Latitud	43.066466
Longitud	-2.489145
Capacidad	60

- Servicios:

Nombre	Girona
Latitud	41.981796
Longitud	2.8237
Demanda	10

Nombre	Tortosa
Latitud	40.811004
Longitud	0.520997
Demanda	10

Nombre	LLoret
Latitud	41.698785
Longitud	2.8475839
Demanda	10

Nombre	Vic
Latitud	41.930291
Longitud	2.25435
Demanda	10

Nombre	Sant Celoni
Latitud	41.690185
Longitud	2.49167
Demanda	20

Nombre	Manresa
Latitud	41.725175
Longitud	1.823353
Demanda	10

Nombre	Vilafranca
Latitud	41.34509
Longitud	1.69985
Demanda	20

Nombre	Terrassa
Latitud	41.560953
Longitud	2.01044
Demanda	20

Nombre	Arbúcies
Latitud	41.817054
Longitud	2.514368
Demanda	10

Nombre	Igualada
Latitud	41.578765
Longitud	1.617193
Demanda	10

Nombre	Reus
Latitud	41.154818
Longitud	1.108676
Demanda	10

Nombre	Lleida
Latitud	41.614152
Longitud	0.625782
Demanda	10

Salida

Ruta 1 - Color Verde

Recurso	Camión 1 (70)
Itinerario	Depósito → Terrassa(20) → Manresa(10) → Igualada(10) → Vilafranca (20) → Reus(10) → Depósito
Ocupación	100%

Ruta 2 - Color Amarillo

Recurso	Camión 2 (20)
Itinerario	Depósito → Sant Celoni(20) → Depósito
Ocupación	100%

Ruta 3 - Color Azul

Recurso	Camión 3 (40)
Itinerario	Depósito → Vic(10) → Arbúcies(10) → Girona(10) → Lloret de Mar(10) → Depósito
Ocupación	100%

Ruta 4 - Color Rojo

Recurso	Camión 4 (60)
Itinerario	Depósito → Lleida(10) → Tortosa(10) → Depósito
Ocupación	33.33%

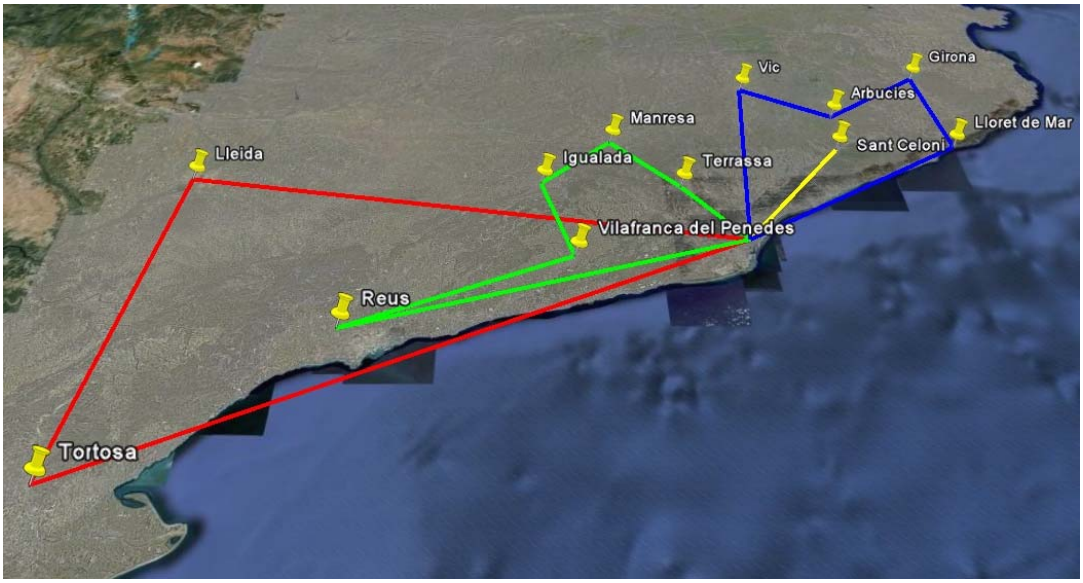


Figura 59: Solución ejemplo 2 - NN en Google Maps

```
### Lets start the NNA ###

Fitness: 5.8850538867822015

XML Instance Document saved at : src\es\csic\iiia\udt\plan\test\catalunya.kml
Your CSV file has been written at: src\es\csic\iiia\udt\plan\test\catalunya.csv

 ### The NNA ends (1078ms) ###
```

Igual que en el ejemplo anterior vemos que la ejecución del algoritmo es muy rápida (1078ms). También las rutas obtenidas (Figura 59) aparentemente parecen correctas aunque se podrían mejorar aplicando operadores de búsqueda local.

Una posible mejora seria (Figura 60) la utilización del camión 4 de capacidad 60 para cubrir la ruta del norte (Vic, Arbúcies, Girona, Lloret y Sant Celoni) y utilizar el camión 2 de capacidad 20 para cubrir Lleida i Tortosa. De esta manera nos ahorraríamos el camión 3 de capacidad 40, con un ahorro considerable en carburante y peajes.

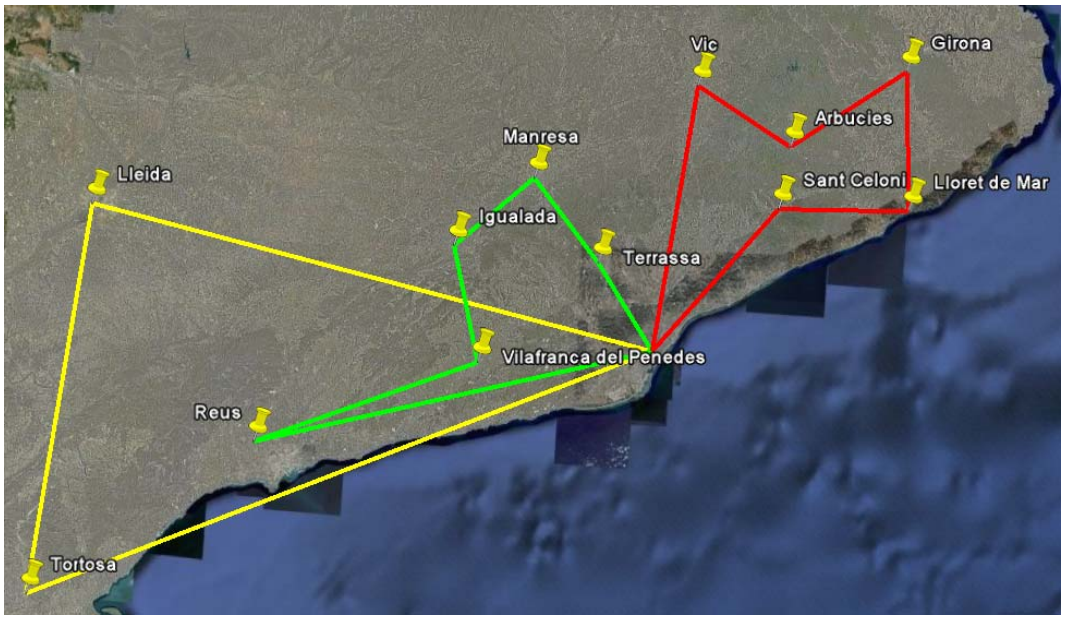


Figura 60: Posible mejora de la solución al ejemplo 2 - NN en Google Maps

3.2.2. Algoritmos Genéticos

Desde la UDT-IA se ha llevado a cabo el desarrollo integral de una aplicación basada en algoritmos genéticos, centrada en resolver el problema del TSP, en primera instancia, y el problema del VRP a continuación, dadas sus equivalencias. El hecho que este esfuerzo extra se haya realizado con los GA se debe a la evidencia concluida, que los GA – una modificación de los mismos – son los más adecuados para este tipo de problemas.

Si bien se conoce la existencia de multitud de implementaciones de GA para resolver el problema del TSP – como ya se ha mostrado en apartados anteriores – el objetivo de este desarrollo es el de definir un entorno (cromosomas, operadores, …) adaptado a las necesidades del problema concreto, pudiendo comprobar las diferencias entre ellos, implementándolos en el lenguaje de programación Java.

Esta adaptación e implementación, a partir de los estudios realizados, ha facilitado la creación de un *benchmark* en un entorno definido y controlado (tipo *sandbox*), que ha dado interesantes estadísticas.

La imagen siguiente, muestra uno de los mejores resultados obtenidos, al resolver un problema tipo TSP con el programa desarrollado. Como base para las pruebas, se ha utilizado un estándar de facto, fácilmente localizable por Internet, llamado berlin52.tsp. Este archivo contiene las coordenadas geográficas (latitud , longitud) de 52 *hotspots* de la ciudad de Berlín.

Crossover	Time (ms)	Fitness	Best result found at generation
ERO (t1)	149856	7544	172
ERO (t2)	150910	7544	648
ERO (t3)	148681	7544	294
ERO M	149816	7544	371

Computer:	PIV-ht 3.2GHz
	1GB RAM

NOTE: Initial population generated randomly.

Tests done with ...	
Genes (cities):	52+1
Generations:	1000
Population Size:	1000
Mod.Population:	30%
|- Elitism:	5%
|- Crossover:	80%
`- Mutation:	15%

Figura 61 - Resultado Ejecuciones

Este problema en concreto, el mejor *fitness* que puede llegar a dar es de 7544 y, como se muestra en la imagen anterior, el software desarrollado ofrece esa *mejor* solución. Esta implementación concreta posee estas características: Modificando sólo el 30% de la población en cada generación,

- **Operador de cruce**: *Edge Recombination Operator* – 80% del total de la modificación.
- **Mutación**: *Swap, Inversion, Insertion* y *Dispersion* – 15% del total de la modificación, escogiendo cualquiera de ellas de forma aleatoria.
- **Elitismo**: 5% del total de la modificación.

Gráficamente podemos ver la evolución del *fitness* encontrado, de la siguiente forma:

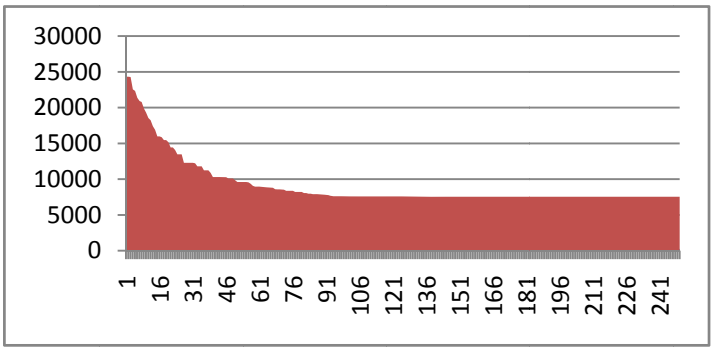


Figura 62 - Evolución fitness

Finalmente, para entender el tipo de resultado que ofrece este software, podemos dibujar el mapa que ofrecen las coordenadas, como si de un mapa se tratara:

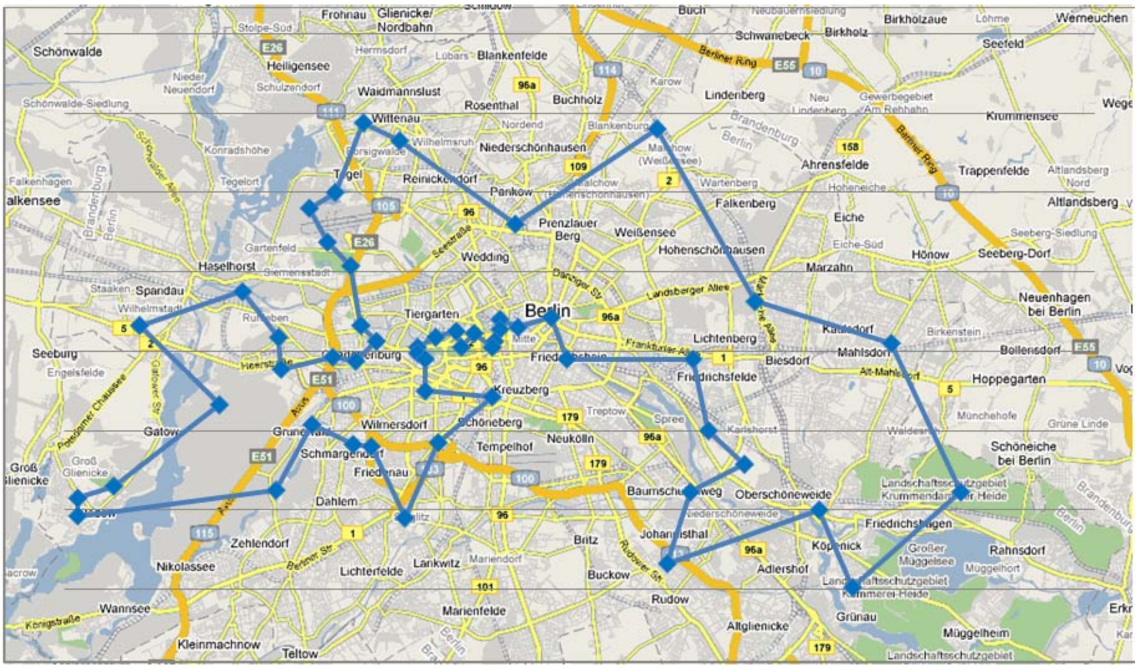


Figura 63 - Representación de la solución

En lo que respecta al problema tipo VRP, dado que todo lo mostrado hace referencia al TSP, se llegó a un estado de desarrollo en el que aún no se podían sacar conclusiones, pero sería sólo cuestión de tiempo.

4. Replanificación

Dado que el sector de la logística es un medio “vivo”, dinámico, existen multitud de situaciones, tales como averías de vehículos, obras, atascos en carretera, nuevos avisos de pedidos urgentes, etc, la planificación inicial de las rutas de los vehículos, puede sufrir cambios inesperados que hay que introducir en el sistema para atenderlos.

Actualmente, este proceso se realiza de forma totalmente manual, donde un operador va comunicándose con todos los vehículos a ver quien puede hacerse cargo del nuevo pedido, puede ir a buscar la carga de otro vehículo averiado, etc.

El objetivo es preparar un mecanismo que nos permita realizar esta acción de forma automática.

Una posibilidad para enfocar este problema consiste en utilizar sistemas dinámicos de planificación, en lugar de sistemas estáticos.

Por ejemplo en [Monte03] presentan un algoritmo dinámico basado en hormigas que funciona mediante un Event Manager. El día de trabajo se divide en franjas horarias o slots de tiempo. Las órdenes recibidas en una franja, se tienen en cuenta a partir de la siguiente. Al comienzo de cada slot, se resuelve un VRP estático con las órdenes nuevas, donde cada vehículo empieza en el último cliente visitado y con la capacidad de carga actual. Las órdenes que llegan fuera de una hora final de planificación, se dejan para el día siguiente.

En [Zeddi08] proponen un sistema multi agente con sistema de subastas, con agentes para los clientes y para los vehículos. En esta ocasión, también consiste en resolver un VRP estático, pero en vez de hacerlo por franjas horarias, en esta ocasión se realiza cada vez que ocurre una incidencia. Aquí se separan los posibles cambios en tres categorías distintas: adición de un servicio, cancelación de un servicio o cambio en un servicio. En el primer y tercer caso, se inserta el nuevo/modificado servicio en una de las rutas ya existentes. En el segundo caso, se resuelve el VRP de nuevo pero quitando el servicio cancelado.

Los autores de [Fleis04] utilizan un algoritmo en el que para cada nuevo evento vuelven a planificar las órdenes restantes y después insertan la nueva orden donde minimice el coste.

Otra posibilidad es hacer una planificación estática inicial que seguirán todos los vehículos si no hay problemas. En el momento en que ocurra una incidencia, se pasa al sistema la planificación realizada, el vehículo que ha tenido problemas y los nuevos servicios que hay que realizar, además de los que aún no se han realizado. Con estos datos, se hace una reparación de las rutas ya realizadas para poder cumplir con las nuevas características del problema.

Esta reparación consiste en una inserción de los nodos afectados por la incidencia en las otras rutas donde se minimice el coste de la asignación y siga creando soluciones viables. Una vez asignados todos los nuevos nodos se devuelve la nueva planificación.

5. Arquitectura propuesta

Una vez comentadas las diferentes soluciones que se encuentran tanto a nivel comercial como a nivel académico o de investigación, en este apartado pasaremos a proponer una solución para un planificador de recursos móviles.

Una vez realizado el estudio de las diferentes tecnologías de Inteligencia Artificial y teniendo en cuenta las limitaciones y suposiciones anteriores, creemos que la técnica más apropiada para la resolución del problema CVRPTW es un algoritmo memético (combinación de algoritmos genéticos y procesos de búsqueda local).

Las razones de que el algoritmo elegido sea éste y no otro son las siguientes:

- Dado que desconocemos el alcance del dominio del problema, no podemos aplicar una técnica heurística dependiente del problema.
- Los algoritmos genéticos dan buenos resultados en un tiempo razonable. El objetivo principal no es encontrar la solución óptima, sino encontrar una buena solución en poco tiempo. Además, nos ofrecen herramientas (mutación) para salir de mínimos locales que están tapando mejor mínimos cercanos.
- Los operadores de búsqueda local nos ofrecen la posibilidad de explotar las soluciones encontradas para intentar encontrar otras que sean mejores pero cercanas en el espacio de búsqueda, de esta manera podemos llegar a encontrar mínimos locales que pueden ser interesantes como solución final.

Una vez decidido la técnica a emplear, pasaremos a continuación a explicar los elementos principales de los que se compondrá el algoritmo. Al ser la base un algoritmo genético, los elementos que se tienen que definir son los siguientes:

- Representación de los cromosomas y soluciones
- Construcción de la población inicial
- Elitismo
- Selección de individuos
- Operador de cruce
- Operador/es de mutación

También tenemos que definir qué operadores de búsqueda local se van a aplicar y a qué soluciones se aplican, así como el orden en que se ejecutarán si hay más de uno.

5.1. Representación de los cromosomas

El factor determinante de si los algoritmos genéticos van a dar mejores o peores soluciones y si va a ser más sencillo o más complicado trabajar con ellos es la representación de los cromosomas.

Una posibilidad de representación muy utilizada de los cromosomas es mediante un array de enteros de longitud *n*, donde *n* es el número de clientes [Tan01], [Perei02], [Geiger08], [Ghos09], [Fal08]. Un ejemplo podría ser el siguiente:

[2, 4, 3, 1, 5, 6, 9, 7, 10, 8]

donde no tenemos ningún marcador de los vehículos. De esta manera, el operador de cruce y los operadores de mutación, son muy fáciles de construir, ya que no se tiene que diferenciar entre vehículos y servicios.

En otro array o lista, tenemos los diferentes vehículos que podemos utilizar.

Por un lado esto es muy bueno porque simplifica las operaciones del genético, pero por otro lado, hay que tener en cuenta que a la hora de verificar si las soluciones son válidas o no y calcular su coste, hay que tener un procedimiento de cración de rutas para indicar a cada vehículo que nodos ha de visitar y en qué orden. Esto se suele realizar de la siguiente manera:

- De forma secuencial, se van añadiendo nodos al primer vehículo hasta que dejen de cumplir las restricciones de capacidad y/o tiempo.
- Se va repitiendo este proceso hasta que ya no quedan nodos por visitar.

En el ejmplo anterior, si tenemos tres vehículos (V1, V2, V3), tendríamos esta posible solución:

- V1 = [2, 4, 3, 1]
- V2 = [5, 6]
- V3 = [9, 7, 10, 8]

Una vez tenemos las rutas creadas, podemos comprobar si son viables o no y calcular su coste. Además, se pueden aplicar los operadores de búsqueda local.

Al finalizar todas las modificaciones de las rutas, se realiza el proceso inverso para volver a construir el cromosoma a partir de la ruta, que consiste en crear un array añadiendo de manera secuencial todos los nodos del vehículo V1, luego del vehículo V2, …, hasta Vm, donde m es el número de vehículos.

Otra posibilidad de representación la tenemos en [Alba06], [Dorro07]. Consiste en tener un vector de enteros en el que se encuentran tanto los clientes a visitar como marcadores de separación de rutas, que equivaldrían a los camiones. Si tenemos *c* clientes y *k* camiones, la longitud del cromosoma será n= c + k. Los clientes serán representados con los valores [o … c-1] y los marcadores de ruta con los valores [c … n-1]. De esta manera, si tenemos 8 clientes y 3 marcadores tendríamos la representación siguiente:

Clientes = [0, 1, 2, 3, 4, 5, 6, 7] Marcadores = [8, 9, 10]

Cromosoma = [8, 0, 4, 5, 10, 3, 2, 1, 9, 6, 7]

Ruta1 = [0, 4, 5] Ruta2 = [3, 2, 1] Ruta3 = [6, 7]

En este tipo de representación tenemos en el mismo cromosoma las rutas ya definidas, por las que no se necesitaría un proceso de construcción de rutas ni de cromosoma, pero nos añade genes al cromosoma y lo complica. Tenemos que tener en cuenta que, dado que una ruta se define como los elementos que hay entre dos separadores, no podemos permitir soluciones del tipo [0, 1, 2, 8, 5, 4, …] donde la primera ruta no tiene ningún separador anterior.

Esto se podría solucionar si suponemos que en la posición "-1" del vector, tenemos el primer separador. Si lo hacemos de esta manera, en lugar de tener *k* separadores, tenemos *k – 1*. Así no hace falta tener un separador siempre en la primera posición del vector.

Al tener varias posibilidades de representación, una buena opción es compararlas entre ellas para ver cual es mejor. Esto lo encontramos en [Xu05]. Aquí se comparan tres posibles representacaiones de los cromosomas.

En la primera representación (GR1) cada posición del vector representa el servicio a realizar y el contenido el vehículo que va a realizar dicho servicio. Por ejemplo, si tenemos el vector [1, 2, 1, 1, 2, 2], tendríamos seis servicios y dos camiones donde el camión 1 realizaría los servicios [1, 3, 4] y el camión 2 los servicios [2, 5, 6]. El problema de esta representación, es que no nos da el orden de visita. El camión 1 podría realizar seis rutas distintas para visitar los tres servicios asignados: [1, 3, 4], [1, 4, 3], [3, 1, 4], [3, 4, 1], [4, 1, 3], [4, 3, 1].

La segunda representación (GR2) comparada es la de [Alba07], pero aquí especifican que si no hay separador en la primera posición del vector, estos nodos iniciales se añaden a la ruta del último separador encontrado del vector. Por ejemplo si tenemos el cromosoma [1, 2, 8, 3, 4, 9, 5, 6, 10, 7], donde hay tres separadores (8, 9, 10) y 7 servicios (1, 2, 3, 4, 5, 6, 7), las rutas encontradas serían las siguientes:

Ruta1 = [3, 4] Ruta2 = [5, 6] Ruta3 = [7, 1, 2]

La última representación (GR3) que se compara es la primera que se explicó, la más utilizada consistente en un vector de *n* servicios sin separadores de ruta y con la construcción de rutas cuando sea necesario comprobar coste, viabilidad, etc.

El resultado de la comparación resulta ser que con GR3 se consiguen los mejores resultados y GR1 es la que da peores resultados. En principio, que GR3 de mejores resultados que GR2 en términos de complejidad se debe a que el tamaño de los cromosomas es menor en GR3, pero el proceso que se necesita de decodificación de las rutas, puede ser más costoso en tiempo que en GR2 que es directo.

Una vez vistas diferentes posibilidades para la representación de los cromosomas, nosotros optamos por utilizar la representación de cromosomas **GR2**.

5.2. Construcción de la población inicial

Para la construcción de la población inicial, podemos utilizar varios métodos de construcción para crear diversos subconjuntos de soluciones y con ellas formar la población.

En nuestro caso, la población inicial se construirá de la manera siguiente:

- Una parte de la solución (alrededor del 30%) se hará mediante aplicación de diferentes operadores de búsqueda local sobre la solución del vecino más cercano.
- Para asegurarnos soluciones viables de un inicio un 20% de la población se creará mediante el algoritmo del 2-vecino más cercano, que es una extensión del vecino más cercano en el que en vez de escoger el nodo más cercano, se elige aleatoriamente uno de los dos nodos más cercanos.
- El resto de soluciones (un 50%) se realizarán de manera aleatoria.

5.3. Elitismo

En el algoritmo propuesto, vamos a usar el elitismo para asegurarnos que las mejores soluciones de una población formen parte de la siguiente, para ir teniendo una población de individuos cada vez mejor.

5.4. Selección de parientes para el cruce

Una vez disponemos de una población evaluada y lista para dar nacimiento a una nueva generación, procedemos a seleccionar los individuos que serán tomados como padres. Para esta tarea, se ordena toda la población actual, de mejor a peor *fitness*, y se escogen los mejores individuos, en un porcentaje equivalente al del volumen de población que queremos modificar.

Cuando queremos generar un nuevo cromosoma, seleccionamos aleatoriamente dos de los padres de la selección anterior y procedemos a ejecutar la operación que nos interese. Al mismo tiempo, se deben marcar los padres seleccionados como “usados” en una lista paralela, con tal de evitar repetir padres. Lo mismo haremos con los hijos a reemplazar, para no sobrescribirlos.

5.5. Operador de cruce

Dada la codificación seleccionada para los cromosomas, el operador de cruce que más se puede ajustar a nuestras necesidades es el que encontramos en [Perei02], ya que tiene en cuenta las rutas y de esta manera no tenemos problemas con tener los vehículos integrados en el cromosoma.

5.6. Operadores de mutación

En esta propuesta, no se va a utilizar sólo un operador de mutación sino que se usarán cuatro tipos de mutación distinta. Una vez el algoritmo decida mutar un cromosoma, entonces se elegirá una de estas cuatro mutaciones distintas de modo aleatorio para ser aplicada.

Los distintos tipos de mutación utilizados son los siguientes:

- *Swap*
- *Inversion*
- *Insertion*
- *Dispersion*

5.7. Búsqueda local

Una vez realizadas las diferentes operaciones del genético, a un cierto porcentaje de soluciones (alrededor del 10%) se le aplicará un proceso de búsqueda local para intentar mejorarlas buscando en su vecindario del espacio de búsqueda posibles soluciones que sean mejores que la actual.

Los operadores utilizados serán los siguientes:

- 2-Opt intra ruta.
- Exchange inter ruta
- Relocate inter ruta

Estos son los operadores que se han demostrado más eficaces para este tipo de problemas.

5.8. Replanificación

Aquí entramos en el terreno de la investigación. Aunque las técnicas CBR expuestas en este documento no fueron concebidas para tal fin, desde la UDT-IA creemos que existe la posibilidad de utilizar alguna de las técnicas de la etapa “Reutilizar” con la finalidad de reparar rutas ya creadas.

Este es un punto poco explorado desde el punto de vista de la investigación y por lo tanto, esta propuesta, debe tomarse con total precaución.

La idea consiste en la utilización de una aplicación externa con acceso a los resultados de la aplicación principal, con la finalidad de añadir ciudades a la s rutas ya creadas, tal y como se expone en el apartado CBR de este documento.

El algoritmo aplicado para la **reutilización** es el siguiente:

```
function Add-cities(tour, rem-cities)
{
  if rem-cities
  {
    best-city <-Select-best-city(tour, rem-cities)
    rem-cities <-Remove-city(best-city, rem-cities)
    tour <-Insert-city(tour, best-city)
    Add-cities(tour, rem-cities)
  }
}
```

Donde `tour` es la ruta a replanificar, `y rem-cities` es el conjunto de ciudades externas a insertar en la ruta.

`Select-best-city` selecciona las ciudades a insertar (extrayéndola de la lista `Remove-city`) y las inserta en el lugar de la ruta que ofrece mínimo coste (`Insert-city`).

6. Conclusiones

En este proyecto se ha realizado un estudio de la problemática que tienen actualmente las empresas del sector de la logística para la planificación de los recursos móviles y como la inteligencia artificial puede facilitar dicha tarea, que actualmente se hace de manera sino completamente manual, semi-automática en el mejor de los casos.

La variante más sencilla de todas es la del TSP (Traveling Salesman Problem) que consiste en que un comercial debe visitar diferentes puntos y se quiere minimizar la distancia recorrida. A partir del TSP, sale una variante más compleja que ya se acerca más a nuestras necesidades, el conocido como VRP (Vehicle Routing Problem) en la que la diferencia con el TSP es que se pueden añadir restricciones al sistema para acercarlo más a la realidad, tales ocmo restricciones de capacidad de los vehículos (CVRP – Capacitated Vehicle Routing Problem), restricciones de tiempo (VRPTW – Vehicle Routing Problem with Timie Windows), que indican que un vehículo debe llegar a cada cliente en un intervalo de tiempo determinado o una combinación de las dos, el conocido como CVRPTW (Capacitated Vehicle Routing Problem with Time Windows). Existen más variantes pero creemos que son demasiado específicas para nuestro caso, así que nos inclinamos más hacia que el problema que debemos resolver consiste en el CVRPTW.

Una vez encontrado y definido el problema, pasamos a investigar en la bibliografía, que técnicas basadas en inteligencia artificial se utilizaban para resolverlo.

Existe una gran variedad de posibles técnicas a utilizar, que podríamos definir en tres grandes grupos:

- Óptimos/Exactos
- Técnicas heurísticas
- Técnicas metaheurísticas

Las técnicas exactas son aquellas que buscan todas las soluciones posibles al problema para acabar devolviendo la mejor existente. Esto implica que si el dominio donde se van a aplicar es muy grande, el tiempo necesario para poder encontrar la solución óptima es extremadamente grande, aunque existan algoritmos que evitan realizar búsquedas en regiones donde seguro que no se encuentra la mejor solución posible, como el Branch and Bound.

Las técnicas heurísticas, son aquellas que dependen del dominio y que pueden aprovechar este conocimiento para encontrar mejores soluciones, por lo que es necesario este conocimiento para poder aplicarlas. Normalmente no acaban dando la solución óptima, pero dan buenas soluciones.

Estas técnicas se pueden dividir en tres categorías distintas:

- Heurísticas constructivas
- Heurísticas de dos fases
- Heurísticas de mejora o de búsqueda local

Los métodos constructivos son aquellos que, a partir de una solución vacía, van construyendo poco a poco una solución viable que intenta ser la mejor posible. Entre este tipo de algoritmos tenemos el del vecino más cercano (NN – Nearest Neighbor) o el algoritmo de Clarke and Wright (o de los ahorros).

Los de dos fases, son métodos que primero separan los clientes en conjuntos y luego se buscan rutas dentro de cada conjunto (cluster first- route second), o los que primero hacen una ruta global para todos los clientes y luego separan en subrutas para cada vehículo (route first – cluster second). Los primeros suelen dar mejores resultados que los primeros. Un ejemplo de estos es el algoritmo de barrido o de Gillet y Miller.

Por último tenemos los de mejora o de búsqueda local. Estos se utilizan para explotar ciertas regiones del espacio de búsqueda donde puede haber buenas soluciones (mínimos locales) y así mejorar las soluciones encontradas por otras cercanas o vecinas. Existen dos familias de operadores: los que operan en una sola ruta (intra ruta) o entre diferentes rutas (inter rutas). Entre estos operadores, uno de los más utilizados es el 2-Opt que pertenece a la familia de intra ruta.

En cambio, las metaheurísticas, son independientes del dominio en el que se aplican, por lo que son técnicas mucho más generales que las anteriores y que no necesitan, a priori, ningún conocimiento del dominio. Este conocimiento, no obstante puede aprovecharse para optimizar estos algoritmos.

Hoy en día las investigaciones y los paquetes comerciales tienden a basarse más en técnicas de este tercer grupo que del segundo y que por supuesto del primero.

De técnicas meteheurísticas existen multitud de familias, con orígenes bien diversos, como los basados en la naturaleza, en procesos técnicos, etc.

Los algoritmos basados en Colonias de Hormigas o ACO (Ants Colony Optimization) intentan reflejar y copiar el sistema que utilizan las hormigas para encontrar los caminos más cortos del hormiguero a las diferentes fuentes de comida. A partir de esta filosofía, se pueden aplicar al VRP para encontrar los caminos más cortos para visitar todos los nodos.

Otros algoritmos basados en la naturaleza son los genéticos, que tratan de imitar la teoría de la evolución de Darwin aplicada al problema concreto del VRP. A partir de una población de soluciones iniciales, mediante sistemas de cruce entre soluciones (reproducción) y mutaciones, se va mejorando la población y sus miembros para intentar encontrar la mejor solución posible. Este tipo de algoritmos es muy utilizado en este tipo de problemas y los productos comerciales suelen recurrir a este tipo de soluciones para crear el producto.

Otro tipo de algoritmo es el Recocido Simulado o Simulated Annealing que está basado en un proceso de tratamiento de metales del mismo nombre. Estos algoritmos comienzan permitiendo mucha exploración en el espacio de búsqueda y poco a poco van cambiando la exploración por la explotación de la región donde parece que se encuentran las mejores soluciones.

Por último tenemos la Búsqueda Tabú o Tabu Search, que intenta recorrer todo el espacio de búsqueda prohibiendo volver a recorrer caminos que ya se han investigado antes.

Esta técnica no suele utilizarse sola, sino que normalmente va acompañando a otra metaheurística para mejorarla.

Este concepto, el de mezclar técnicas entre ellas para mejorar la calidad de las soluciones obtenidas, es algo habitual hoy en día y no se concibe la aplicación de ninguna técnica metaheurística pura. Se hacen híbridos entre diferentes técnicas, ya sean metaheurísticas, o heurísticas y metaheurísticas.

Entre los híbridos más utilizados tenemos los algoritmos Meméticos, que combinan algoritmos genéticos con operadores de búsqueda local heurística.

Una vez hemos visto que técnicas existen para resolver el VRP, se ha realizado un test de diferentes aplicaciones que implementan dichas técnicas para poder valorar y comparar sus resultados y su comportamiento, así como el desarrollo de algunas de ellas para poder adaptarlas a nuestras necesidades y pruebas.

Uno de los objetivos de este proyecto consistía en un estudio de las diferentes tecnologías existentes para resolver el problema que ha quedado definido como el CVRPTW y proponer una arquitectura para el nuevo sistema.

El otro objetivo del proyecto era el de añadir la posibilidad de replanificar cuando ocurren emergencias o eventos inesperados para poder corregir las rutas y adaptarlas a estas nuevas condiciones.

Hemos estudiado las diferentes posibilidades que hay a la hora de realizar este proceso y hemos visto que todos los artículos estudiados dan soluciones parecidas, que consisten en cada vez que ocurre una incidencia o cada cierto tiempo, se prepara un VRP estático con los servicios pendientes y los nuevos y se resuelve, dando lugar a una nueva planificación, normalmente utilizando un proceso de inserción de los nuevos servicios en las rutas ya existentes minimizando en sobrecoste de la operación.

Otra posibilidad, que todavía está en una fase prematura es la utilización de la tecnología CBR para replanificar, pero para ello es necesario disponer de una base de casos suficientemente amplia que se haya recogido durante un cierto periodo de tiempo para poder encontrar soluciones similares que se puedan reutilizar para las nuevas incidencias.

Por último hemos propuesto una arquitectura para el sistema basada en un algoritmo memético, a la cual le vemos distintas posibilidades futuras como:

- Creación de rutas abiertas. De esta manera un vehículo no debe volver siempre al depósito al finalizar los servicios.
- División de rutas por jornadas. Si una planificación tiene una duración mayor de las horas que puede operar un vehículo (o su chofer), se puede hacer que pase noche en algún sitio y continúe la ruta al día siguiente.
- Intentar balancear el uso de los vehículos, de tal manera que en vez de minimizar los vehículos, se intenta que todos los vehículos trabajen.
- Posibilidad de hacer cargas y descargas. Cargar en un punto, descargar en otro una parte, volver a cargar, etc.
- Posibilidad de restringir que a algunos servicios sólo puedan ir determinados vehículos, por ejemplo cubas, depósitos, etc.

7. Bibliografía

[AntSim09] AntSim v1.1**.** http://www.nightlab.ch/antsim.php . 2009

[Alba06] Alba, E. and Dorronsoro, B. *Computing nine new best-so-far solutions for capacitated VRP with a cellular Genetic algorithm*. Inf. Process. Lett. 98, 6 (Jun. 2006), 225-230.

[ACODemo09] ACOpt - Ant Colony Optimization Demonstration. http://www.borgelt.net/acopt.html . 2009

[Bara06] Barajas, N. *Estado del Arte del Problema de Ruteo de Vehículos (VRP)*. 2006. http://www.geocities.com/nicolasbarajasseminario/Archivos/EstadoDelArteBorrador.pdf

[Barba99] Barbarosoglu, G. and Ozgur, D. *A tabu search algorithm for the vehicle routing problem*. Comput. Oper. Res. 26, 3 (Mar. 1999), 255-270.

[Bian02] Bianchi, L., Gambardella, L. M., and Dorigo, M. *An ant colony optimization approach to the probabilistic traveling salesman problem*. In J. J. Guerv´s, o P. Adamidis, H. Beyer, J. L. Mart´ and H. Schwefel, editors, In Proceedings ın, of the 7th International Conference on Parallel Problem Solving From Nature, PPSN VII, volume 2439 of LNCS, pages 883–892, London, UK, 2002b. Springer-Verlag.

[Bouh08] Bouhafs, L., Hajjam, A., Koukam, A., *A Tabu Search and Ant Colony System Approach for the Capacitated Location-Routing Problem.* Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, *2008. SNPD '08. Ninth ACIS International Conference on* , vol., no., pp.46-50, 6-8 Aug. 2008

[Bran06] Brandao de Oliveira, H. C., Crispim Vasconcelos, G., Bastos Alvarenga, G. *A Multi-Start Simulated Annealing Algorithm for the Vehicle Routing Problem with Time Windows*. In Proceedings of the Ninth Brazilian Symposium on Neural Networks (October 23 - 27, 2006). SBRN. IEEE Computer Society, Washington, DC, 24. 2006.

[Bray02] Bräysy, O., Gendreau, M. *Tabu Search heuristics for the Vehicle Routing Problem with Time Windows.* TOP: An Official Journal of the Spanish Society of Statistics and Operations Research, Springer, vol. 10(2), pages 211-237, December. 2002.

[Caric08] Caric, T. et al. *Vehicle Routing Problem.* i-Tech, Vienna, Austria, 2008.

[Concorde09] Concorde TSP Solver. http://www.tsp.gatech.edu/concorde.html. 2009

[Cunn97] Cunningham, P. and Smyth, B. *Case-based reasoning in scheduling: Reusing solution components*. Internationa Journal of Production Resarch, 35(11): 2947 – 2962, Novmeber 1997,

[Czar02] Czarnas, P., Czech, Z. J. *Parallel simulated annealing for the vehicle routing problem with time windows*. 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing, Canary Islands–Spain. Pages 376—383. 2002.

[Dona08] Donati, A. V., Montemanni, R., Casagrande, N., Rizzoli, A. E. and Gambardella, L. M.*Time dependent vehicle routing problem with a multi ant colony system.* European Journal of Operational Research, Elsevier, vol. 185(3), pages 1174-1191, March 2008.

[Dorigo96] Dorigo, M., Maniezzo, V., Colorni, A. *Ant system: optimization by a colony of cooperating agents*, Systems, Man, and Cybernetics, Part B, IEEE Transactions on , vol.26, no.1, pp.29-41, Feb 1996

[Dorro07] Dorronsoro, B., Arias, D., Luna, F., Nebro, A.J., Alba, E. *A Grid-Based Hybrid Cellular Genetic Algorithm for Very Large Scale Instances of the CVRP*.High Performance Computing & Simulation Conference (HPCS 2007). Prague, Czech Republic, June 2007

[Fal08] Fallahi, A. E., Prins, C., and Wolfler Calvo, R. *A memetic algorithm and a tabu search for the multi-compartment vehicle routing problem*. Comput. Oper. Res. 35, 5 (May. 2008), 1725-1741.

[Fleis04] Fleischmann, B., Gnutzmann, S., and Sandvoβ, E. Dynamic Vehicle Routing Based on Online Traffic Information. Transportation Science 38, 4, 420-433. 2004.

[Gamb99] Gambardella, L. M., Taillard, É., and Agazzi, G. *MACS-VRPTW: a multiple ant colony system for vehicle routing problems with time windows*. In New Ideas in Optimization, D. Corne, M. Dorigo, F. Glover, D. Dasgupta, P. Moscato, R. Poli, and K. V. Price, Eds. Mcgraw-Hill'S Advanced Topics In Computer Science Series. McGraw-Hill Ltd. UK, Maidenhead, UK, 63-76, 1999.

[Gomez09] Gomez, M. and Plaza, E. *Case-based Reasoning for Scheduling Problems*. Waiting to final version, 2009.

[Geig08] Geiger, M. *A Computational Study of Generic Crossover Operators for Multi-Objective with Soft Time Windows*, 2008. http://arxiv.org/pdf/0809.0410

[Gend99] Gendreau, M., Guertin, F., Potvin, J., and Taillard, E. *Parallel Tabu Search for Real-Time Vehicle Routing and Dispatching*. Transportation Science 33, 4 (Apr. 1999), 381-390.

[Ghos09] Ghoseiri, K. and Ghannadpur, S.F. *Hybrid Genetic Algorithm for Vehicle Routing and Scheduling Problem*. Journal of applied sciences [1812-5654] Ghoseiri v:9 n:1 p:79, 2009.

[Hur97] Hurtado, R.V., Paz, J. *Framework en Java para el Desarrollo de Sistemas de Colonia de Hormigas*, 1997. http://www.postgradoinformatica.edu.bo/enlaces/investigacion/pdf/INGSW3_97.pdf

[JOpt09] JOpt Advanced Vehicle Routing Components.

http://www.dna-evolutions.com/joptproductlineoverview.html . 2009

[Karo05] Karova, M., Smarkov, V., Penev, S. *Genetic operators crossover and mutation in solving the TSP problem*. International Conference on Computer Systems and Technologies- CompSysTech 2005.

[Lau03] Lau, H. C., Sim, M., Teo, K. M. *Vehicle routing problem with time windows and a limited number of vehicles.* European Journal of Operational Research, Elsevier, vol. 148(3), pages 559-569, August 2003.

[Leong06] Leong, H. W. and Liu, M. *A multi-agent algorithm for vehicle routing problem with time window*. In *P*roceedings of the 2006 ACM Symposium on Applied Computing (Dijon, France, April 23 - 27, 2006). SAC '06. ACM, New York, NY, 106-111.

[Lin09] Lin, Shih-Wei, Yu, V. F., Chou, Shuo-Yan. *Solving the truck and trailer routing problem based on a simulated annealing heuristic.* Computers & Operations Research, Volume 36, Issue 5, Selected papers presented at the Tenth International Symposium on Locational Decisions (ISOLDE X), May 2009, Pages 1683-1692. 2009.

[lin09b] Lin, Shih-Wei. Lee, Zne-Jung. Ying, Kuo-Ching. Lee, Chou-Yuan. *Applying hybrid meta-heuristics for capacitated vehicle routing problem.* Expert Systems with Applications, Volume 36, Issue 2, Part 1, March 2009, Pages 1505-1512. 2009.

[Monte03] Montemanni R., Gambardella L. M., Rizzoli A. E., Donati A. V. A new algorithm for a Dynamic Vehicle Routing Problem based on Ant Colony System. In Second International Workshop on Freight Transportation and Logistics, pages 27-30. 2003.

[Monte05] Montemanni, R., Gambardella, L.M., Rizzoli, A.E. And Donati A.V. *Ant colony system for a dynamic vehicle routing problem*. Journal of combinatorial optimization [1382-6905] MONTEMANNI año:2005 v:10 n:4 p:327

[nana07] Nanayakkara, S.C.; Srinivasan, D.; Lai Wei Lup; German, X.; Taylor, E.; Ong, S.H., *Genetic Algorithm based route planner for large urban street networks*, Evolutionary Computation, 2007. CEC 2007. IEEE Congress on , vol., no., pp.4469-4474, 25-28 Sept. 2007

[OpenTS09] OpenTS - Java Tabu Search. http://www.coin-or.org/Ots/ . 2009

[Perei02] Pereira, F. B., Tavares, J., Machado, P., and Costa, E. *GVR: A New Genetic Representation for the Vehicle Routing Problem*. In Proceedings of the 13th Irish international Conference on Artificial intelligence and Cognitive Science (September 12 - 13, 2002). M. O'Neill, R. F. Sutcliffe, C. Ryan, M. Eaton, and N. Griffith, Eds. Lecture Notes In Computer Science, vol. 2464. Springer-Verlag, London, 95-102.

[Qi08] Qi, C. *An Ant Colony Algorithm with Stochastic Local Search for the VRP*. In Proceedings of the 2008 3rd international Conference on innovative Computing information and Control - Volume 00 (June 18 - 20, 2008). ICICIC. IEEE Computer Society, Washington, DC, 464. 2008.

[Qiang08] Qiang, Z., Qiuwen, Z. *An Improved Ant Colony Algorithm for the Logistics Vehicle Scheduling Problem.* Intelligent Information Technology Application, 2008. IITA '08. Second International Symposium on , vol.2, no., pp.55-59, 20-22 Dec. 2008

[Robus05] Robusté, F., Glaván, D. *e-Logistics*. Edicions UPC. 2005.

[Solo87] Solomon, M. M. *Algorithms for the vehicle routing and scheduling problems with time window constraints*. *Oper. Res.* 35, 2 (Apr. 1987), 254-265.

[Tan01] Tan, K. C., Lee, L. H., Zhu, Q. L., Ou, K. *Heuristic methods for vehicle routing problem with time windows*, Artificial Intelligence in Engineering, Volume 15, Issue 3, July 2001, Pages 281-295, ISSN 0954-1810.

[Tavak07] Tavakkoli-Moghaddam, R., Safaei, N., Kah, M.M.O., Rabbani, M. *A New Capacitated Vehicle Routing Problem with Split Service for Minimizing Fleet Cost by Simulated Annealing*, Journal of the Franklin Institute, Volume 344, Issue 5, Modeling, Simulation and Applied Optimization Part II, August 2007, Pages 406-425, ISSN 0016-0032. 2007.

[TSPGA09] Traveling Salesman Problem Genetic Algorithm. http://www.mathworks.com/matlabcentral/fileexchange/13680 . 2009

[Vacic02] Vacic, V., Sobh, T.M., *Vehicle Routing Problem*, 2002.

http://www.vacic.org/lib/vrptw.pdf

[Voudouris03] Voudouris, C. and Tsang, E., Guided local search., En: Glover, F. and Kochenberger, G. (Eds.), *Handbook of metaheuristics*, Kluwer academic publisher, 2003.

[Xu05] Yi-Liang Xu; Mene-Hiot Lim; Meng-Joo Er, *Investigation on genetic representations for vehicle routing problem.* Systems, Man and Cybernetics, 2005 IEEE International Conference on , vol.4, no., pp. 3083-3088 Vol. 4, 10-12 Oct. 2005

[Yepes00] Yepes, V. Medina, J.R. *Optimización del problema generalizado de las rutas con restricciones temporales y de capacidad (CVRPSTW)*. Actas del IV Congreso de Ingeniería del Transporte, 2000. http://personales.upv.es/vyepesp/00YMX13.pdf

[Zabala05] Zabala, C.A. *Implementación del Sistema de Colonia de Hormigas con Búsqueda Local al Problema de Ruteo de Vehículos con Capacidad y Ventanas de Tiempo (CVRPTW)*, 2005. http://hdl.handle.net/1992/851

[Zeddi08] Zeddini, B., Temani, M., Yassine, A., and Ghedira, K. *An Agent-Oriented Approach for the Dynamic Vehicle Routing Problem*. In Proceedings of the 2008 international Workshop on Advanced information Systems For Enterprises - Volume 00 (April 19 - 20, 2008). IWAISE. IEEE Computer Society, Washington, DC, 70-76.

[Zhang08] Zhang, X., Tang, L. 2008. *A new hybrid ant colony optimization algorithm for the vehicle routing problem.* Pattern Recognition Letters, In Press, Corrected Proof, Available online 12 June 2008, ISSN 0167-8655.

[Zhen08] Zhen, T., Zhang, Q., Zhang, W., Ma, Z. 2008. *Hybrid Ant Colony Algorithm for the Vehicle Routing with Time Windows.* Computing, Communication, Control, and Management, 2008. CCCM '08. ISECS International Colloquium on , vol.1, no., pp.8-12, 3-4 Aug. 2008

A. Anexo 1

RECURSOS	
Posición conocida	Latitud
	Longitud
	Hora
Capacidad	Peso
	Volumen
Horario	Horario laboral
	Calendario trabajo
Costes	Coste/hora
	Coste/Km.
	Coste/hora extra
Condicionantes	Tipos de pedido admitidos

SERVICIOS (carga-descarga o visita)	
Origen (carga o visita)	Latitud
	Longitud
	Horario laboral
	Calendario trabajo
	Duración de carga/visita
Destino (descarga) *(solo para servicios carga-descarga)*	Latitud
	Longitud
	Horario laboral
	Calendario trabajo
	Duración de descarga
Detalle del servicio	Tipo
	Peso
	Volumen
Horario finalización	Hora mínima
	Hora máxima

FUNCIÓN DE COSTE	
Criterios de optimización (ajustados a necesidades del cliente)	Coste de horas
	Coste de Km.
	Coste de horas extras
	Coste por impuntualidad
	Nº vehículos utilizados

RESULTADOS
Relación de recursos con los servicios asignados a cada uno de ellos
Coste total del resultado obtenido

B. Anexo 2

El CD que acompaña a la documentación incluye:

proyecto → Codigo fuente del software desarrollado.

video → Video demostración de los algoritmos desarrollados por la UDT-IA.