Trabajo Fin de Grado
Grado en Ingeniería de Tecnologías Industriales

Resolución heurística de un problema de rutado con aplicaciones para el comercio electrónico

Autora: Sara Medrán Medrán
Tutor: Alejandro Escudero Santana

Dep. Organización Industrial y Gestión de Empresas II
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2017

Trabajo Fin de Grado
Grado en Ingeniería en Tecnologías Industriales

Resolución heurística de un problema de rutado con aplicaciones para el comercio electrónico

Autora:
Sara Medrán Medrán

Tutor:
Alejandro Escudero Santana
Profesor Ayudante Doctor

Dep. Organización Industrial y Gestión de Empresas II
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2017

A mi madre.

Mi ejemplo a seguir.

Agradecimientos

Agradecer la ayuda recibida a lo largo de estos años es lo mínimo que puedo hacer por todos aquellos que han contribuido, directa o indirectamente, para que haya sido posible llegar al final de esta etapa.

En primer lugar, agradecer a mis padres su apoyo y paciencia infinita, solo comparable con el número de tuppers con los que me han cuidado durante todo este tiempo, y a mis hermanos, por el "sufrimiento" compartido y por todas esas horas estivales de biblioteca que acabaron dando sus frutos.

También me gustaría dar las gracias a mis amigos, tanto a los compañeros de fatigas como a los que no tienen nada que ver con este mundillo, por su ayuda, sus consejos y sus ánimos a la hora de superar las decepciones. En especial a Mari Paz, porque aun a cientos de kilómetros, ha sido un gran apoyo, creyendo en mí cuando ni yo misma lo hacía. Tarta de queso para celebrarlo.

Por último, agradecer a Don Alejandro Escudero Santana, por la oportunidad de trabajar con él, por guiarme durante la realización de este trabajo y por su absoluta disponibilidad para atenderme cada vez que ha sido necesario.

Mención aparte merece él. Gracias por aguantarme como solo tú puedes, por quererme como solo tú sabes. Por todos estos años empujándome hacia adelante. Fran, este logro también es tuyo.

Índice

Índice de Tablas

Índice de Figuras

1 INTRODUCCIÓN

Nunca andes por el camino trazado pues te conducirá a donde otros ya fueron.

- Alexander Graham Bell -

El sector del transporte se enfrenta en nuestros días a una serie de dificultades que hacen que este tenga que volverse cada vez más eficiente. Debido al incremento del precio de los combustibles, a la creciente globalización de los mercados y al objetivo de mejorar cada vez más el servicio al cliente, entre otras cosas, se hace necesario que el transporte de mercancías se realice de forma óptima para abaratar costes.

1.1 Motivación del trabajo

En la siguiente ilustración, Figura 1, se puede observar la gran importancia que tiene en nuestro país el transporte de mercancías por carretera. En otros países europeos la diferencia con los otros métodos no es tan abismal, pero igualmente el transporte por carretera representa un porcentaje importante de las toneladas distribuidas anualmente.

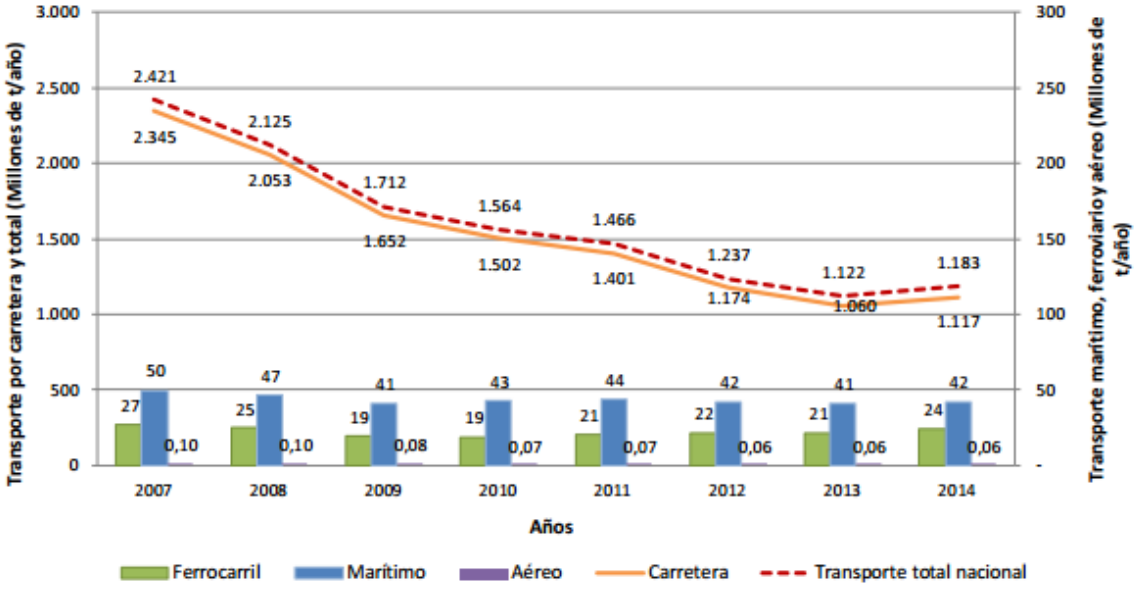

Figura 1: Evolución del transporte de mercancías en España (OTLE. Ministerio de Fomento)

Dentro de los costes logísticos de una empresa, los costes de transporte de mercancías constituyen una de las partes más importantes, ya que representan un porcentaje mayoritario de dichos costes. Esto se puede ver en la Figura 2:

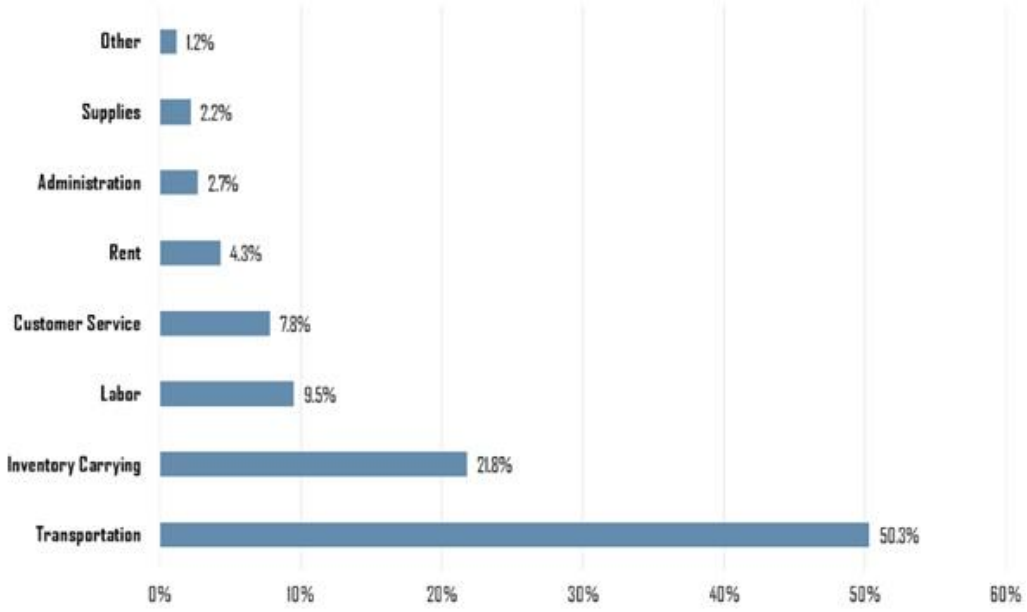

Figura 2: Porcentajes de costes logísticos en una empresa (Establish, Inc.)

A la vista de estos datos, se puede entender la necesidad de que las rutas de transporte tengan que ser cada vez mejor diseñadas, pues además, el entorno es cada vez más competitivo.

La resolución óptima del problema de rutas de transporte se trata de una tarea difícil, puesto que se encuentra dentro de los problemas de tipo NP-Hard. Cuando el tamaño de problema es pequeño, se pueden encontrar soluciones númericamente exactas en un tiempo razonable, pero debido a la complejidad de dicho problema, conforme el tamaño va creciendo, los tiempos se disparan y su resolución se vuelve inviable. Es en estos casos cuando se hace interesante el uso de técnicas heurísticas, ya que se pueden obtener soluciones cercanas al óptimo del problema en tiempos considerados aceptables.

1.2 Objetivos

En la realización de este trabajo se plantea como objetivo principal la minimización de los costes de transporte mediante la implementación de herramientas de optimización de rutas.

Como se puede ver en la Figura 3, debido al crecimiento del volumen de transporte se ha producido un aumento de las emisiones de gases invernaderos a la atmósfera, por lo que con la persecución de este objetivo no solo se conseguiría una reducción de costes, sino que también se reduciría el consumo de combustible y la mencionada emisión de gases, por el hecho de que los vehículos circularían durante un tiempo menor. Además, una adecuada planificación de rutas también se traduciría en un mejor servicio al cliente, por lo que se perfilan entonces como objetivos secundarios una reducción de la huella ecológica del transporte de mercancías por carretera y el aumento de la satisfacción de los clientes.

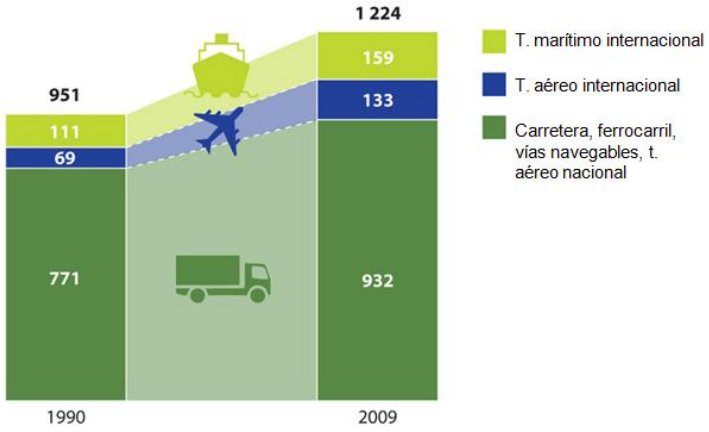

Figura 3: Gases de efecto invernadero emitidos por el transporte, EU-27, expresados en millones de toneladas equivalentes de CO_2 (Agencia Europea del Medio Ambiente)

1.3 Estructura del trabajo

Para la consecución de estos objetivos, los pasos a seguir durante el siguiente proyecto serán los descritos a continuación.

En primer lugar se hará una breve introducción al comercio electrónico y al problema de rutas de vehículos, con el fin de que se pueda comprender la importancia que tienen en la actualidad. El diseño de rutas objeto del presente trabajo está enfocado a este tipo de comercio.

Posteriormente, se desarrollará dicho problema de rutas de vehículos, explicando detalladamente en qué consiste. Se profundizará en el caso concreto del VRPTW (problema con ventanas temporales), ya que el problema a resolver pertenece a este tipo. Cada uno de los clientes contará con tres localizaciones posibles donde podrá ser visitado, siendo necesario solo visitar una de ellas. El cliente se encontrará un determinado intervalo de tiempo en cada una de las localizaciones, conociéndose dicho intervalo con el nombre de ventana temporal.

A continuación, se presentarán los tres grupos en los que se clasifican los métodos de resolución usados para hallar una solución a los problemas de rutas (métodos exactos, heurísticas y metaheurísticas), y se verán algunos de los ejemplos más utilizados para cada una de las categorías.

Tras esto, se detallarán las características de los problemas a resolver, especificándose todas aquellas consideraciones que se deban tener en cuenta y todas aquellas particularidades que presentan frente al problema clásico de VRPTW.

Para finalizar se procederá a la obtención de resultados y al análisis de estos, con el fin de obtener conclusiones acerca de cómo de ventajosas son las técnicas no exactas aplicadas en la resolución del problema y si son válidas para obtener soluciones cercanas al óptimo.

En resumen, mediante el uso de la informática, implementando una serie de heurísticas mediante un lenguaje de programación, se quiere intentar llegar a una solución aceptable del problema planteado que cumpla los objetivos marcados.

2 EL COMERCIO ELECTRÓNICO

Si tu negocio no está en internet, tu negocio no existe.

- Bill Gates -

La aparición de las nuevas tecnologías ha servido para favorecer cambios en muchos aspectos de la sociedad, siendo uno de ellos el comercio. Se puede hacer llegar los productos a un mayor número de consumidores potenciales, ya que las barreras del tiempo y de la distancia desaparecen, y aquellas empresas que saben aprovecharlo pueden aumentar enormemente su competitividad en un mercado cada vez más globalizado.

2.1 Introducción al comercio electrónico

El comercio electrónico, también conocido como e-commerce (electronic commerce en inglés), se ha convertido en nuestros días en una gran vía de negocio. Los consumidores se han familiarizado con esta nueva forma de realizar sus compras y gracias a la comodidad, al gran número de posibilidades que ofrece y al hecho de que los procedimientos de pago sean cada vez más seguros y diversos ha conseguido alcanzar un gran éxito.

Se define el comercio electrónico como cualquier actividad de intercambio comercial, tanto de productos físicos como virtuales, en la que las órdenes de compra, de venta y los pagos se realizan a través de un medio telemático. Se trata de una venta a distancia haciendo uso de las ventajas que proporciona la tecnología con la que se cuenta hoy en día.

Se distinguen tres tipos de comercio electrónico atendiendo a los participantes:

- Business to Business (B2B): se lleva a cabo entre empresas. Un ejemplo sería el aprovisionamiento de materiales de una empresa mediante un proveedor.
- Business to Consumer (B2C): se lleva a cabo entre una empresa y un consumidor. Un ejemplo sería la compra de un libro por un consumidor final a una empresa.
- Consumer to Consumer (C2C): se lleva a cabo entre consumidores. Un ejemplo de este tipo de comercio sería la compra-venta de productos de segunda mano directamente entre particulares.

2.2 Historia del comercio electrónico

El comercio electrónico no es más que una fase a la que se ha llegado durante el proceso de desarrollo del comercio.

Se puede considerar que el nacimiento del comercio electrónico tuvo lugar hacia el año 1920 en Estados Unidos, cuando apareció la venta por catálogo impulsada por los grandes mayoristas. Este sistema supuso una gran revolución en aquel momento, puesto que fue la primera forma de comprar sin necesidad de ver antes el producto de forma física. En su lugar se mostraba un catálogo con fotos ilustrativas de los productos a vender. Este tipo de venta permitió a las empresas captar cuotas del mercado a las que antes no podían acceder, por ejemplo zonas rurales no demasiado comunicadas, ya que no existía la necesidad de que los clientes acudieran a los locales de venta. Además estos podían elegir los productos tranquilamente en sus hogares, sin la presión de un vendedor.

En el año 1960 se originó en Estados Unidos una importante forma de intercambio de datos de manera electrónica, el EDI (Electronic Data Interchange). Este sistema permitía a las empresas la realización de transacciones electrónicas e intercambio de información comercial, por lo que se puede decir que la historia del e-commerce comienza en esta fecha, aunque el término como tal no fuera usado hasta muchos años después.

Sin embargo, no fue hasta el año 1970 cuando aparecieron las primeras relaciones comerciales que hacían uso de un ordenador para la transmisión de datos (como facturas y órdenes de compras), aunque aún no habían aparecido los ordenadores tal y como se conocen hoy en día, por lo que ofrecían un servicio muy limitado. A pesar de dichas limitaciones, este hecho supuso una mejora de los procesos de fabricación de empresas de un mismo sector en el ámbito privado. En esta década, gracias a la transferencia electrónica de fondos monetarios a través de redes de seguridad privadas de las instituciones financieras, se expandió el uso de las nuevas tecnologías de telecomunicación con fines comerciales, lo que permitió el intercambio entre ordenadores de información financiera, más específicamente, la transferencia de pagos y giros.

Fue en 1979 cuando el empresario inglés Michael Aldrich conectó una televisión y un ordenador mediante las líneas telefónicas, habilitando las transacciones comerciales entre empresas o entre una empresa y un consumidor, iniciando así la venta online, por lo que es considerado el padre del comercio electrónico.

A mediados de 1980, con la ayuda de la televisión, se modernizó la venta por catálogo con la aparición de las televentas. En ellas se exhibía el producto, resaltando sus características más importantes y consiguiendo un mayor realismo que en la venta por catálogo. La compra del producto se llevaba a cabo mediante una llamada telefónica y el pago se realizaba usualmente mediante tarjeta de crédito, lo que proporcionaba un gran anonimato al comprador.

En 1981 apareció el primer modelo B2B (Business to Businnes) de compras online, en 1982 la compañía francesa de telecomunicaciones PTT inventó Minitel, considerado el servicio en línea pre-WWW más destacado y donde los usuarios podían hacer compras en línea. En 1984 la empresa Tesco lanzó la primera plataforma B2C (Business to Consumer), introduciendo el concepto de cesta de la compra online.

A principios de los años 90 se produjo en el mundo un cambio radical en la forma de comunicación y comercialización debido a la aparición de la World Wide Web (WWW), creada por el inglés Tim Berners-Lee. Supuso un gran avance por su alto nivel de accesibilidad, ya que no era necesario un alto conocimiento de la informática por sus usuarios. Con Internet funcionando, el e-commerce tuvo un crecimiento muy destacado y se crearon plataformas dedicadas exclusivamente a esta actividad, como Amazon e eBay,que siguen operativos y en crecimiento en nuestros días.

En el año 1995 se produjo la consolidación del comercio electrónico. Por iniciativa de los integrantes del G8, se creó un mercado global para Pymes, cuyo propósito era consolidar el uso del e-commerce entre las empresas de todo el mundo, dando buenos resultados.

Tras la aparición de Google y Paypal en el año 1998, se potenció el comercio electrónico entre los consumidores y se creó un nuevo hábito de consumo, que sigue evolucionando a día de hoy. Sin duda, se puede decir que la razón principal por la que el comercio electrónico ha tenido tanto éxito es por la facilidad con la que se puede llevar a cabo, y acabará por imponerse en un futuro no muy lejano.

2.3 Ventajas y desventajas del comercio electrónico

El comercio electrónico presenta un gran número de ventajas, tanto para las empresas como para los consumidores. Algunas de ellas son las siguientes:

- **Desaparecen los límites geográficos y temporales.** La clientela potencial es mucho más amplia que la de un comercio tradicional, ya que mediante Internet se pueden captar clientes de cualquier lugar del mundo.
- **Es cómodo.** El consumidor puede acceder de forma cómoda y sencilla, y desde cualquier parte (casa, lugar de trabajo, etc) a un amplio catálogo de productos y servicios. Tan solo necesita una conexión a Internet y una tarjeta de crédito para la realización del pago.
- **Gran variedad de productos.** Se le presenta al consumidor la oportunidad de adquirir artículos que no están disponibles de forma física en el área geográfica donde vive, ya que mediante Internet puede acceder a infinidad de tiendas virtuales repartidas a lo largo del planeta.
- **Gran variedad de ofertas.** El consumidor se encuentra ante una mayor y diversa oferta del mismo producto o servicio, puede comparar precio y calidad entre un elevado número de vendedores, que pueden estar situados en cualquier parte del mundo.
- **Menores precios.** Al no haber intermediarios, locales físicos ni personal de cara al público, el consumidor puede adquirir el producto a un precio más competitivo que en un comercio tradicional.
- **Mayores márgenes.** Las empresas disponen de un mayor margen de beneficio sin necesidad de incrementar el precio del producto final por el mismo motivo de no existencia de intermediarios, local físico, etc.
- **Búsqueda más fácil de proveedores.** Mediante el comercio electrónico las empresas pueden ponerse en contacto con gran cantidad de proveedores, por lo que el tiempo de búsqueda de estos es menor.
- **Eliminación de intermediarios.** Se agiliza la cadena de suministro y todo el proceso logístico gracias a los mercados electrónicos entre empresas, pudiendo dar lugar este hecho a una reducción de costes.
- **Menor coste de transacciones.** La tramitación en línea de pagos, facturas y pedidos resulta un procesamiento más eficiente en el comercio electrónico entre empresas.
- **Transparencia de precios.** Debido a la competitividad del mercado, los proveedores necesitan detallar los precios y cómo llevan a cabo el procesamiento de las transacciones, teniendo el comprador la oportunidad de comparar precios y servicios en el momento de tomar decisiones de compra.

En contraposición, también presenta ciertos inconvenientes, algunos de los cuales han limitado enormemente su desarrollo, como por ejemplo:

- **Inseguridad en las transacciones.** Aunque hoy en día existen pasarelas de pago online homologadas por los protocolos de Comercio Electrónico Seguro (CES), la presencia del fraude electrónico sigue provocando desconfianza entre los compradores, preocupados por el robo o uso fraudulento de información sensible, como datos personales o número de tarjeta bancaria.
- **Dificultad de marketing.** Al existir tal cantidad de información en Internet, es difícil dar a conocer una empresa si no se diseña de forma adecuada el plan de marketing. Es necesario presentar los productos y servicios ofrecidos de forma que se consiga captar un número de clientes fijos para asegurar la subsistencia de la empresa en Internet.
- **Marco jurídico y fiscalidad.** Cuando se produce un conflicto debido a una transacción entre países distintos se plantean cuestiones legales difíciles de resolver, como por ejemplo qué leyes se tienen que aplicar, cómo imponer el cumplimiento de la decisión, etc.

- **Genera desconfianza.** Los consumidores no tienen la capacidad de visualizar el producto, no pueden tener un conocimiento físico de él al no estar en contacto directo.
- **Quejas o devoluciones.** La distancia entre vendedor y cliente puede resultar un inconveniente a la hora de realizar estos trámites o de hacer valer la garantía de un producto. Se hace necesaria una buena atención al cliente por canales electrónicos.
- **Retos logísticos.** Se han llevado a cabo modelos específicos logísticos para el e-commerce, ya que se ha de gestionar de forma eficiente una gran cantidad de información. Al ser el comercio electrónico considerado más ágil que el comercio tradicional, se espera de él un menor tiempo de distribución.

3 PROBLEMA DE RUTEO DE VEHÍCULOS (VRP)

Ningún conocimiento humano puede ir más allá de su experiencia.

- John Locke -

En el siguiente capítulo se va a hacer una introducción al problema del rutado de vehículos, describiendo sus principales características y elementos implicados. También se hará un repaso histórico acerca de las variantes del problema, se plantearán brevemente otros problemas similares y se desarrollará en profundidad el problema de rutado de vehículos con ventanas temporales.

3.1 Introducción al VRP

El problema de planificación de rutas de vehículos (*Vehicle Routing Problem* o VRP), se trata de un problema de optimización combinatoria muy importante en diferentes entornos logísticos, ya que cada vez es más difícil conseguir una diferenciación en el mercado a través únicamente del producto ofertado, por tanto es en dicho entorno logístico donde se pueden lograr ventajas competitivas.

"El problema de distribuir productos desde ciertos depósitos a sus usuarios finales juega un papel central en la gestión de algunos sistemas logísticos, y su adecuada planificación puede significar considerables ahorros. Esos potenciales ahorros justifican en gran medida la utilización de técnicas de investigación operativa como facilitadoras de la planificación, dado que se estima que los costos del transporte representan entre el 10% y el 20% del costo final de los bienes" (Toth & Vigo 2001).

El objetivo del VRP consiste en hallar las rutas que deben recorrer cada uno de los vehículos (que parten de un depósito) de manera que se consigan satisfacer los pedidos de una serie de clientes (que se sitúan de forma dispersa geográficamente), las restricciones operativas y se minimice el coste total del transporte.

Durante los últimos años, debido al desarrollo de la sociedad actual, se ha producido un incremento de la necesidad del transporte de mercancías. Los problemas de rutado de vehículos se han desarrollado a gran velocidad y han sido aplicados en una gran cantidad de sectores, tanto logísticos como de telecomunicaciones, planificación y control de la producción, dirección de proyectos, etc.

Debido al gran número de aplicaciones industriales que posee, ha sido un problema estudiado enormemente, produciéndose una evolución constante en la calidad de las metodologías resolutivas usadas, tanto en las númericamente exactas como en las heurísticas.

Al ser un problema de tipo NP-Hard, resolverlo de manera exacta se vuelve imposible para tamaños grandes (a partir de 50 clientes), por lo que se requiere la utilización de otros métodos aproximados para conseguir una solución cercana al óptimo en un tiempo aceptable, las denominadas heurísticas y metaheurísticas.

3.1.1 Principales características y elementos implicados

En el problema VRP más sencillo se tienen en cuenta los siguientes puntos:

- Existe un único depósito central.
- El depósito cuenta con una flota de vehículos homogénea.
- Se debe atender a una serie de clientes dispersos de forma geográfica.
- Las demandas de cada cliente son conocidas.
- Las rutas se inician en el depósito y terminan en él.
- Los vehículos tienen una capacidad máxima que no se puede exceder.
- Los clientes solo serán visitados una única vez. Se tiene que tener en cuenta por tanto que la capacidad del vehículo que sirva al cliente sea superior a la cantidad total demandada.

El objetivo que se busca es el de diseñar un número de rutas para servir a todos los clientes, minimizando el coste, esto es, minimizando el número de vehículos usados y el tiempo que estos pasan circulando.

En la Figura 4 se puede observar gráficamente y de forma sencilla el funcionamiento de dicho problema.

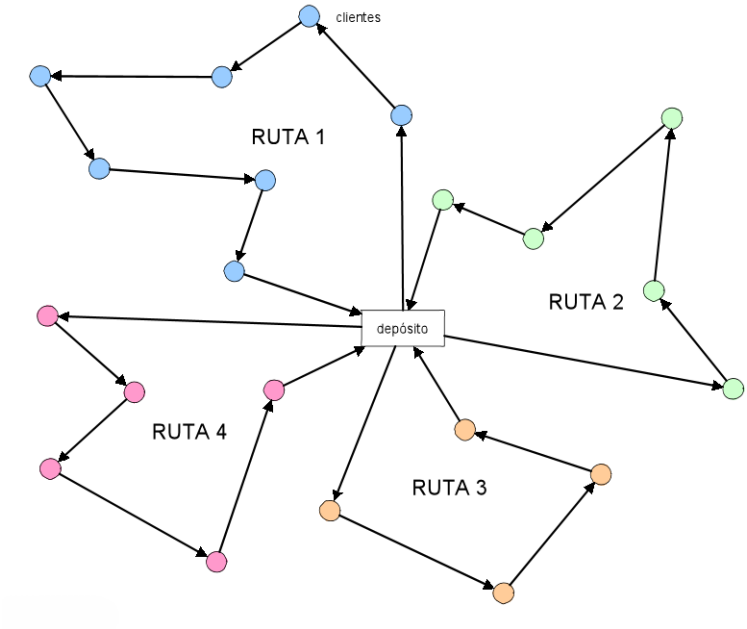

Figura 4: Representación genérica del VRP

Se llama ruta a un ciclo simple con origen y destino en el depósito del problema, y representa la secuencia de visitas llevada a cabo por el vehículo al que pertenece la ruta. El coste total de una ruta se puede calcular como la suma de los costes de los arcos que forman dicha ruta. Se procede de la misma forma para calcular el tiempo de una ruta.

Se describen a continuación los elementos principales de los problemas tipo VRP:

1. **La red de transporte:** se considera generalmente una red terrestre, por ser la más habitual, aunque también se puede considerar una red marímita, aérea o una combinación de redes en problemas en los que se usen distintos tipos de transporte. Se describe la red como un grafo donde los arcos representan las vías y los vértices se corresponden con los nodos de la red, donde se encuentran situados los clientes o los depósitos. Se puede tener un grafo dirigido o no, dependiendo de si se permite la circulación en uno o en ambos sentidos.

Se asocia a cada arco un coste, que puede representar la longitud de la distancia, el tiempo de viaje o el coste monetario de realizar el mismo. Este puede depender de diversos factores, como el tipo de vehículo o el momento en el que se recorra el arco.

2. **La flota de vehículos:** se puede conocer el número de vehículos disponibles o puede ser una variable de decisión. Los vehículos suelen tener una capacidad limitada, que se puede expresar, por ejemplo, mediante peso y volumen. Cuando todos los vehículos comparten los mismos atributos se habla de flota homogénea, mientras que si hay diferencias entre ellos se trata de una flota heterogénea. Se puede asociar un coste fijo a cada vehículo por el hecho de usarlo y un coste variable proporcional a la distancia recorrida.

   Generalmente, cada vehículo recorre una sola ruta durante el horizonte temporal del problema, durante la cual tendrá que ir sirviendo a los clientes que formen parte de dicha ruta. Se recomienda el equilibrio de carga de trabajo de los conductores.

3. **Los clientes y/o proveedores:** se entiende la demanda como un producto que ocupa un volumen en un vehículo o como un servicio a realizar que dura cierto tiempo. Cada cliente tiene una demanda que se debe satisfacer por un solo vehículo. Se puede requerir la entrega de un producto a los clientes o la recogida de mercancía para transportarla hasta el depósito.

   Se puede dar el caso de que se deban realizar ciertas entregas a los clientes, pero la mercancía no esté inicialmente en el depósito. Se hace necesario entonces visitar primeramente a los proveedores y después a los clientes.

   Existen muchas otras variantes del problema, por ejemplo el caso en el que la demanda de los clientes puede ser satisfecha por varios vehículos diferentes, aquel en el que los clientes solo pueden ser servidos durante un intervalo de tiempo o la variante donde existen restricciones de compatibilidad entre los clientes y los vehículos.

4. **El depósito central (o depósitos):** los vehículos y las mercancías a distribuir se suelen situar inicialmente en depósitos. Se suele imponer que cada ruta tenga su inicio y su fin en un mismo depósito, aunque hay casos en los que también se permiten rutas entre depósitos diferentes.

   En los problemas con varios depósitos, cada uno de ellos puede tener una situación geográfica diferente y tener una capacidad máxima. Cada depósito puede tener asignada una flota vehículos determinada o puede que dicha asignación necesite ser determinada.

   Al igual que en el caso de los clientes, se puede dar el hecho de que los depósitos tengan ventanas de tiempo asociadas.

5. **Las rutas solución:** en este tipo de problemas se tratan de determinar las rutas para cada uno de los vehículos, de manera que se cumplan todas las restricciones y se consigan los objetivos propuestos. Se suelen tomar como objetivo la minimización de los costes, del número de los vehículos totales, del tiempo total de transporte, de la distancia recorrida, etc.

   En general se asume que un vehículo, durante el período de planificación, solo realizará una ruta, pero hay variantes del problema en las que esto no es así y el mismo vehículo puede participar en más de una ruta.

3.2 Variantes del VRP

El problema VRP tuvo sus inicios en el año 1959, cuando Dantzing y Ramser modelaron la entrega de combustible a diferentes estaciones de servicio mediante una flota de camiones, partiendo desde una terminal.

Según se han ido modificando los requisitos logísticos, el problema VRP ha ido evolucionando. Se describen a continuación algunas de las variantes más conocidas.

3.2.1 Problema con restricciones de capacidad (CVRP)

El problema CVRP (*Capacitated VRP*) se trata de una variante del VRP donde uno o varios vehículos con capacidad limitada y constante deben encargarse de servir a un número de clientes desde un depósito. Se conoce la demanda de cada cliente, la cual no debe superar nunca la capacidad del vehículo encargado de servir a dicho cliente. El objetivo es el de minimizar el número de vehículos y la suma de los tiempos de las rutas.

3.2.2 Problema con ventanas temporales (VRPTW)

El VRPTW (*VRP with Time Windows*) se trata de un problema VRP con una restricción adicional: cada cliente tiene asociada una ventana temporal dentro de la cual tendrá que ser atendido. El depósito desde el que parten los vehículos también dispone de una ventana temporal. Como objetivo se plantea la minimización del número de vehículos, de la suma de los tiempos de las rutas y del tiempo de espera para servir a los clientes en sus ventanas temporales.

En este tipo de problemas se plantean unas restricciones adicionales:

- Un cliente no puede ser atendido después del fin de su ventana temporal.
- Un vehículo que llegue a un cliente antes de que comience su ventana temporal tendrá que esperar para poder servirlo.
- Las rutas deben empezar y terminar dentro de la ventana temporal del depósito.

Hay casos en los que se relajan las restricciones de cumplimiento de las ventanas temporales y se permite la llegada tardía para servir a un cliente, pero se penaliza la solución aumentando el valor de la función objetivo.

3.2.3 Problema con múltiples depósitos (MDVRP)

El MDVRP (*Multiple Depot VRP*) se trata de un problema VRP donde se dispone de varios depósitos para atender a los clientes. Si se pudiera asignar a los clientes a depósitos concretos se podría modelar el problema como varios problemas VRP independientes, pero en este caso no es posible, ya que los clientes se encuentran entremezclados con los depósitos.

En este tipo de problemas se requiere la asignación de cada cliente a un depósito. Cada depósito dispone de su flota particular de vehículos, que deben salir de dicho depósito, servir a un cliente asociado al mismo depósito y retornar a su depósito correspondiente. Al igual que en casos anteriores, el objetivo se trata de minimizar el número de vehículos a usar y la distancia de las rutas de los mismos.

3.2.4 Problema con entregas y devoluciones (VRPPD)

El VRPPD (*VRP with Pickup and Delivery*) se trata de una variante del VRP donde existe la posibilidad de que un cliente que ha sido servido disponga de una mercancía que necesita ser recogida. Se debe tener especial cuidado ya que los clientes introducen carga en los vehículos y la capacidad de estos no puede ser superada. Se dificulta mucho el problema, por lo que se suele impedir el intercambio de mercancía entre clientes, todos los

envíos comienzan en el depósito y todas las devoluciones vuelven a él.

Se puede relajar la restricción de que todos los clientes tengan que ser visitados al menos una vez, o se puede considerar que, antes de comenzar con las devoluciones, cada vehículo debe haber entregado toda la mercancía.

Se busca como objetivo la minimización del número de vehículos usados y de la suma de los tiempos de transporte, y para que la solución sea factible se debe cumplir la restricción de que cada vehículo debe disponer de capacidad suficiente para llevar la mercancía a entregar y poder recoger las devoluciones.

3.2.5 Problema con entregas parciales (SDVRP)

El SDVRP (*Split Delivery VRP*) se trata de una relajación del VRP en donde se permite que un mismo cliente sea servido por diferentes vehículos, siempre y cuando se vea reducido el coste total. Esta variación es muy interesante cuando la demanda de un cliente es mayor que la capacidad de los vehículos.

Se busca como objetivo la minimización del número de vehículos usados y de la suma de los tiempos de transporte.

3.2.6 Problema con valores al azar (SVRP)

El SVRP (*Stochastic VRP*) se trata de un problema VRP en el que uno o varios componentes del problema (número de clientes, tiempo de servicio, tiempo de recorrido, etc) son aleatorios. Se pueden encontrar los siguientes tipos de SVRP:

- Clientes estocásticos: cada cliente v_i tiene una probabilidad p_i de estar presente y una probabilidad (1-p_i) de encontrase ausente.
- Demandas estocásticas: la demanda d_i de cada cliente es un una variable aleatoria.
- Tiempos estocásticos: los tiempos de transporte tt_i y los tiempos de servicio ts_i son variables aleatorias.

En este tipo de problemas se lleva a cabo la realización de dos etapas para obtener una solución. En la primera etapa se determina una solución antes de conocer el valor de las variables. En la segunda, una vez conocidos los valores de las variables, se lleva a cabo una acción correctiva.
Se busca como objetivo la minimización del número de vehículos usados y de la suma de los tiempos de transporte.

3.2.7 Problema periódico VRP (PVRP)

El PVRP (*Periodic VRP*) se diferencia del problema VRP en el periodo de planificación. Mientras que en el VRP este es de un único día, en el PVRP se extiende el horizonte de planificación hasta *M* días.

Para que una solución sea factible, se tienen que cumplir todas las restricciones del problema, durante el periodo de planificación cada cliente debe ser visitado al menos una vez y no es necesario que los vehículos vuelvan al depósito en el mismo día en el que salieron de él, sino que pueden hacerlo en cualquiera de los días del horizonte temporal.

Igualmente, se busca como objetivo la minimización del número de vehículos usados y de la suma de los tiempos de transporte para servir a los clientes.

3.2.8 Problema con viajes de regreso (VRPB)

El VRPB (*VRP with Backhauls*) se trata de una modificación del VRP donde los clientes pueden ser servidos o entregar productos. Se divide el conjunto de clientes en dos, el primero se corresponde con los clientes que tienen que ser servidos y en el segundo se encuentran aquellos donde una mercancía tiene que ser recogida. En

cada ruta, primero se tiene que entregar la mercancía y después se realizarán las recogidas, y las cantidades a ser servidas y recogidas son fijas y conocidas con antelación.

Para que la solución sea factible se requiere que la capacidad de los vehículos al recoger mercancía no sea excedida y que cada cliente sea visitado por una sola ruta. Como objetivo se busca un conjunto de rutas que minimicen la distancia total recorrida.

3.2.9 Problema con instalaciones satélites (VRPSF)

El VRPSF (*VRP with Satellite Facilities*) se trata de un problema VRP en el que se permite el reabastecimiento de los vehículos sin necesidad de que vuelvan al depósito. Se dispone de instalaciones satélites donde los vehículos pueden reponer su carga y continuar con las entregas hasta el final de su turno sin necesidad de volver al depósito inicial.

3.2.10 Otros problemas similares

Los problemas de rutas se pueden dividir en dos grandes categorías: una en la cual los clientes se encuentran sobre los arcos y otra en la que se encuentran en los nodos. En la primera de ellas, la ruta óptima que se busca tiene que recorrer todos los arcos del grafo, mientras que en la segunda, se debe visitar todos los nodos que forman parte del problema. Dicho de otra forma, en los problemas sobre arcos se asemejan los arcos con calles que se tienen que visitar, mientras que en los problemas sobre nodos se considera que cada nodo representa un cliente.

El origen de los problemas de rutas sobre arcos tiene lugar en el siglo XVIII, cuando los habitantes del pueblo ruso de Königsberg, actual Kaliningrado, se preguntaron si sería posible el trazado de una ruta que atravesase de forma única los 7 puentes que unían las distintas partes en las que se dividía la población, debido al paso del río Pregel, y volviera al punto de partida (Figura 5). En el año 1736, el matemático suizo Leonhard Euler resolvió la cuestión, demostrando que no era posible. A pesar de ello, se extrajeron interesantes conclusiones para su aplicación en problemas posteriores.

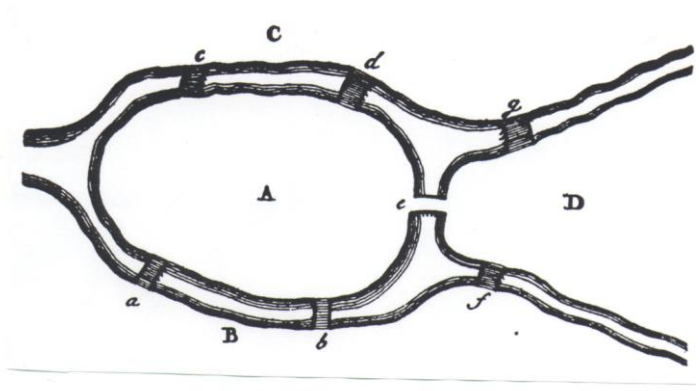

Figura 5: Los puentes de Königsberg

El origen de los problemas de rutas sobre nodos tiene lugar en el siglo XIX, con la aparición del "Icosian Game" (Figura 6), juego inventado por el británico T. Kirkman y el irlandés W.R. Hamilton. El objetivo del juego era el de encontrar una ruta entre los 20 puntos que lo formaban, haciendo uso únicamente de los caminos permitidos, volviendo al nodo de partida. Es obvio que el juego no tenía como finalidad la búsqueda del camino óptimo, bastaba con encuentrar una ruta que visitase todos los nodos tan solo una vez.

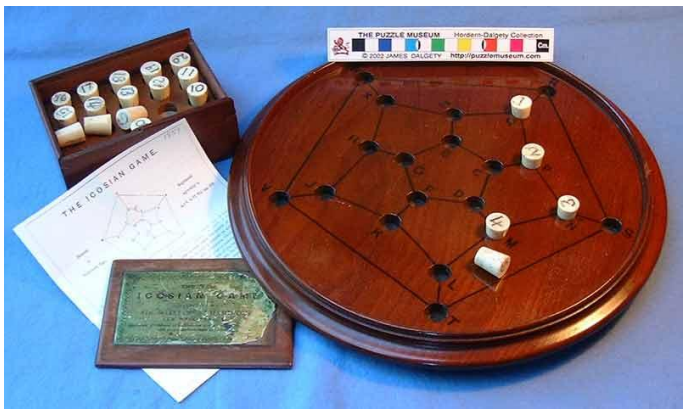

Figura 6: Icosian Game

A continuación se muestra un ejemplo de problema para cada una de estas categorías. Se hablará del problema del cartero chino en el caso de los problemas sobre arcos y del problema del viajante de comercio en el de los problemas sobre nodos.

3.2.10.1 Problema del cartero chino (CPP)

Este problema fue definido en el año 1962 por Meigu Guan. Su objetivo es el de hallar una ruta, cuyo coste sea mínimo, que recorra al menos una vez todos los arcos que forman parte del grafo, siendo el nodo de partida y el de finalización el mismo.

Recibe este nombre porque lo que se planteaba era el problema al que se tenía que enfrentar un cartero a la hora de repartir la correspondencia, buscando recorrer la menor distancia posible. Si se tiene una zona determinada de una ciudad, representada a través de un grafo, el cartero a lo largo de su recorrido debe recorrer cada arista de dicho grafo al menos una vez. A cada una de estas aristas se le asocia un peso, dependiendo este de la longitud que presente, para poder determinar el recorrido de coste mínimo.

3.2.10.2 Problema del viajante de comercio (TSP)

Históricamente, se ha entendido el problema de rutas de vehículos como una generalización del problema del viajante de comercio, por lo que se va describir brevemente en qué consiste el TSP. Este problema es, además, base de una gran parte de los problemas de distribución física de productos.

Los orígenes desde el punto de vista del modelado matemático de este problema se remotan a los inicios de la década de 1930. En dicho problema, un agente de ventas debe visitar una cierta cantidad de ciudades en un único viaje. Comienza desde su cuidad de residencia y debe determinar la ruta a seguir para que cada ciudad sea visitada en una única ocasión, debiendo regresar al final a su ciudad, de forma que la longitud total del viaje sea la mínima posible.

Se puede modelar como un grafo formado por N nodos unidos por arcos, considerándose los nodos como los clientes a visitar y los arcos como los trayectos para ir de unos a otros. Conociendo el coste de los arcos que unen los clientes, se tiene como objetivo encontrar el orden de visita a las ciudades que minimice el valor de la función objetivo.

El problema del viajante de comercio, TSP (de sus siglas en inglés *Travelling Salesman Problem*), se trata de uno de los problemas clásicos de optimización combinatoria y se usa en numerosos métodos de optimización. A pesar de su aparente sencillez, se trata de un problema complejo computacionalmente. Esto no ha sido inconveniente para que se hayan desarrollado en los últimos años un gran número de métodos exactos y heurísticas para darle solución, pudiendo resolverse incluso para una gran cantidad de ciudades.

Entre las diversas aplicaciones del TSP se encuentran la logística (planificación de rutas), la industrial (secuenciación de operaciones) o la genética (secuenciación de genes).

Para este proyecto, que se centra en el desarrollo de rutas de transporte con aplicaciones para el comercio

electrónico, la variante del problema del viajante de comercio que será interesante es la variante múltiple (m-TSP) en su forma simétrica (la distancia para ir del nodo i al nodo j es la misma que la de ir del nodo j al nodo i) y con ventanas temporales.

El m-TSP se trata de una generalización del problema del viajante de comercio donde se tiene un déposito (ciudad de partida) y un número m de vehículos (número de vendedores). Se tiene como objetivo el diseño de una ruta para cada uno de los vendedores, de forma que cada ciudad sea visitada únicamente una vez por uno de los vendedores. Todas las rutas deben partir y finalizar en la ciudad de origen.

A la hora del modelo matemático, se define un coste c_{ij} que representa el coste de ir de la ciudad i a la ciudad j. Al tratarse de la forma simétrica, se tiene que $c_{ij} = c_{ji}$. Este coste se tiene en cuenta en la función objetivo cuando el agente de ventas lleva a cabo el trayecto entre los nodos i y j.

Tras añadir las ventanas temporales, se obtiene el Symmetric Multiple Travelling Salesman Problem with Time Windows (m-TSPTW), que presenta una formulación similar al problema que se pretende dar solución, el VRPTW.

3.3 Modelado matemático del VRP

3.3.1 Modelo matemático del CVRP

A continuación se va a presentar el modelo matemático para el problema general del ruteo de vehículos.

Se tiene como objetivo principal la minimización del coste del recorrido de los vehículos. Se calcula un número determinado de vehículos con una capacidad conocida y homogénea, que parten desde un depósito y tienen que servir a todos los clientes, cuya demanda es conocida. Cada cliente solo puede ser visitado por un único vehículo. Cada vehículo se corresponde con una ruta.

1. Índices del problema.

   i: nodo de partida del vehículo.

   j: nodo de llegada del vehículo.

   k: vehículo del conjunto K.

   0: nodo correspondiente al depósito.

2. Parámetros del problema.

   V: conjunto de nodos del problema. Cada nodo representa a un cliente. El nodo 0 representa al depósito.

   E: conjunto de aristas del problema. Cada arista representa la distancia entre dos nodos.

   K: conjunto suficientemente grande de vehículos para servir a todos los clientes.

   *C:*capacidad de cada vehículo. La flota es homogénea.

   c_{ij}: coste de ir desde el nodo i el nodo j.

   $\Delta^+(i)$: conjunto de nodos formado por los nodos a los que llegan las aristas que parten del nodo i. $\Delta^+(i) = \{j \in V |\ (\mathrm{i,j}) \in \mathrm{E}\}$.

   $\Delta^-(i)$: conjunto de nodos formado por los nodos de los que parten las aristas que llegan al nodo i. $\Delta^-(i) = \{j \in V |\ (\mathrm{j,i}) \in \mathrm{E}\}$.

d_i: demanda asociada al cliente i.

$d(S)$: demanda total del conjunto de clientes.

3. Variables del problema.

m: número de vehículos usados en la solución.

$$x_{ij} = \begin{cases} 1, & si\ se\ va\ desde\ el\ nodo\ i\ al\ nodo\ j \\ 0, & en\ cualquier\ otro\ caso \end{cases}$$

$$x_{0j} = \begin{cases} 1, & si\ se\ va\ desde\ el\ depósito\ al\ nodo\ j \\ 0, & en\ cualquier\ otro\ caso \end{cases}$$

$$x_{i0} = \begin{cases} 1, & si\ se\ va\ desde\ el\ nodo\ i\ al\ depósito \\ 0, & en\ cualquier\ otro\ caso \end{cases}$$

$$y_k = \begin{cases} 1, & si\ ha\ sido\ usado\ el\ vehículo\ k \\ 0, & en\ cualquier\ otro\ caso \end{cases}$$

$$x_{ik} = \begin{cases} 1, & si\ el\ cliente\ i\ es\ servido\ por\ el\ vehículo\ k \\ 0, & en\ cualquier\ otro\ caso \end{cases}$$

$r(S)$: número mínimo de vehículos necesarios para poder servir a todos los clientes.

4. Modelo del problema.

$$\min \sum_{(i,j)\in E} c_{ij}x_{ij} \qquad (1)$$

s.a:

$$\sum_{j\in\Delta+(0)} x_{0j} = m \qquad (2)$$

$$\sum_{i\in\Delta-(0)} x_{i0} = m \qquad (3)$$

$$\sum_{j\in\Delta+(i)} x_{ij} = 1 \qquad \forall\, i \in V\setminus\{0\} \qquad (4)$$

$$\sum_{i\in\Delta-(i)} x_{ij} = 1 \qquad \forall\, j \in V\setminus\{0\} \qquad (5)$$

$$\sum_{i\in S, j\in\Delta+(i)\setminus S} x_{ij} \geq r(S) \qquad \forall\, S \subset V\setminus\{0\} \qquad (6)$$

$$m \geq 1$$

$$x_{ij}, x_{0j}, x_{i0} \in \{0,1\} \qquad \forall (i,j) \in E$$

La función objetivo (1) representa el coste total de la solución. Las restricciones (2) y (3) hacen referencia a que m es el número de vehículos utilizados para servir a todos los clientes, y que además, todos los vehículos parten del depósito y regresan a él al finalizar la ruta. Las restricciones (4) y (5) sirven para asegurar que todos los clientes son nodos intermedios de alguna ruta, es decir, que tienen un cliente que les precede y otro que les sucede. La restricción (6) sirve para eliminar los sub-tours (todo subconjunto de nodos S tiene que ser abandonado al menos una vez) y para asegurar que la demanda total de los clientes pertenecientes a una ruta no supera la capacidad C de un vehículo.

Para determinar el valor de $r(S)$ se requiere resolver el siguiente problema:

$$r(S) = \min \sum_{k \in K} y_k$$

s.a:

$$\sum_{i \in S} d_i x_{ik} \leq C y_k \qquad \forall\, k \in K$$

$$\sum_{k \in K} x_{ik} = 1 \qquad \forall\, i \in S$$

$$x_{ik} \in \{0,1\} \qquad \forall\, i \in S, \forall\, k \in K$$

$$y_k \in \{0,1\} \qquad \forall\, k \in K$$

donde, como ya se ha dicho previamente, K es un conjunto suficientemente grande de vehículos para servir la demanda de todos los clientes (se podría tomar, por ejemplo, tantos vehículos como clientes haya). Se necesita establecer una cota inferior para el número de vehículos y esta viene dada por el cociente $\left\lceil \frac{d(S)}{C} \right\rceil$.

En esta forma de expresar el modelo, m es una variable que no tiene cota superior, es decir, se considera que se tienen disponibles un número ilimitado de vehículos. Si se estuviera en el caso de que se dispusiera de una flota finita, se agregaría una cota superior para m o se fijaría directamente su valor.

3.3.2 Modelo matemático del VRPTW

Como se ha comentado previamente, el VRPTW (*VRP with Time Windows*) se trata de una variante del problema VRP, al que se le añade una restricción adicional: cada cliente tiene asociada una ventana temporal dentro de la cual tendrá que ser atendido. El depósito desde el que parten los vehículos también dispone de una ventana temporal. El objetivo que se plantea es el de encontrar para cada vehículo la ruta que tiene que llevar a cabo de forma que el coste sea mínimo.

En este tipo de problemas se plantean unas restricciones adicionales:

- Un cliente no puede ser atendido después del fin de su ventana temporal.
- Un vehículo que llegue a un cliente antes de que comience su ventana temporal tendrá que esperar para poder servirlo.
- Las rutas deben empezar y terminar dentro de la ventana temporal del depósito.

Hay casos en los que se relajan las restricciones de cumplimiento de las ventanas temporales y se permite la llegada tardía para servir a un cliente, pero se penaliza la solución aumentando el valor de la función objetivo.

Además, cada cliente tiene asociado un tiempo de servicio, que se define como el tiempo que tarda en realizarse la entrega. Para que el cliente pueda ser atendido dentro de su ventana temporal, se tiene que tener en cuenta que sumando al instante de tiempo en el que llega el vehículo al cliente el tiempo de servicio de dicho

cliente, se tiene que obtener un valor menor o igual que el instante de tiempo en el que finaliza la ventana temporal del cliente.

Al igual que en el caso del VRP, los vehículos parten del depósito y tienen que acabar en él al final de su ruta correspondiente.

A continuación se va a presentar el modelo matemático para el problema del ruteo de vehículos con ventanas temporales.

Se tiene como objetivo principal la minimización del coste del recorrido de los vehículos. Se considera una flota heterogénea de vehículos con una capacidad conocida, que parten desde un depósito y tienen que servir a todos los clientes, cuya demanda también es conocida. Cada cliente solo puede ser visitado por un único vehículo. Cada vehículo se corresponde con una ruta.

1. Índices del problema.

   i: nodo de partida del vehículo.

   j: nodo de llegada del vehículo.

   k: vehículo del conjunto K.

   $0\ y\ n+1$: nodos correspondientes al depósito.

2. Parámetros del problema.

   V: conjunto de nodos del problema. Cada nodo representa a un cliente. Los nodos 0 y $n+1$ representan al depósito.

   E: conjunto de aristas del problema. Cada arista representa la distancia entre dos nodos.

   K: conjunto suficientemente grande de vehículos para servir a todos los clientes.

   c_{ij}^{k}: coste de ir el vehículo k desde el nodo i al nodo j.

   $\Delta^{+}(i)$: conjunto de nodos formado por los nodos a los que llegan las aristas que parten del nodo i. $\Delta^{+}(i) = \{j \in V |\ (\mathrm{i,j}) \in \mathrm{E}\}$.

   $\Delta^{-}(i)$: conjunto de nodos formado por los nodos de los que parten las aristas que llegan al nodo i. $\Delta^{-}(i) = \{j \in V |\ (\mathrm{j,i}) \in \mathrm{E}\}$.

   d_i: demanda asociada al cliente i.

   e_i: inicio de la ventana temporal del cliente i.

   l_i: fin de la ventana temporal del cliente i.

   s_i: tiempo de servicio que se tarda en atender al cliente i.

   q^{k}: capacidad de vehículo k. En caso de que la flota fuese homogénea todas las capacidades serían iguales.

   t_{ij}^{k}: tiempo que tarda el vehículo k en ir desde el nodo i al nodo j.

   M: constante suficientemente grande.

3. Variables del problema.

   y_i^{k}: hora de llegada del vehículo k al cliente i.

   y_j^{k}: hora de llegada del vehículo k al cliente j.

$$x_{ij}^k = \begin{cases} 1, & si\ el\ vehículo\ k\ va\ desde\ el\ nodo\ i\ al\ nodo\ j \\ 0, & en\ cualquier\ otro\ caso \end{cases}$$

$$x_{ji}^k = \begin{cases} 1, & si\ el\ vehículo\ k\ va\ desde\ el\ nodo\ j\ al\ nodo\ i \\ 0, & en\ cualquier\ otro\ caso \end{cases}$$

$$x_{0j}^k = \begin{cases} 1, & si\ el\ vehículo\ k\ va\ desde\ el\ depósito\ al\ nodo\ j \\ 0, & en\ cualquier\ otro\ caso \end{cases}$$

4. Modelo del problema.

$$\min \sum_{k\in K} \sum_{(i,j)\in E} c_{ij}^k x_{ij}^k \qquad (7)$$

s.a:

$$\sum_{k\in K} \sum_{j\in\Delta-(i)} x_{ij}^k = 1 \qquad \forall\, i \in V\setminus\{0,n+1\} \qquad (8)$$

$$\sum_{j\in\Delta+(0)} x_{0j}^k = 1 \qquad \forall\, k \in K \qquad (9)$$

$$\sum_{j\in\Delta+(i)} x_{ij}^k - \sum_{j\in\Delta-(i)} x_{ji}^k = 0 \qquad \forall\, k \in K, i \in V\setminus\{0,n+1\} \qquad (10)$$

$$\sum_{i\in V\setminus\{0,n+1\}} d_i \sum_{j\in\Delta+(i)} x_{ji}^k \le q^k \qquad \forall\, k \in K \qquad (11)$$

$$y_j^k - y_i^k \ge s_i + t_{ij}^k - M\left(1 - x_{ij}^k\right) \qquad \forall (i,j) \in V\setminus\{0,n+1\}, \forall k \in K \qquad (12)$$

$$e_i \le y_i^k \le l_i \qquad \forall\, i \in V\setminus\{0,n+1\}, \forall\, k \in K \qquad (13)$$

$$x_{ij}^k, x_{ji}^k, x_{0j}^k \in \{0,1\} \qquad \forall (i,j) \in E, \forall\, k \in K$$

$$y_i^k, y_j^k \ge 0 \qquad \forall\, (i,j) \in V\setminus\{0,n+1\}, \forall\, k \in K$$

La función objetivo (7) representa el coste total de la solución. La restricción (8) sirve para que todos los clientes sean visitados por un vehículo. Las restricciones (9) y (10) hacen referencia a que cada vehículo realiza una ruta con origen y fin en el depósito. La restricción (11) sirve para asegurar que la demanda total de los clientes pertenecientes a una ruta no supera la capacidad q_k del vehículo que realiza dicha ruta. La restricción (12) sirve para eliminar los sub-tours (todo subconjunto de nodos S tiene que ser abandonado al menos una vez) y para asegurar que un vehículo que va del nodo i al nodo j no puede llegar a j antes del instante de tiempo que viene dado por la suma del instante de tiempo en el que se llega a i más el tiempo de servicio en dicho cliente i más el tiempo de desplazamiento entre i y j ($y_i + s_i + t_{ij}^k$), siendo M una constante lo suficientemente grande. La restricción (13) impone el cumplimiento de las ventanas temporales.

En esta formulación del problema se ha considerado que se dispone de un número ilimitado de vehículos, por lo que se cuenta con vehículos suficientes para atender a todos los clientes. En caso de que dicho número de vehículos fuese finito, habría que añadir la restricción al modelo.

El problema a resolver en este proyecto se basa en el modelo anteriormente desarrollado del problema de rutado de vehículos con ventanas temporales, pero presenta una serie de particularidades que serán descritas en un capítulo posterior.

4 MÉTODOS DE RESOLUCIÓN

Si buscas resultados distintos, no hagas siempre lo mismo.

- Albert Einstein -

El diseño de rutas de vehículos se ha convertido en un importante problema de optimización en la actualidad, por lo que se han desarrollado a lo largo de las últimas décadas una gran cantidad de métodos para intentar darle solución. Estos algoritmos para resolver las distintas variantes del problema VRP se pueden clasificar en función del aspecto que se considere; por ejemplo, si se considera el enfoque de optimización utilizado pueden ser de optimización local o global, y si se considera la clase de algoritmo al que pertenece se pueden dividir en las siguientes familias: métodos de resolución exacta, heurísticas clásicas o metaheurísticas.

A continuación se van a describir algunos de los métodos más usados para la resolución de problemas de tipo VRP. Estos algoritmos se encuentran en constante mejora y los tiempos de computación necesarios para resolver los problemas dependen de dicho algoritmo y de las características del problema.

4.1 Métodos exactos

Se considera como método exacto a aquel que parte de una formulación como modelo de programación lineal, entera, mixta, cuadrática, etc., y acaba llegando a una solución posible para el problema, haciendo uso de algoritmos para acotar el conjunto de soluciones factibles. Recientemente se ha realizado un enfoque unificado para la resolución de las distintas variantes del VRP.

Resolver un modelo matemático se trata de una tarea compleja, por lo que el uso de estos métodos solo es adecuado para problemas pequeños (alrededor de 50 clientes), teniendo en cuenta que se suele realizar alguna relajación del problema para llevar a cabo la resolución.

Para problemas grandes, aunque se encuentre un algoritmo que halle la solución exacta, el tiempo que emplearía en dar con ella sería tan grande que lo hace inaplicable por completo, por lo que este tipo de métodos de resolución no suelen ser aplicados en la realidad, solo son usados para resolver problemas con un número pequeño de nodos.

Se hablará a continuación de dos tipos de métodos exactos de resolución: el método de ramificación y poda y el método de ramificación y corte.

4.1.1 Ramificación y poda (Branch and Bound)

Este algoritmo consiste en dividir el problema en un conjunto de subproblemas, ramificándolo como si se tratase de las ramas de un árbol, de forma que cada una de estas ramificaciones se corresponda con una posible solución posterior a la actual. Se establecen unas cotas superior e inferior y se va examinando cada subconjunto para hallar la mejor solución. El algoritmo detecta si en una ramificación las soluciones que se están obteniendo no podrán ser óptimas y elimina esa rama del árbol, reduciendo el espacio de búsqueda para no malgastar tiempo en buscar soluciones que se alejen del óptimo, de ahí la palabra poda.

4.1.2 Ramificación y corte (Branch and Cut)

Este método se considera un híbrido, ya que se basa en el método de ramificación y poda, pero además incorpora planos de corte. Se empieza resolviendo el problema lineal sin restricciones enteras. Al obtener una solución óptima con un valor no entero para una variable que ha de serlo, mediante el uso de planos de corte se busca una restricción lineal de forma que se cumpla para todos los puntos factibles enteros y que no se cumpla por la solución óptima no entera obtenida. Si se puede encontrar dicha restricción, se incorpora al problema lineal, llegando a una nueva solución. Se repite este proceso hasta que se encuentre una solución entera y se pueda demostrar que es óptima o hasta que no se encuentren más planos de corte. A partir de entonces, se aplica el algortimo de ramificación y poda.

4.2 Métodos heurísticos

Se conoce como heurística a un algoritmo que para un problema dado permite obtener soluciones con una buena calidad. No garantiza que la solución alcanzada sea óptima (aunque se le acerca bastante), pero los tiempos de ejecución se ven reducidos considerablemente.

Se pueden clasificar las heurísticas usadas para el VRP en tres categorías:

- Constructivas: van dando forma a la solución a medida que avanzan, en lugar de partir de una primera solución factible. Uno de los algoritmos más conocidos es la heurística de ahorros (también conocida como heurística de Clarke and Wright), que será desarrollada a continuación. Como otro ejemplo típico se tienen las heurísticas angulares, como el método de barrido.
- De mejora: la solución se forma a partir de una solución factible. Se pueden encontrar heurísticas del tipo intra-ruta y extra-ruta. En las primeras se hacen variaciones entre los arcos de una ruta, siendo la heurística de Lin-Kernighan un ejemplo de ello, mientras que en las segundas los intercambios se producen entre dos o más rutas distintas.
- Técnicas de relajación: estas heurísticas se encuentran asociadas a la programación lineal entera. La técnica de Relajación Lagrangiana es la más conocida de ellas. Esta heurística descompone un modelo lineal entero en dos conjuntos de restricciones, difíciles y fáciles. Las primeras son relajadas al incluirlas en la función objetivo multiplicándolas por un factor de penalización, tal y como se hace en el método de los multiplicadores de Lagrange, sirviendo esto para acotar el problema original, lo que reduce el tiempo empleado en el proceso de resolución.

A continuación se describen con más detalle tres de las heurísticas más usadas.

4.2.1 Algoritmo de Clarke & Wright

Se conoce también como el método de los ahorros. Fue definido por Clarke y Wright en el año 1964, tratándose de la heurística más significativa para llevar a cabo la resolución de problemas de tipo VRP. Se cuenta con un depósito central y un número de vehículos no limitado para atender la demanda conocida de un número n de clientes, siendo el objetivo el de encontrar las rutas que deben realizar dichos vehículos de manera que los costes sean mínimos y se satisfaga la demanda.

El depósito se denota como 0 y los clientes van desde 1 hasta n. Los costes de ir desde el depósito a cada cliente (c_{0j}) y los costes de desplazamiento entre cada pareja de clientes (c_{ij}) son conocidos y simétricos. Los pasos que sigue el algoritmo son los siguientes:

1. Se crean n rutas, una por cada cliente, de la forma (0,i,0). Los vehículos parten del depósito, realizan la entrega a un cliente y vuelven al depósito.
2. Se calcula el ahorro para cada unión de clientes como:

   $s_{ij} = 2 * (c_{0i} + c_{0j}) - (c_{0i} + c_{ij} + c_{0j}) = c_{i0} + c_{0j} - c_{ij}$.
3. Se ordenan los resultados de forma decreciente.
4. Se escoge la pareja de clientes que han dado lugar al ahorro de mayor valor y, si se cumplen todas las restricciones, se unen, introduciéndose el arco (i, j) y eliminándose los arcos $(i, 0)$ y $(j, 0)$.
5. Se repite el paso anterior hasta que no se encuentren más uniones factibles.

Este algoritmo es muy utilizado porque se obtienen buenos resultados mediante la exploración limitada del espacio de búsqueda, siendo su aplicación bastante simple y con un tiempo de resolución pequeño.

Por otra parte, el método no contempla un número limitado de vehículos. Si este fuera el caso, el paso 4 sería realizado hasta alcanzar el número deseado de vehículos, incluso si esto conlleva hacer uso de uniones que presenten ahorros negativos.

4.2.2 Técnicas de búsqueda local

En estas heurísticas, se parte de una solución inicial dada y se va mejorando esta de forma progresiva. En cada iteración, el algoritmo tiene que hallar una solución mejor a la anterior, finalizando el proceso cuando no se encuentre ningún movimiento que mejore la solución actual.

Tal y como su nombre indica, se trata de un procedimiento de búsqueda local, por tanto es de esperar que la solución obtenida sea un óptimo local. Para escapar de dicho óptimo, se tendrá que permitir al algoritmo la posibilidad de desplazarse hacia soluciones de peor valor, para así conseguir llegar a un óptimo distinto al anterior, creándose un algoritmo más complejo.

Lin y Kernighan fueron los responsables de la propuesta de un algoritmo basado en la realización de movimientos compuestos. La función objetivo puede ser mejorada o empeorada por cada uno de los movimientos simples, pero el movimiento global debe ser de mejora, con el fin de mantener el control sobre el proceso de búsqueda.

La mezcla de diferentes movimientos para alterar la estructura de la solución que se lleva a cabo en este método hace que se alcance el óptimo de manera lenta y gradual, e introduce una componente de diversificación.

4.2.3 Método de barrido

Se trata de un método sencillo que puede ser resuelto en poco tiempo, e incluso mediante cálculos manuales, hasta para problemas de gran tamaño. Por esto, es muy usado cuando se necesita la obtención de buenos resultados en un período pequeño de tiempo.

El algoritmo se lleva a cabo de la siguiente manera:

1. Se presenta el mapa con la localización del depósito y de los clientes en un plano en coordenadas polares (r, θ). La línea recta entre el depósito y el cliente se corresponde con r, siendo θ el ángulo que forma dicha línea con respecto a un eje de referencia.
2. Una recta cuyo origen es el depósito y con un determinado ángulo θ va barriendo el plano hasta que la suma de las demandas de los clientes que ya han sido "barridos" supere la capacidad máxima del vehículo, obteniéndose varios conjuntos de clientes.

3. Se emplea un algoritmo para obtener la mejor ruta posible dentro de cada uno de dichos conjuntos. Para obtener esto podría usarse, por ejemplo, el algoritmo de ahorros descrito anteriormente.

Se puede ver en la figura 7 el proceso de formación de los distintos conjuntos. Se supone una capacidad de 10.000 unidades para los vehículos y se va barriendo (en este caso en sentido positivo) hasta alcanzar la capacidad máxima de los vehículos. Cuando no se cumpla la restricción de capacidad, se hace una separación y se define una ruta.

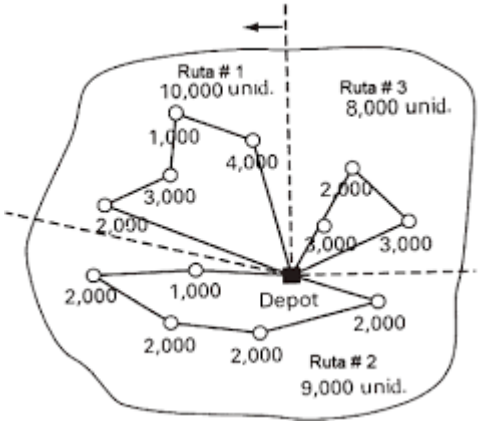

Figura 7: Ejemplo cálculo de conjuntos con método de barrido (Ballou, 2004)

4.3 Métodos metaheurísticos

Se conoce como metaheurísticas a aquellos métodos aproximados diseñados para la resolución de problemas complejos de optimización combinatoria, para los que las heurísticas no son capaces de obtener una buena solución. En general, conllevan un mayor tiempo de ejecución que las heurísticas clásicas, pero este aún sigue siendo menor que el empleado por los métodos exactos. Mediante la combinación de mecanismos estadísticos y de conceptos derivados de la inteligencia artificial y de la evolución biológica se genera un marco donde se crean nuevos algoritmos híbridos.

Se pueden clasificar los métodos metaheurísticos en las siguientes categorías:

- Constructivos: se parte de una solución vacía en la que se van introduciendo elementos. El ejemplo más destacado es el algoritmo GRASP.
- Evolutivos: se realizan varios grupos de soluciones completas y, basada en el valor de algunos atributos, se lleva a cabo una selección de ellas. Tras esto, se combinan entre sí algunas de las soluciones escogidas para dar lugar a nuevas soluciones y se reemplazan soluciones anteriores. En esta categoría se puede encontrar el Algoritmo Genético.
- De búsqueda: se da por hecho la existencia de una solución óptima y el algoritmo realiza una serie de pasos que, aunque no la alcance, se queda bastante cerca. El problema que se presenta en estos métodos es el de quedar atrapado en un óptmico local, por tanto se hace necesaria la aplicación de ciertos procedimientos para escapar de él. Se puede actuar de tres formas, según se lleve a cabo dicha salida:
    1. Comenzar de nuevo el algoritmo tomando una solución inicial distinta. Un ejemplo de ello sería la metaheurística multi-start.
    2. Una vez alcanzado el óptimo, proceder a la variación de la estructura del entorno de la solución. Como ejemplo se tiene la metaheurística de búsqueda de entornos variables.

3. Aceptar movimientos que empeoren el valor de la solución obtenida para poder salir del óptimo local. En este caso se encuentran el recocido simulado y la búsqueda tabú.

Se describen con más detalle a continuación cinco de las metaheurísticas más usadas.

4.3.1 Búsqueda local

Las técnicas de búsqueda local intentan encontrar una mejora de la solución actual entre un número de soluciones candidatas. Para ello se mueven de una solución a otra mediante la realización de cambios locales hasta que se encuentra la solución considerada óptima o hasta que transcurre un plazo establecido.

El algoritmo de búsqueda local parte de una solución inicial y se va moviendo de forma iterativa a una solución vecina. Para esto es necesario definir una relación de vecindad en el espacio de búsqueda. Por ejemplo, una solución vecina de una solución formada por un conjunto de nodos podría ser otra solución en la que varía uno de ellos. Esto es, si se tiene una solución formada por los nodos (1-2-3-4), una solución vecina podría ser la formada por (1-2-3-5). Cada solución candidata tiene más de una solución vecina, formándose estas únicamente a partir de la solución actual, de ahí el nombre de búsqueda local.

Cuando se toma como criterio para movernos a una solución vecina la maximización local, la metaheurística recibe el nombre de *hill climbing*. Puede darse el caso de que en los vecinos no se encuentren mejoras, quedando la búsqueda local atrapada en un óptimo local. Para salir de él se puede hacer uso de las opciones descritas previamente para los algoritmos de búsqueda.

La conclusión del algoritmo puede llevarse a cabo en un tiempo limitado. Es común escoger la finalización cuando en un número determinado de pasos no se produce mejora en la solución que se considere actualmente la mejor encontrada. Esta técnica puede proporcionar una solución que valga aún habiéndose interrumpido el proceso antes de acabar.

4.3.2 GRASP (greedy randomized adaptative search procedure)

Traducido como "Procedimiento de búsqueda voraz aleatoria y adaptativa", se trata de una técnica para resolver problemas de optimización combinatoria. Primeramente, se intenta la construcción de soluciones con una calidad elevada, para que sean a continuación procesadas, obteniendo otras soluciones de mejor calidad.

Como ya se ha mencionado, se trata de un algoritmo de tipo constructivo. En cada iteración se tienen dos fases: una primera fase de construcción de la solución y una segunda en la que se intenta optimizar la solución generada en la fase anterior, haciendo uso de alguna técnica de búsqueda local.

El procedimiento que sigue el algoritmo es el siguiente:

1. Mediante el uso de un procedimiento de tipo Greedy, se construye una solución aleatoria.
2. Para mejorar esta solución, se lleva a cabo una búsqueda local.
3. Se procede a la actualización de la mejor solución obtenida.
4. Se repiten estos pasos hasta la satisfacción del criterio de parada definido.

Se puede considerar qu existe cierta similitud con el problema de Programación Lineal, en el que se realiza la construcción de una solución factible y posteriormente, se le aplica el algoritmo Simplex. Pero se presenta una gran diferencia, ya que en la técnica GRASP la calidad de la solución generada es muy importante.

4.3.3 Algoritmo Genético

Esta técnica, introducida por Holland, hace uso de las ideas de la evolución natural de los seres vivos para resolver problemas de optimización. El algoritmo trabaja sobre una población (conjunto de soluciones codificadas) donde cada una de las soluciones recibe el nombre de cromosoma. Mediante la aplicación de una serie de pasos se modifican en cada iteración las soluciones, creándose soluciones nuevas formadas a partir de las anteriores.

Se puede considerar el método como una analogía de la teoría de la evolución de Darwin, puesto que las nuevas soluciones mantienen algunas de las características de las soluciones de las que han sido derivadas, dependiendo del nivel de mutación en el que se haya incurrido en cada iteración.

El proceso general que sigue el algoritmo es:

1. Inicialización: se genera una población inicial, formada por un conjunto de cromosomas que representan distintas soluciones del problema. Dicha población inicial se puede crear de forma aleatoria o no, pero para los casos en los que no sea generada aleatoriamente, las soluciones deben ser diversas para tener representada a la mayor parte de la población y evitar que el método converja antes de lo esperado.
2. Evaluación: se evalúa cada uno de los cromosomas haciendo uso de la función objetivo para establecer cómo de buenas son las soluciones.
3. Desarrollo del algoritmo: se realizan los siguientes pasos hasta que se alcance un número fijado de iteraciones o no se produzcan cambios en la población:
    a. Selección: se eligen los cromosomas con mejor evaluación para ser cruzados y generar una nueva tanda de soluciones.
    b. Cruzamiento: se generan dos nuevos "descendientes" a partir de la combinación de dos cromosomas que se consideran "padres".
    c. Mutación: se modifica de forma aleatoria alguna parte de algunos de los individuos, con el fin de poder explorar zonas del espacio de soluciones a las que no se podía acceder con la población hasta el momento.
    d. Reemplazo: tras realizar los pasos previos, se procede a seleccionar los mejores individuos para que formen la siguiente población.

4.3.4 Búsqueda Tabú

Atribuida a Fred Glover, la *Tabú Search* es una de las técnicas metaheurísticas más importantes para la resolución de problemas de optimización combinatoria, siendo considerada como una de las que mejores resultados consigue para resolver problemas de tipo VRP. Como característica fundamental presenta el empleo de estructuras de memoria como base para su estrategia, llevando a cabo el almacenamiento de toda la información posible para conseguir la mejora de la solución.

Este algoritmo presenta, en la solución que se quierer mejorar, una serie de movimientos tabú (movimientos prohibidos), que ayudan a que el algoritmo no se atasque en óptimos locales y pueda buscar en una zona más amplia. Por esto es por lo que se considera que el algoritmo cuenta con memoria, aunque sea a corto plazo.

Se dice que es una técnica inteligente, ya que va almacenando información mientras que se producen iteraciones del algoritmo, actuando posteriormente en consonancia con dichos datos almacenados. Para salir de óptimos locales se puede llevar a cabo una diversificación para guiar a la búsqueda hacia las zonas inexploradas.

4.3.5 Recocido simulado

Esta técnica, también conocida como *Simulated Annealing*, trata de asemejarse al proceso al que se someten los metales para mejorar su estructura cristalina. El metal es calentado hasta alcanzar una temperatura muy elevada para ser enfriado de forma lenta, lo que permite que su estructura cristalina se reorganice en una configuración de mínima energía. Este proceso se tiene que realizar de forma adecuada, ya que el exceso de temperatura provoca la ruptura del orden alcanzado, mientras que un calentamiento insuficiente no permite que los átomos se muevan para reconfigurar su posición.

La aplicación de esta idea para resolver problemas de optimización se basa en la siguiente analogía:

Se considera una configuración cristalina de los átomos metálicos como una solución, donde la energía asociada a dicha configuración será la longitud de la solución y la temperatura será una distancia.

Para cada iteración del algoritmo, se evalúan algunos vecinos del estado actual y se decide, de forma probabilísitica, si transicionar hacia un nuevo estado o si mantenernos en el que nos encontramos. Se puede definir un estado como la posición que ocupan los atómos en el momento actual, mientas que un vecino se formaría a partir del desplazamiento de un átomo de la solución primera. El algoritmo repite la comparación entre estados vecinos hasta que se alcancen las condiciones de parada impuestas o hasta que se llegue a un estado óptimo que minimice la energía del sistema.

Dependiendo de la calidad de la solución (energía) y de la temperatura establecida en la iteración, se escogerá una configuración u otra:

- Si la temperatura es muy elevada, todas las configuraciones tienen la misma probabilidad de ser escogidas, sin importar la calidad de estas.
- Si la temperatura es muy baja, la probabilidad de ser escogida será no nula solo para las configuraciones cuyo coste sea mínimo.

Si el sucesor que se ha elegido de forma aleatoria conlleva una pérdida de energía, se convierte en la solución actual, por el hecho de que se está minimizando. Dependiendo de la temperatura, también podría ser escogido un sucesor que provoque un incremento de energía.

Se van sucediendo pequeños cambios en el proceso hasta que no existan posibilidades de mejora. Se reduce lentamente la temperatura, lo que implica que disminuye la probabilidad de que se produzcan movimientos que puedan empeorar la función objetivo, hasta llegar a la total prohibición.

5 TÉCNICAS APLICADAS AL PROBLEMA DE RUTADO DE VEHÍCULOS

Imposible significa que no has encontrado la solución.

- Henry Ford -

En la realización de este trabajo se ha planteado como objetivo principal la resolución de problemas de rutas de transporte mediante el uso de técnicas heurísticas, de forma que el coste de la solución sea el mínimo posible. Para la búsqueda de dichas soluciones se hará uso concretamente del algoritmo de Clarke & Wright, con el que se obtendrá una primera solución candidata, a la que posteriormente se le aplicará una búsqueda local para ver si es posible hallar una solución con menor coste.

A continuación se van a describir las características de los problemas a resolver, además de la codificación de los métodos heurísticos, haciendo hincapié en las simplificaciones que se han asumido para la implementación de estos.

5.1 Descripción del problema

El problema a resolver es el siguiente: se tiene una serie de clientes, situados de forma dispersa geográficamente, cuyos pedidos deben ser satisfechos. Se dispone de una serie de vehículos, que deben partir de un depósito y volver a él tras finalizar su ruta, para llevar a cabo las entregas a dichos clientes. Se considera una flota homogénea, teniendo todos los vehículos las mismas características y contando con la misma capacidad, sin ser esta limitante (un único vehículo podría servir a todos los clientes si se cumpliesen las restricciones de tiempo, los productos se consideran de tamaño pequeño). Se supone un número de vehículos suficiente para poder servir a todos los clientes.

A la hora de servir a los clientes esto se puede realizar, para cada uno de ellos, en tres localizaciones distintas. Cada una de estas localizaciones tiene asociada una ventana temporal, que se conoce como el intervalo de tiempo en el que podrá ser atendido en dicha localización. El hecho de que cada cliente tenga tres posibles lugares donde poder entregarle la mercancía se debe a que estos pueden cambiar su situación a lo largo del tiempo. Por ejemplo, un cliente podría encontrarse en su casa a primera hora de la mañana, pero transcurrido cierto periodo de tiempo podría desplazarse hasta su lugar de trabajo, cambiando de esta forma su posición. Si un cliente ha sido atendido en una de sus posibles localizaciones, las otras dos dejan de estar activas, ya que dicho cliente ya ha sido servido y solo es necesario visitar una vez a cada uno de ellos. El depósito desde el que parten los vehículos también tiene asociada una ventana temporal.

En la Figura 8 se puede ver un ejemplo de lo explicado previamente. Se representa un problema con 3 clientes, cada uno de ellos con 3 ventanas temporales. El depósito aparece denotado como "0". Una solución posible podría ser la representada en la Figura 9, donde se parte desde el depósito y se visita al cliente uno en su tercera localización, después se pasa a servir al cliente dos en la primera y se acaba visitando al cliente tres en la tercera, volviendo tras esto al depósito.

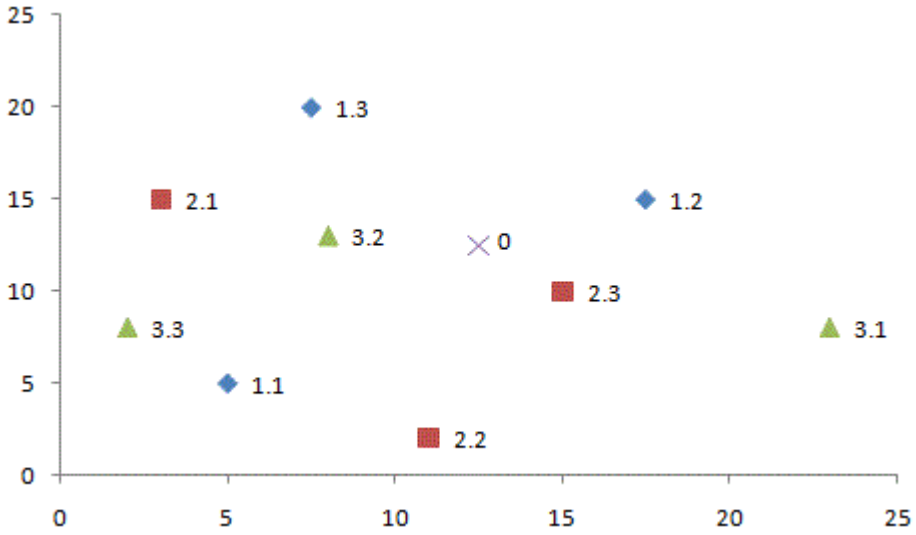

Figura 8: Representación gráfica problema de 3 clientes con 3 ventanas temporales cada uno

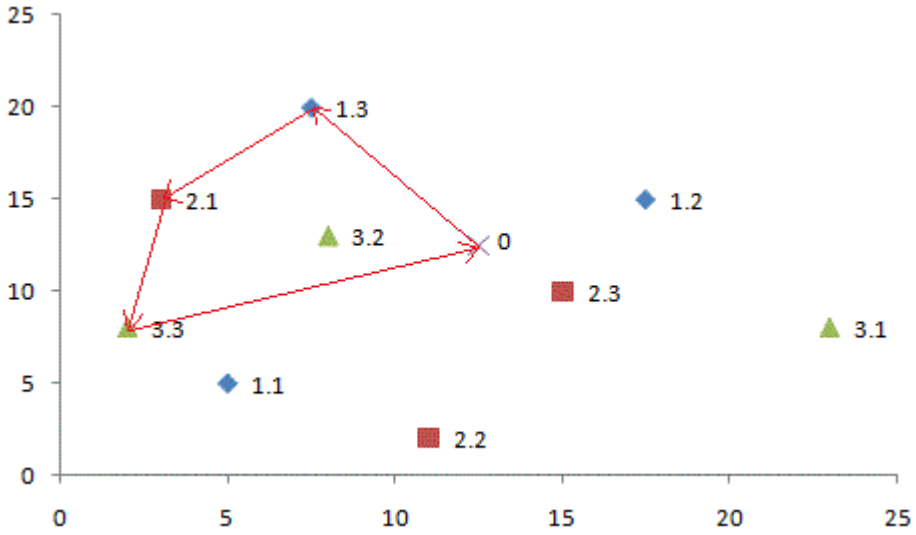

Figura 9: Representación gráfica solución problema de 3 clientes con 3 ventanas temporales cada uno

La resolución del problema conlleva la determinación de las rutas a realizar para servir a todos los clientes, de forma que el coste sea mínimo. Cada una de las rutas se corresponde con un vehículo. En la Figura 10 se puede observar un ejemplo de solución para un problema VRPTW con 30 clientes. En el centro, representado mediante un cuadrado rojo, se encuentra el depósito. Todas las rutas parten de él y vuelven aquí al finalizar, siendo representada cada una de ellas por un color distinto y rodeándose en rojo el primer cliente a visitar para conocer la orientación de la misma.

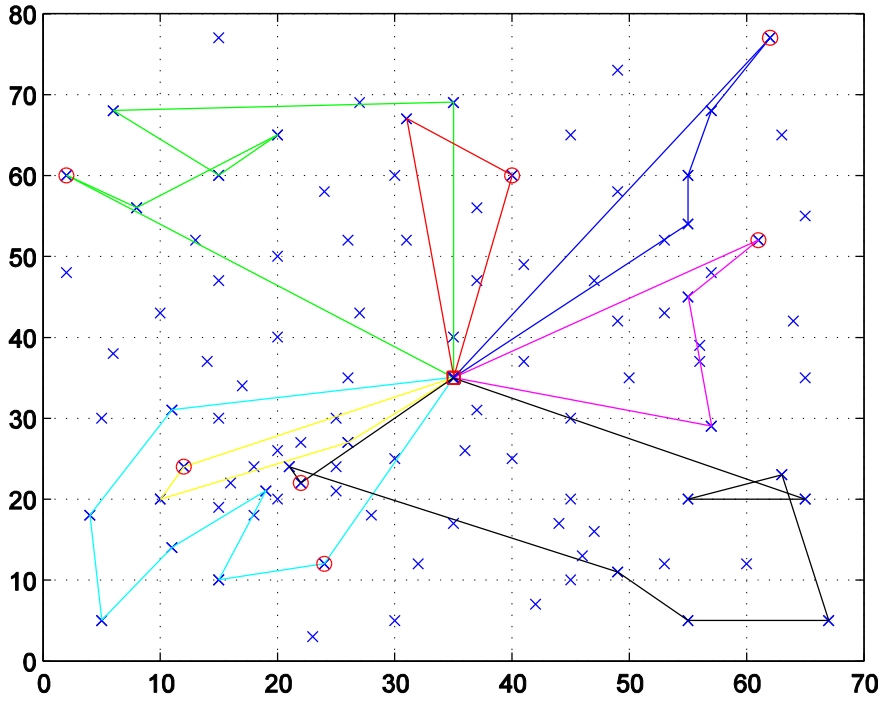

Figura 10: Ejemplo solución problema VRPTW

5.2 Adaptación de las técnicas a problema bajo estudio

Como ya se ha comentado al inicio del capítulo, lás técnicas usadas para resolver el problema serán el algoritmo de Clarke & Wright, para obtener una primera solución, y la búsqueda local, para ver si, aplicándola a partir de la solución obtenida por el algoritmo de ahorros, es posible hallar una solución con menor coste. Ambas técnicas fueron descritas con detalle en el capítulo anterior, encontrándose el algoritmo de Clarke & Wright dentro de las heurísticas y la búsqueda local dentro de las metaheurísticas.

5.2.1 Resolución mediante algoritmo de Clarke & Wright

A continuación se va a proceder a describir la implementación del algoritmo de ahorros. Previamente se van a aclarar algunos conceptos para que sea más fácil la posterior comprensión del desarrollo de dicho algoritmo. En este caso, las restricciones que se deben cumplir y que determinarán las rutas no se tratan de restricciones de capacidad, sino que se tratan de restricciones temporales, ya que cada uno de los clientes cuenta con una ventana temporal dentro de la cual tiene que ser atendido.

Como ya se ha comentado anteriormente, el algoritmo de ahorros es un método heurístico, por lo que no proporciona una solución óptima del problema con certeza. Sin embargo, sí que obtiene una buena solución, cuyo desviación con respecto a la óptima es pequeña.

5.2.1.1 Consideraciones importantes

Previo al desarrollo del algoritmo se hace necesario tener en cuenta lo siguiente:

- Número de clientes del problema (numCustomer): el número de clientes del problema es uno de los valores que se necesita para determinar el problema a resolver.
- Ventanas temporales de las rutas: al igual que los clientes, las rutas también cuentan con un periodo de tiempo determinado en el que se deben desarrollar. Todas ellas cuentan con un inicio y fin más temprano y con un inicio y fin más tardío. Los inicios más temprano (iE) y más tardío (iL) se conocen como los instantes de tiempos más temprano y más tardío, respectivamente, en los que puede comenzar la ruta, mientras que los finales más temprano (fE) y más tardío (fL) se tratan de los instantes más temprano y tardío, respectivamente, en los que puede finalizar. Todos ellos están condicionados por las restricciones temporales que se han de cumplir.
- Distancias: es importante conocer las distancias entre las distintas posiciones para poder calcular el tiempo de desplazamiento entre cada una de ellas. Se crea una matriz simétrica donde se almacenan los valores de distancia que hay entre cada par de localizaciones. También se cuenta con otra matriz simétrica donde se guardan las distancias existentes entre los clientes y el depósito.
- Orígenes y destinos: se conoce como origen a la última tarea de cada ruta intermedia (rutas que se van formando según se desarrolla el algoritmo) y como destino a la primera tarea de dichas rutas intermedias. Esto es así porque durante el desarrollo del algoritmo se produce la unión de rutas intermedias, de tal forma que los clientes que se encuentren en la última posición (orígenes) solo pueden buscar una tarea sucesora, ya que la predecesora la tienen fijada. Para el caso de la primera posición (destinos) ocurre lo contrario, solo pueden ser precedidos por otra tarea porque la sucesora ya se encuentra definida.
- Factibilidad de las uniones: se conoce como unión factible aquella unión de rutas intermedias que cumple las restricciones del problema y, por tanto, se puede llevar a cabo. Para que sea factible se deben cumplir los siguientes requisitos:
    1. No pueden tratarse de uniones entre distintas localizaciones de un mismo cliente.
    2. El instante de tiempo resultante de la suma del instante más temprano de finalización de la ruta más el tiempo de desplazamiento entre la última tarea de la ruta intermedia actual y la primera tarea de la ruta intermedia que se está considerando tiene que ser menor o igual que el instante de inicio más tardío de la ruta intermedia que se está considerando (fE+distanceOD≤iL). En otras palabras, se han de cumplir las restricciones de las ventanas temporales, los clientes tienen que ser servidos dentro de su ventana temporal.
    3. Una vez unidas las dos rutas intermedias, a la finalización de la ruta nueva se parte hacia el depóstito y se tiene que poder llegar antes de la finalización del horizonte temporal de este.
- Ahorro: se conoce el concepto de ahorro como el ahorro de costes que se obtiene al unir dos rutas en una como se representa en la Figura 11. En primer lugar, todas las rutas tienen la forma $(0,i,0)$, los vehículos parten del depósito, sirven al cliente y vuelven al depósito. Si se realiza la ruta como en la imagen de la derecha se consigue un ahorro, que viene dado por la diferencia de costes entre las dos situaciones:
    1. Coste en situación de la izquierda: $C_{izq} = c_{0i} + c_{i0} + c_{0j} + c_{j0}$.
    2. Coste en situación de la derecha: $C_{der} = c_{0i} + c_{ij} + c_{j0}$.
    3. Ahorro: $S_{ij} = (c_{0i} + c_{i0} + c_{0j} + c_{j0}) - (c_{0i} + c_{ij} + c_{j0}) = c_{i0} + c_{0j} - c_{ij}$.

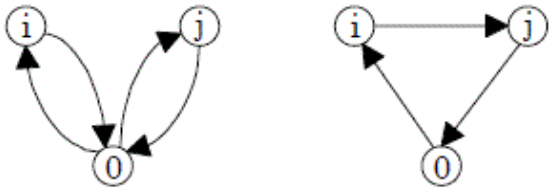

Figura 11: Concepto de ahorro

- Tiempo de espera: si se llega a un cliente antes de que su ventana temporal haya comenzado se tendrá que esperar un tiempo hasta que esto ocurra para poder atender al cliente.
- Solución: en ella se encuentran definidas todas las rutas que componen la solución final. Cada vehículo realiza únicamente una ruta. Parten del depósito, sirven a la secuencia de clientes que les corresponda y vuelven al punto de partida. En la Figura 12 se puede ver un ejemplo de codificación de una solución para un problema resuelto mediante el algoritmo de Clarke & Wright. Cada una de las filas se interpreta como una ruta, siendo los números distintos de 0 el orden de las localizaciones de los clientes que tiene que visitar el vehículo. Por ejemplo, la primera fila se correspondería con la ruta (0-65-71-9-81-0) y la quinta con la ruta (0-78-3-54-0). Se recuerda que al depósito se le denota con el número 0.

```
routes =

    65    71     9    81     0     0     0     0
    36    47    11    19    49    32     0     0
    30    90     0     0     0     0     0     0
    42    14    98    44    38    86    84     0
    78     3    54     0     0     0     0     0
    61    16    94     0     0     0     0     0
    92    59    75    23    67    55     4    25
```

Figura 12: Ejemplo representación solución Clarke & Wright

5.2.1.2 Desarrollo del algoritmo

Una vez aclarados ciertos conceptos para el mejor entendimiento de la implementación del método heurístico, en este apartado se van a describir los pasos que se siguen en el desarrollo del algoritmo de ahorros. Para facilitar la comprensión de la heurística, se puede pensar en cada ruta no como en un conjunto de clientes, sino como si siguiera siendo un único cliente, ya que contará también con su ventana temporal.

En primer lugar, se calculan las distancias desde cada uno de los clientes al depósito y las distancias entre pares de localizaciones de clientes distintos (las distancias entre las posiciones de un mismo cliente no son interesantes). Al comienzo del algoritmo se supone que se tienen tantas rutas como ventanas temporales haya, por ejemplo, si se tuvieran 10 clientes que servir en el problema, al inicio se supondrían 30 rutas.

A continuación, se buscan los orígenes y los destinos posibles y se calculan las distancias entre ellos. Para cada ruta se calculan los tiempos de inicio y fin más tempranos y más tardíos. Se debe tener cuidado con esto, ya que no siempre el comienzo más temprano de la ruta coincidirá con el inicio de la ventana temporal del primer cliente, ya que es necesario desplazarse desde el depósito, lo que conlleva un tiempo. Por ejemplo, si el comienzo de la ventana temporal del primer cliente de la ruta fuese el instante 15 pero el vehículo desplazándose desde el depósito llegase en el instante 18, el inicio más temprano de la ruta se correspondería con el instante 18. Lo mismo ocurriría para el caso de la última tarea de la ruta y su finalización más tardía, ya que el vehículo tiene que finalizar su recorrido volviendo de nuevo al depósito.

En el siguiente paso, se procede a buscar entre todas las uniones de rutas posibles aquellas que sean factibles y se calculan los ahorros que se producirían al unirlas y el mínimo tiempo de espera entre orígenes y destinos, ya que la unión a realizar se escogerá mediante una combinación de estos factores. Para calcular el margen de mejora se multiplica el ahorro por un coeficiente y se le resta el mínimo tiempo de espera multiplicado por otro coeficiente, siendo los coeficientes escogidos en función de la importancia que se le atribuya a cada uno de los factores. La unión factible de rutas con mayor margen de mejora será la escogida y se procede a su unión para constituir una nueva ruta.

Tras la creación de dicha nueva ruta, se actualizan los datos del problema: inicios y finalizaciones más tempranos y tardíos de cada ruta y nuevos orígenes y destinos posibles. Se vuelve a calcular la factibilidad entre rutas y sus márgenes de mejora para proceder a elegir una nueva unión de rutas. Este proceso se repite hasta que no se puedan realizar más uniones debido a que ya no exista margen de mejora entre rutas.

Llegados a este punto, si tras no poder realizarse más uniones de rutas ha quedado algún cliente sin unir, se necesitará un vehículo por cada cliente que haya quedado suelto para que puedan ser atendidos. En este caso, la ventana temporal no será una restricción condicionante para escoger la localización donde visitarlo, ya que el vehículo únicamente servirá a ese cliente. Por tanto, se escoge la localización más cercana al depósito para que el coste debido al desplazamiento sea menor.

Al final del algoritmo se tendrá un conjunto de rutas que se corresponderá con la mejor solución del problema mediante la heurística de Clarke & Wright.

5.2.2 Resolución mediante búsqueda local

Como se comentó al inicio del capítulo, para la búsqueda de soluciones para los problemas de rutado se haría uso del algoritmo de ahorros con el fin de obtener una primera solución candidata, que posteriormente sería usada para llevar a cabo una búsqueda local con el fin de intentar hallar una solución mejor a la ya encontrada. Dicha búsqueda local podría realizarse también partiendo de una solución aleatoria, pero se decide usar la proporcionada por la heurística desarrollada anteriormente porque se obtendrá una nueva solución en un tiempo menor, aunque no existan garantías de que sea mejor que la solución que se podría alcanzar al partir desde la aleatoria.

5.2.2.1 Vecindad

Como ya se comentó en el capítulo de los métodos de resolución, el algoritmo de búsqueda local parte de una solución inicial y se va moviendo de forma iterativa a una solución vecina. Lo primero que será necesario será definir la batería de vecinos que componen el espacio de búsqueda. En este caso, una solución vecina de una solución formada por un conjunto de nodos será otra solución en la que variará uno de ellos. Por ejemplo, si se tiene una solución formada por los nodos (1-2-3-4), una solución vecina sería la formada por (2-1-3-4).

Para llevar a cabo la búsqueda local se ha decidido realizar la creación de dos vecindades distintas. Se describen a continuación los métodos usados para cada una de ellas:

- Vecindad mediante permutación: las soluciones vecinas se construyen permutando para cada una de ellas únicamente la posición de dos elementos de la solución inicial. Para mayor claridad se hace uso de un ejemplo. Para el caso de la solución inicial (1-2-3), las soluciones que constituirían la vecindad serían (2-1-3), (3-2-1) y (1-3-2). Para un número n de elementos en el vector solución inicial, el número de permutaciones posibles para la creación de la vecindad es de $\left(n * \frac{n+1}{2}\right) - n$.

- Vecindad mediante inserción: las soluciones vecinas se construyen mediante la inserción de un elemento de la solución inicial en una posición distinta a la posición que ocupa en la solución de partida. Al igual que en el caso anterior, se produce una única modificación con respecto a la solución inicial en cada solución que constituye la vecindad. Para el mismo caso de antes, las soluciones vecinas de la solución inicial (1-2-3) mediante este método serían (2-1-3), (2-3-1), (1-3-2) y (3-1-2). Para un número n de elementos en el vector solución inicial, el número de permutaciones posibles para la creación de la vecindad es de $(n^2 - 2 * n + 1)$.

5.2.2.2 Consideraciones importantes

En primer lugar, previo al desarrollo del algoritmo, se hace necesario especificar la forma que debe tener la solución inicial de partida para que se pueda hacer uso de ella. En la Figura 13 se puede ver la estructura que tiene la solución que se obtiene tras aplicar la heurística de Clarke & Wright. Se tiene una matriz donde cada una de sus filas representa una ruta y se necesita modificar la forma de dicha solución con el fin de obtener un vector con el que poder trabajar para crear las vecindades.

```
routes =

     2    15     0
     5    11     7
```

Figura 13: Ejemplo de solución inicial

En la Figura 14 se puede ver la nueva estructura que presenta la solución anterior tras haber sido transformada. Dicha forma se obtiene representando una a continuación de otra las rutas de la solución obtenida por el algoritmo de ahorros y añadiendo tras esto las ventanas temporales que habían sido descartadas. Esto se hace para que cuando se creen las vecindades se encuentren representadas todas las posibles soluciones vecinas del espacio de búsqueda.

```
solucion =

     2    15     5    11     7     1     3     4     6     8     9    10    12    13    14
```

Figura 14: Ejemplo de solución inicial transformada para búsqueda local

También es importante aclarar que a la hora de la búsqueda de una nueva solución mediante la técnica de búsqueda local se han usado dos métodos para la construcción de las soluciones. La diferencia entre ellos se encuentra en la forma de ordenar los clientes en los vehículos. Se habla de ordenar clientes en vehículos porque una ruta se corresponde con la secuencia de clientes que tiene que visitar un vehículo. Se explican ambos métodos a continuación, haciendo uso del ejemplo de la Figura 14 para una mayor comprensión:

- Método uno: se coloca el primer elemento del vector solución en el primer vehículo (se inicia la primera ruta con la primera localización que aparece en la solución que se está considerando). Se considera la siguiente posición: si se cumplen las restricciones de las ventanas temporales, es decir, si desde la primera localización se puede llegar a la segunda y servir al cliente, se añade este a la ruta. Se lleva a cabo este proceso hasta que se encuentre el primer elemento que incumpla las restricciones, momento en el que se comenzará una nueva ruta. Se tiene que tener en cuenta que en el mismo vector se encuentran las tres ventanas temporales de cada cliente, por lo que cuando ya se haya añadido una a una ruta, las otras dos dejarán de tener validez y serán saltadas cuando se esté explorando el vector.

  En el ejemplo: se empezaría una primera ruta con la localización 2, es decir, el vehículo que realice esa ruta deberá visitar la localización 2 (que se corresponde con la segunda ventana temporal del primer cliente) en primer lugar. A continuación se comprueba el siguiente elemento: suponemos que es posible servir la posición 15 tras haber servido la 2, por tanto se añadiría a la ruta actual. Se pasa ahora al siguiente: se va a considerar ahora que no se puede servir la localización 5 tras haber servido la 15. En este caso la posición 5 pasará a ser la primera tarea a realizar en una segunda ruta. Siguiendo este método se construyen las rutas correspondientes hasta que todos los clientes estén incluidos en alguna de ellas.

- Método dos: se coloca el primer elemento del vector solución en el primer vehículo (se inicia la primera ruta con la primera localización que aparece en la solución que se está considerando). Se considera la siguiente posición: si se cumplen las restricciones de las ventanas temporales, es decir, si desde la primera localización se puede llegar a la segunda y servir al cliente, se añade este a la ruta. La diferencia con el método anterior es que no se salta a la creación de una nueva ruta tras encontrar el primer elemento que incumpla las restricciones, sino que se sigue explorando el vector saltando el elemento en cuestión y continuando con los demás hasta que no se encuentre ningún elemento que se puede añadir a la ruta o hasta que se ocupe todo el horizonte temporal del vehículo. Al igual que antes, se tiene que tener en cuenta que en el mismo vector se encuentran las tres ventanas temporales de cada cliente, por lo que cuando ya se haya añadido una a una ruta, las otras dos dejarán de tener validez y serán saltadas cuando se esté explorando el vector.

  En el ejemplo: se empezaría una primera ruta con la localización 2, es decir, el vehículo que realice

esa ruta deberá visitar la localización 2 (que se corresponde con la segunda ventana temporal del primer cliente) en primer lugar. A continuación se comprueba el siguiente elemento: suponemos que es posible servir la posición 15 tras haber servido la 2, por tanto se añadiría a la ruta actual. Se pasa ahora al siguiente: se va a considerar ahora que no se puede servir la localización 5 tras haber servido la 15. En este caso se salta la posición 5 y se pasará a comprobar la posición 11. En el momento en el que no se puedan añadir más localizaciones a la ruta actual, por cualquiera de los motivos que se han comentado anteriormente, se abrirá y comenzará a llenar con el mismo procedimiento la segunda ruta. Siguiendo este método se construyen las rutas correspondientes hasta que todos los clientes estén incluidos en alguna de ellas.

Por último, se puede observar en la Figura 15 la estructura que presentan las soluciones obtenidas mediante el procedimiento de búsqueda local. Dicha estructura es la misma que la que tienen las obtenidas tras la heurística de Clarke & Wright, correspondiéndose cada fila con una ruta a realizar por un vehículo.

```
goodcarspm1 =

    65    71     9    81     0     0     0
    16    61     0     0     0     0     0
    36    47    11    19    49    32    89
    28    75    23    67    56     4     0
    53     0     0     0     0     0     0
    59    42    14    44    38    86    84
    76     3    26     0     0     0     0
```

Figura 15: Ejemplo representación solución Búsqueda Local

5.2.2.3 Desarrollo del algoritmo

La búsqueda de soluciones mediante la aplicación de la búsqueda local en este trabajo se ha llevado a cabo de varias formas distintas, combinando la forma de crear la vecindad con los métodos para construir las rutas. Se mencionan a continuación los cuatro procedimientos empleados:

1. Vecindad creada mediante permutación y método uno de construcción de soluciones.
2. Vecindad creada mediante permutación y método dos de construcción de soluciones.
3. Vecindad creada mediante inserción y método uno de construcción de soluciones.
4. Vecindad creada mediante inserción y método dos de construcción de soluciones.

Se va a proceder a explicar de forma desarrollada el procedimiento que sigue el algoritmo para el primer caso. Únicamente se describirá este porque todos siguen los mismo pasos, con la única diferencia de que puede variar la forma de crear la vecindad o la forma de construir las soluciones.

En primer lugar, se transforma la solución inicial de partida en una solución que tenga una forma adecuada con la que se pueda trabajar, como se ha explicado anteriormente.

A continuación, se crea la batería de soluciones vecinas a partir de permutaciones del vector obtenido en el paso anterior.

Una vez creada la vecindad, se procede a analizar las soluciones que la componen. Haciendo uso del método uno de construcción de soluciones se van ordenando una por una las soluciones y se evalúan.

Para finalizar, se busca entre los valores resultantes de esta evaluación el menor de ellos, ya que corresponderá con la mejor solución. Se toma esta solución como nueva solución inicial, se vuelve a crear una vecindad mediante permutaciones y se vuelven a evaluar las soluciones tras aplicar el método uno. Se repite el proceso hasta que no se pueda encontrar ninguna solución con menor coste.

6 RESULTADOS

Mejor que de nuestro juicio, debemos fiarnos del cálculo algebraico.

- Leonhard Euler -

En el presente capítulo se hablará de la batería de problemas elegida para la validación de las codificaciones de las técnicas de resolución y se mostrarán los resultados obtenidos tras la aplicación del algoritmo de ahorros, y de una posterior búsqueda local, a dicha batería. Además se presentarán las distintas soluciones a uno de los problemas a modo de ejemplo y se finalizará el capítulo con la evolución que sigue un problema resuelto mediante una búsqueda local cuando se parte desde una solución inicial aleatoria.

6.1 Batería de problemas elegida para el testeo

La batería de problemas que se ha tomado para llevar a cabo la prueba de los algoritmos implementados ha sido la compuesta por los 12 problemas correspondientes al tipo R1 del conjunto de problemas de Solomon. Este conjunto se encuentra constituido por 56 problemas, que se agrupan en distintos tipos en función del ancho de sus ventanas temporales, el número de clientes que cuentan con dicha ventana temporal y la ubicación geográfica de los clientes. Se han escogido un conjunto de los problemas de Solomon por ser estos muy conocidos y un buen punto de referencia para la validación de técnicas heurísticas en problemas de tipo VRPTW.

En los problemas de tipo R1, las coordenadas geográficas son generadas de forma aleatoria y poseen un corto horizonte de programación. Esto quiere decir que los clientes no se agrupan por zonas, como si lo hacen en los problemas de tipo C, y que las ventanas temporales tienen una amplitud pequeña, al contrario que en los problemas de tipo 2 donde debido al largo horizonte temporal se generan menos rutas, ya que un mayor número de clientes puede ser atendido por el mismo vehículo. El motivo de la elección de los problemas de tipo R1 para este trabajo se debe a la aleatoriedad de la situación geográfica de los clientes, al corto horizonte de programación y a que un elevado número de los clientes que conforman los problemas de este tipo cuentan con una ventana temporal que comienza en un valor distinto a 0. Igualemente, podrían haber sido usados cualquiera de los otros tipos para la validación de las heurísticas.

Para cada uno de los grupos, Solomon diseñó problemas con 50 y 100 clientes, siendo los segundos los escogidos para este trabajo. Se muestra a continuación un fragmento correspondiente al primer problema de tipo R1 con 100 clientes (Figura 16), para que se pueda entender con mayor claridad toda la información que se conoce acerca de cada uno de los problemas.

CUSTOMER CUST NO.	XCOORD.	YCOORD.	DEMAND	READY TIME	DUE DATE	SERVICE TIME
0	35	35	0	0	230	0
1	41	49	10	161	171	10
2	35	17	7	50	60	10
3	55	45	13	116	126	10
4	55	20	19	149	159	10
5	15	30	26	34	44	10
6	25	30	3	99	109	10
7	20	50	5	81	91	10
8	10	43	9	95	105	10
9	55	60	16	97	107	10

Figura 16: Fragmento problema R101 Solomon

La primera columna corresponde al número de cliente, denotándose el depósito como cliente número 0. Las columnas segunda y tercera recogen las coordenadas x e y, respectivamente, del depósito y de cada uno de los clientes, mientras que las quinta y sexta almacenan los datos de inicio y fin de las ventanas temporales de cada uno de ellos. Se puede ver como el depósito también cuenta con un horizonte temporal, que va desde el instante 0 hasta el 230.

En la cuarta columna aparecen los datos relativos a la demanda de cada uno de los clientes y en la séptima el tiempo que se tardaría en atender a cada uno de ellos. En este trabajo no se tendrán en cuenta estos valores, se supone que la capacidad de los vehículos no es limitante (un único vehículo podría servir a todos los clientes si se cumpliesen las restricciones de tiempo, los productos se suponen pequeños) y se considera que la entrega de las mercancías es instantánea, no llevan un tiempo de servicio asociado.

La matriz completa estaría formada por 101 filas, la primera correspondiente al depósito y las 100 restantes a clientes. En la batería de Solomon cada fila se corresponde con un cliente, pero en el caso de nuestro trabajo esto no es así. Cada uno de los clientes cuenta con tres localizaciones donde puede ser servido, contando cada una de ellas con su ventana temporal correspondiente. En el fragmento anterior (Figura 16), se tiene una primera fila correspondiente al depósito y los datos de tres clientes. Las filas primera, segunda y tercera serían las tres posiciones donde puedo atender al primer cliente, las filas cuatro, cinco y seis las localizaciones del segundo, y así sucesivamente. Por tanto, haciendo uso de los problemas de Solomon con 100 clientes, se cuenta con 33 clientes como máximo en este trabajo.

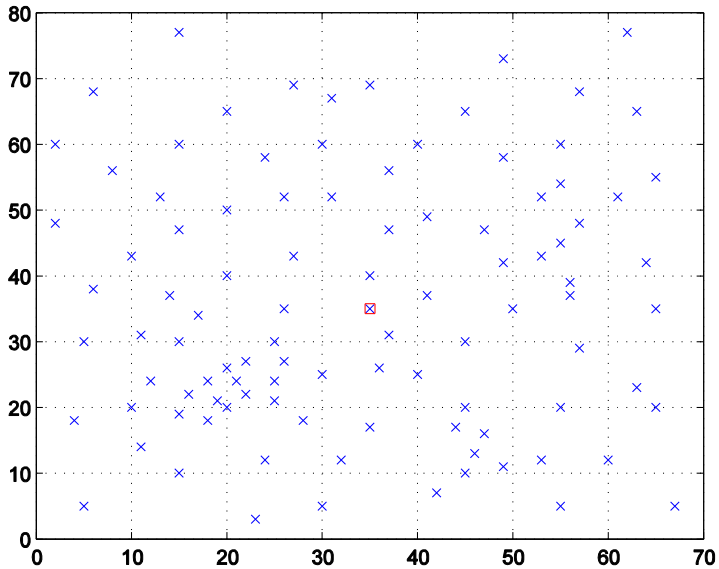

Figura 17: Localización clientes problema R101 Solomon

En la Figura 17 se puede ver la localización geográfica de los posibles lugares donde pueden ser atendidos cada uno de los 33 clientes del problema R101 de Solomon. Se representan 99 posiciones mediante cruces de color azul, 3 posibles por cada uno de ellos, además de un cuadrado rojo que se corresponde con la posición del depósito, lugar desde el que parten y al que tienen que volver todos los vehículos. Todos los problemas de la batería usada para este trabajo presentan la misma distribución, siendo la apertura de las ventanas temporales la variación existente entre ellos.

6.2 Resolución batería de problemas

Para la resolución de los problemas se supone un número de vehículos suficiente para poder servir a todos los clientes. La velocidad a la que se considerará que se desplazan los vehículos se tomará como 60 km/h. Esto se hace así para poder calcular fácilmente los tiempos de desplazamiento a partir de las distancias, ya que todas las distancias son euclídeas y se miden en km, por tanto se conoce que se tardará un minuto en recorrer un km.

Para poder comparar si una solución es mejor o peor que otra se debe determinar el coste de esta. Para ello se han tenido en cuenta dos factores: la distancia recorrida total por todos los vehículos (1 unidad por cada km), y un coste fijo que se imputa por cada vehículo que ha sido usado (10 unidades por cada uno).

A continuación, se muestran los resultados obtenidos tras la aplicación del algoritmo de ahorros, y de una posterior búsqueda local, a la batería de problemas de tipo R1 del conjunto de Solomon. Se han realizado pruebas con 10, 20 y 30 clientes para ver cómo se comportan los algoritmos según crece el tamaño del problema. Se presentan también, tras dichos resultados, otras tablas donde aparecen agrupadas a modo de resumen las puntuaciones para todos los problemas, resueltos mediante todos los métodos y para cada uno de los tres grupos de clientes tomados para el testeo.

Clientes	Nº problema	Clarke & Wright	
		Puntuación	Tiempo
10	1	299,5324	0,120800
	2	261,7715	0,144137
	3	236,6559	0,148346
	4	236,6559	0,120174
	5	294,6157	0,161380
	6	278,8182	0,143671
	7	237,6769	0,138276
	8	237,6769	0,128615
	9	336,1200	0,135063
	10	310,3354	0,173699
	11	237,6769	0,134511
	12	237,6769	0,137493

Tabla 1: Puntuaciones y tiempos para 10 clientes heurística Clarke & Wright

Clientes	Nº problema	Búsqueda Local: vecindad con permutaciones			
		Método 1		Método 2	
		Puntuación	Tiempo	Puntuación	Tiempo
10	1	230,1024	0,544201	226,8004	0,534911
	2	222,4417	0,307768	173,8618	0,791768
	3	172,0568	0,828918	157,1564	0,895075
	4	153,1206	0,784287	157,1564	0,854467
	5	222,8282	0,399199	215,5272	0,952287
	6	204,7843	0,650338	187,6667	0,952482
	7	185,1513	0,629742	179,5326	0,745585
	8	185,1513	0,614036	179,5326	0,704258
	9	235,3178	0,482361	236,3660	0,950523
	10	170,5967	0,956124	193,3566	1,081488
	11	185,1513	0,545467	179,4387	0,735201
	12	199,9191	0,309020	226,4893	0,430103

Tabla 2: Puntuaciones y tiempos para 10 clientes búsqueda local vecindad permutaciones

Clientes	Nº problema	Búsqueda Local: vecindad con inserciones			
		Método 1		Método 2	
		Puntuación	Tiempo	Puntuación	Tiempo
10	1	213,5967	1,020859	208,8731	1,339768
	2	166,3538	1,456839	184,4970	1,554461
	3	157,3350	2,258426	159,6624	2,272354
	4	160,5986	1,443341	158,6263	2,200572
	5	178,4291	1,267500	209,1687	1,826815
	6	222,9418	0,756811	206,5499	1,324202
	7	174,3156	1,054716	174,3156	1,435226
	8	174,3156	1,049616	174,3156	1,706955
	9	201,7007	2,003090	205,9174	3,319945
	10	181,3914	2,175412	188,8487	2,320402
	11	168,0337	1,601020	174,3156	1,868969
	12	169,8829	1,192320	152,4829	2,075403

Tabla 3: Puntuaciones y tiempos para 10 clientes búsqueda local vecindad inserciones

Clientes	Nº problema	Clarke & Wright	
		Puntuación	Tiempo
20	1	499,2771	0,120647
	2	465,1558	0,139641
	3	433,8996	0,140746
	4	354,4680	0,183882
	5	462,5223	0,129614
	6	446,1052	0,136174
	7	396,4921	0,143673
	8	343,2103	0,151602
	9	428,3147	0,143256
	10	416,2919	0,160913
	11	348,8699	0,147235
	12	330,6314	0,161096

Tabla 4: Puntuaciones y tiempos para 20 clientes heurística Clarke & Wright

Clientes	Nº problema	Búsqueda Local: vecindad con permutaciones			
		Método 1		Método 2	
		Puntuación	Tiempo	Puntuación	Tiempo
20	1	449,3557	3,313167	442,7754	9,758577
	2	341,6704	6,257906	373,8618	18,767067
	3	392,7888	2,643773	351,1125	11,742195
	4	316,5148	4,702098	276,9803	16,158000
	5	426,9198	2,245087	402,2678	18,335908
	6	386,5800	5,638611	474,2103	9,725565
	7	349,9076	2,947249	306,9216	9,802524
	8	291,6906	4,893160	341,4986	10,190698
	9	322,1337	6,658891	344,1149	10,094013
	10	301,0320	3,289544	335,5300	10,826490
	11	316,2528	4,011574	333,3913	15,603638
	12	311,9029	4,197758	280,6832	6,873702

Tabla 5: Puntuaciones y tiempos para 20 clientes búsqueda local vecindad permutaciones

Clientes	Nº problema	Búsqueda Local: vecindad con inserciones			
		Método 1		Método 2	
		Puntuación	Tiempo	Puntuación	Tiempo
20	1	369,6772	15,063326	453,0402	34,480022
	2	350,4290	11,882719	397,6339	33,593238
	3	331,0721	15,467865	339,7774	29,026689
	4	312,5281	8,369545	282,7157	27,539147
	5	384,4196	11,552373	374,7220	22,220702
	6	277,4442	15,899480	340,6327	43,804717
	7	274,0377	15,437032	293,4319	20,930106
	8	258,6673	13,244439	353,8508	20,588894
	9	275,5616	15,732113	323,7402	24,392258
	10	271,9747	13,622400	302,4128	30,854022
	11	295,2587	9,835189	358,3748	22,596307
	12	267,3160	11,952818	271,2816	14,753135

Tabla 6: Puntuaciones y tiempos para 20 clientes búsqueda local vecindad inserciones

Clientes	Nº problema	Clarke & Wright	
		Puntuación	Tiempo
30	1	752,0748	0,299049
	2	744,3181	0,171605
	3	684,2338	0,209229
	4	480,5293	0,188022
	5	591,2685	0,176133
	6	591,0510	0,144623
	7	589,1629	0,154543
	8	487,0556	0,190494
	9	570,2964	0,175256
	10	566,4384	0,186731
	11	480,5368	0,210877
	12	445,1042	0,193675

Tabla 7: Puntuaciones y tiempos para 30 clientes heurística Clarke & Wright

Clientes	Nº problema	Búsqueda Local: vecindad con permutaciones			
		Método 1		Método 2	
		Puntuación	Tiempo	Puntuación	Tiempo
30	1	658,5182	10,637752	640,2997	68,989474
	2	524,2344	23,072997	506,0029	125,147536
	3	467,3081	28,977166	472,0137	105,002750
	4	472,0665	18,808303	466,5601	42,497344
	5	577,3722	4,313338	620,6514	54,307804
	6	556,9581	15,173694	425,8257	146,712113
	7	518,2532	26,370181	468,5290	141,042426
	8	405,9979	30,198077	463,1006	49,351860
	9	514,0804	23,831370	521,9453	93,290880
	10	521,7652	18,187979	502,3470	65,778759
	11	454,8862	12,781340	511,5893	85,670411
	12	404,4078	21,138148	467,9180	48,019208

Tabla 8: Puntuaciones y tiempos para 30 clientes búsqueda local vecindad permutaciones

Clientes	Nº problema	Búsqueda Local: vecindad con inserciones			
		Método 1		Método 2	
		Puntuación	Tiempo	Puntuación	Tiempo
30	1	567,9909	43,763168	630,7430	150,778130
	2	430,8172	76,656596	539,7220	266,917077
	3	400,8346	67,118492	425,2416	317,052486
	4	444,2794	36,592508	420,1155	259,411624
	5	509,5244	36,053514	503,3740	277,999274
	6	435,7751	76,128718	472,9432	296,695067
	7	466,1763	55,066591	448,7839	146,078629
	8	433,5506	44,583999	423,2852	203,468471
	9	458,5636	59,323014	576,6667	165,160691
	10	433,7428	79,341376	500,4314	345,159802
	11	430,6126	44,853385	438,4423	275,921511
	12	364,3193	69,461244	440,5659	162,473748

Tabla 9: Puntuaciones y tiempos para 30 clientes búsqueda local vecindad inserciones

Clientes	Nº problema	Puntuación				
		C&W	Búsqueda Local			
			PM1	PM2	IM1	IM2
10	1	299,5324	230,1024	226,8004	213,5967	208,8731
	2	261,7715	222,4417	173,8618	166,3538	184,4970
	3	236,6559	172,0568	157,1564	157,3350	159,6624
	4	236,6559	153,1206	157,1564	160,5986	158,6263
	5	294,6157	222,8282	215,5272	178,4291	209,1687
	6	278,8182	204,7843	187,6667	222,9418	206,5499
	7	237,6769	185,1513	179,5326	174,3156	174,3156
	8	237,6769	185,1513	179,5326	174,3156	174,3156
	9	336,1200	235,3178	236,3660	201,7007	205,9174
	10	310,3354	170,5967	193,3566	181,3914	188,8487
	11	237,6769	185,1513	179,4387	168,0337	174,3156
	12	237,6769	199,9191	226,4893	169,8829	152,4829

Tabla 10: Puntuaciones para 10 clientes para cada uno de los métodos de resolución

Clientes	Nº problema	Puntuación				
		C&W	Búsqueda Local			
			PM1	PM2	IM1	IM2
20	1	499,2771	449,3557	442,7754	369,6772	453,0402
	2	465,1558	341,6704	373,8618	350,4290	397,6339
	3	433,8996	392,7888	351,1125	331,0721	339,7774
	4	354,4680	316,5148	276,9803	312,5281	282,7157
	5	462,5223	426,9198	402,2678	384,4196	374,7220
	6	446,1052	386,5800	474,2103	277,4442	340,6327
	7	396,4921	349,9076	306,9216	274,0377	293,4319
	8	343,2103	291,6906	341,4986	258,6673	353,8508
	9	428,3147	322,1337	344,1149	275,5616	323,7402
	10	416,2919	301,0320	335,5300	271,9747	302,4128
	11	348,8699	316,2528	333,3913	295,2587	358,3748
	12	330,6314	311,9029	280,6832	267,3160	271,2816

Tabla 11: Puntuaciones para 20 clientes para cada uno de los métodos de resolución

Clientes	Nº problema	Puntuación				
		C&W	Búsqueda Local			
			PM1	PM2	IM1	IM2
30	1	752,0748	658,5182	640,2997	567,9909	630,7430
	2	744,3181	524,2344	506,0029	430,8172	539,7220
	3	684,2338	467,3081	472,0137	400,8346	425,2416
	4	480,5293	472,0665	466,5601	444,2794	420,1155
	5	591,2685	577,3722	620,6514	509,5244	503,3740
	6	591,0510	556,9581	425,8257	435,7751	472,9432
	7	589,1629	518,2532	468,5290	466,1763	448,7839
	8	487,0556	405,9979	463,1006	433,5506	423,2852
	9	570,2964	514,0804	521,9453	458,5636	576,6667
	10	566,4384	521,7652	502,3470	433,7428	500,4314
	11	480,5368	454,8862	511,5893	430,6126	438,4423
	12	445,1042	404,4078	467,9180	364,3193	440,5659

Tabla 12: Puntuaciones para 30 clientes para cada uno de los métodos de resolución

Se deben especificar de forma detallada las características del ordenador con el que se lleven a cabo las pruebas de los algoritmos, ya que esto influirá sobre el tiempo de computación. Para este trabajo se ha usado:

- Ordenador: HP ProBook 4520s.
- Sistema operativo: Windows 7 Home Premium.
- Memoria RAM: 2,86 GB.
- Procesador: Intel Celeron CPU P4500 a 1,87 GHz.

A la vista de los resultados, se puede deducir que para problemas pequeños se encuentran soluciones muy rápidamente, pero el tiempo de computación se dispara según va creciendo el tamaño del problema. El algoritmo de ahorros no presenta apenas variaciones en el tiempo necesario para alcanzar una solución según aumenta el número de clientes. El problema se encuentra en la posterior aplicación de la búsqueda local (siendo los tiempos mayores cuando la vecindad ha sido creada mediante inserción), ya que el espacio de búsqueda de soluciones es más amplio. Los tiempos en las tablas vienen dados en segundos. Estos tiempos podrían verse reducidos fácilmente haciendo uso de un ordenador más potente.

6.3 Ejemplo de solución

Se muestra a continuación un ejemplo de solución para uno de los grupos de pruebas. Dichas soluciones han sido obtenidas tras hacer uso de la heurística de Clarke and Wright y de una posterior búsqueda local. Se desarrolla como ejemplo el problema número 3 con un grupo de 30 clientes, ya que la variación de la solución entre cada uno de los métodos es significativa. Se recuerda que el cliente rodeado por un círculo rojo se corresponde con el primer cliente a visitar en cada ruta, para que quede definida de forma clara la orientación de estas.

```
Solución algoritmo Clarke & Wright

routes =

    71    65    81     9     0     0     0
    32    90    63    11    49    19     0
    47    36    82     0     0     0     0
     3    78    29     0     0     0     0
    75    23    67    54    55     4    25
    42    16    44    86    38    14    59


puntuacion =

   684.2338

Elapsed time is 0.209229 seconds.
```

Figura 18: Solución problema 3, 30 clientes, algoritmo Clarke and Wright

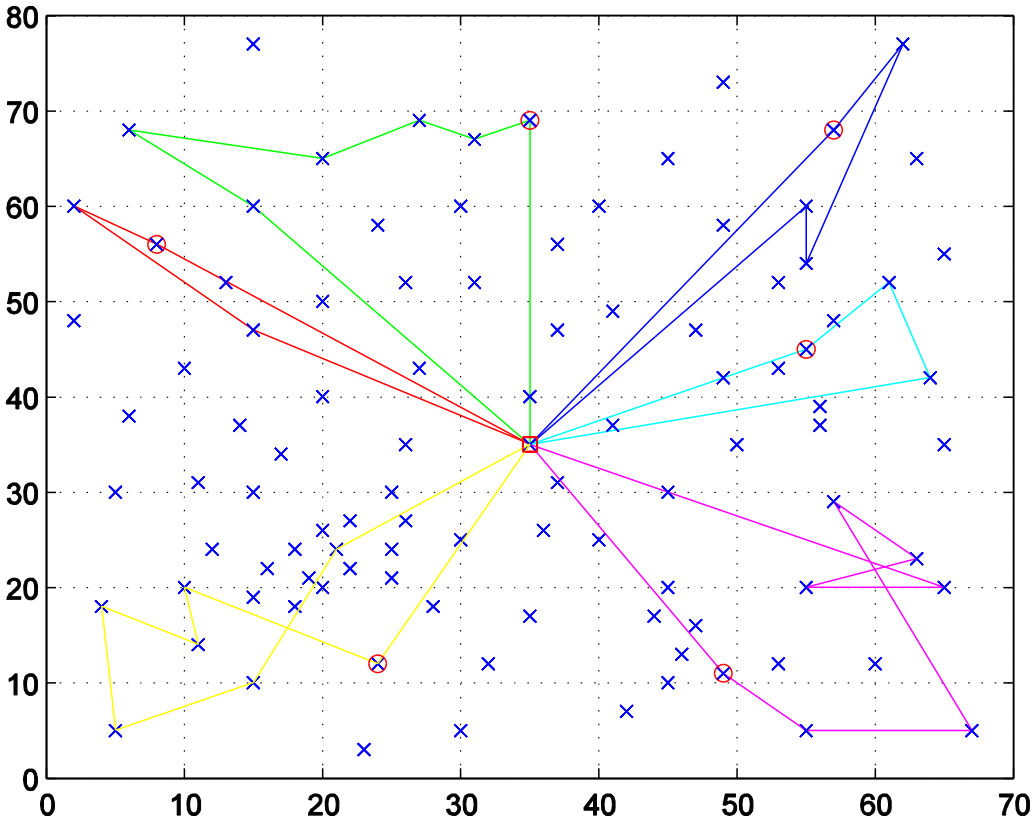

Figura 19: Representación gráfica solución problema 3, 30 clientes, algoritmo Clarke and Wright

```
Solución Búsqueda Local
Solución a partir de batería obtenida con permutaciones, método uno

goodcarspm1 =

    71    65    35    81    50     0     0     0     0     0
    90    63    11    19    48     7    31     1    77    28
    75    23    67    55    54    26     0     0     0     0
    42    37    44    86    16    84     5    59    13     0


bestpuntp1 =

  467.3081

Elapsed time is 28.977166 seconds.
```

Figura 20: Solución problema 3, 30 clientes, búsqueda local vecindad permutación, método uno

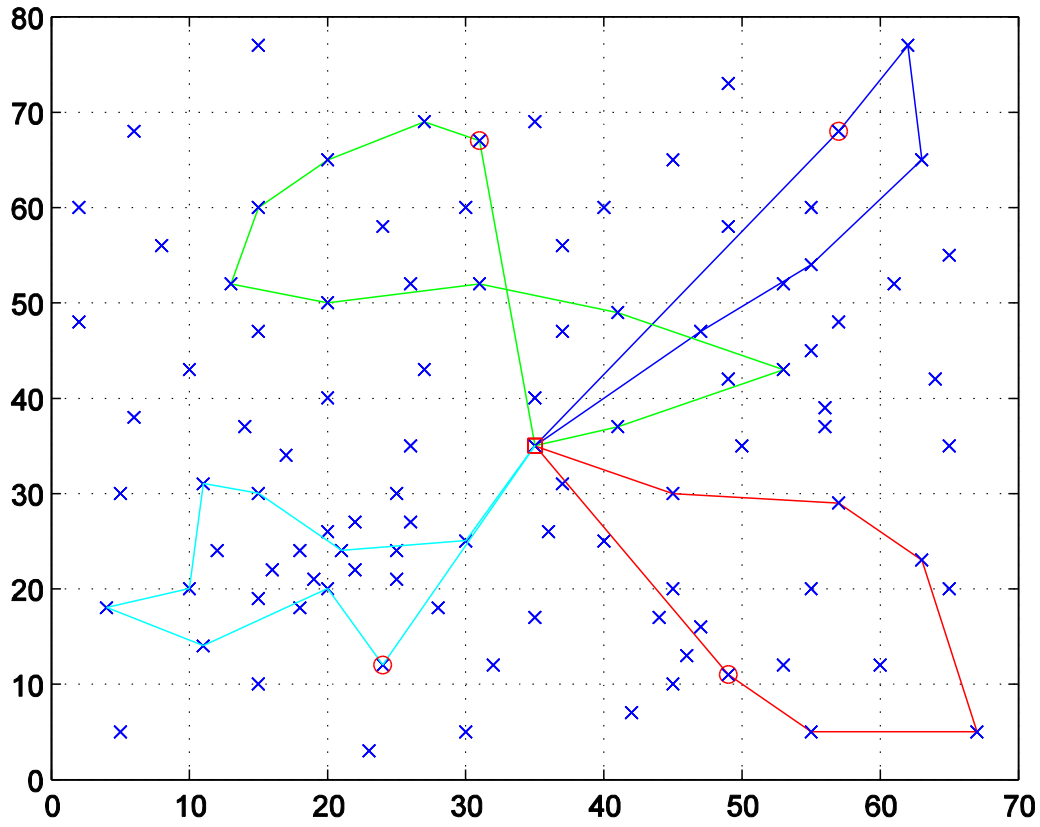

Figura 21: Representación gráfica solución problema 3, 30 clientes, búsqueda local vecindad permutación, método uno

```
Solución Búsqueda Local
Solución a partir de batería obtenida con permutaciones, método dos

goodcarspm2 =

    81    65    71     9    32    90    63    19    48    83    16    85    59
    76    68     3    34    29    24    54    26     0     0     0     0     0
    40    73    56     4    12    50     0     0     0     0     0     0     0
    44    37    13     0     0     0     0     0     0     0     0     0     0


bestpuntp2 =

  472.0137

Elapsed time is 105.002750 seconds.
```

Figura 22: Solución problema 3, 30 clientes, búsqueda local vecindad permutación, método dos

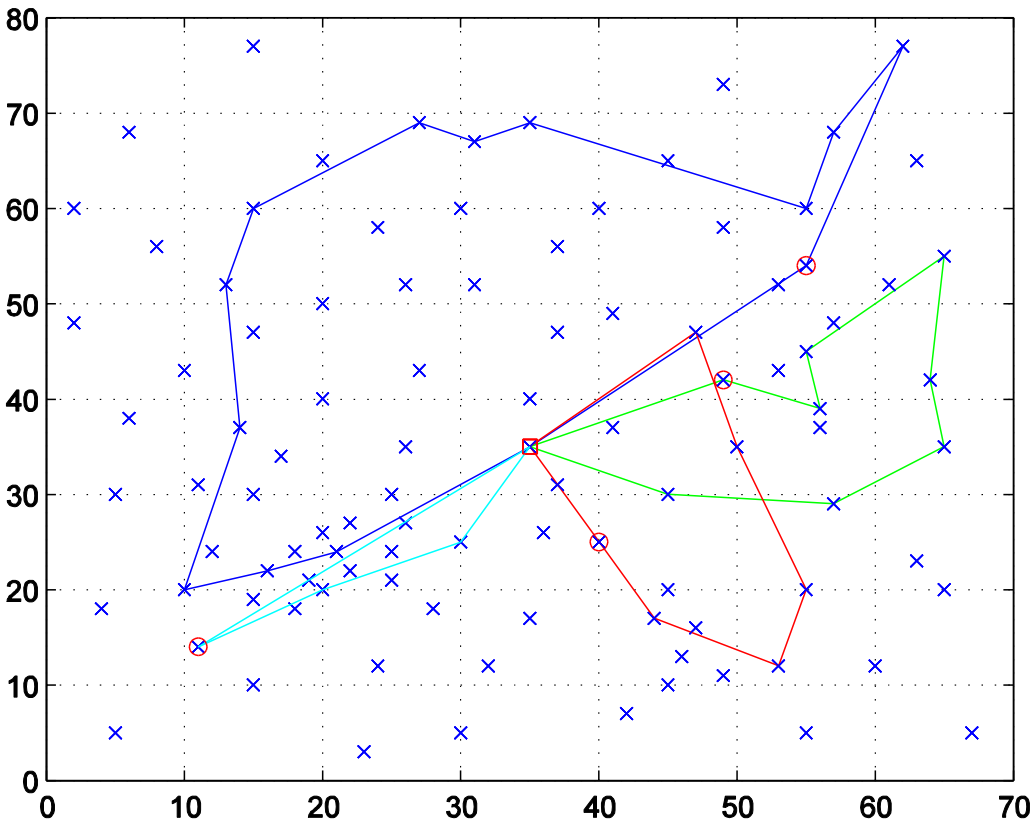

Figura 23: Representación gráfica solución problema 3, 30 clientes, búsqueda local vecindad permutación, método dos

```
Solución Búsqueda Local
Solución a partir de batería obtenida con inserciones, método uno

goodcarsim1 =

    51    71    65    34     9    81     3    68    77    28
    31    90    63    11    19    48    82    52     0     0
    40    22    75    56     4    26     0     0     0     0
    44    16    85    37    59    13     0     0     0     0


bestpunti1 =

  400.8346

Elapsed time is 67.118492 seconds.
```

Figura 24: Solución problema 3, 30 clientes, búsqueda local vecindad inserción, método uno

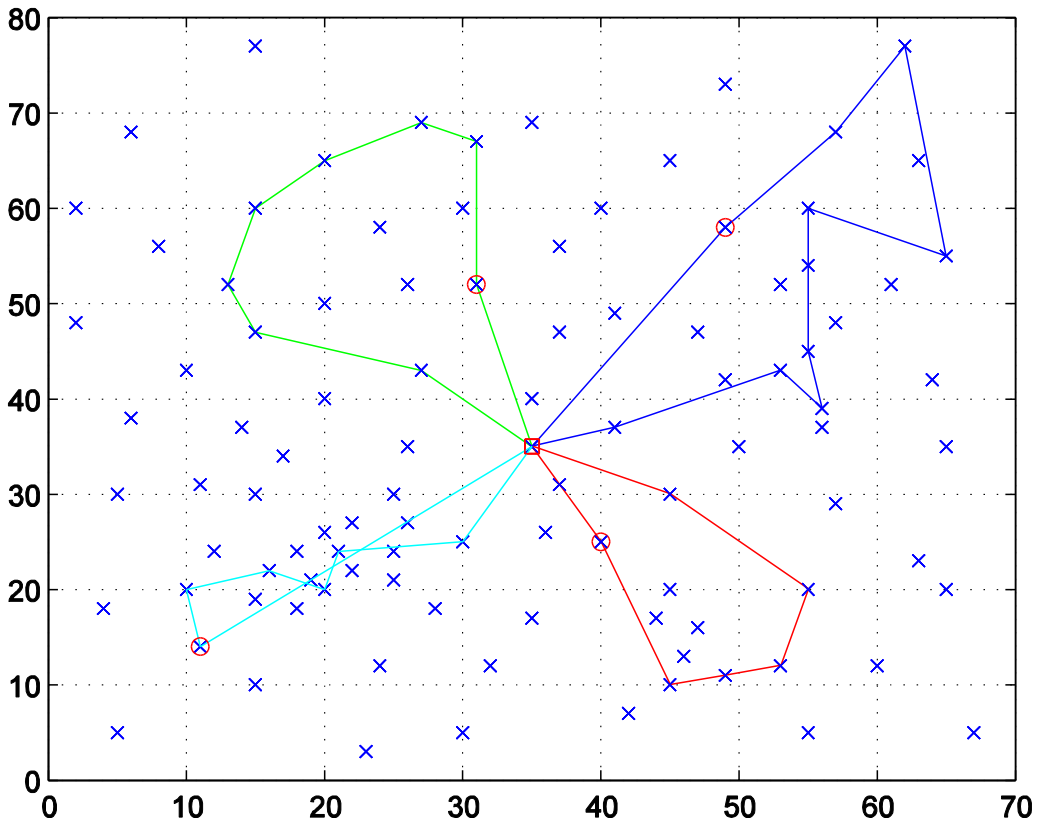

Figura 25: Representación gráfica solución problema 3, 30 clientes, búsqueda local vecindad inserción, método uno

```
Solución Búsqueda Local
Solución a partir de batería obtenida con inserciones, método dos

goodcarsim2 =

    34    65    71     9    81    68    77     3    12    26    28
    45    49    63    32    90    19    47    82    18    52     0
    40    73    22    57    13    59    37    85     5     0     0


bestpunti2 =

  425.2416

Elapsed time is 317.052486 seconds.
```

Figura 26: Solución problema 3, 30 clientes, búsqueda local vecindad inserción, método dos

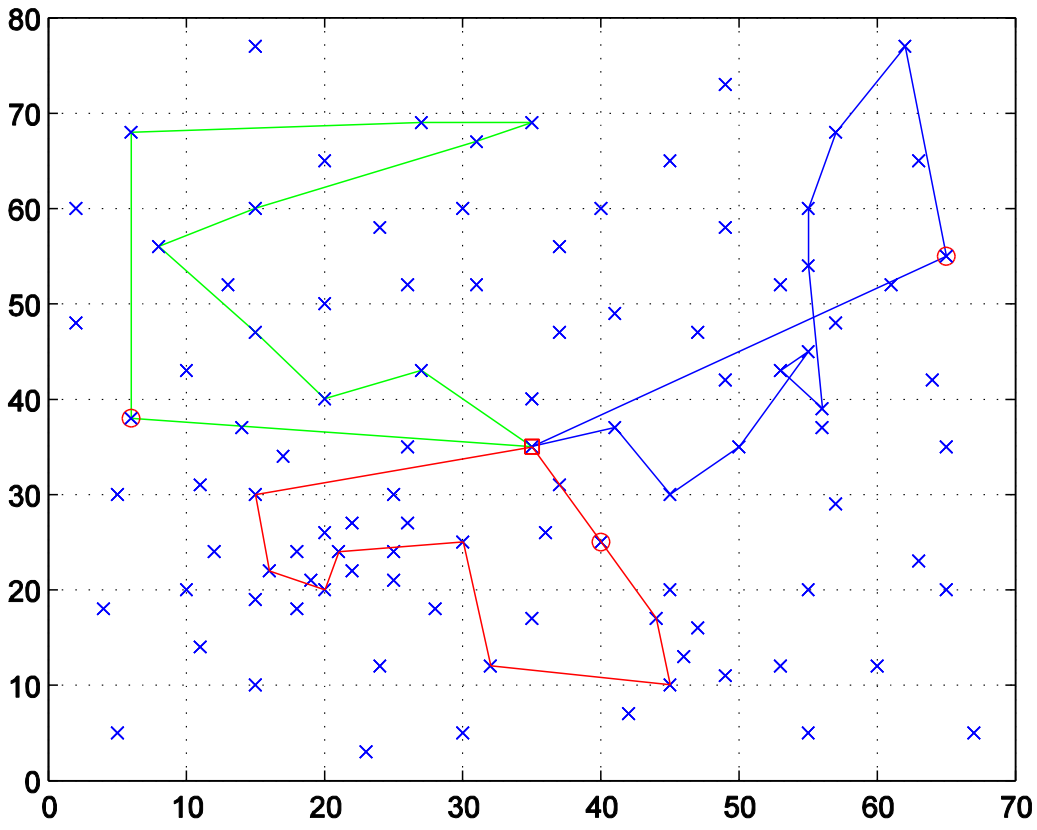

Figura 27: Representación gráfica solución problema 3, 30 clientes, búsqueda local vecindad inserción, método dos

Se observa como para cada uno de los métodos se llega a una solución distinta, siendo la mejor de ellas la obtenida mediante la aplicación de la búsqueda local donde la vecindad se crea mediante inserción y se construyen las soluciones siguiendo el método uno, ya que se trata de la que incurre en un menor coste.

6.4 Búsqueda local partiendo desde solución inicial aleatoria

Las soluciones dadas por la búsqueda local en este trabajo se encuentran condicionadas por la solución obtenida por el algoritmo de ahorros, ya que se toma esta como solución de partida para la aplicación de las búsquedas locales.

En este apartado se desarrolla un ejemplo de evolución de una búsqueda local que parte de una solución inicial aleatoria para 30 clientes. Se toman los datos del problema 1. Se hace uso de la búsqueda local donde la vecindad se crea mediante permutación y se construyen las soluciones siguiendo el método uno, pero podría haberse escogido cualquiera de los otros tres métodos de búsqueda local usados en el presente proyecto.

En primer lugar, se crea la solución aleatoria de la que se partirá, tal y como se muestra en la Figura 28.

```
solucion=[1:90];
solucion=solucion(randperm(length(solucion)));
```

Figura 28: Creación solución aleatoria de 30 clientes

Posteriormente se comienza a iterar, explorando el espacio de soluciones vecinas. En las Figuras 29, 30 y 31 se pueden ver las soluciones intermedias cuando se han realizado 5, 10 y 20 iteraciones respectivamente. Se conoce como iteración el proceso en el que se crea una vecindad a partir de la modificación de una solución inicial, y se escoge la mejor solución dentro de dicha vecindad, que pasará a ser la solución inicial en la siguiente iteración.

```
goodcarspm1 =

    65    81    10
    61    85    48
    56    89     0
    18     6     0
    44    13     0
    31    37     0
    22    74     0
    69     0     0
    27    52     0
     2    41    58
     7    34     0
    78    50    70
    28    21    84


bestpuntp1 =

  1.0224e+003
```

Figura 29: Solución búsqueda local, 30 clientes, solución inicial aleatoria, tras 5 iteraciones

```
goodcarspm1 =

    65    81    34     0     0     0     0
    61    85    48     0     0     0     0
    69    31    18     6    37    13    89
    55     0     0     0     0     0     0
    44    22    74     0     0     0     0
    27    52     0     0     0     0     0
     2    41    58     0     0     0     0
    82     7    10     0     0     0     0
    78    50    70     0     0     0     0
    28    21     0     0     0     0     0


bestpuntp1 =

  827.5050
```

Figura 30: Solución búsqueda local, 30 clientes, solución inicial aleatoria, tras 10 iteraciones

```
goodcarspm1 =

    65    81    34     0     0     0     0
    59    87    57     0     0     0     0
    69    52    18     6    37    13    89
    44    41    22    74     0     0     0
    28    27    31    62    19    48     0
    82     7     0     0     0     0     0
    12    76    50     1    70     0     0


bestpuntp1 =

  620.5186
```

Figura 31: Solución búsqueda local, 30 clientes, solución inicial aleatoria, tras 20 iteraciones

Se finaliza la búsqueda local cuando no se puede encontrar ninguna solución con un coste menor. En la Figura 32 se muestra la solución final para este problema. Se puede comprobar que se ha llegado a una solución mejor incluso que la que se obtuvo partiendo de la solución dada por el algoritmo de ahorros para 30 clientes, que era de 658,5182. Sin embargo, el tiempo empleado para alcanzar la solución final ha sido mucho mayor, ya que el obtenido haciendo uso de una solución inicial dada fue de 10,637752 segundos.

```
goodcarspm1 =

    65    81    34     0     0     0
    59    87    57     0     0     0
    27    18     6    37    13    89
    44    41    22    74     0     0
    28    69    31    62    19    48
    52    82     7     0     0     0
    12    76    50     1    70     0


bestpuntp1 =

   609.8292

Elapsed time is 121.473553 seconds.
```

Figura 32: Solución final búsqueda local, 30 clientes, solución inicial aleatoria

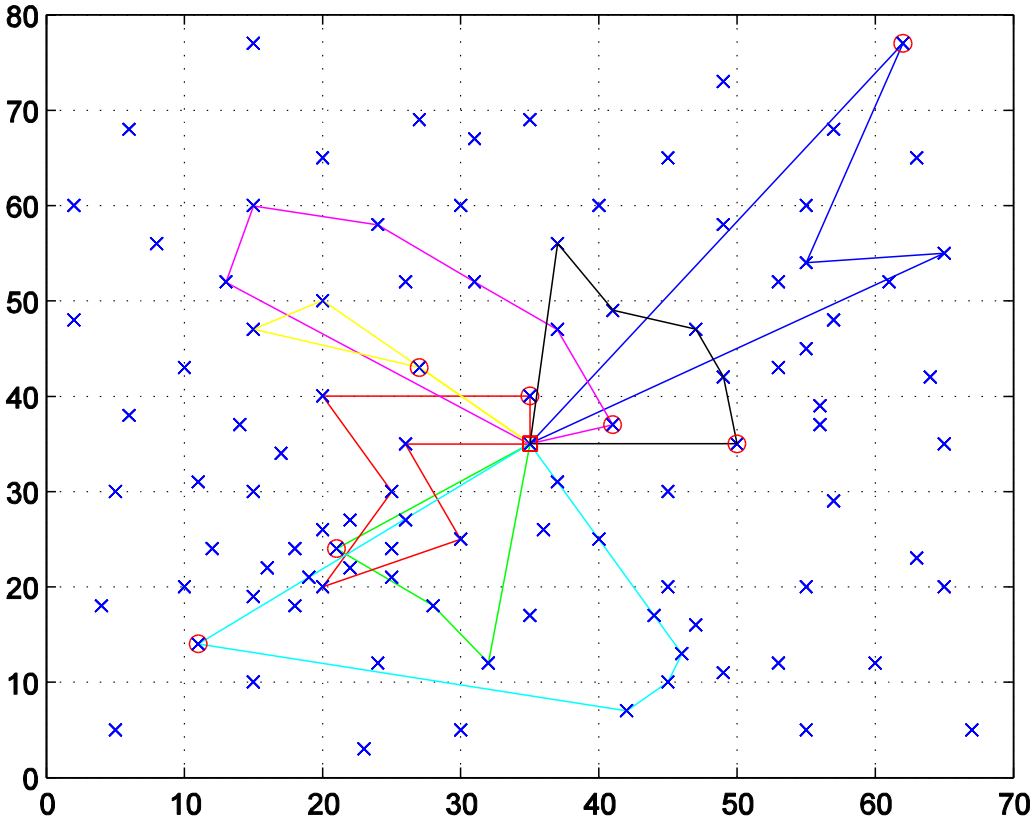

Figura 33: Representación gráfica solución final búsqueda local, 30 clientes, solución inicial aleatoria

7 CONCLUSIONES

Poca gente es capaz de prever hacia donde les lleva el camino hasta que llegan a su fin.

- J.R.R. Tolkien -

Una vez mostrados los resultados que se han obtenido tras la resolución de la batería de problemas mediante la heurística de ahorros y la aplicación posterior de diversas búsquedas locales, se pueden realizar una serie de conclusiones para determinar si se ha cumplido con los objetivos marcados al inicio de este trabajo.

Se presenta a continuación un breve resumen de los pasos dados durante la realización del proyecto:

- Para comenzar, se explica el problema general de diseño de rutas de vehículos y se hace una breve introducción al comercio electrónico, con el fin de establecer el escenario del que se parte.
- A continuación, se profundiza en la explicación del problema VRP y en el caso concreto del VRPTW, ya que el problema a resolver se encuentra dentro de dicha categoría.
- Posteriormente, se ha hablado de los tres grupos en los que se clasifican los métodos de resolución usados para hallar una solución a este problema (métodos exactos, heurísticas y metaheurísticas), viéndose algunos de los ejemplos más utilizados para cada una de las categorías.
- Tras esto, se han explicado detalladamente las características de los problemas a resolver, especificando todas aquellas consideraciones que se deben tener en cuenta y todas aquellas particularidades que presentan frente al problema clásico de VRPTW.
- Para finalizar, se ha aplicado el algoritmo de ahorros a la batería de problemas escogida y se han realizado búsquedas locales partiendo de las soluciones obtenidas, de modo que al final se obtienen cinco soluciones distintas para cada uno de los problemas a resolver. También se ha hecho una búsqueda local partiendo desde una solución inicial aleatoria para ver cómo reaccionaría el algoritmo implementado en dicho caso.

En la realización de este trabajo se planteó como objetivo principal la optimización de rutas de transporte, con el fin de obtener una minimización de los costes asociados al problema. Se pretendía conseguir esto mediante la implementación de técnicas no exactas para llevar a cabo la resolución del problema.

A la vista de los resultados, resumidos en las Tablas 10, 11 y 12 presentadas en el capítulo anterior, se puede concluir diciendo que las técnicas usadas en este trabajo para el diseño de rutas de transporte son buenos métodos de resolución, ya que se pueden obtener buenas soluciones en un tiempo de computación pequeño. Haciendo uso del algoritmo de Clarke & Wright se consiguen soluciones en un intervalo de tiempo muy corto, pudiéndo mejorarse estas con la aplicación posterior de una búsqueda local para explorar los alrededores. Se aprecia esto en dichas Tablas 10, 11 y 12 mencionadas anteriormente, donde para cada uno de los cuatro tipos de búsquedas locales realizadas se ve que las soluciones obtenidas tienen un menor coste que las del algoritmo de ahorros.

En el caso de la búsqueda local donde se parte de una solución inicial aleatoria, la solución a la que se llega suele ser mejor porque el espacio de búsqueda de soluciones explorado es mayor, pero se tarda un poco más de tiempo en alcanzarla.

Para testar la calidad de estos resultados, sería necesaria la comparación con resultados obtenidos a partir de otros métodos de resolución. Si se comparasen con los resultados obtenidos mediante un método exacto, se vería que las soluciones se acercan bastante al óptimo y que los tiempos de computación son mucho menores. Además, hay que tener en cuenta que los tiempos de computación en la búsqueda de soluciones por métodos exactos se disparan conforme se hace mayor el tamaño del problema, llegando a veces incluso a no converger si el problema es muy grande, por lo que se hace interesante el uso de estos métodos heurísticos, ya que la solución obtenida es bastante buena y se halla de forma rápida.

Como último detalle se comenta que a los algoritmos desarrollados se le podrían hacer una serie de mejoras, ya que la búsqueda local puede quedar atrapada en un óptimo local. Para escapar de estas situaciones, se puede llevar a cabo alguno de los procedimientos descritos en el apartado 3 del capítulo 4 de este trabajo, que pueden ser de ayuda para conseguir explorar de forma más extensa el espacio de búsqueda de soluciones.

8 BIBLIOGRAFÍA

[1] Aarts, E. & Lenstra, J. *Local Search in combinatorial optimization.* John Wiley & Sons, 2003.

[2] Ashour, S., Vega, J. & Parker, R.G. "A heuristic for travelling salesman problems". *Transportation Research, V6*, 1972, pp. 187-195.

[3] Blanton, J. & Wainwright, R. "Multiple vehicle routing with time and capacity constraints using genetic algorithms". *Proceedings of the 5th International Conference on Genetic Algorithms*. 1993, pp. 452–459.

[4] Bodin, L., Golden, B., Assad, A. & Ball, M. "Routing and scheduling of vehicles and crews – the state of the art", *Computers and Operations Research, V10(2)*, 1983, pp. 63–211.

[5] Breedam, V. "Improvement heuristics for the vehicle routing problem based on simulated annealing". *European Journal of Operational Research, V86*, 1995, pp. 480-490.

[6] Clarke, G. & Wright, J.W. "Scheduling of vehicles from a central depot to a number of delivery points". *Operations Research, V12*, 1964, pp. 568-581.

[7] Christofides, N., Mingozzi, A. & Toth, P. "The vehicle routing problem". *Combinatorial Optimization*, Wiley, Chichester, UK, 1979, pp. 315-338.

[8] Cordeau, J-F., Gendreau, M., Laporte, G., Potvin, J-Y. & Semet, F. "A guide to vehicle routing heuristics". *Journal of the Operational Research Society, V53*, 2002, pp. 512–522.

[9] Díaz, A., Glover, F., Ghaziri, H.M., González, J.L., Laguna, M., Moscato, P. & Tseng, F.T. *Optimización Heurística y Redes Neuronales.* Madrid: Paraninfo, 1996.

[10] Dumas, Y., Desrosiers, J., Gelinas, E. & Solomon, M. "An optimal algorithm for the Travelling Salesman Problem." *Operations Research, V43*, 1995, pp. 367-371.

[11] Escobar, M. *El Comercio Electrónico, perspectiva presente y futura en España*. Fundación AUNA, 2000.

[12] Ghiani, G. & Improta, G. *An efficient transformation of the generalized vehicle routing problem. European Journal of Operational Research, V122*, 2000, pp. 11–17.

[13] Glover, F. & Kochenberger, G.A. *Handbook of Metaheuristics*. Boston: Kluwer Academic Publishers, 2003.

[14] Guting, G. & Punnen, A. *The Traveling Salesman Problem and its variations*. Dordrecht: Kluwer Academic Publishers, 2002.

[15] Kallehauge, B. "Formulations and exact algorithms for the vehicle routing problem with time windows". *Computers & Operations Research, V35*, 2008, pp. 2307–2330.

[16] Kontoravdis, G., Bard, J.F. "A GRASP for the Vehicle Routing Problem with Time Windows". *ORSA Journal on Computing, V7(1)*, 1995, pp. 10-23.

[17] Laporte, G., Gengreau, M., Potvin, J. & Semet, F. "Classical and modern heuristics for the vehicle routing problem". *International Transactions in Operational Research, V7*, 2000, pp. 285-300.

[18] Laporte, G. “The vehicle routing problem: an overview of exact and approximate algorithms”. *European Journal of Operational Research, V59,* 1992, pp. 345-358.

[19] Martí, R. "Procedimientos Metaheurísticos en Optimización Combinatoria." *Matemàtiques, V1,* pp. 3-62.

[20] Ministerio de Fomento de España. "Informe anual 2015 del Observatorio del Transporte y la Logística en España". 2016.

[21] Ministerio de Fomento de España. "Observatorio del transporte de mercancías por carretera. Oferta y demanda". 2016.

[22] Osman, I.H., & Kelly, J.P. *Meta-Heuristics.* 1996.

[23] Perboli, G., Tadei, R. & Vigo, D. “The two-echelon capacitated vehicle routing problem: models and math-based heuristics”. *Transportation Science, V45,* 2011, pp. 364–380.

[24] Rodriguez, J.P., Slack, B. & Comtois, C. *The geography of Transport systems.* Hofstra University, 2006.

[25] Savelsbergh, M. W. P. “Local search in routing problems with time windows”. *Annals of Operations Research, V4,* 1986, pp. 285-305.

[26] Solomon, M. “Algorithms for the vehicle routing and scheduling problem with time window constraints”. *Operations Research, V35(2),* 1987, pp. 254–264.

[27] Suárez, A. *Nueva Sociedad y Nueva Economía. Los grandes desafíos del S. XXI*. Pearson Educación, 2001.

[28] Tan, K.C., Lee, L.H. & Zhu, K.Q. *Heuristic Methods for Vehicle Routing Problem with Time Windows.* Fort Lauderdale, Florida: Proceedings of the 6th International Symposium on Artificial Intelligence and Mathematics, 2000.

[29] Toth, P. & Vigo, D. "The vehicle routing problem." *SIAM Monographs on Discrete Mathematics and Applications, V9,* 2002, pp. 157-193.

ANEXO: CÓDIGOS DE PROGRAMACIÓN

Se presentan a continuación los algoritmos implementados en lenguaje de programación M usados para resolver el problema de diseño de rutas de vehículos en el presente trabajo.

1. Heurística Clarke & Wright.

```
%Algoritmo de Clarke & Wright
%Las restricciones son las ventanas temporales en lugar de la capacidad
%Se ordenan en función de distancia y tiempo de espera
clear
clc

%Se calcula el tiempo de ejecución
tic

%Se cargan los datos correspondientes a un problema
numprob=input('Introducir número de problema del que queremos cargar los
datos [1-12]: ');
[px]=cargar_datos(numprob);

% Se pide el número de clientes, para poder limitar el tamaño de los
% problemas en las iteraciones de prueba
numCustomer=input('Introducir número de clientes [1-33]: ');

%% Definición del problema
id=(1:3*numCustomer)';                % Número de identificación unívoco
customer=ones(3,1)*(1:numCustomer);
customer=customer(:);                 % Cliente al que pertenecen los datos
customerExe=(1:3)'*ones(1,numCustomer);
customerExe=customerExe(:);           % Posibilidad dentro del cliente
x=px(2:3*numCustomer+1,2);            % Coordenada x
y=px(2:3*numCustomer+1,3);            % Coordenada y

%Se calcula la distancia al depósito desde cada cliente
x1dep=px(1,2);
y1dep=px(1,3);
x2cliente=px(2:3*numCustomer+1,2)';
y2cliente=px(2:3*numCustomer+1,3)';
x1dep=x1dep*ones(1,3*numCustomer);
y1dep=y1dep*ones(1,3*numCustomer);
distanciadep=sqrt((x2cliente-x1dep).^2+(y2cliente-y1dep).^2);


%Ventanas temporales reales (no siempre se llega desde el depósito antes de
%que empiecen, se tiene que tener en cuenta el tiempo de desplazamiento)
for vtw=1:length(distanciadep)
    if distanciadep(vtw)>px(vtw+1,4)
        E1(vtw)=distanciadep(vtw);
        E2(vtw)=distanciadep(vtw);
    else
        E1(vtw)=px(vtw+1,4);
        E2(vtw)=px(vtw+1,4);
    end
```

```
end
E1=E1';
E2=E2';
for vtw2=1:length(distanciadep)
    if px(vtw2+1,5)+distanciadep(vtw2)<px(1,5)
        L1(vtw2)=px(vtw2+1,5);
        L2(vtw2)=px(vtw2+1,5);
    else
        L1(vtw2)=px(1,5)-distanciadep(vtw2);
        L2(vtw2)=px(1,5)-distanciadep(vtw2);
    end
end
L1=L1';
L2=L2';


x1=px(2:3*numCustomer+1,2);
y1=px(2:3*numCustomer+1,3);
x2=px(2:3*numCustomer+1,2);
y2=px(2:3*numCustomer+1,3);
x1=x1*ones(1,3*numCustomer);
y1=y1*ones(1,3*numCustomer);
x2=(x2*ones(1,3*numCustomer))';
y2=(y2*ones(1,3*numCustomer))';

distancia=sqrt((x2-x1).^2+(y2-y1).^2); %necesario para poder actualizar
tiempos


%% Construcción de la solución

 % Solución inicial - Se sirven todos los clientes y todas las
 % realizaciones de los mismos a través de rutas individuales
 % No es una solución factible dado que de cada cliente sólo es necesario
 % servir una realización

routes=(1:3*numCustomer)'; %Los que contiene son id
bandera1=1;
bandera2=1;
inicio=0;


% Se empieza una heurística que trata de unir rutas factibles con máximo
% ahorro
if numCustomer>1
while(bandera1==1) %condición de parada, que no se puedan hacer más uniones,
no
    %haya ahorro

    % Número actual de rutas existentes
    numRoute=size(routes,1);

    % Se buscan los orígenes y destinos de las uniones de rutas (r1r2)
    % El destino coincide con la primera tarea de una ruta
    % El origen coincide con el final de una ruta
    D=routes(:,1);    % Destino (r2)

    O=D;              % Origen (r1)

    for i=2:size(routes,2)
        ind=routes(:,i)~=0;
```

```
        O=routes(:,i).*ind+O.*(~ind);
    end

    % Para cada ruta se calculan los tiempos iE, iL, fE, fL
    % fE fL posibilidad de terminar en r1 (Origen)
    % iE iL posibilidad de iniciar la ruta r2 (Destino)

    % Al inicio coinciden con L y E de O y D.
    if (inicio==0)
    iE=E1(D);
    iL=L1(D);
    fE=E1(O);
    fL=L1(O);
    inicio=1;
    elseif (inicio==1)
    iE=E1(D);
    iL=L2(D);
    fE=E2(O);
    fL=L1(O);
    end



    % Se calculan las distancias desde el final de la última tarea de r1
    % (O) al principio de la ruta r2 (D).
    xO=x(O);
    yO=y(O);
    xD=x(D);
    yD=y(D);

    xO=xO*ones(1,numRoute);      % Se generan unas matrices que facilitan
    yO=yO*ones(1,numRoute);      % los cálculos
    xD=(xD*ones(1,numRoute))';
    yD=(yD*ones(1,numRoute))';

    distanceOD=sqrt((xD-xO).^2+(yD-yO).^2);

    % Se analizan las uniones factibles en tiempo fE+distanceOD<=iL
    iL=(iL*ones(1,numRoute))';      % Se generan matrices por facilidad de
    fE=fE*ones(1,numRoute);  % cálculo
    feasible1=fE+distanceOD<=iL;

    % Tampoco son factibles las uniones entre realizaciones de un mismo
    % cliente
    aux1=customer(O)*ones(1,numRoute);
    aux2=(customer(D)*ones(1,numRoute))';
    feasible2=aux1-aux2;
    feasible2(feasible2~=0)=1;
    feasible2=feasible2.*(ones(numRoute)-eye(numRoute)); %corrección para
cuando se forman rutas
    %Al formarse nuevas rutas, se modifica la matriz de factibilidad

    % Se calculan los ahorros de las uniones de rutas
    xDepot=px(1,2).*ones(numRoute,1)*ones(1,numRoute);  % Posición de
Depósito
    yDepot=px(1,3).*ones(numRoute,1)*ones(1,numRoute);
    distanceODepot=sqrt((xDepot-xO).^2+(yDepot-yO).^2); % Distancia a
Depósito
    distanceDepotD=sqrt((xDepot-xD).^2+(yDepot-yD).^2);

    %Se comprueba que de tiempo a volver al depósito
    feasible3=fE+distanceOD+distanceODepot<=px(1,5);
```

```
feasible=feasible1.*feasible2.*feasible3;

saving=distanceODepot+distanceDepotD-distanceOD;    % Ahorro

saving=saving.*feasible;    % Sólo de aquellas uniones factibles

% Cálculo de tiempo de espera mínimo de las uniones
fL=fL*ones(1,numRoute);
iE=(iE*ones(1,numRoute))';

minwait=iE-(distanceOD+fL);

minwait=max(0,minwait).*feasible;   % Sólo de las uniones factibles


% Se elige una unión de parámetros

%PARÁMETROS DE CONFIGURACIÓN DEL ALGORITMO
alfa=1; %coeficiente para el ahorro
beta=1; %coeficiente para el tiempo de espera

%Ojo con esto, tiene que ser máximo y mayor que 0
bestmerging=alfa*saving-beta*minwait;
%Condición: si hay bestmerging>0, coja ese, si no que mire max saving
if (bestmerging(bestmerging>0))
    [r1ind, r2ind]=find(bestmerging==max(max((bestmerging))),1,'first');
else
    if (saving(saving>0))
        [r1ind, r2ind]=find(saving==max(max((saving))),1,'first');
    else
        bandera1=0;
        bandera2=0;
        iE=iE(1,:)';
        fL=fL(:,1);
        iL=iL(1,:)';
        fE=fE(:,1);
    end
end

if bandera2==1;
% Se extrae la r1 a unir
r1=routes(r1ind,:);
r1(r1==0)=[];

% Se extrae la r2 a unir
r2=routes(r2ind,:);
r2(r2==0)=[];

% Se crea la nueva ruta unión de r1 r2
newRoute=[r1 r2];

% De la lista de rutas, se extrae todo lo relacionado con r1 y r2

% Si son rutas formadas
if(r1ind>r2ind)
    if length(r1)>1
        routes(r1ind,:)=[];
    end
    if length(r2)>1
        routes(r2ind,:)=[];
    end
```

```
    else
        if length(r2)>1
            routes(r2ind,:)=[];
        end
        if length(r1)>1
            routes(r1ind,:)=[];
        end
    end


    %Todavía rutas de un solo cliente - Hay que buscar todas las posibles
    %realizaciones del customer y quitarlo.
    if length(r1)==1
        customer1=customer(r1);
        customer1id=id(customer1==customer);
        for i=1:length(customer1id)
            [row column]=find(routes==customer1id(i));
            routes(row,:)=[];
        end
    end
    if length(r2)==1
        customer2=customer(r2);
        customer2id=id(customer2==customer);
        for i=1:length(customer2id)
            [row column]=find(routes==customer2id(i));
            routes(row,:)=[];
        end
    end


    % Se añade la nueva ruta a la lista
    %las rutas se añaden al final del todo
    routes(size(routes,1)+1,1:length(newRoute))=newRoute


    %Actualizar tiempos conforme se construyen las rutas
    %El inicio más temprano se corresponde con el inicio de la ventana
    %temporal de la primera tarea de la ruta
    iE=E1(D);
    iE(size(iE,1)+1)=E1(newRoute(1));



    %La finalización más tardía se corresponde con el fin de la ventana
    %temporal de la última tarea de la ruta
    fL=L1(O);
    fL(size(fL,1)+1)=L1(newRoute(length(newRoute)));



    %Inicio más tardío
    iL=L2(D);
    iL(size(iL,1)+1)=L2(newRoute(length(newRoute)));
    for it=length(newRoute):-1:2
        if iL(size(iL,1))-distancia(newRoute(it-
1),newRoute(it))>L2(newRoute(it-1));
            iL(size(iL,1))=L2(newRoute(it-1));
        else
            iL(size(iL,1))=iL(size(iL,1))-distancia(newRoute(it-
1),newRoute(it));
        end
    end
```

```
    L2(newRoute(1))=iL(size(iL,1));


    %Finalización más temprana
    fE=E2(O);
    fE(size(fE,1)+1)=E2(newRoute(1));
    for it=1:length(newRoute)-1
        if
fE(size(fE,1))+distancia(newRoute(it),newRoute(it+1))<E2(newRoute(it+1));
            fE(size(fE,1))=E2(newRoute(it+1));
        else

fE(size(fE,1))=fE(size(fE,1))+distancia(newRoute(it),newRoute(it+1));
        end
    end
    E2(newRoute(size(newRoute,2)))=fE(size(fE,1));

    end
end

%Clientes que queden sueltos, criterio para seleccionar una de las tres
%posibilidades: cliente que esté más cerca
s=1;
c=1;
t=size(routes,1);
corregidos=0;
while  s<t
    if (routes(s,1)~=0) && (routes(s,2)==0) && (routes(s,1)~=corregidos)
        %Buscar de las 3 posibilidades cuál está más cerca
        dis1w=sqrt((px(find(px(:,1)==routes(s,1)),2)-
px(1,2)).^2+(px(find(px(:,1)==routes(s,1)),3)-px(1,3)).^2); %primera ventana
        dis2w=sqrt((px(find(px(:,1)==routes(s+1,1)),2)-
px(1,2)).^2+(px(find(px(:,1)==routes(s+1,1)),3)-px(1,3)).^2); %segunda
ventana
        dis3w=sqrt((px(find(px(:,1)==routes(s+2,1)),2)-
px(1,2)).^2+(px(find(px(:,1)==routes(s+2,1)),3)-px(1,3)).^2); %tercera
ventana
        disw=[dis1w,dis2w,dis3w];

        if(min(disw)==dis1w)
            fila=s;
        elseif(min(disw)==dis2w)
            fila=s+1;
        elseif(min(disw)==dis3w)
            fila=s+2;
        end

        routes(size(routes,1)+1,1:length(routes(s,:)))=routes(fila,:);
        corregidos(c)=routes(fila,1);
        c=c+1;
        routes(s+2,:)=[];
        routes(s+1,:)=[];
        routes(s,:)=[];
        t=size(routes,1);
        s=s-1;
    end
    s=s+1;
end
else
    s=1;
    %Buscar de las 3 posibilidades cuál está más cerca
        dis1w=sqrt((px(find(px(:,1)==routes(s,1)),2)-
px(1,2)).^2+(px(find(px(:,1)==routes(s,1)),3)-px(1,3)).^2); %primera ventana
```

```
        dis2w=sqrt((px(find(px(:,1)==routes(s+1,1)),2)-
px(1,2)).^2+(px(find(px(:,1)==routes(s+1,1)),3)-px(1,3)).^2); %segunda
ventana
        dis3w=sqrt((px(find(px(:,1)==routes(s+2,1)),2)-
px(1,2)).^2+(px(find(px(:,1)==routes(s+2,1)),3)-px(1,3)).^2); %tercera
ventana
        disw=[dis1w,dis2w,dis3w];

        if(min(disw)==dis1w)
            fila=s;
        elseif(min(disw)==dis2w)
            fila=s+1;
        elseif(min(disw)==dis3w)
            fila=s+2;
        end

        routes=routes(fila);
end
display('Solución algoritmo Clarke & Wright')
routes
[puntuacion]=evaluar(px,routes,numCustomer)

%Fin tiempo ejecución
toc
%% Consideraciones importantes:
% cuando termina el bloque antes, hay que ver si existe algún customer que
% sigue sin unirse, en tal caso dejar una sola realización.
% A la hora de calcular la mejor unión, hay que ser conscientes de:
% - Primero aquellas que son positivas
% - Si no quedan positivas, ver si alguna es positiva obviando el waiting
% time
```

2. Algoritmo Búsqueda Local vecindad creada mediante permutación y construcción de soluciones mediante método uno.

```
%Algoritmo de Búsqueda Local
%Permutación para crear vecindad
% 1)rellenar coche mientras se pueda, saltar a nuevo cuando no

%Se calcula el tiempo de ejecución
tic

%Se convierte la solución inicial en un vector con el que se pueda trabajar
[solucion]=transformar(routes,numCustomer);
solucionpm1=solucion;

%Se reserva memoria para una matriz donde se almacenan las evaluaciones
resultados=zeros((numCustomer*3)*((numCustomer*3+1)/2)-numCustomer*3,1);

%Se evalua la solución de partida
[nscars1]=cars1(px,solucionpm1,numCustomer);
resultados(1)=evaluar(px,nscars1,numCustomer);
bestpuntp1=resultados(1);

bandera=0; %bandera para acabar cuando no se encuentre mejor solución

%% Se generan nuevas soluciones
while bandera==0
```

```
    %Se crea vecindad mediante permutaciones
    [bateria]=permuta(solucionpm1,numCustomer);

    for bat=2:size(bateria,1)
        %Se rellena vehículo mediante orden de clientes hasta que haya que
        %saltar a nuevo (se tendrá una solución ya codificada)
        [nscars1]=cars1(px,bateria(bat,:),numCustomer);
        %Se evalúan las nuevas soluciones, a mayor puntuación, peor solución
        resultados(bat)=evaluar(px,nscars1,numCustomer);
    end

    %Se busca entre todas cuál es la puntuación menor (mejor solución)
    puntuacion1=min(resultados); %mejor por 1)

    if puntuacion1>=bestpuntp1
        bandera=1;
    else
        bestpuntp1=puntuacion1;
        fila1=find(resultados==bestpuntp1);
        solucionpm1=bateria(fila1(1),:);
    end

end

%Se expresan las soluciones en el formato matriz de rutas
display('Solución Búsqueda Local')
display('Solución a partir de batería obtenida con permutaciones, método
uno')
[goodcarspm1]=cars1(px,solucionpm1,numCustomer)
bestpuntp1

%Fin tiempo ejecución
toc
```

3. Algoritmo Búsqueda Local vecindad creada mediante permutación y construcción de soluciones mediante método dos.

```
%Algoritmo de Búsqueda Local
%Permutación para crear vecindad
% 2)rellenar coche mientras quede tiempo, saltando a siguiente cliente

%Se calcula el tiempo de ejecución
tic

%Se convierte la solución inicial en un vector con el que se pueda trabajar
[solucion]=transformar(routes,numCustomer);
solucionpm2=solucion;

%Se reserva memoria para una matriz donde se almacenan las evaluaciones
resultados=zeros((numCustomer*3)*((numCustomer*3+1)/2)-numCustomer*3,1);

%Se evalua la solución de partida
[nscars2]=cars2(px,solucionpm2,numCustomer);
resultados(1)=evaluar(px,nscars2,numCustomer);
bestpuntp2=resultados(1);

bandera=0; %bandera para acabar cuando no se encuentre mejor solución
```

```
%% Se generan nuevas soluciones
while bandera==0

    %Se crea vecindad mediante permutaciones
    [bateria]=permuta(solucionpm2,numCustomer);

    for bat=2:size(bateria,1)
        %Se rellena vehículo mediante orden de clientes hasta que haya que
        %saltar a nuevo (se tendrá una solución ya codificada)
        [nscars2]=cars2(px,bateria(bat,:),numCustomer);
        %Se evalúan las nuevas soluciones, a mayor puntuación, peor solución
        resultados(bat)=evaluar(px,nscars2,numCustomer);
    end

    %Se busca entre todas cuál es la puntuación menor (mejor solución)
    puntuacion2=min(resultados); %mejor por 2)

    if puntuacion2>=bestpuntp2
        bandera=1;
    else
        bestpuntp2=puntuacion2;
        fila2=find(resultados==bestpuntp2);
        solucionpm2=bateria(fila2(1),:);
    end

end

%Se expresan las soluciones en el formato matriz de rutas
display('Solución Búsqueda Local')
display('Solución a partir de batería obtenida con permutaciones, método
dos')
[goodcarspm2]=cars2(px,solucionpm2,numCustomer)
bestpuntp2

%Fin tiempo ejecución
toc
```

4. Algoritmo Búsqueda Local vecindad creada mediante inserción y construcción de soluciones mediante método uno.

```
%Algoritmo de Búsqueda Local
%Inserción para crear vecindad
% 1)rellenar coche mientras se pueda, saltar a nuevo cuando no

%Se calcula el tiempo de ejecución
tic

%Se convierte la solución inicial en un vector con el que se pueda trabajar
[solucion]=transformar(routes,numCustomer);
solucionim1=solucion;

%Se reserva memoria para una matriz donde se almacenan las evaluaciones
resultados=zeros((3*numCustomer)^2-2*3*numCustomer+2,1);

%Se evalua la solución de partida
[nscars1]=cars1(px,solucionim1,numCustomer);
resultados(1)=evaluar(px,nscars1,numCustomer);
bestpunti1=resultados(1);
```

```
bandera=0; %bandera para acabar cuando no se encuentre mejor solución

%% Se generan nuevas soluciones
while bandera==0

    %Se crea vecindad mediante inserciones
    [bateria]=inserta(solucionim1,numCustomer);

    for bat=2:size(bateria,1)
        %Se rellena vehículo mediante orden de clientes hasta que haya que
        %saltar a nuevo (se tendrá una solución ya codificada)
        [nscars1]=cars1(px,bateria(bat,:),numCustomer);
        %Se evalúan las nuevas soluciones, a mayor puntuación, peor solución
        resultados(bat)=evaluar(px,nscars1,numCustomer);
    end

    %Se busca entre todas cuál es la puntuación menor (mejor solución)
    puntuacion1=min(resultados); %mejor por 1)

    if puntuacion1>=bestpunti1
        bandera=1;
    else
        bestpunti1=puntuacion1;
        fila1=find(resultados==bestpunti1);
        solucionim1=bateria(fila1(1),:);
    end

end

%Se expresan las soluciones en el formato matriz de rutas
display('Solución Búsqueda Local')
display('Solución a partir de batería obtenida con inserciones, método uno')
[goodcarsim1]=cars1(px,solucionim1,numCustomer)
bestpunti1

%Fin tiempo ejecución
toc
```

5. Algoritmo Búsqueda Local vecindad creada mediante inserción y construcción de soluciones mediante método dos.

```
%Algoritmo de Búsqueda Local
%Inserción para crear vecindad
% 2)rellenar coche mientras quede tiempo, saltando a siguiente cliente

%Se calcula el tiempo de ejecución
tic

%Se convierte la solución inicial en un vector con el que se pueda trabajar
[solucion]=transformar(routes,numCustomer);
solucionim2=solucion;

%Se reserva memoria para una matriz donde se almacenan las evaluaciones
resultados=zeros((3*numCustomer)^2-2*3*numCustomer+2,1);

%Se evalua la solución de partida
[nscars2]=cars2(px,solucionim2,numCustomer);
```

```
resultados(1)=evaluar(px,nscars2,numCustomer);
bestpunti2=resultados(1);

bandera=0; %bandera para acabar cuando no se encuentre mejor solución

%% Se generan nuevas soluciones
while bandera==0

    %Se crea vecindad mediante inserciones
    [bateria]=inserta(solucionim2,numCustomer);

    for bat=2:size(bateria,1)
        %Se rellena vehículo mediante orden de clientes hasta que haya que
        %saltar a nuevo (se tendrá una solución ya codificada)
        [nscars2]=cars2(px,bateria(bat,:),numCustomer);
        %Se evalúan las nuevas soluciones, a mayor puntuación, peor solución
        resultados(bat)=evaluar(px,nscars2,numCustomer);
    end

    %Se busca entre todas cuál es la puntuación menor (mejor solución)
    puntuacion2=min(resultados); %mejor por 2)

    if puntuacion2>=bestpunti2
        bandera=1;
    else
        bestpunti2=puntuacion2;
        fila2=find(resultados==bestpunti2);
        solucionim2=bateria(fila2(1),:);
    end

end

%Se expresan las soluciones en el formato matriz de rutas
display('Solución Búsqueda Local')
display('Solución a partir de batería obtenida con inserciones, método dos')
[goodcarsim2]=cars2(px,solucionim2,numCustomer)
bestpunti2

%Fin tiempo ejecución
toc
```

6. Función para cargar los datos al inicio de la resolución del problema.

```
function [px]=cargar_datos(numprob)

%Se cargan los datos numéricos de los problemas
%Los ficheros tienen que estar en la misma carpeta que el .m
%De no ser así, se indica la dirección
    if numprob==1
        load r101_1.txt %si está en la misma carpeta
        %load D:\Sara\Universidad\Grado\Proyecto\In\r101_1.txt en caso de
tener
        %que indicar la dirección
        r101_1(:,4)=[];
        r101_1(:,7)=0;
        px=r101_1;

    elseif numprob==2
```

```
    load r102_1.txt
    r102_1(:,4)=[];
    r102_1(:,7)=0;
    px=r102_1;

elseif numprob==3
    load r103_1.txt
    r103_1(:,4)=[];
    r103_1(:,7)=0;
    px=r103_1;

elseif numprob==4
    load r104_1.txt
    r104_1(:,4)=[];
    r104_1(:,7)=0;
    px=r104_1;

elseif numprob==5
    load r105_1.txt
    r105_1(:,4)=[];
    r105_1(:,7)=0;
    px=r105_1;

elseif numprob==6
    load r106_1.txt
    r106_1(:,4)=[];
    r106_1(:,7)=0;
    px=r106_1;

elseif numprob==7
    load r107_1.txt
    r107_1(:,4)=[];
    r107_1(:,7)=0;
    px=r107_1;

elseif numprob==8
    load r108_1.txt
    r108_1(:,4)=[];
    r108_1(:,7)=0;
    px=r108_1;

elseif numprob==9
    load r109_1.txt
    r109_1(:,4)=[];
    r109_1(:,7)=0;
    px=r109_1;

elseif numprob==10
    load r110_1.txt
    r110_1(:,4)=[];
    r110_1(:,7)=0;
    px=r110_1;

elseif numprob==11
    load r111_1.txt
    r111_1(:,4)=[];
    r111_1(:,7)=0;
    px=r111_1;

elseif numprob==12
    load r112_1.txt
    r112_1(:,4)=[];
```

```
        r112_1(:,7)=0;
        px=r112_1;

    else
        px=0;
        display('No se ha introducido un número correcto de problema')
    end
end
```

7. Función para evaluar el coste de los soluciones.

```
%Función para evaluar el coste de las soluciones

%Las distancias serán euclídeas
%Se toma 60 km/h como velocidad de los vehículos
%1 punto por cada unidad de distancia recorrida
%20 puntos por cada entrega no realizada
%10 puntos por cada vehículo usado

function [puntuacion]=evaluar(px,coches,numCustomer)

customer=ones(3,1)*(1:numCustomer);
customer=customer(:);                % Cliente al que pertenecen los datos

%Se cuenta el número de coches
nc=size(coches,1);

%Se guarda la localización de cada coche en cada momento
%En un principio todos parten del depósito
 for i=1:nc
    poscoches(i,1)=px(1,2);
    poscoches(i,2)=px(1,3);
 end

%Vector para guardar el tiempo en el que se encuentra cada coche
timecar=zeros(1,nc);

entregado=zeros(1,numCustomer);
puntuacion=0;

%Ahora se comprueba para cada coche cuantos clientes se pueden atender
    for j=1:nc
        k=1;

        while k<=size(coches,2) && coches(j,k)>0

            %Si el cliente ya ha sido atendido
            if entregado(customer((coches(j,k))))==1;
                k=k+1;

            %Si el cliente no ha sido atendido
            else
            %Tiempo que se tardaría en llegar al depósito desde la posición
en
            %la que se encuentra
            vuelta1=sqrt((px(1,2)-poscoches(j,1))^2+(px(1,3)-
poscoches(j,2))^2);
```

```
            %Tiempo hasta llegar al siguiente cliente (es igual a la
distancia)
            dist=sqrt((px(coches(j,k)+1,2)-
poscoches(j,1))^2+(px(coches(j,k)+1,3)-poscoches(j,2))^2);

            %Tiempo que se tardaría en llegar al depósito desde la posición a
            %la que se quiere ir
            vuelta2=sqrt((px(1,2)-px(coches(j,k)+1,2))^2+(px(1,3)-
px(coches(j,k)+1,3))^2);

                %Si se puede atender al cliente
                if timecar(j)+dist<px(coches(j,k)+1,4) &&
timecar(j)+vuelta1<px(1,5) && timecar(j)+dist+vuelta2<px(1,5)  %se llega
antes de que empiece ventana temporal y se puede volver al depósito
                    timecar(j)=px(coches(j,k)+1,4);
                    poscoches(j,1)=px(coches(j,k)+1,2);
                    poscoches(j,2)=px(coches(j,k)+1,3);
                    puntuacion=puntuacion+dist;

                    %Se marca el cliente como servido
                    entregado(customer((coches(j,k))))=1;
                    k=k+1;

                elseif px(coches(j,k)+1,4)<timecar(j)+dist &&
timecar(j)+dist<px(coches(j,k)+1,5) && timecar(j)+vuelta1<px(1,5) &&
timecar(j)+dist+vuelta2<px(1,5) %llego en ventana temporal y puedo volver al
depósito
                    timecar(j)=timecar(j)+dist;
                    poscoches(j,1)=px(coches(j,k)+1,2);
                    poscoches(j,2)=px(coches(j,k)+1,3);
                    puntuacion=puntuacion+dist;

                    %Se marca el cliente como servido
                    entregado(customer((coches(j,k))))=1;
                    k=k+1;

                %Si no se puede atender al cliente
                else
                    k=k+1;
                end
            end
        end
    end

    %Se penalizan los clientes no entregados
    for p=1:numCustomer
        if entregado(p)==0
        puntuacion=puntuacion+20;
        end
    end

    %Se llevan los coches al depósito
    retorno=0;
    for r=1:nc
        distdep=sqrt((px(1,2)-poscoches(r,1))^2+(px(1,3)-poscoches(r,2))^2);
        retorno=retorno+distdep;
    end
    puntuacion=puntuacion+retorno;

    %Se imputa un coste por el uso de vehículos
     for s=1:nc
```

```
        if timecar(s)>0
            puntuacion=puntuacion+10;
        end
    end
end
```

8. Función para transformar las soluciones dadas en forma de matriz a vector.

```
%Función para transformar matriz solución inicial en un vector
function [vec]=transformar(routes,numCustomer)
vec=[];
for i=1:size(routes,1)
    v=routes(i,:);
    v(v==0)=[];
    vec=[vec v];
end

%se añaden al final las ventanas temporales no escogidas
v1=[1:3*numCustomer];
for i=1:length(vec)
    j=1;
    bandera=0;
   while bandera==0
    if(vec(i)==v1(j))
        v1(j)=[];
        bandera=1;
    else
        j=j+1;
    end
   end
end

vec=[vec v1];

end
```

9. Función para la creación de la vecindad mediante permutaciones en la búsqueda local.

```
%Función para crear vecindad Búsqueda Local mediante permutaciones
function [bateria]=permuta(solucion,numCustomer)
m=1; %índice para realizar permutaciones
j=1;
k=2; %índice para filas de la batería
v=1;
%Matriz donde se almacenan las nuevas soluciones, se reserva memoria
bateria=zeros((numCustomer*3)*((numCustomer*3+1)/2)-
numCustomer*3+1,numCustomer*3);
bateria(1,:)=solucion; %la primera fila corresponde a la solución de partida

bandera=0;

while bandera==0
    for i=v:length(solucion)-1
        newsol=solucion;
        aux1=newsol(i+1);
```

```
            newsol(i+1)=newsol(m);
            newsol(j)=aux1;
            bateria(k,:)=newsol;
            k=k+1;
        end
        v=v+1;
        j=j+1;
        m=m+1;
        if bateria(size(bateria,1),1)~=0
            bandera=1;
        end
    end
end
```

10. Función para la creación de la vecindad mediante inserciones en la búsqueda local.

```
%Función para crear vecindad Búsqueda Local mediante inserciones
function [bateria]=inserta(solucion,numCustomer)
k=2; %índice para filas de la batería
v=1; %índice para ventanas del vector
n=1; %índice para el último caso
l=1; %índice para los casos centrales
h=0; %contador para el primer caso
%Matriz donde se almacenan las nuevas soluciones, se reserva memoria
bateria=zeros((3*numCustomer)^2-2*3*numCustomer+2,numCustomer*3);
bateria(1,:)=solucion; %la primera fila corresponde a la solución de partida

bandera=0;

while bandera==0
    for i=v:length(solucion)-1
        if v==1 %el primer caso se trata diferente
            newsol=solucion;
            aux1=newsol(v);
            aux2=newsol(v+1:v+h+1);
            aux3=newsol(v+h+2:length(newsol));
            newsol(v+h+1)=aux1;
            newsol(1:v+h)=aux2;
            newsol(v+h+2:length(newsol))=aux3;
            bateria(k,:)=newsol;
            k=k+1;
            h=h+1;
        else %clientes que están en medio del vector
            for l=1:length(solucion)-1
                newsol=solucion;
                aux1=newsol(v);
                vecaux=newsol;
                vecaux(v)=[];
                if l==1 && v-1~=l
                    %aux2=[];
                    aux3=vecaux;
                    newsol(1)=aux1;
                    newsol(2:length(newsol))=aux3;
                    bateria(k,:)=newsol;
                    k=k+1;
                elseif l==length(solucion)-1
                    aux2=vecaux;
                    %aux3=[];
                    newsol(l+1)=aux1;
                    newsol(1:l)=aux2;
                    bateria(k,:)=newsol;
```

```
                    k=k+1;
                elseif v-1==l %no hacer nada
                elseif l>=v
                    aux2=vecaux(1:l);
                    aux3=vecaux(l+1:length(vecaux));
                    newsol(l+1)=aux1;
                    newsol(1:l)=aux2;
                    newsol(l+2:length(newsol))=aux3;
                    bateria(k,:)=newsol;
                    k=k+1;
                else
                    aux2=vecaux(1:l-1);
                    aux3=vecaux(l:length(vecaux));
                    newsol(l)=aux1;
                    newsol(1:l-1)=aux2;
                    newsol(l+1:length(newsol))=aux3;
                    bateria(k,:)=newsol;
                    k=k+1;
                end
            end
            v=v+1;
        end
    end
    if v==1
        v=v+1;
    end

    if v==length(solucion) %el último caso se trata diferente
        for n=1:length(solucion)-2 %el último caso es repetido
            newsol=solucion;
            aux1=newsol(v);
            if n==1
                %aux2=[];
                aux3=newsol(1:length(newsol)-1);
                newsol(1)=aux1;
                newsol(2:length(newsol))=aux3;
            else
                aux2=newsol(1:n-1);
                aux3=newsol(n:length(newsol)-1);
                newsol(n)=aux1;
                newsol(1:n-1)=aux2;
                newsol(n+1:length(newsol))=aux3;
            end
            bateria(k,:)=newsol;
            k=k+1;
        end
    end
    if bateria(size(bateria,1),1)~=0
        bandera=1;
    end
end
end
```

11. Función para la construcción de soluciones según método uno en la búsqueda local.

```
%Función correspondiente al método 1 Búsqueda Local
function [nscars]=cars1(px,newsol,numCustomer)
% 1)rellenar coche mientras se pueda, saltar a nuevo cuando no

i=1; %índice para leer el vector solución
```

```
j=1; %índice para las filas de la matriz coches
k=1; %índice para las columnas de la matriz coches
entregado=zeros(1,numCustomer); %vector de clientes ya servidos
%Vector para guardar el tiempo en el que se encuentra cada coche
timecar=[];
timecar(1)=0;

%Se guarda la localización de cada coche en cada momento
poscoches=[]; %se desconoce a priori el número de coches
%En un principio todos parten del depósito
poscoches(1,1)=px(1,2);
poscoches(1,2)=px(1,3);

%Cliente al que pertenecen los datos
customer=ones(3,1)*(1:numCustomer);
customer=customer(:);

bandera=0;

while (bandera==0)
    if sum(entregado==1)==length(entregado)
        bandera=1;
    end
    if bandera==0
    %Si el cliente ya ha sido atendido, se salta al siguiente
    if entregado(customer(newsol(i)))==1;
        i=i+1;
    %Si el cliente no ha sido atendido
    else

    %Tiempo hasta llegar al siguiente cliente (es igual a la distancia)
    dist=sqrt((px(newsol(i)+1,2)-poscoches(j,1))^2+(px(newsol(i)+1,3)-
poscoches(j,2))^2);

    %Tiempo que tardaría en llegar al depósito desde la posición a la que se
quiere ir
    vuelta=sqrt((px(1,2)-px(newsol(i)+1,2))^2+(px(1,3)-px(newsol(i)+1,3))^2);

    %Si se puede atender al cliente
    if timecar(j)+dist<px(newsol(i)+1,4) && timecar(j)+dist+vuelta<=px(1,5)
%se llega antes de que empiece ventana temporal y se puede volver al depósito
        timecar(j)=px(newsol(i)+1,4);
        poscoches(j,1)=px(newsol(i)+1,2);
        poscoches(j,2)=px(newsol(i)+1,3);
        nscars(j,k)=newsol(i);
        k=k+1;

        %Se marca el cliente como servido
        entregado(customer(newsol(i)))=1;
        i=i+1;

    elseif px(newsol(i)+1,4)<=timecar(j)+dist &&
timecar(j)+dist<=px(newsol(i)+1,5) && timecar(j)+dist+vuelta<=px(1,5) %llego
en ventana temporal y puedo volver al depósito
        timecar(j)=timecar(j)+dist;
        poscoches(j,1)=px(newsol(i)+1,2);
        poscoches(j,2)=px(newsol(i)+1,3);
        nscars(j,k)=newsol(i);
        k=k+1;

        %Se marca el cliente como servido
        entregado(customer(newsol(i)))=1;
```

```
        i=i+1;

    %Si no se puede atender al cliente
    else
        j=j+1;
        k=1;
        timecar(j)=0;
        poscoches(j,1)=px(1,2);
        poscoches(j,2)=px(1,3);

    end
    end
    end
end
end
```

12. Función para la construcción de soluciones según método dos en la búsqueda local.

```
%Función correspondiente al método 2 Búsqueda Local
function [nscars]=cars2(px,newsol,numCustomer)
% 2)rellenar coche mientras quede tiempo, saltando a siguiente cliente

i=1; %índice para leer el vector solución
j=1; %índice para las filas de la matriz coches
k=1; %índice para las columnas de la matriz coches
entregado=zeros(1,numCustomer); %vector de clientes ya servidos
%Vector para guardar el tiempo en el que se encuentra cada coche
timecar=[];
timecar(1)=0;

%Se guarda la localización de cada coche en cada momento
poscoches=[]; %se desconoce a priori el número de coches
%En un principio todos parten del depósito
poscoches(1,1)=px(1,2);
poscoches(1,2)=px(1,3);

%Cliente al que pertenecen los datos
customer=ones(3,1)*(1:numCustomer);
customer=customer(:);

bandera=0;

while (bandera==0) && i<=length(newsol)
    %Si todos los clientes han sido servidos ya
    if sum(entregado==1)==length(entregado)
        bandera=1;
    end

    if bandera==0
    %Si el cliente ya ha sido atendido, se salta al siguiente
    if entregado(customer(newsol(i)))==1;
        i=i+1;
    %Si el cliente no ha sido atendido
    else

    %Tiempo hasta llegar al siguiente cliente (es igual a la distancia)
    dist=sqrt((px(newsol(i)+1,2)-poscoches(j,1))^2+(px(newsol(i)+1,3)-
poscoches(j,2))^2);
```

```
    %Tiempo que tardaría en llegar al depósito desde la posición a la que se
quiere ir
    vuelta=sqrt((px(1,2)-px(newsol(i)+1,2))^2+(px(1,3)-px(newsol(i)+1,3))^2);

    %Si se puede atender al cliente (cabe en el tiempo sobrante del coche)
    if timecar(j)+dist<px(newsol(i)+1,4) && timecar(j)+dist+vuelta<=px(1,5)
%se llega antes de que empiece ventana temporal y se puede volver al depósito
        timecar(j)=px(newsol(i)+1,4);
        poscoches(j,1)=px(newsol(i)+1,2);
        poscoches(j,2)=px(newsol(i)+1,3);
        nscars(j,k)=newsol(i);
        k=k+1;

        %Se marca el cliente como servido
        entregado(customer(newsol(i)))=1;
        i=i+1;

    elseif px(newsol(i)+1,4)<=timecar(j)+dist &&
timecar(j)+dist<=px(newsol(i)+1,5) && timecar(j)+dist+vuelta<=px(1,5) %llego
en ventana temporal y puedo volver al depósito
        timecar(j)=timecar(j)+dist;
        poscoches(j,1)=px(newsol(i)+1,2);
        poscoches(j,2)=px(newsol(i)+1,3);
        nscars(j,k)=newsol(i);
        k=k+1;

        %Se marca el cliente como servido
        entregado(customer(newsol(i)))=1;
        i=i+1;

    %Si no se puede atender al cliente, se comprueba si puedo el siguiente
    else
        i=i+1;
    end
    end
    end
    %Si se ha llenado el coche pero no se han servido todos los clientes
    if i==3*numCustomer+1 && bandera==0
        i=1;
        k=1;
        j=j+1;
        timecar(j)=0;
        poscoches(j,1)=px(1,2);
        poscoches(j,2)=px(1,3);
    end
end
end
```

13. Función para representar gráficamente las soluciones.

```
%Función para representar de forma gráficas las rutas
function representa(px,routes)
x0=px(1,2);
y0=px(1,3);
xc=px(:,2);
yc=px(:,3);
x=[];
y=[];
for i=1:size(routes,1)
```

```
    n=sum(routes(i,:)~=0);
    x=[];
    y=[];
    for j=1:n
        x1=px(routes(i,1)+1,2);
        y1=px(routes(i,1)+1,3);
        x=[x px(routes(i,j)+1,2)];
        y=[y px(routes(i,j)+1,3)];
    end
x=[x0 x x0];
y=[y0 y y0];
plot(xc,yc,'x')
grid
hold on
if i==1
    plot(x,y,'b')
    plot(x1,y1,'ro')
elseif i==2
    plot(x,y,'g')
    plot(x1,y1,'ro')
elseif i==3
    plot(x,y,'r')
    plot(x1,y1,'ro')
elseif i==4
    plot(x,y,'c')
    plot(x1,y1,'ro')
elseif i==5
    plot(x,y,'m')
    plot(x1,y1,'ro')
elseif i==6
    plot(x,y,'y')
    plot(x1,y1,'ro')
elseif i==7
    plot(x,y,'k')
    plot(x1,y1,'ro')
elseif i==8
    plot(x,y,'w')
    plot(x1,y1,'ro')
end
plot(x0,y0,'rs')
end
```