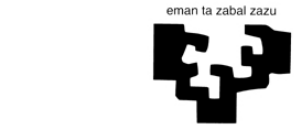

ZTF-FCT
Zientzia eta Teknologia Fakultatea
Facultad de Ciencia y Tecnología

OPTIMIZACIÓN ENTERA MIXTA PARA PROBLEMAS DE GENERACIÓN DE RUTAS DE VEHÍCULOS

Trabajo Fin de Grado
Grado en Matemáticas

JOSU IRCIO FERNÁNDEZ

Trabajo dirigido por
Gloria Pérez Sainz De Rozas

Leioa, 3 de septiembre de 2014

Índice general

Agradecimientos

A Gloria Pérez Sainz De Rozas, por todo el tiempo que me ha dedicado desde que un día aparecí en su despacho, por su total implicación y su entusiasmo, por sus sugerencias e indicaciones, sin las cuales este trabajo no hubiera sido posible y sobre todo por iniciarme en el fascinante mundo de la optimización y la programación.

A la UPV-EHU, por ofrecerme un ambiente propicio y los medios necesarios para la realización de este trabajo.

A mis padres, porque me han dado todo y todo lo que soy se lo debo a ellos, por apoyarme y darme ánimos en todo momento.

Introducción

Los problemas del viajero y de rutas de vehículos son dos de los problemas más estudiados en Investigación Operativa. Estos problemas de optimización se plantean cuando existen unos clientes que demandan un servicio y se debe encontrar la mejor ruta para satisfacerles. Es decir, realizar el servicio en el menor tiempo y/o con el menor gasto posible.

Aún no se ha encontrado ningún algoritmo que logre resolver el problema generalizado de rutas en un tiempo polinómico. A primera vista trataríamos de calcular todas las posibilidades y seleccionar la de coste mínimo. Tal algoritmo, llamado algoritmo de fuerza bruta, crece exponencialmente cuando aumentamos el número de puntos a visitar, por lo tanto, no es nada efectivo. Por consiguiente, se buscan soluciones aproximadas que puedan ser obtenidas con una rapidez relativa y que sean lo suficientemente parecidas a la solución óptima.

Debido a la gran utilidad del problema en distintos ámbitos, los esfuerzos realizados en mejorar los algoritmos de resolución no son pocos.

0.1. Desarrollo histórico y Evolución

Las primeras herramientas de diseño óptimo de rutas y frecuencias de vehículos surgen en la década del 70. Estas están basadas en ideas intuitivas, sin una formulación del modelo y su función objetivo. Incluso en algunos casos sin exploración del espacio de soluciones.
En la década del 80 se formulan algunas funciones objetivo, y se incorporan nuevos parámetros como el cubrimiento de la demanda o el factor de carga (proporción de pasajeros respecto a la cantidad de asientos).
El problema de optimización de rutas de vehículos implica determinar un plan de recorridos, frecuencias, horarios, asignación de personal y flota, en lo posible óptimas. Este proceso se puede descomponer en etapas [Ceder y Wilson, 1986] de la siguiente manera:

- 1)Diseño de las rutas: cantidad de líneas y el trazado de sus recorridos.
- 2) Determinación de frecuencias: de pasadas para cada línea, eventualmente variable en el tiempo.

- 4) Asignación de flota: en base a los vehículos disponibles para realizar los viajes.
- 5) Asignación de personal y recursos disponibles a los viajes programados por línea.

La planificación del transporte basada en herramientas de apoyo a la decisión cobra cada vez más importancia, tanto en los países desarrollados como en los en vías de desarrollo. Problemas como la asignación de flota y personal, han recibido amplio tratamiento con muchos resultados publicados; se modelan como problemas clásicos de optimización combinatoria, programación lineal entera, y, en muchos casos, se resuelven de forma exacta a partir de la creación de modelos de optimización para los cuales se dispone de algoritmos eficientes de resolución.

En cambio el problema generalizado de optimización de rutas y frecuencias posee varias fuentes de complejidad (no linealidad, no convexidad, múltiples objetivos) que dificultan tanto su formulación como la derivación de algoritmos eficientes de resolución. La complejidad de gran parte de estos problemas hace que no se puedan resolver casos de tamaño realista en un tiempo de cómputo razonable. Para superar estos obstaculos se utilizan técnicas de relajación, que permiten abordar el problema inicial a partir de otros problemas más sencillos, además de utilizar algoritmos heurísticos también llamados quasióptimos, que no garantizan la obtención de soluciones óptimas pero que pueden ofrecer soluciones de buena calidad en poco tiempo de cómputo.
Ya en estos últimos años han aparecido otros enfoques, como la utilización de metaheurísticas y la exploración del espacio de soluciones. La facilidad de integrar módulos existentes y de incorporar interfaces gráficas, estimulan el desarrollo de nuevos métodos. Se desarrollan las siguientes características:

- **Adaptabilidad:** respecto de los datos disponibles.
- **Interactividad:** con el usuario, el modo de permitir la incorporación de conocimiento humano (técnico humano) en el proceso de toma de decisiones.
- **Eficiencia:** calidad en los resultados y tiempos de procesamiento razonables.
- **Flexibilidad:** en cuanto al horizonte de planificación, los primeros métodos refirieron a planificaciones de corto y mediano plazo.

Capítulo 1

Problemas Enteros Mixtos

1.1. Problemas lineales deterministas

Un problema lineal determinista trata de encontrar la solución de un problema de optimización de una función lineal, que depende de un conjunto de restricciones también lineales. Tiene la expresión siguiente:

$$Z = \min c_1x_1 + c_2x_2 + ... + c_nx_n$$

$$\begin{aligned} \text{sujeto a} \quad & b_1^1 \leq a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n \leq b_2^1 \\ & b1^2 \leq a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n \leq b_2^2 \\ & \vdots \\ & b_1^m \leq a_{m1}x_1 + a_{m2}x_2 + ... + a_{mn}x_n \leq b_2^m \\ & l_i \leq x_i \leq u_i \qquad i = 1, ..., n. \qquad\qquad \exists k \in \{1, ..., p\},\ p \leq n,\ x_k \in \mathbb{Z} \end{aligned} \tag{1.1}$$

Empleando notación matricial:

$$\begin{gathered} Z = \min c^t x \\ \text{s.a. } b_1 \leq Ax \leq b_2 \\ l \leq x \leq u \end{gathered} \tag{1.2}$$

Donde $x \in \mathbb{R}^{n+}$ es un vector columna de variables de las cuales p son enteras, $c^t \in \mathbb{R}^n$ es un vector fila de costos. $A \subset M_{m\times n}$ es la matriz de restricciones y $b \in \mathbb{R}^m$ es el vector columna de términos independientes, la *función objetivo,* $c^t x$, es una función lineal. El conjunto de *soluciones factibles* son aquellas que cumplen el conjunto de restricciones del modelo. El objetivo es encontrar la solución factible que obtenga el mejor valor de la función objetivo, que se llama óptima.

Los problemas lineales se pueden clasificar según el caracter de las variables:

- **Problemas Lineales Generales:** Todas las variables de decisión son continuas. Se llamarán simplemente Problemas lineales y se denotarán por PL.

- **Problemas Enteros:** Todas las variables de decisión son enteras. Se donotarán por PE. Un caso particular son los **problemas 0-1 puros** si todas las variables son binarias.

- **Problemas Mixtos:** Las variables de decisión son tanto continuas como enteras. También son conocidos como *problemas enteros mixtos lineales* y sedenotarán por PM.

En lo sucesivo, trataremos problemas Enteros Mixtos, más concretamente, los problemas del agente viajero y el problema de rutas.

1.2. Modelización del problema del agente viajero y de las rutas.

En general, estos dos problemas se modelizan como problemas de optimización entera mixta. Los modelos presentados buscan maximizar el nivel de servicio, minimizando el uso de los recursos, según determinadas restricciones. Estos objetivos son generalmente contrapuestos, una mejora en uno implica un detrimento en el otro.

Baaj y Mahmassani (1991)

Se busca minimizar los tiempos totales de transferencia de la carga y el tamaño de la flota requerida.

Los principales aspectos del problema son tenidos en cuenta, así como una variedad de parámetros y restricciones (factor de carga, por ejemplo). Es flexible, ya que permite la incorporación del conocimiento de los usuarios. Pueden ser agregado al momento de aplicar un método de resolución. Los componentes de la función objetivo se expresan en distintas unidades, obligando a utilizar coeficientes de conversión.

Israeli y Ceder (1993)

Este modelo es similar al propuesto por Baaj y Mahmassani (1991), pero se formula como un problema de optimización multiobjetivo.

Ngamchai y Lovell (2000)

Con una formulación similar a la propuesta por Baaj y Mahmassani (1991), este modelo permite calcular frecuencias de rutas; aunque requiere del uso de coeficientes de conversión a la misma unidad (€/hora) de todas las componentes de la función objetivo.

Gruttner, Pinninghoff, Tudela y Díaz (2002)

Este modelo difiere de todos los anteriores en la especificación de los componentes del sistema. Propone un modelo de asignación alternativo, que usa el método logit mediante el cálculo de utilidades de cada línea para cada par origen-destino (i,j). No se contemplan aspectos tales como la determinación de frecuencias y dimensionamiento de flota; requiere la utilización de coeficientes de conversión y de valores subjetivos del tiempo.

Capítulo 2

Métodos de Resolución

El problema de transporte es uno de los problemas más conocidos y desafiantes en la programación lineal entera, que cae en la categoría denominada NP-Hard, esto es, los problemas que no se pueden resolver en un tiempo polinomial como función del tamaño de la entrada en una máquina de Turing determinística. Esto ocurre pues el tiempo y esfuerzo computacional requerido para resolver este problema aumenta exponencialmente respecto al tamaño del problema, es decir, la cantidad de vértices a ser visitados por los vehículos.
Actualmente, los algoritmos para resolver las distintas instancias del problema de transporte son muy variados en distintos aspectos, como el enfoque de optimización utilizado: local o global, a qué clase de algoritmos pertenece, por ejemplo si están basados en programación lineal, son heurísticas clásicos o metaheurísticas.
Debido a la complejidad computacional en este tipo de problemas es a menudo deseable obtener soluciones aproximadas, para que puedan ser encontradas suficientemente rápido y que sean suficientemente buenas para llegar a ser útiles en la toma de decisiones. Por esto se han ideado algoritmos que no garantizan optimalidad, pero que logran entregar buenas soluciones a estos problemas difíciles de resolver. Estos son los algoritmos heurísticos, que conforman una clase de métodos muy extensa y taxonómicamente compleja tal como se describe en ver[1], que en la útima década han tenido un gran éxito resolviendo problemas pertenecientes a la clase NP-Hard.
Por último, los últimos avances en la resolución de estos problemas son los Algoritmos híbridos. En ellos se combinan aspectos de varias heurísticas, metaheurísticas o algoritmos exactos para obtener lo mejor de ellos. Algunos ejemplos recientes son la combinación de recocido simulado y búsqueda tabú, ver[2].

2.1. Métodos exactos

Son aquellos que parten de una formulación como modelos de programación lineal (enteros) o similares, y llegan a una solución factible (entera) gracias a algoritmos de acotamiento del conjunto de soluciones factibles. Se han realizado avances recientes en este campo, ver[3].

Branch and Bound

El método de Branch and Bound (o Ramificación y Acotamiento) es un algoritmo diseñado para la resolución de modelos de programación entera. Su operatoria consiste en "linealizar" el modelo de programación entera, es decir, resolver éste como si fuese un modelo de programación lineal y luego generar cotas en caso que al menos una variable de decisión adopte un valor fraccionario. El algoritmo genera en forma recursiva cotas (o restricciones adicionales) que favorecen la obtención de valores enteros para las variables de decisión. En este contexto resolver el modelo lineal asociado a un modelo de programación entera se conoce frecuentemente como resolver la relajación continua del modelo entero. Se continua bifurcando y añadiendo condiciones adicionales o secundarias hasta que el óptimo continuo también cumple las condiciones de integralidad.

2.2. Heurísticas

Inicialmente las heurísticas se concebían como algoritmos hechos a la medida del problema que se quería tratar, por lo que su aplicabilidad estaba acotada a los supuestos de quien las diseñaba.

Una heurística es un algoritmo que permite obtener soluciones de buena calidad para un problema dado. Esto permite tener menores tiempos de ejecución, pero sin asegurar la optimalidad de la solución.
Dependiendo de cómo acometen su labor, las heurísticas (para el problema de rutas de vehículos) pueden clasificarse, como se expone, ver[4]:

- **Constructivas:** no parten de una solución factible, sino que la van elaborando a medida que progresan. Una de las más conocidas es la heurística de ahorros propuesta en ver[5], donde se crean n rutas factibles, y se va probando a unir una ruta que termina en i con otra que comienza en j , agregando el arco $[i, j]$ y calculando el ahorro de cada posible movimiento.

- **De mejora:** trabajan sobre una solución factible. Un ejemplo es la heurística de Lin-Kernighan, ver[6].

- **Técnicas de relajación:** son métodos asociados a la programación lineal entera. La más conocida es la llamada Relajación Lagrangeana, que consisten en descomponer un modelo lineal entero en un conjunto de restricciones difíciles y otras más fáciles, relajando las primeras, al pasarlas a la función objetivo multiplicándolas por una penalidad, de forma análoga al método de multiplicadores de Lagrange. Esto sirve para obtener cotas al problema original, acelerando el proceso de resolución. Algunas revisiones progresivamente más actualizadas del tema son las presentes ver[7, 8, 9].

2.3. Metaheurísticas

Las metaheurísticas nacieron de la búsqueda de enfoques generales que fueran capaces de resolver una clase de problemas. Su concepción fue inspirada por la observación de la naturaleza.

Una metaheurística es una estrategia (heurística) general para la resolución de una gran variedad de problemas para los que no existe un algoritmo confiable de resolución, ya sea por la complejidad del problema, o por falta de estudios en la resolución de éste, según lo expresado en ver[10]. Tienen un rol fundamental en la Investigación de Operaciones, pues pueden ser aplicadas a problemas de Optimización Combinatorial, con resultados muy cercanos al óptimo. Se basan en la observación de la naturaleza, la evolución biológica, procesos físicos asociados a la manufactura, etc. Dentro de las características deseables de una metaheurítica, mencionadas en ver[11], están:

(I) Ser algoritmos de optimización global. Esto implica la existencia de mecanismos que le permitan escapar de óptimos locales.

(II) Brindar suficiente libertad a quien la implemente, mediante la posibilidad de trabajar con distintos parámetros, estrategias de paralelización, adición de heurísticas complementarias, etc.

(III) Lograr un rendimiento consistente y estable en los problemas de la clase que resuelven.

Algunas de las metaheurísticas más comúnmente utilizadas en problemas de optimización combinatoria, son:

- **Algoritmos genéticos:** corresponden a una clase de algoritmos evolutivos, los cuales fueron descritos por primera vez ver[12].
  Han sido aplicados recientemente en su forma pura para el VRP original por ejemplo ver[13], y en forma híbrida ver[14, 15], combinando características de otras metaheurísticas.

- **Búsqueda tabú:** en esta metaheurística, se busca en la proximidad de la solución actual otra que mejore la evaluación de la función objetivo, almacenando las soluciones anteriores (o alguna característica de éstas), las que son marcadas como tabú. Esto evita que el algoritmo entre en un ciclo, pudiendo escapar de óptimos locales. Un ejemplo reciente se presenta en ver[16] para el VRP con una flota de vehículos heterogénea. Hasta antes el trabajo de ver[14], era la metaheurística que obtenía las mejores soluciones para los problemas de gran tamaño.
- **Colonias de hormigas:** basadas en la naturaleza, varias hormigas (procesos, hilos, agentes, etc.) exploran distintas direcciones del espacio de soluciones factibles, dejando tras de síun rastro de feromonas, que le indican a la siguiente hormiga las direcciones más "interesantes"de ser exploradas, las que toma con una probabilidad proporcional al nivel de feromona existente, en un intento por no caer en un óptimo local.

A continuación, para solucionar el problema del viajero, el problema de rutas de vehículos y sus generalizaciones debido a sus extensión utilizaremos el método de Branch and Bound.

Capítulo 3

Travelling Salesman Problem (TSP)

3.1. Desarrollo Histórico

Hay diferentes versiones sobre el origen del problema del viajero. En el siglo XIX el matemático irlandés W. R. Hamilton propone un juego (llamado el Icosaedro de Hamilton) que guarda mucha relación con el TSP. ver[17].

Ya en 1832 se publica en Alemania el libro "El viajante de comercio: cómo debe ser y qué se debe hacer para conseguir comisiones y triunfar en el negocio. Por un viajante de comercio veterano" donde se proponen algunas rutas por Alemania y Suiza a modo de ejemplo. Pero en ningún caso se realiza un tratamiento matemático del problema.

La primera vez que fue propuesto el problema desde el punto de vista matemático se remontan a principios de la década de 1930. Sobresale el trabajo de Karl Menger, ver[18], que define el problema con precisión y comprobó que pudiera ser resuelto en un número finito de pruebas. Pero se desconocía la existencia de reglas que permitiesen reducir este número de pruebas por debajo del número de permutaciones. Además propone un algoritmo sencillo para encontrar soluciones factibles, el Algoritmo del vecino más próximo. Consiste en comenzar por el nodo origen e ir visitando cada vez el punto más cercano sin volver a un punto ya visitado.

En los 50 y 60 el problema fue ganando popularidad muy rápido, destacando el trabajo de George Dantzig, Delbert Ray Fulkerson y Selmer M. Johnson (1954), ver[19], que modelizaron el TSP como un problema de programación lineal entera y desarrollaron un algoritmo de planos de corte para su resolución con el cual fueron capaces de resolver un caso con 49 ciudades. Teniendo en cuenta la falta de programas informáticos una gran cifra. La idea fue aplicar el recien desarrollado, por Dantzig en 1947, algoritmo del simplex, ver[20]. Lo que supuso un gran avance en las técnicas de optimización del momento. A partir de entonces comenzaron a desarrollarse otros algoritmos

que fuesen aplicables al problema con un número cada vez más grande de ciudades.
En 1972 Richard M. Karp, ver[21], demostró que el TSP es NP-duro, explicando la gran dificultad computacional con la que investigadores anteriores se habían encontrado. En los años 70 y 80 se realizaron grandes progresos y se consiguieron resolver instancias de hasta 2392 ciudades, utilizando planos de corte y algoritmos de ramificación y acotación.
En los 90 Applegate, Bixby, Chvátal, y Cook, ver[22], desarrollaron el software Concorde, ver[23], que aún se utiliza, y en 1991 Gerhard Reinelt publicó la TSPLIB, ver[24], una colección de casos de distintos tamaños que ha sido muy utilizada por investigadores posteriores para comparar la eficiencia de distintos algoritmos.
En la última década se han ido resolviendo problemas de dimensiones cada vez mayores; en 2006 David Applegate, Robert Bixby, Va ek Chvátal, William Cook, Daniel Espinoza y Marcos Goycoolea (Applegate et al., 2006), resolvieron con el Concorde una instancia de 85900 vértices de forma exacta.

El Problema del Viajero (conocido por sus siglas en inglés TSP: Traveling Salesman Problem) es uno de los problemas de optimización combinatoria más estudiado a lo largo de la historia, dando lugar a multitud de variantes y extensiones con infinidad de aplicaciones prácticas en la vida real.
Se centra en estudiar problemas de la siguiente clase: un comercial debe visitar varios clientes y desea conocer cúal es el camino de mínima distancia que, partiendo de su lugar de trabajo, vaya a todas las ciudades y regrese.
A pesar de la aparente sencillez de su planteamiento, el TSP es un problema computacionalmente complejo.

3.2. Aplicaciones

La importancia de esta clase de problemas se debe al gran número de situaciones reales en las que se puede aplicar. La mayor parte de estas se encuentran en el campo de la logística (reparto de mercancías, correo, rutas escolares), aunque también podemos encontrar aplicaciones de estos problemas en industria (producción de circuitos integrados) o en genética. Algunos ejemplos típicos:

- **Logística:** Las aplicaciones más directas y más abundantes del TSP. Entre las múltiples destacamos:

  - *Vendedores y turistas*: Se utiliza en los planificadores de rutas para determinar cuál es el mejor camino para visitar los puntos que desean y volver al punto de origen (nótese que los turistas desean visitar los monumentos o lugares emblemáticos y después

regresar al hotel).

- *Rutas escolares*: Fueron las primeras aplicaciones del TSP pues permite reducir gastos de una manera significativa.

- *Reparto de correo*: Aunque generalmente el reparto de correo se ajusta mejor a un problema de rutas sobre arcos, cuando las casas están muy alejadas unas de otras o cuando sólo se debe visitar algunas de ellas es adecuado utilizar este método.

- **Industria:** Las aplicaciones en industria no son tan numerosas. Estas son las más conocidas:

  - *Secuenciación de tareas*: Se utiliza en máquina debe realizar una serie de tareas en el mínimo tiempo posible y sin importar el orden de realización de las mismas. Se aplica un TSP suponiendo que cada tarea es uno de los nodos a visitar, han de realizarse todas las tareas para producir el producto y que la distancia entre la tarea i y la j es tij.

  - *Producción de circuitos electrónicos*: Se dividen en dos grandes grupos.

    (I) Problemas de perforado: Los circuitos integrados se encuentran en muchos dispositivos electrónicos. Estas placas han de ser perforadas un número relativamente grande de ocasiones. Para ello se utilizan máquinas que si no son programadas correctamente, el tiempo que se tarda en recorrer la placa de un orificio a otro puede aumentar significativamente aumentando el coste al disminuir el rendimiento. Ajustamos a un problema TSP tomando como ciudades cada una de las posiciones donde debe realizarse una perforación y las distancias entre ellas como el tiempo que necesita la máquina en trasladarse de una a otra. Trataremos de minimizar el tiempo que pierde la taladradora en moverse de una posición a otra. Estas soluciones han sido utilizadas por grandes empresas, como son Siemens e IBM, dando lugar a mejoras de aproximadamente el 10 % del rendimiento total de las líneas de producción.

    (II) Conexión de chips: Los chips son herramientas fundamentales en el diseño de ordenadores y de otros dispositivos digitales. Deben ser conectados entre sí por cables, entonces se trata

de minimizar la cantidad de cable necesaria para unir todos los puntos. Claramente este modelo puede ser modelizado como un TSP tomando los pins como las ciudades y la distancia entre ellas, la cantidad de cable necesario para unirlas.

- *Creación de cluster de datos*: La organización de datos en grupos (clusters) de elementos con propiedades similares es un problema básico en análisis de datos que se puede ajustar como un TPS.

3.3. Modelización y Formulación del TSP.

Problema básico TSP.

Sea $G = (V, A)$ es un grafo completo no dirigido con el conjunto de n vértices $V = \{1, ..., n\}$ y conjunto de arcos $A = \{[i, j] : i, j \in V, i \neq j\}$. donde $c_{i,j}$ será el costo asociado con cada arco $a = [i, j]$ de A. El vértice 1 será el vértice depósito desde donde comenzará el trayecto y donde concluirá. Para recorrer todos los vértices serán necesarios $|L| = n$ tramos. Por tanto, se tendrá el tramo $l \in L = \{1, ..., n\}$ que irá de i a j.

Parámetros:
$c_{i,j}$ es la distancia entre dos ciudades i, j o el costo de ir de la ciudad i a la ciudad j

$$X_{ijl} = \begin{cases} 1 & \text{si en el tramo } l \text{ se va de } i \text{ a } j. \\ 0 & \text{en otro caso.} \end{cases}$$

Función a optimizar: $\min \sum_{i \in V} \sum_{j \in V} \sum_{l \in L} c_{ij} X_{ijl}$

Restricciones:

- El tramo 1, $l = 1$, partirá desde el origen, $i = 1$, a otro vértice j del conjunto total de vértices V. Se traduce a expresiones matématicas dejando fijos los índices origen y tramo y variando el índice destino de la variable X_{ijl}.

  $$\sum_{j \in V / j \neq 1} X_{1j1} = 1$$

- Saliendo de i se llega a un único j por un único tramo l. Se fija el índice de origen y el sumatorio de la variable X_{ijl} para todo posible destino y tramo tiene que ser 1 debido a la unicidad.

  $$\sum_{j \in V} \sum_{l \in L} X_{ijl} = 1 \quad i = 1, \cdots, n$$

- A cada tramo l se le asigna una ruta única. Se fija el índice para cada tramo y el sumatorio de la variable X_{ijl} para cada ruta posible será 1 debido a la unicidad.
  $\sum_{i \in V} \sum_{j \in V} X_{ijl} = 1 \quad l = 1, \cdots, n$

- A j se llega desde un único origen i en un único tramo l. Se fija el índice de destino y el sumatorio de la variable X_{ijl} para todo posible origen y tramo tiene que ser 1 debido a la unicidad.
  $\sum_{i \in V} \sum_{l \in L} X_{ijl} = 1 \quad j = 1, \cdots, n$

- El punto final j del tramo l debe de ser el punto de partida del tramo $l + 1$. Por lo tanto, se fija el índice de destino j y el tramo l y el sumatorio de X_{ijl} para todo origen posible tiene que ser igual al sumatorio de X_{ijl} para el tramo siguiente, es decir $l + 1$, para origen el destino del tramo l y todo destino posible.
  $\sum_{i \in V / i \neq j} X_{ijl} = \sum_{r \in V / r \neq j} X_{jr(l+1)} \quad j, k = 1, \cdots, n$

EJEMPLO

Una empresa de producción tiene 9 filiales establecidas en una región y debe proveer a cada una de ellas con los materiales necesarios para su funcionamiento. Desde su sede central o depósito, 1 se encargan de la logística en el cumplimiento de las demandas. Para ello con el único camión que posee esta empresa en construcción debe visitar cada una de las sedes y regresar de nuevo al déposito. Como tiene varias alternativas para recorrer las filiales, se quiere encontrar en qué orden debe hacer su recorrido para minimizar el costo total del trayecto recorrido. En la siguiente tabla se muestran las dependencias, las rutas existentes y el coste de ir de una a otra:

c_{ij}	1	2	3	4	5	6	7	8	9	10
1	0	4	3	1	1	1	1	1	5	7
2	4	0	2	1	2	4	1	3	8	5
3	3	2	0	5	1	1	1	4	10	2
4	1	1	1	0	1	7	1	1	1	1
5	1	2	5	1	0	1	4	1	2	5
6	1	4	1	7	1	0	1	1	4	$1e^{20}$
7	1	1	1	1	4	1	0	1	$1e^{20}$	$1e^{20}$
8	10	3	3	1	10	10	1	1	1	2
9	2	10	4	3	1	1	3	6	7	9
10	2	10	3	10	10	2	5	$1e^{20}$	0	7

(3.1)

Como se observa en la tabla de costos al no ser simétrica los costos de ir de la filial i a la filial j no son los mismos que los de ir de la j a la i lo que hace más realista el problema pues, el costo en el consumo de combustible en la realidad varía dependiendo de muchos factores como por ejemplo la inclinación y pendiente del camino. Si en un sentido es cuesta arriba en el otro será cuesta abajo.
También se observa que en la tabla tenemos costos muy altos, $1e^{20}$, estos indican que el costo de recorrer ese camino es infinito, es decir, que no podemos recorrerlo bien porque no existe el camino o bien porque en este momento está cortada esa ruta.

Este es un problema básico del agente viajero y por lo tanto se modeliza como se ha planteado anteriormente. Se obtiene que hay:

- 1000 variables a tener en cuenta.

- 121 condiciones que se deben cumplir.

Debido a la extensión del problema se utiliza el método exacto Branch and Bound para resolverlo y mediante el software libre Coin-or y su paquete de optimización se busca una solución.
El departamento de logística de la empresa encuentra que la solución óptima para minimizar el gasto del servicio es la siguiente ruta:

$TRAMO$	1	2	3	4	5	6
$(origen, destino)$	$(1,8)$	$(8,7)$	$(7,2)$	$(2,4)$	$(4,10)$	$(10,9)$

(3.2)

$TRAMO$	7	8	9	10
$(origen, destino)$	$(9,6)$	$(6,3)$	$(3,5)$	$(5,1)$

(3.3)

El costo total de realizar esta ruta es: Función objetivo $z = 9$.
Se presenta una recreación de lo que sería la ruta y los gastos empleados en cada trayecto.

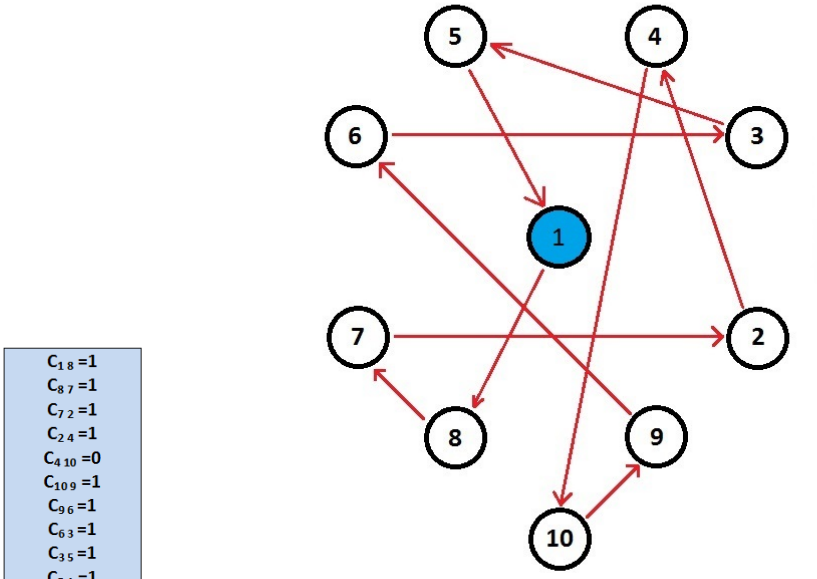

Figura 3.1: Esquema de la solución del TSP básico.

3.4. Variantes del TSP

Estas son unas de las variantes más conocidas del TSP
MAX-TSP
TSP con cuello de botella
TSP gráfico
TSP agrupado
TSP generalizado
TSP con múltiples viajantes

Se estudiará el TSP generalizado, el TSP con múltiple viajantes y la combinación de ambos el TSP generalizado múltiple.

3.4.1. TSP Generalizado (GTSP)

El problema del viajero generalizado (GTSP) es una variación del problema del TSP muy conocido en el que no todos los vértices tienen por qué ser visitados. En particular, el conjunto de nodos V se divide en c clusters o subconjuntos, $V_1, ..., V_c$ con $V_1 \cup ... \cup V_c = V$ y $V_l \cap V_h = \emptyset$ $l \neq h \quad \forall l, h \in \{1, ..., c\}$. El objetivo es encontrar un recorrido de longitud mínima que contenga exactamente un vértice de cada conjunto V_l.
El GTSP y sus variantes aparecen en la vida real en aplicaciones como el diseño del flujo de materiales, reparto de correos, las operaciones de computacionales, la logística de fabricación, distribución de mercancías por mar para el número potencial de puertos etc. El GTSP puede ser modelizado como sigue:

Sea $G = (V, A)$ un grafo dirigido donde $V = \{1, ..., n\}$ es el conjunto de vértice y $A = \{(i, j) : i, j \in V, \ i \neq j\}$ es el conjunto de arcos dirigidos. El grafo G tiene unos pesos c_{ij} asociados a cada arco (i, j) que representan la distancia o coste del viaje del vértice i al j. Sea $K = \{1, ..., c\}$ el conjunto de clusters o subconjuntos de vétices, $|K| = c$. Donde $k \in K = \{1, ..., c\}$ denotará el cluster en el que nos encontraremos. Los clusters en los que se dividen el conjunto de vértices seran $V_1, ..., V_c$ con $V_1 \cup ... \cup V_c = V$ y $V_l \cap V_h = \emptyset \ l \neq h \quad \forall l, h \in \{1, ..., c\}$.
Se buscará el recorrido de longitud mínima que se inicia en el vértice 1 o vértice de inicio y visita exactamente un vértice de cada uno de los clúster regresando finalmente al vértice de inicio. Se describe la formulacion basada en los vértices:

Variables:

$$X_{ij} = \begin{cases} 1 & \text{si se recorre el arco que va de } \ i \ \ a \ \ j. \\ 0 & \text{en otro caso.} \end{cases}$$

$$w_{sq} = \begin{cases} 1 & \text{si se va del cluster } \ s \ \text{ al cluster } \ q. \\ 0 & \text{en otro caso.} \end{cases}$$

$u_s = \{t \in \{1, ..., c\} \mid t$ es el orden de clasificación del cluster s en la gira de un vehículo$\}$.

Función a optimizar: $\min \sum_{i \in V} \sum_{j \in V} c_{ij} X_{ij}$

Restricciones:

- Restricciones de flujo en los clusters:

  De cada cluster sale un único camino. Se fija el cluster V_k y el sumatorio de X_{ij} para todo origen que pertenezca al cluster y todo destino posible fuera del cluster tiene que ser 1 debido a la unicidad.

  $$\sum_{i \in V_k} \sum_{j \in V \setminus V_k} X_{ij} = 1 \quad k = 1, ..., c$$

  A cada cluster llega un único camino. Se fija el cluster V_k y el sumatorio de X_{ij} para todo origen posible fuera del cluster y todo destino

que pertenezca al cluster tiene que ser 1 debido a la unicidad.

$$\sum_{i \in V \setminus V_k} \sum_{j \in V_k} X_{ij} = 1 \quad k = 1, ..., c$$

- Restricciones de equilibrio de flujo en los vértices:

  El número de caminos que llegan a un vértice tiene que ser igual al número de los caminos que salen. Se fija un vértice y el sumatorio de X_{ij} para origen el vertice fijado y todo destino posible será igual al sumatorio de X_{ij} para todo origen posible y destino el vértice fijado.

  $$\sum_{j \in V \setminus \{i\}} X_{ij} = \sum_{r \in V \setminus \{i\}} X_{ri} \quad \forall i \in V$$

- Restricciones de relación entre las variables:

  El flujo del cluster s al cluster q se define por w_{sq}. Por lo tanto, w_{sq} debe ser igual a la suma de los caminos que van del vértice i al j, X_{ij}, donde $i \in V_s$ y $j \in V_q$.

  $$w_{sq} = \sum_{i \in V_s} \sum_{j \in V_q} X_{ij}, \qquad s \neq q;\ s, q \in K = \{1, ..., c\}$$

- Restricciones de eliminación de subciclos:

  Impide que en un mismo viaje se pase dos veces por el mismo cluster es decir se realice un subciclo.

  $$u_s - u_q + (c) w_{sq} + (c-2) w_{qs} \leq c - 1, \qquad s \neq q;\ s, q \in K - \{1\} = \{2, ..., c\}$$

  No hay caminos entre los vértices del mismo cluster. Si los hubiera u_s sería igual que u_q y $w_{sq} = 1$. Entonces se tendría que $c \leq c - 1$ lo que es absurdo pues $c \geq 0$.

- Restricciones de limitación de las variables auxiliares:

  $$u_s - \sum_{q \in K - \{1\};\ q \neq s} w_{qs} \geq 1, \qquad s \in K - \{1\} = \{2, ..., c\}$$

  $$u_s + (c-1) w_{1s} \leq c \qquad s \in K - \{1\} = \{2, ..., c\}$$

- Restricciones de integralidad: $X_{ij} \in \{0, 1\}$ $w_{sq} \in \{0, 1\}$ $u_s \in \{1, ..., c\}$

EJEMPLO

La empresa de producción, en la busqueda de mejorar el rendimiento y abaratar los costos de abastecer a las filiales, estudia diferentes posibilidades. La primera, es agrupar en distintos clusters o conjuntos las sedes según la proximidad de estas y de esta forma repartir el trabajo de organización de ellas por grupos. De esta manera, la empresa sólo tiene que abastecer a cada uno de los cluster de filiales y ellas se encargan de distribuir la carga entre su propio grupo.

La empresa finalmente organiza las filiales en 5 clusters de este modo

cluster \ sede	1	2	3	4	5	6	7	8	9	10
cluster 1	✓									
cluster 2		✓	✓	✓						
cluster 3					✓	✓				
cluster 4							✓	✓		
cluster 5									✓	✓

(3.4)

Se distinguen 5 cluster diferenciados donde el cluster 1 estará formado únicamente por la sede 1, el déposito, que es de donde partirá el vehículo hacia los demás clusters y donde regresará una vez recorridos todos. El cluster 2 contará con las sedes 2, 3 y 4, el cluster 3 con las sedes 5 y 6, el cluster 4 con las sedes 7 y 8 y el cluster 5 con las sedes 9 y 10.
Por consiguiente los costos de ir de una sede a otra siguen siendo los mismo que en la tabla de costes del anterior apartado y se sigue contando con un único vehículo.

Este es un problema generalizado del agente viajero y por lo tanto se modeliza como se ha planteado anteriormente. Se obtiene que hay:

- Numero de variables: 130
- Numero de condiciones: 60

Se observa que el problema se ha reducido bastante y la dificultad de organizar ha decrecido como se preveía así como el tiempo de cómputo. Aún y todo, debido a la extensión del problema se utiliza el método exacto Branch and Bound para resolverlo y mediante el software libre Coin-or y su paquete de optimización se busca una solución.
El departamento de logística de la empresa de nuevo encuentra que la solución óptima para minimizar el gasto del servicio es la siguiente ruta:

Origen 1 Destino 4 (cluster 1 $\rightarrow$ cluster 2)
Origen 4 Destino 9 (cluster 2 $\rightarrow$ cluster 5)
Origen 9 Destino 6 (cluster 5 $\rightarrow$ cluster 3)
Origen 6 Destino 7 (cluster 3 $\rightarrow$ cluster 4)
Origen 7 Destino 1 (cluster 4 $\rightarrow$ cluster 1)

El coste total de realizar esta ruta será: Función objetivo $z = 5$.
Por lo tanto, agrupando las sedes en clusters el coste será de 4 unidades menos frente a tener que recorrer todas las sedes indistintamente.
Se presenta una recreación de lo que serían los clusters, la ruta óptima y gastos empleados en cada trayecto.

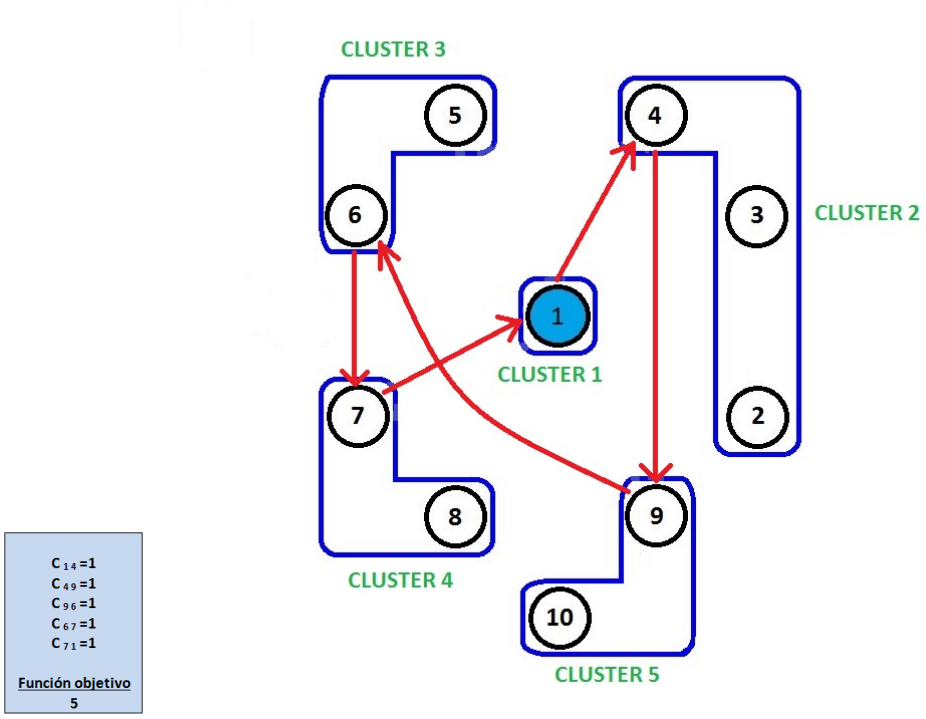

Figura 3.2: Esquema de la solución del GTSP.

3.4.2. TSP de multiples viajantes (mTPS)

El problema del viajero para múltiples viajantes (mTPS), consiste en determinar un conjunto de rutas para m vendedores los cuales salen desde un vértice depósito (ciudad de origen) y regrasan a él al terminar la ruta. El mTSP, en general, puede definirse de la siguiente manera:
Dado un conjunto de vértices, ciudades, donde hay un depósito (ciudad origen) con m vendedores ubicados. Los demás vértices (ciudades) deben ser visitados por los vendedores. Luego, el plan estratégico consiste en encontrar tours para todos los vendedores que empiecen y terminen en la ciudad depósito, de forma que cada vértice intermedio es visitado una única vez y el costo total de la visita de todos los vértices sea el mínimo.

En comparación con el TSP, el mTPS es más adecuado para modelar situaciones de la vida real, ya que es capaz de manejar más de un vendedor. Estas situaciones se presentan en su mayoría en problemas de enrutamiento y programación. Por ejemplo, el mTPS se puede utilizar para calcular el número mínimo de vehículos requeridos para servir a un conjunto de clientes, en el que cada cliente tiene una demanda, no negativa, asociada y cada vehículo no puede viajar más de una cantidad predefinida de kilometros. El mTPS puede ser modelado como sigue:

Sea $G = (V, A)$ un grafo dirigido donde $V = \{1, ..., n\}$ es el conjunto de vértice y $A = \{(i, j) : i, j \in V,\ i \neq j\}$ es el conjunto de arcos dirigidos. El grafo G tiene unos pesos c_{ij} asociados a cada arco (i, j) que representan la distancia o coste del viaje del vértice i al j.
Desde el vértice origen, 1, salen los m vendedores que realizarán los recorridos. Sea $M = \{1, ..., m\}$ el conjunto de m vendedores.

Variables:

$$X_{ij} = \begin{cases} 1 & \text{Si se recorre el arco que va de } i \text{ a } j. \\ 0 & \text{en otro caso.} \end{cases}$$

$u_s = \{t \in \{1, ..., n\} \mid t \text{ es el orden de visitado de vértice } s.\}$

Función a optimizar: $\min \sum_{i \in V} \sum_{j \in V} c_{ij} X_{ij}$

Restricciones:

- Restricciones de flujo en los vértices:

  De cada vértice a excepción del origen sale un único camino.

  $$\sum_{j \in V;\ j \neq i} X_{ij} = 1, \quad i \in V - \{1\} = 2, ..., n.$$

  A cada vértice a excepción del origen llega un único camino.

  $$\sum_{i \in V;\ i \neq j} X_{ij} = 1, \quad j \in V - \{1\} = 2, ..., n.$$

- Restricciones de flujo en el depósito:

  Del depósito deben salir m vehículos. Es decir, se fija como origen el vertice 1 y saldran m rutas distintas. Por lo tanto, el sumatorio de X_{1j} para todo destino posible será m.

  $$\sum_{j \in V;\ j \neq 1} X_{1j} = m$$

Al depósito deben llegar los mismos vehículos que han salido, m vehículos. Es decir, se fija como destino el vertice 1 y llegarán m rutas distintas. Por lo tanto, el sumatorio de X_{i1} para todo origen posible será m.

$$\sum_{i \in V;\ i \neq 1} X_{i1} = m$$

- Restricciones de eliminación de subciclos:

  El único subciclo posible es que un vehículo salga del origen a un cierto vértice y vuelva directamente al origen. Es necesario pues, si en el depósito se tiene al menos el número de vehículos igual a la parte entera de la mitad del número de vértices, $m \geq \lceil n/2 \rceil$, el problema sería infactible porque al menos un vehículo tendría que recorrer un único vértice aparte del depósito y no podría cumplir la restricción. Por lo tanto tenemos,

  $$u_s + (n-2)X_{1s} - X_{s1} \leq n-1, \qquad s \in V - \{1\} = \{2, ..., n\}$$
  $$u_s + X_{1s} + X_{s1} \geq 2, \qquad s \in V - \{1\} = \{2, ..., n\}$$
  $$u_s - u_q + (n)X_{sq} + (n-2)X_{qs}, \qquad s \neq q; s, q \in V - \{1\} = \{2, ..., n\}$$

- Restricción de integralidad: $X_{ij} \in \{0, 1\}$, $m \geq 1$ entero, $u_s \in \{1, ..., n\}$.

EJEMPLO

La segunda opción para mejorar el rendimiento de la empresa de producción y de este modo ser más competitiva, es aumentar la flota de camiones que transporten la materia prima necesaria para el funcionamiento de las filiales.
Desde el departamento de investigación se estudia si introduciendo nuevos vehículos el costo de abastecer todas las filiales descendería haciendo rentable la compra de nuevos camiones.

Este es un problema del viajero para múltiples viajantes y por lo tanto, se modeliza como se ha planteado anteriormente, se tienen:

- 110 variables binarias.
- 110 condiciones que se deben cumplir.

En el departamento de investigación se varía el número de vehículos, m, desde 1 hasta 6 observando si se abarata el proceso de abastecimiento. Debido a la extensión del problema se utiliza el método exacto Branch and Bound para resolverlo y mediante el software libre Coin-or y su paquete de optimización se busca una solución.

(I) **Solución para un vehículo, $m = 1$:**

Corresponde con el problema inicial del agente viajero. Costo de realizar la ruta óptima: $z = 9$ con el siguiente recorrido,

$1 \xrightarrow{1} 8 \xrightarrow{1} 7 \xrightarrow{1} 2 \xrightarrow{1} 4 \xrightarrow{1} 10 \xrightarrow{0} 9 \xrightarrow{1} 6 \xrightarrow{1} 3 \xrightarrow{1} 5 \xrightarrow{1} 1$

Contabilizamos también el coste total del recorrido del camión que más filiales recorre. Se toma como el tiempo de realizar el proceso de reparto. En este caso al solo haber un camión será el costo total que es $z = 9$.

(II) **Solución para dos vehículos, $m = 2$:**

Corresponde con el problema del agente viajero para dos viajantes. Costo de realizar la ruta óptima: $z = 11$ con el siguiente recorrido,

Vehículo 1: $1 \xrightarrow{1} 6 \xrightarrow{1} 3 \xrightarrow{1} 5 \xrightarrow{1} 1$.

Vehículo 2: $1 \xrightarrow{1} 8 \xrightarrow{1} 7 \xrightarrow{1} 2 \xrightarrow{1} 4 \xrightarrow{1} 10 \xrightarrow{0} 9 \xrightarrow{2} 1$.

En este caso el tiempo de realizar la ruta lo determina el vehículo 2 con un coste de $z = 7$.

(III) **Solución para tres vehículos, $m = 3$:**

Corresponde con el problema del agente viajero para tres viajantes. Costo de realizar la ruta óptima: $z = 13$ con el siguiente recorrido,

Vehículo 1: $1 \xrightarrow{1} 4 \xrightarrow{1} 1$.

Vehículo 2: $1 \xrightarrow{1} 6 \xrightarrow{1} 3 \xrightarrow{2} 2 \xrightarrow{1} 7 \xrightarrow{1} 1$.

Vehículo 3: $1 \xrightarrow{1} 8 \xrightarrow{2} 10 \xrightarrow{0} 9 \xrightarrow{1} 5 \xrightarrow{1} 1$.

En este caso el tiempo de realizar la ruta lo determina el vehículo 2 con un coste de $z = 6$.

(IV) **Solución para cuatro vehículos, $m = 4$:**

Corresponde con el problema del agente viajero para cuatro viajantes. Costo de realizar la ruta óptima: $z = 15$ con el siguiente recorrido,

Vehículo 1: $1 \xrightarrow{1} 5 \xrightarrow{1} 1$.

Vehículo 2: $1 \xrightarrow{1} 6 \xrightarrow{1} 3 \xrightarrow{2} 2 \xrightarrow{1} 4 \xrightarrow{1} 1$.

Vehículo 3: $1 \xrightarrow{1} 7 \xrightarrow{1} 1$.

Vehículo 4: $1 \xrightarrow{1} 8 \xrightarrow{2} 10 \xrightarrow{0} 9 \xrightarrow{2} 1$.

En este caso el tiempo de realizar la ruta lo determina el vehículo 2 con un coste de $z = 6$.

(v) **Solución para cinco vehículos, $m = 5$:**

Corresponde con el problema del agente viajero para cinco viajantes. Costo de realizar la ruta óptima: $z = 18$ con el siguiente recorrido,

Vehículo 1: $1 \xrightarrow{1} 4 \xrightarrow{1} 2 \xrightarrow{2} 3 \xrightarrow{3} 1$.

Vehículo 2: $1 \xrightarrow{1} 5 \xrightarrow{1} 1$.

Vehículo 3: $1 \xrightarrow{1} 6 \xrightarrow{1} 1$.

Vehículo 4: $1 \xrightarrow{1} 7 \xrightarrow{1} 1$.

Vehículo 4: $1 \xrightarrow{1} 8 \xrightarrow{2} 10 \xrightarrow{0} 9 \xrightarrow{2} 1$.

En este caso el tiempo de realizar la ruta lo determina el vehículo 1 con un coste de $z = 7$.

(vi) **Solución para seis vehículos, $m = 6$:**

Corresponde con el problema del agente viajero para seis viajantes. Costo de realizar la ruta óptima: $z = 22$ con el siguiente recorrido,

Vehículo 1: $1 \xrightarrow{4} 2 \xrightarrow{2} 3 \xrightarrow{3} 1$.

Vehículo 2: $1 \xrightarrow{1} 4 \xrightarrow{1} 1$.

Vehículo 3: $1 \xrightarrow{1} 5 \xrightarrow{1} 1$.

Vehículo 4: $1 \xrightarrow{1} 6 \xrightarrow{1} 1$.

Vehículo 5: $1 \xrightarrow{1} 7 \xrightarrow{1} 1$.

Vehículo 6: $1 \xrightarrow{1} 8 \xrightarrow{2} 10 \xrightarrow{0} 9 \xrightarrow{2} 1$.

En este caso el tiempo de realizar la ruta lo determina el vehículo 1 con un coste de $z = 9$.

Se concluye que el menor coste de realizar la ruta es con un solo vehículo, $z = 9$. Por lo tanto, a nivel de gastos no es necesario aumentar la flota de camiones puesto que el gasto sería mayor y no sería una inversión rentable. Sin embargo, si nos fijamos en el tiempo de realizar la ruta, es decir, el camión con mayor costo de cada ruta, obviamente con un solo camión el tiempo empleado es el mayor, de 9 concretamente. En este sentido, tenemos que para dos vehículos el tiempo en realizar la ruta será de 7 y para tres y cuatro camiones será de 6 teniendo un gasto total de 11, 13, 15 respectivamente. De todas las posibilidades de aumentar la flota, se observa, que la que obtiene mejor proporción tiempo/gasto con respecto a no aumentarla es la de dos vehículos.

Número de vehículo	Proporción *tiempo/ gasto* respecto a un vehículo
2	$(9-7)/(11-9)=1$
3	$(9-6)/(13-9)=0{,}75$
4	$(9-6)/(15-9)=0{,}5$
5	$(9-7)/(18-9)=0{,}223$
6	$(9-9)/(22-9)=0$

(3.5)

Por consiguiente, el departamento, teniendo en cuenta el acortamiento en el tiempo de entrega, lo que hará que la producción aumente en las sedes generando mayores beneficios, y el reducido aumento del gasto de transporte se decide por la mejor opción en proporción tiempo/gasto, que es aumentar la flota en 2 vehículos.
Por lo tanto, la opción elegida para mejorar el rendimiento es aumentar la flota en dos vehículos. El departamento de logística de la empresa de nuevo encuentra que la solución óptima para minimizar el gasto del servicio es la siguiente ruta:

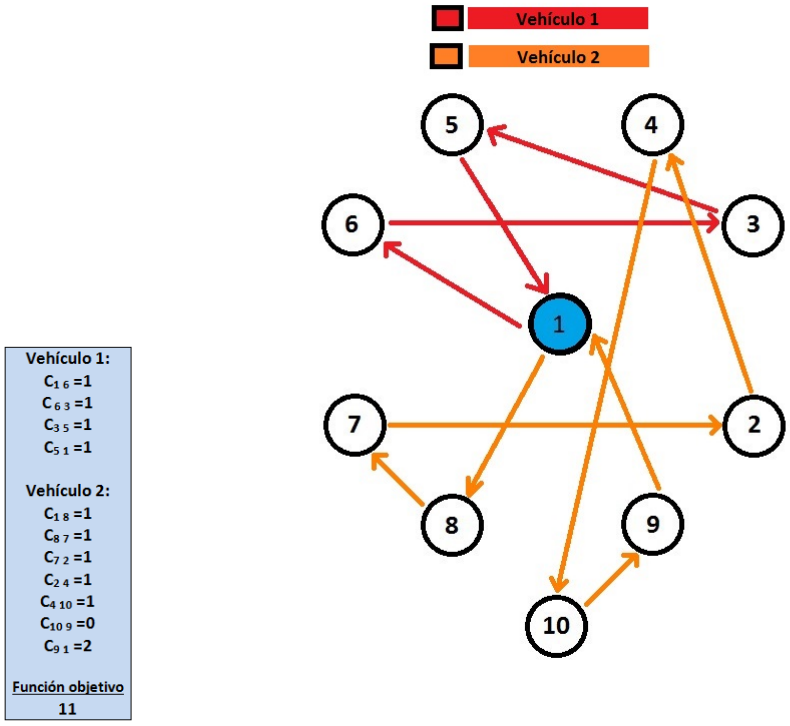

Figura 3.3: Esquema de la solución del 2-TSP

Vehículo 1:			
Origen	1	Destino	6
Origen	6	Destino	3
Origen	3	Destino	5
Origen	5	Destino	1

Vehículo 2:			
Origen	1	Destino	8
Origen	8	Destino	7
Origen	7	Destino	2
Origen	2	Destino	4
Origen	4	Destino	10
Origen	10	Destino	9
Origen	9	Destino	1

El costo total de realizar esta ruta es: Función objetivo $z = 11$.
Se presenta una recreación de lo que sería la ruta y los gastos empleados en cada trayecto.

3.4.3. TPS Generalizado múltiple (GmTSP)

En esta variante del TSP combinamos el GTSP y el mTSP. Es decir, se tiene los vértices divididos en clusters y se cuenta con m vehículos.

Sea $G = (V, A)$ un grafo dirigido con $V = \{1, 2, ..., n\}$ el conjunto de vértices. Se tiene que $|V| = n$, donde el vértice 1 representa el depósito y el resto de $n - 1$ vértices representan geográficamente clientes dispersos. Sea $A = \{(i, j) : i, j \in V; i \neq j\}$ el conjunto de arcos donde el costo, c_{ij}, no negativo es asociado a cada arco $(i, j) \in A$ representando el costo de ir desde el vérice i al vértice j. El conjunto de vértices V se agrupan en $|K| = c$ clusters mutuamente excluyentes, tal que, $V_1 = \{1\}$. Cada uno de los clusters tiene que ser visitado una única vez, para ello, existen m vehículos idénticos. El GmTSP consiste en encontrar m recorridos de m vehículos de mínimo costo total que comiencen y terminen en el vértice 1, depósito, y de tal manera que cada cluster o subgrupo sea visitado exactamente una vez, es decir, se visite un vértice de cada cluster.

El GmTSP y sus casos especiales se encuentran en aplicaciones de la vida real, tales como las operaciones de ordenador, fabricación, logística, distribución de mercancías etc.

El problema puede ser modelizado como sigue:

V se divide en $|K| = c$ clusters $V_1, ..., V_c$, cada uno de los cuales representa a un grupo de clientes y donde V_1 es el orígen.

Parámetros:

- c_{ij}: costos de viajar desde el vértice i al vértice j, $i \neq j$; $i \in V_l$, $j \in V_h$; $l \neq h$; $l, h = 1, ..., c$.

- m: número de vehículos (número de tours).

Variables:

$X_{ij} = \begin{cases} 1 & \text{Si se recorre el arco que va de } \ i \in V_p \ \ a \ \ j \in V_l, \ p \neq l; \ p, l = \{0, ..., c\}. \\ 0 & \text{en otro caso.} \end{cases}$

$w_{sq} = \begin{cases} 1 & \text{si se va del cluster } \ s \ \text{ al cluster } \ q. \\ 0 & \text{en otro caso.} \end{cases}$

$u_s = \{t \in \{1, ..., c\} \mid t \text{ el orden de clasificación del cluster } s \text{ en la gira de un vehículo.}\}$

Función a optimizar: $\min \sum_{i \in V} \sum_{j \in V} c_{ij} X_{ij}$

Restricciones:

- Restricciones de flujo en los clusters:

  De cada cluster sale un único camino hacia otro cluster excepto desde el cluster depósito que saldrán m.

  $$\sum_{i \in V_k} \sum_{j \in V \setminus V_k} X_{ij} = 1 \quad k \in K - \{1\} = 2, ..., c$$

  A cada cluster llega un único camino excepto excepto al cluster depósito que llegarán m.

  $$\sum_{i \in V \setminus V_k} \sum_{j \in V_k} X_{ij} = 1 \quad k \in K - \{1\} = 2, ..., c$$

- Restricciones de equilibrio de flujo en los vértices:

  El número de caminos que llegan a un vértice tiene que ser igual al número de los caminos que salen.

  $$\sum_{j \in V \setminus \{i\}} X_{ij} = \sum_{r \in V \setminus \{i\}} X_{ri} \quad \forall i \in V$$

- Restricciones de flujo en el depósito:

  Del depósito deben salir m vehículos. Es decir, se fija como origen el vertice 1 y saldran m rutas distintas.

  $$\sum_{j \in V; \ j \neq 1} X_{1j} = m$$

Al depósito deben llegar los mismos vehículos que han salido, m vehículos. Es decir, se fija como destino el vertice 1 y llegarán m rutas distintas.

$$\sum_{i \in V;\ i \neq 1} X_{i1} = m$$

- Restricciones de relación entre las variables:

El flujo del cluster s al cluster q se define por w_{sq}. Por lo tanto, w_{sq} debe ser igual a la suma de los caminos que van del vértice i al j, X_{ij}, donde $i \in V_s$ y $j \in V_q$.

$$w_{sq} = \sum_{i \in V_s} \sum_{j \in V_q} X_{ij}, \qquad s \neq q;\ s, q \in K = \{1, ..., c\}$$

- Restricciones de eliminación de subciclos:

No hay caminos entre los vértices del mismo cluster.

$$\sum_{i \in V_k} \sum_{j \in V_k;\ j \neq i} X_{ij} = 0 \quad k = 1, ..., c$$

A cada vértice a lo sumo llega un camino y saldrá otro con excepción de el depósito que como se indica en las restricciones anteriores sale y llegan m.

$$\sum_{j \in V \setminus \{i\}} X_{ij} \leq 1 \quad \forall i \in V \setminus \{1\}$$

$$\sum_{i \in V \setminus \{j\}} X_{ij} \leq 1 \quad \forall j \in V \setminus \{1\}$$

El único subciclo posible es que un vehículo salga del origen a un cierto cluster y vuelva directamente al origen. Es necesario pues, si en el depósito se tiene al menos el número de vehículos igual a la parte entera de la mitad del número de clusters, $m \geq \lceil c/2 \rceil$, el problema sería infactible porque al menos un vehículo tendría que recorrer un único cluster aparte del depósito y no podría cumplir la restricción. Además toda ruta debe empezar y terminar en el origen es decir, no puede haber subciclos independientes, para ello añadimos,

$u_s + (c-2)w_{1s} - w_{s1} \leq c - 1, \qquad s \in K - \{1\} = \{2, ..., c\}$
$u_s + w_{1s} + w_{s1} \geq 2, \qquad s \in K - \{1\} = \{2, ..., c\}$
$u_s - u_q + (c)w_{sq} + (c-2)w_{qs}, \qquad s \neq q; s, q \in K - \{1\} = \{2, ..., c\}$

EJEMPLO

El departamento de investigación despúes de estudiar los dos ámbitos de mejora mencionados y optar por las mejores soluciones en los dos casos, decide combinar las dos soluciones. De este modo, lograr maximizar el rendimiento de la empresa y convertirla en competitiva.
Por lo tanto, la empresa de producción organiza sus filiales en 5 clusters y aumenta su flota de camiones en dos. Se consigue abaratar los gastos de transporte debido a la organización en clusters y se reduce el tiempo de reparto aunque aumentando sensiblemente el gasto añadiendo otro camión a la flota.

Este es un problema del viajero generalizado múltiple, para resolverlo se modeliza como hemos propuesto anteriormente, se tienen:

- 130 variables binarias.
- 83 condiciones que se deben cumplir.

Debido a la extensión del problema se utiliza el método exacto Branch and Bound para resolverlo y mediante el software libre Coin-or y su paquete de optimización se busca una solución.
El departamento de logística e investigación de la empresa encuentra que la solución óptima para minimizar el gasto del servicio con 5 clusters y 2 vehículos es la siguiente ruta:

Vehículo 1:

Origen	1	Destino	4	(cluster 1	→	cluster 2)
Origen	4	Destino	9	(cluster 2	→	cluster 5)
Origen	9	Destino	6	(cluster 5	→	cluster 3)
Origen	6	Destino	1	(cluster 3	→	cluster 1)

Vehículo 2:

Origen	1	Destino	7	(cluster 1	→	cluster 4)
Origen	7	Destino	1	(cluster 4	→	cluster 1)

El coste total de realizar esta ruta será: Función objetivo $z = 6$.
Por lo tanto, agrupando las sedes en 5 clusters y utilizando dos vehículos el coste será de 3 unidades menos frente al problema del viajero inicial con 10 ciudades y un único vehículo. Además, en hacer la ruta solo tardaremos 4 unidades de tiempo frente a las 9 del problema inicial. Asique, la reducción de costo y tiempo será considerable, haciendo que la empresa sea mucho má competitiva y potente.

Se presenta una recreación de lo que serían los clusters,las distintas rutas óptimas para cada vehículo y gastos empleados en cada trayecto.

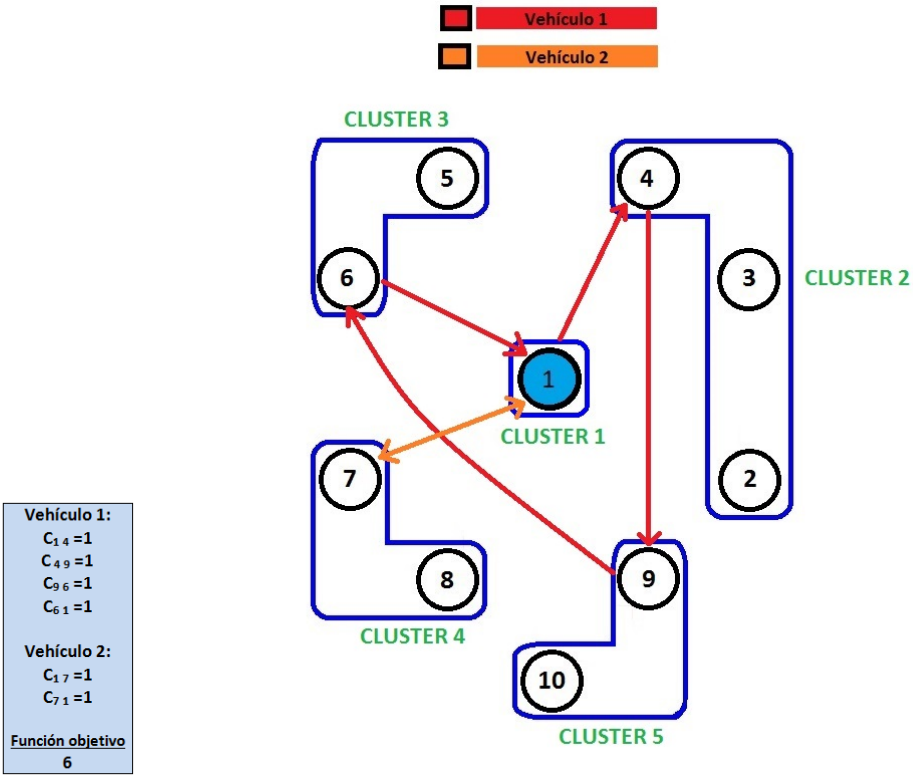

Figura 3.4: Esquema de la solución del GmTSP para 5 clusters y dos vehículos

CONCLUSIÓN

Se ha observado una posible aplicación del problema de transporte y sus variantes a la vida real.
En un principio, se tenía una empresa de reparto en construcción, que desde un depósito tenía que distribuir los materiales para que sus 9 filiales pudieran trabajar. Primero se organizo la ruta más favorable en cuanto al gasto de modo que se cumpliera el reparto en cada filial. Despúes, con el objetivo de abaratar gastos y reducir el tiempo de ejecución de la tarea se estudio por separado agrupar las filiales en clusters y añadir más vehículos a la flota de la empresa. Por último, una vez obtenidos los mejores resultados para cada una de las opciones se trato de combinar ambas. De este modo, con este proceso hemos logrado que la empresa con los recursos que tiene sea lo más competitiva posible.

Otros ámbitos de mejora donde también entran variantes del TSP son por ejemplo, el agrupado de la carga en el camión según el orden de reparto y conseguir tanto el mínimo gasto en el agrupado de la carga en containers combinado con el menor gasto en el reparto y rutas. También teniendo en cuenta la necesidad volver a recoger los productos fabricados en las filiales para devolverlos al depósito se pueden plantear nuevos problemas.

3.5. Métodos de resolución

Se presentan los métodos de resolución más conocidos para el TSP y sus variantes. Existirán algunas heurísticas creadas únicamente para algunas variantes. Aquí se nombran los más usados,
Método de los planos de corte
Método de ramificación y acotación
Programación dinámica
Heurísticas
Heurística del vecino más próximo
Heurística de Christofides
Heurísticas de intercambio

Como se ha ido explicando en cada ejemplo, para resolverlos se utiliza el método exacto Branch and Bound.

Capítulo 4

Problema de Ruta (VRP)

4.1. Desarrollo Histórico

Los problemas de rutas tienen su origen en el siglo XIX cuando el irlandés W.R. Hamilton y el británico T. Kirkman inventaron el denominado "Icosian Game". Este juego consistía en encontrar una ruta entre los 20 puntos del juego usando sólo los caminos permitidos y regresando al nodo origen. Obviamente, este juego no se centraba en la búsqueda del camino óptimo, sino en la búsqueda de un camino que visitase todos los nodos una única vez. Este tipo de caminos o ciclos recibirían el nombre de Hamiltonianos en honor a W.R. Hamilton.
Dantzig y Ramser fueron los primeros autores que trataron este tema, cuando estudiaron la aplicación real en la distribución de gasolina para estaciones de carburante (1959). Proponían una solución basada en una formulación de programación lineal que daba lugar a un solución quasi-óptima. No es más que una generalización del problema del viajero en el que se obliga a este a visitar la ciudad de origen cada vez que haya visitado m de las $n-1$ ciudades restantes.

Los problemas de rutas de vehículos (más conocidos por sus siglas en inglés VRP) tratan de resolver problemas como el que sigue: una empresa debe repartir cierto producto entre sus m-clientes y desea encontrar la ruta (o rutas) de menor coste que, partiendo del almacén, visite cada cliente satisfaciendo su demanda y regrese al almacén. Es de gran utilidad en problemas reales y considerable dificultad. Cuando el tamaño del mismo es excesivamente grande, es deseable obtener soluciones aproximadas que puedan ser obtenidas con una rapidez relativa y que sean lo suficientemente buenas respecto a la solución óptima, se consiguen con métodos heurísticos.

4.2. Aplicaciones

Las aplicaciones más comunes son la planificación de sistemas de recogida de basuras, limpieza de calles, rutas de vendedores, etc. Aunque la aplicación por excelencia es la optimización en el proceso de transporte de mercancías. El transporte de mercancías se encuentra presente en muchos de los sistemas de producción, representando una parte importante (entre el 10 % y el 20 %) del coste final del producto. De manera que una reducción del gasto en el transporte de mercancías puede suponer un aumento considerable de las ganancias.

4.3. Modelización y Formulación del VRP

Parámetros:

- Conjunto de vertices $V = \{1, \cdots, n\}$ lugares de paso. Donde el 1 es el depósito donde se cargan los vehículos y los vértices restantes son los lugares o clientes que se deben visitar.
- $M = \{1, ..., m\}$ es el conjunto de m vehículos y estos tienen capacidad Q.
- d_i es la demanda del mismo producto de cada cliente i. Esta cantidad parte inicialmente del vértice 1(depósito). Por lo tanto el vértice 1 tendrá demanda 0.
- $c_{ij} \geq 0$ es el coste o distancia de ir desde el punto i al punto j.
- I es el conjunto de clientes a visitar.

Variables: $X_{ijv} = \begin{cases} 1 & \text{Si un vehículo } v \in M \text{ va de } i \text{ a } j. \\ 0 & \text{en caso contrario.} \end{cases}$

Función Objetivo: $\text{mín} \sum_{v \in M} \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij}$

Restricciones:

- Cada una de las rutas tiene que empezar y finalizar en el vértice 1 (depósito).

- Restricciones de Grado:

  (I) Cada vehículo debe visitar a cada cliente una vez.

$$\sum_{j \in V} x_{ijv} = 1 \qquad \forall i \in I, \forall v \in M \qquad \text{Cada vehículo } v \text{ del vértice } i \text{ sale a un único } j.$$

$$\sum_{j \in V} x_{jiv} = 1 \qquad \forall i \in I, \forall v \in M \quad \text{Cada vehículo } v \text{ llega al vértice } i \text{ desde un único } j.$$

  (II) No se pueden usar más vehículos de los disponibles en el depósito. Los vehículos que salen del depósito son los mismos que llegan sin superar m, el máximo.

$$\sum_{j \in V} x_{1j} = \sum_{j \in V} x_{j1} \leq m$$

- Restricciones de capacidad:

  La demanda que transporta cada vehículo no puede superar la capacidad de estos. Además, hay que asegurarse de que cada cliente reciba vehículos suficientes como para satisfacer su demanda.

$$\sum_{i \in V} x_{ji} \geq \sum_{i \in V} \frac{d_i}{Q}$$

Si se establece la capacidad de los vehículos como infinta o mayor que la suma de las demandas de todos los vértices la restricción de capacidad no influye y por tanto el problema se reduce al problema del viajero mTSP. Además, si el número de vehículos es uno el problema es el mismo que el TSP original.

4.4. Variantes del VRP

Estas son algunas de las variantes de VRP,
VRP clásico
VRP con capacidad limitada (CVRP)
VRP con ventanas de tiempo (VRPTW)
VRP de ida y vuelta (VRPB)
VRP con recogida y reparto (VRPPD)

Debido a la complejidad y la extensión de los problemas se centrará el estudio en el VRP con capacidad limitada.

4.4.1. VRP Con capacidad limitada (CVRP)

Este problema es un VRP clásico en el que los vehículos de la flota tienen una capacidad determinada e igual para todos ellos esta es menor que la suma de las demandas de todos los vértices. El objetivo es el mismo pero debemos añadir la restricción de capacidad: la suma de las demandas de las ciudades que se visitan en cada ruta no puede exceder la capacidad del vehículo. Formalmente, el CVRP se define como:
Sea $G = (V, A)$ un grafo donde $V = \{1, ..., n\}$ es el conjunto de vértices, o clientes, con el depósito ubicado en el vértice 1, y A es el conjunto de arcos que los unen. Para cada arco $(i, j); i \neq j \ \forall i, j \in \{1, ..., n\}$, existe un coste no negativo asociado c_{ij} que suele ser interpretado como el coste o el tiempo en el que se incurre al viajar de i a j.
Generalmente, el uso de arcos de la forma (i, i) no está permitido lo que se suele imponer fijando $c_{ii} = \infty \ \forall i \in V$. Además, a cada cliente i se le asocia una demanda no negativa d_i que le debe ser entregada. Al depósito se le asigna una demanda ficticia $d_1 = 0$.
Dado el conjunto $S \subseteq V$, se define $d(S) = \sum_{i \in S} d_i$ como la demanda total del conjunto S.
Se tiene un conjunto $M = \{1, ..., m\}$ de m vehículos iguales, todos con la misma capacidad Q, disponibles en el depósito. Para asegurar la viabilidad se asume que $d_i \leq Q$ y que M no es más pequeño que un M_{min}. Donde se toma $|M_{min}| = \lceil d(V)/Q \rceil$.

Variables:

$$X_{ijv} = \begin{cases} 1 & \text{Si el arco } (i,j) \text{ es recorrido por el vehículo } v \in M. \\ 0 & \text{en otro caso.} \end{cases}$$

$$y_{iv} = \begin{cases} 1 & \text{Si el vértice } i \text{ es recorrido por el vehículo } v. \\ 0 & \text{en otro caso.} \end{cases} \quad i \in V, \ v \in M.$$

$$w_{sq} = \begin{cases} 1 & \text{si se va del vértice } s \text{ al vértice } q. \\ 0 & \text{en otro caso.} \end{cases}$$

$u_s = \{t \in \{1, ..., n\} \mid t \text{ es el orden de visitado de vértice } s.\}$

Función a optimizar: $\min \sum_{i \in V} \sum_{j \in V} \sum_{v \in M} c_{ij} X_{ijv}$

Restricciones:

- Restricciones de flujo en los vértices:

  Cada vértice debe ser visitado una única vez.

Además, de cada vértice a excepción del origen sale un único camino.

$$\sum_{j\in V;\ j\neq i}\sum_{v\in M} X_{ijv} = 1, \quad i \in V - \{1\} = 2, ..., n.$$

El mismo número de vehículos que entran saldrán de cada cliente, respectivamente.

$$\sum_{j\in V} X_{ijv} = \sum_{j\in V} X_{jiv} = y_{iv} \qquad \forall i \in V,\ v \in M = \{1, ..., m\}.$$

- Restricciones de flujo en el depósito:

  Del depósito salen m vehículos los cuales regresarán al final del trayecto.

  $$\sum_{j\in V-\{1\}}\sum_{v\in M} X_{0jv} = m$$

  $$\sum_{i\in V-\{1\}}\sum_{v\in M} X_{i0v} = m$$

- Restricciones de capacidad:

  Para cada vehículo la suma de las demandas de los vértices recorridos tiene que ser menor o igual que la capacidad Q de cada uno de los vehículos.

  $$\sum_{i\in V} d_i y_{iv} \leq Q \qquad \forall v \in M = \{1, ..., m\}.$$

- Restricciones de relación entre las variables:

  El flujo del vértice s al vértice q se define por w_{sq}. Por lo tanto, w_{sq} debe ser igual a la suma de los caminos que van del vértice s al q para todos los vehículos, X_{sqv}.

  $$w_{sq} = \sum_{v\in M} X_{sqv}, \qquad s \neq q;\ s, q \in V = \{1, ..., n\}$$

- Restricciones de eliminación de subciclos:

  Toda ruta debe empezar y terminar en el origen es decir, no puede haber subciclos independientes, para ello añadimos,
  $u_s + (n-2)w_{1s} - w_{s1} \leq n-1, \qquad s \in V - \{1\} = \{2, ..., n\}$
  $u_s + w_{1s} + w_{s1} \geq 2, \qquad s \in V - \{1\} = \{2, ..., n\}$
  $u_s - u_q + (n)w_{sq} + (n-2)w_{qs} \leq n-1, \qquad s \neq q; s, q \in\in V - \{1\} = \{2, ..., n\}$

  El primer vértice recorrido siempre será el origen,
  $u_s \in \{2, ..., n\}, \forall s \in V - \{1\} = \{2, ..., n\}$

- Restricciones de no negatividad:

  $X_{ijv} \in \{0,1\}\ \forall i,j \in V,\ v \in M = \{1,...,m\}$.
  $y_{iv} \in \{0,1\}\ \forall i \in V,\ v \in M = \{1,...,m\}$.
  $w_{sq} \in \{0,1\}\ \forall s,q \in V$.
  $u_s \in \{1,...,n\}$

EJEMPLO

Pasados 3 años la empresa, debido a la buena gestión, ha aumentado su producción en las filiales y ahora la carga de materia prima que demandan las filiales ha aumentado considerablemente. Se ha llegado al punto en que un camión no puede abastecer el solo a todas ellas. En la empresa se contaba con dos camiones del tipo C11, es decir, con un eje delantero y un eje trasero que son capaces de transportar 10 toneladas de carga útil.

Por otro lado, las demandas de las filiales para su correcta producción son:

Filiales	I	II	III	IV	V	VI	VII	$VIII$	IX	X	**Total**
Demandas	0	2	3	3	5	5	10	10	7	3	48

(4.1)

Observamos que:

- La medida de las demandas estará en toneladas de materia prima.
- La filial 1 correspondiente al depósito no tendrá demanda obviamente, pues es de donde parte la carga.
- La carga total a distribuir, la suma de las demandas, es de 48 toneladas. Si cada camión puede transportar como máximo 10 toneladas necesitaremos como mínimo 5 camiones.
- Tanto la filial 7 como la 8 necesitarán un camíon cada una exclusivamente para abastecerse.

Desde el departamento de investigación y logística se estudia si al incorporar 5 vehículos más para realizar el transporte, cuál será la ruta y distribución de cada uno de modo que se realice el menor gasto posible y sin aumentar en exceso el tiempo de respuesta, cumpliendo las demandas de cada filial.

Este es un problema de ruta con capacidad limitada y por lo tanto, se modeliza como se ha planteado anteriormente, se tienen:

- 660 variables binarias.
- 376 condiciones que se deben cumplir.

Este ya es un problema de extensión considerable y difícil resolución. Para resolverlo se utiliza el método exacto Branch and Bound para resolverlo y mediante el software libre Coin-or y su paquete de optimización se busca una solución. El departamento de logística e investigación de la empresa de nuevo encuentra que la solución óptima para minimizar el gasto del servicio es la siguiente ruta para cada vehículo.
Se presenta una recreación de lo que serían las distintas rutas óptimas para cada vehículo y gastos empleados en cada trayecto así como las demandas satisfechas.

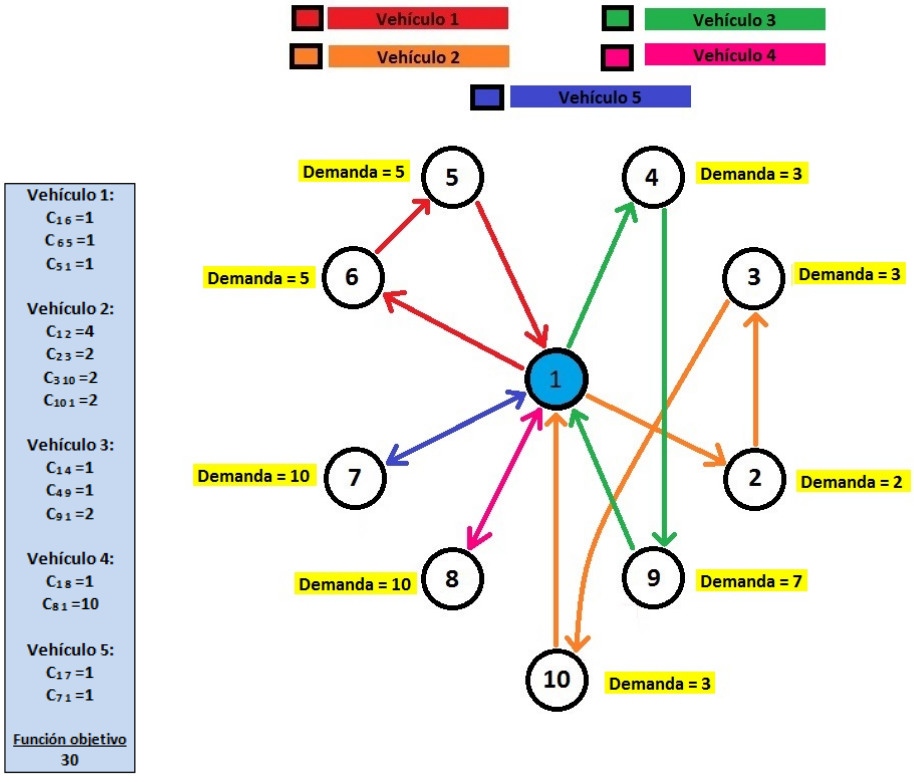

Figura 4.1: Esquema de la solución del CVRP para 5 vehículos con capacidad 10 toneladas.

El coste total de realizar la ruta con 5 vehículos abasteciendo las filiales será: Función objetivo $z = 30$.
Observamos, que el coste de transportar las 48 toneladas de materia prima a las filiales es de 30 unidades frente a las 6 unidades que gastabamos al principio. El único camión que no se lleno con las 10 toneladas es el segundo que distribuye 8. Obviamente los beneficios de procesar esas 48 toneladas de materia prima serán mucho mayores que antes y la empresa seguirá ganando peso e importancia en el mercado.

Vehículo 1:

Origen	1	Destino	6	**Demanda:** 5
Origen	6	Destino	5	**Demanda:** 5
Origen	5	Destino	1	
			Total:	10

Vehículo 2:

Origen	1	Destino	2	**Demanda:** 2
Origen	2	Destino	3	**Demanda:** 3
Origen	3	Destino	10	**Demanda:** 3
Origen	10	Destino	1	
			Total:	8

Vehículo 3:

Origen	1	Destino	4	**Demanda:** 3
Origen	4	Destino	9	**Demanda:** 7
Origen	9	Destino	1	
			Total:	10

Vehículo 4:

Origen	1	Destino	8	**Demanda:** 10
Origen	8	Destino	1	
			Total:	10

Vehículo 5:

Origen	1	Destino	7	**Demanda:** 10
Origen	7	Destino	1	
			Total:	10

4.5. Métodos de resolución

Se enuncian algunos de los últimos métodos conocidos de resolución,
Heurísticas como el Algoritmo de Clarke y Wright y el Algoritmo de Fisher y Jaikumar
Heurísticas de mejora
Heurísticas de intercambio
Recocido simulado
Búsqueda Tabú
Algoritmos genéticos
Optimización de colonias de hormigas

Para resolver el ejemplo de nuevo se utiliza el método exacto Branch and Bound aunque esta vez añadiremos cortes para facilitar la resolución.

4.6. Relación entre todas las variantes y el TSP

El GTSP, si consideramos que el número de clusters en los que dividimos las ciudades es el mismo que el número de ciudades, es decir, cada ciudad es un cluster tendremos de nuevo el problema del viajero inicial.

El mTSP, en el caso de que el número de vehículos, m, sea igual a 1 el problema será de nuevo el problema del viajero inicial y la solución coincidirá.

El GmTSP, al ser una combinación del GTSP y el mTSP ocurrirá lo mismo que con ellos. Si tomamos un vehículo y el número de clusters igual al número de ciudades tendremos el TSP de nuevo y la solución del problema será la misma.

El CVRP y el TSP están muy relacionados entre sí. De hecho, el VRP surgió como una extensión del TSP para el caso en el que la capacidad de los vehículos que realizan la ruta sea limitada y sea, por tanto, necesario realizar varias rutas. Por lo tanto, si tomamos como capacidad de los vehículos infinita y nos limitamos a un solo vehículo tendremos de nuevo el problema del viajero.

La principal diferencia existente entre los VRP y los TPS es que en el primer caso hay un subconjunto de nodos y/o arcos que se deben visitar teniendo en cuenta las demandas de cada uno y, en el segundo, se busca una ruta que una el origen y el destino sin importar los nodos o arcos intermedios.

Apéndice A

Explicación de los Programas

Se trata de explicar el funcionamiento de los programas y los comandos utilizados. Para ello, use utiliza de ejemplo el último programa, el de CVRP, por ser el más completo.

```
#include "pm.h"                    ((1))

int main()
{ ifstream entrada("DatosDeEntradaCVRP.dat",ios::in);                    ((2))
  ofstream salida("salidaRutaCapacidades.txt");//salida.                 ((3))

double *dels,*drowlo, *drowup, *dcollo, *dcolup, *dobj, *costos,*demandas;
int *mcolindx,*mrowindx; //indices.                                      ((4))
int ncols,nelements,nrows,n1,m1,Q,i,j,v,k,p,s;

//máximo 100 nudos y 10000 elementos no nulos.          ((5))
nrows=1500000;ncols=1500000;nelements=100000000;

// reserva de memoria.                                  ((6))
dels=new double[nelements];
mcolindx=new int[nelements];mrowindx=new int[nelements];
drowlo=new double[nrows];drowup=new double[nrows];
dcollo=new double[ncols];dcolup=new double[ncols]; dobj=new double[ncols];
costos=new double[ncols];demandas=new double[ncols];


const int ccortes=1;                    ((7))

double time1;                           ((8))

 //Lectura de datos, índice inicial 0.                ((9))
//Número de nudos n1.
        entrada>>n1;
//Lectura de costos, función objetivo; Los mismos costos en cada tramo.
        for (i=0;i<n1;i++) for (j=0;j<n1;j++) entrada >> costos[i+j*n1];
//Número de vehículos m1.
        entrada>>m1;
//Capacidad de vehículos Q.
        entrada>>Q;
//Demandas de los vértices.
        for(i=0;i<n1;i++) entrada>>demandas[i];

int *enteras; enteras=new int[m1*n1*n1+n1*m1+n1*n1+n1];                  ((10))
```

```
//Pasar los coeficientes de la función objetivo y cotas de las variables.      ((11))
//tramos n1 porque hay que volver al origen.

for(v=0;v<m1;v++){
        for (i=0;i<n1;i++){
            for (j=0;j<n1;j++){
                        dobj[i+j*n1+v*n1*n1]=costos[i+j*n1];
                        dcollo[i+j*n1+v*n1*n1]=0;
                        dcolup[i+j*n1+v*n1*n1]=1;
                        if(i==j) dcolup[i+j*n1+v*n1*n1]=0; // no de i a i.
                } }}

for (k=0;k<n1;k++){
    for(p=0;p<m1;p++){
                        dcollo[m1*n1*n1+k+n1*p]=0;
                        dcolup[m1*n1*n1+k+n1*p]=1;
                }}

for(i=0;i<n1;i++){
        for(j=0;j<n1;j++){
                        dcollo[m1*n1*n1+n1*m1+i+j*n1]=0;
                        dcolup[m1*n1*n1+n1*m1+i+j*n1]=1;
                }}

for(s=0;s<n1;s++){
                        dcollo[m1*n1*n1+n1*m1+n1*n1+s]=0.;
                        dcolup[m1*n1*n1+n1*m1+n1*n1+s]=1.0*(n1);
                }

int neleproblem=0;                          ((12))
int rowsproblem=0;

                        //RESTRICCIONES.                              ((13))

//1) Cada cliente debe ser visitado una vez.

for(j=1;j<n1;j++){
        drowlo[rowsproblem]=1; drowup[rowsproblem]=1;
                for(v=0;v<m1;v++){
                        for(i=0;i<n1;i++){
                                if(i!=j){
                                        dels[neleproblem]=1;
                                        mcolindx[neleproblem]=i+j*n1+v*n1*n1;
                                        mrowindx[neleproblem]=rowsproblem;
                                        neleproblem++;}}}
rowsproblem++;}

//2) Del depósito salen m1 vehículos.
drowlo[rowsproblem]=m1; drowup[rowsproblem]=m1;
        for(v=0;v<m1;v++){
                for(j=1;j<n1;j++){
                                dels[neleproblem]=1;
                                mcolindx[neleproblem]=j*n1+v*n1*n1;
                mrowindx[neleproblem]=rowsproblem;
                                neleproblem++;}}
rowsproblem++;

drowlo[rowsproblem]=m1; drowup[rowsproblem]=m1;
for(p=0;p<m1;p++){
                                dels[neleproblem]=1;
                                mcolindx[neleproblem]=m1*n1*n1+n1*p;
```

```
                mrowindx[neleproblem]=rowsproblem;
                neleproblem++;}
rowsproblem++;

//3)El mismo número de vehículos que entran saldrán de cada cliente.
for (i=0;i<n1;i++){
for (v=0;v<m1;v++){
drowlo[rowsproblem]=0; drowup[rowsproblem]=0;
        for(j=0;j<n1;j++){
                        dels[neleproblem]=1;
                mcolindx[neleproblem]=i+j*n1+v*n1*n1;
                mrowindx[neleproblem]=rowsproblem;
                neleproblem++;}

                        dels[neleproblem]=-1;
                        mcolindx[neleproblem]=m1*n1*n1+i+v*n1;
                mrowindx[neleproblem]=rowsproblem;
                neleproblem++;
rowsproblem++;}}

for (i=0;i<n1;i++){
for (v=0;v<m1;v++){
drowlo[rowsproblem]=0; drowup[rowsproblem]=0;
                        for(j=0;j<n1;j++){
                        dels[neleproblem]=1;
                mcolindx[neleproblem]=j+i*n1+v*n1*n1;
                mrowindx[neleproblem]=rowsproblem;
                neleproblem++;}

                        dels[neleproblem]=-1;
                        mcolindx[neleproblem]=m1*n1*n1+i+v*n1;
                mrowindx[neleproblem]=rowsproblem;
                neleproblem++;
rowsproblem++;}}

//4) Restricciones de capacidad.
for(v=0;v<m1;v++){
        drowlo[rowsproblem]=0; drowup[rowsproblem]=10;
                for(k=0;k<n1;k++){
                        dels[neleproblem]=demandas[k];
                        mcolindx[neleproblem]=m1*n1*n1+k+v*n1;
                        mrowindx[neleproblem]=rowsproblem;
                        neleproblem++;}
rowsproblem++;}

//5) Restricción de relación entre variables Yij.
//El flujo del vértice i al j se define por y_{ij}.
for(i=0;i<n1;i++){
        for (j=0;j<n1;j++){
                if (i!=j){
                        drowlo[rowsproblem]=0.; drowup[rowsproblem]=0.;
                        for(v=0;v<m1;v++){
                                dels[neleproblem]=1.;
                                mcolindx[neleproblem]=i+j*n1+v*n1*n1;
                                mrowindx[neleproblem]=rowsproblem;
                                neleproblem++; }

                        dels[neleproblem]=-1.;
                        mcolindx[neleproblem]=m1*n1*n1+n1*m1+i+j*n1;
                        mrowindx[neleproblem]=rowsproblem;
                        neleproblem++;
rowsproblem++;
```

```
                        }}}

//6) Restricción de eliminación de subciclos.

//1.
for(s=1;s<n1;s++){
        drowlo[rowsproblem]=0; drowup[rowsproblem]=1.0*(n1-1);
                dels[neleproblem]=1;
                mcolindx[neleproblem]=n1*n1*m1+n1*m1+n1*n1+s;
                mrowindx[neleproblem]=rowsproblem;
                neleproblem++;
                dels[neleproblem]=1.0*(n1-2);
                mcolindx[neleproblem]=n1*n1*m1+n1*m1+s*n1;
                mrowindx[neleproblem]=rowsproblem;
                neleproblem++;
                dels[neleproblem]=-1;
                mcolindx[neleproblem]=n1*n1*m1+n1*m1+s;
                mrowindx[neleproblem]=rowsproblem;
                neleproblem++;
rowsproblem++;
                        }

//2.
for(s=1;s<n1;s++){
        drowlo[rowsproblem]=2; drowup[rowsproblem]=10^20;
                dels[neleproblem]=1;
                mcolindx[neleproblem]=n1*n1*m1+n1*m1+n1*n1+s;
                mrowindx[neleproblem]=rowsproblem;
                neleproblem++;
                dels[neleproblem]=1;
                mcolindx[neleproblem]=n1*n1*m1+n1*m1+s*n1;
                mrowindx[neleproblem]=rowsproblem;
                neleproblem++;
                dels[neleproblem]=1;
                mcolindx[neleproblem]=n1*n1*m1+n1*m1+s;
                mrowindx[neleproblem]=rowsproblem;
                neleproblem++;
rowsproblem++;
                        }
//3.
for(s=1;s<n1;s++){
        for(k=1;k<n1;k++){
                if(k!=s){
                        drowlo[rowsproblem]=-1.0*n1; drowup[rowsproblem]=1.0*(n1-1);
                                dels[neleproblem]=1.;
                                mcolindx[neleproblem]=n1*n1*m1+n1*m1+n1*n1+s;
                                mrowindx[neleproblem]=rowsproblem;
                                neleproblem++;
                                dels[neleproblem]=-1.;
                                mcolindx[neleproblem]=n1*n1*m1+n1*m1+n1*n1+k;
                                mrowindx[neleproblem]=rowsproblem;
                                neleproblem++;
                                dels[neleproblem]=1.0*(n1);
                                mcolindx[neleproblem]=n1*n1*m1+n1*m1+s+k*n1;
                                mrowindx[neleproblem]=rowsproblem;
                                neleproblem++;
                                dels[neleproblem]=1.0*(n1-2);
                                mcolindx[neleproblem]=n1*n1*m1+n1*m1+k+s*n1;
                                mrowindx[neleproblem]=rowsproblem;
                                neleproblem++;
 rowsproblem++;
                }}}
```

```
//4.
for(s=1;s<n1;s++){
        drowlo[rowsproblem]=1; drowup[rowsproblem]=n1;
                               dels[neleproblem]=1.;
                               mcolindx[neleproblem]=n1*n1*m1+n1*m1+n1*n1+s;
                               mrowindx[neleproblem]=rowsproblem;
                               neleproblem++;
        rowsproblem++;}


//              RESOLUCIÓN.

//Matriz por indices.
   OsiClpSolverInterface sol1;                          ((14))
   CoinPackedMatrix A(true,mrowindx,mcolindx,dels,neleproblem); ((15))
   sol1.loadProblem(A,dcollo,dcolup,dobj,drowlo,drowup);
   sol1.setObjSense(1);                                      ((16))
   //salvar en formato mps.
      sol1.writeMps("RutaCapacidades");                            ((17))

//Solución inicial lineal.
   sol1.initialSolve();                                 ((18))
   for (i=0;i<m1*n1*n1+n1*m1+n1*n1+n1;i++)  enteras[i]=i;
   for (i=0;i<m1*n1*n1+n1*m1+n1*n1+n1;i++) sol1.setInteger(enteras[i]);

        if(ccortes==1){
        //CORTES.                                           ((19))
      CglKnapsackCover cubrimientos;
      CglGomory gomory;
      CglProbing fijar;
          OsiCuts cortes1;
      double obj = sol1.getObjValue();
      cubrimientos.generateCuts(sol1,cortes1);
      gomory.generateCuts(sol1,cortes1);
      fijar.generateCuts(sol1,cortes1);

      //se aplican si la efectividad es >=0,0 .
      sol1.applyCuts(cortes1,0.0);

   salida<<"\n APLICAMOS CORTES "<<"\n";
   salida<<"\n El número de cortes es: "<<cortes1.sizeCuts()<<"\n";
   if(cortes1.sizeCuts()>0){
   sol1.resolve();
   salida <<"\n La función objetivo cambia de \n"<<obj <<" a " <<sol1.getObjValue();
   salida<< "\nNúmero de variables:"<< sol1.getNumCols();
   salida<< "\nNúmero de condiciones:"<< sol1.getNumRows();
   salida<< "\nFunción objetivo:"<< sol1.getObjValue();
   salida<<"\nLa solución es:";
   for (i=0;i<sol1.getNumCols();i++)salida<<"  "<< sol1.getColSolution()[i];
   salida<< "\n ";
          }
   time1 = CoinCpuTime();
   salida<<"\n El tiempo en realizar los cortes de CPU es: "<<time1;
   salida<<"\n";
   salida<<"\n";}

//  Branch and Bound.          ((21))

   sol1.branchAndBound();
   if (sol1.getObjValue()>1e31){ salida << "MIP Unbounded"; goto fin; }          ((22))
   if (!sol1.isProvenOptimal()){ salida << " The optimum is not found"; goto fin; } ((23))
```

```
   salida<<"\n SOLUCIÓN ENTERA "<<"\n";                                    ((24))
   salida<<"\n Número de nudos:"<< n1;
   salida<<"\n Número de vehículos:"<< m1;
   salida<<"\n Capacidad de vehículos:"<< Q;
   salida<<"\n Demandas:";for(i=0;i<n1;i++){salida<< " "<<demandas[i];}
   salida<< "\n Función objetivo:"<< sol1.getObjValue();
   salida<< "\n Número de variables:"<< sol1.getNumCols()<<" "<<m1*n1*n1+n1*m1+n1*n1+n1;
   salida<< "\n Número de condiciones:"<< sol1.getNumRows();
   salida<<"\n La  solución actual  es:\n";
   salida<< "\n ";
   double gasto;
   for(v=0;v<m1;v++){
         gasto=0.;
         salida<< "vehículo ****** "<< v+1<<"\n";
         for(j=0;j<n1;j++){
                  for(i=0;i<n1;i++){
                           if(abs(sol1.getColSolution()[v*n1*n1+j*n1+i])>1.e-8){
                              gasto=gasto+demandas[i];
                              salida<<"Origen "<<i+1<<"  Destino "<<j+1<<"
"<<"\n ";}
                                    } }
                  salida<<"Gastos en demandas: "<<gasto<<"\n ";}

   salida<<"\n";
   salida<<"\n";

  //X_{ijv}.
  for (v=0;v<m1;v++) {
  salida<<"\n";
  for(j=0;j<n1;j++){
  salida<<"\n";
  for(i=0;i<n1;i++){
  salida<<"("<<i+1<<","<<j+1<<","<<v+1<<")"<<sol1.getColSolution()[v*n1*n1+j*n1+i]<<"";}}}

  salida<<"\n";
  salida << "\n";
  salida << "\n";

  // Y_{ij}.
  salida << "Yij\n";
  for (j = 0; j<n1; j++){
   salida << " \n";
   for (i = 0; i<n1; i++){
    salida <<"("<<j + 1<<","<<i + 1<<")"<< sol1.getColSolution()[n1*n1*m1+n1*m1+i*n1+j]<<"";
                }
                }
   salida << "\n";

 //U_s.
 salida << "\n  ***   Us\n";
 for (j = 0; j<n1; j++){
 salida <<"vértice:"<<j+1<<"orden:"<<1+sol1.getColSolution()[n1*n1*m1+n1*m1+n1*n1+j]<<"";
 salida << "\n";
        }

   double time2= CoinCpuTime();
   salida<<"\n El tiempo resolver el problema entero de CPU es: "<<time2;
   salida<<"\n";
   salida<<"\n";

   if(ccortes==1){
         salida<<"\n El tiempo resolución total de CPU es: "<<(time2 + time1);}
```

```
fin:

entrada.close();                                    ((25))
salida.close();
return 0;
}
```

EXPLICACION

(1) Se incluye el archivo de encabezado pm.h donde se tienen incluidas todas las librerías que se utilizarán para resolver el problema. El archivo pm.h será el mismo para todos los programas.

(2) DatosDeEntradaCVRP.dat será el fichero de entrada de datos por índices. Se tendrán distintos ficheros de entrada según las necesidades de cada problema a resolver y los datos necesarios.

(3) salidaRutaCapacidades.txt será nuestro archivo de texto donde volcaremos la solución del problema. Para ello borramos cualquier dato que tuviera almacenado anteriormente. Para cada problema tendremos un fichero de salida diferente.

(4) Se inicializan las variables que se utilizarán en el programa, tanto índices como variables donde guardaremos los datos.

(5) Se dimensionan las variables de referencia *nrows*, *ncols*, *nelements*. Es decir, se dará el máximo número de elementos, filas y columnas que podrá tener la matriz de restricciones.

(6) A partir de las variables de referencia dimensionamos los vectores que tendremos que mandar al solver para que lea el problema por índices.

(7) Se define la variable *ccortes*. Esta indica que si el valor es 1 se realizan cortes antes de resolver el problema entero y si está en 0 no se realizaran.

(8) Se inicializa la variable de tiempo.

(9) Se leen los datos del fichero de entrada que se ha introducido antes. Es decir, número de nudos, costos, número de vehículos, capacidad de los vehículos, clusters, demandas, etc.

(10) Se define la variable *enteras* donde se almacenarán las variables que sean enteras. Se dimensiona de antemano con el número de variables enteras que va a haber.

(11) Se definen las variables a resolver junto con sus cotas superiores e inferiores y el coeficiente en la función objetivo.

(12) Se inicializan a 0 los contadores *neleproblem* y *rowsproblem*. Uno será el de los elementos y el otro el de las filas de la matriz de restricciones.

(13) Introducimos las restricciones correspondientes a cada problema.

(14) Se indica el solver que se va a utilizar para resolver el problema y la solución se guarda en sol1. Se puede usar el solver de COIN-OR o el de C-PLEX sólo hay que cambiar *OsiClpSolverInterface sol1;* por *OsiCpxSolverInterface sol1;*.

(15) Se define la matriz de restricciones para COIN por el método de índices. Es decir, se le pasan las variables:

- **mrowindx**: vector de índices de cada fila de la matriz de restricciones A.
- **mcolindx**: vector de índices de cada columna de la matriz de restricciones A.
- **dels**: vector de los elementos de la matriz de restricciones A por columnas.
- **neleproblem**: número de elementos de la matriz de restricciones A.

Después se carga el problema para ello se pasan las variables:

- **A**: la matriz de restricciones creada justo antes.
- **dcollo**: vector real con las cotas inferiores de las variables.
- **dcolup**:vector real con las cotas superiores de las variables.
- **dobj**:vector de costes de las variables en la función objetivo.
- **drowlo**: vector de las cotas inferiores de las restricciones.
- **drowup**: vector de las cotas superiores de las restricciones.

(16) Indica si se debe calcular el máximo o el mínimo de la función objetivo. Si es un problema de máximo se pone el -1 si se pone el 1 será un problema de mínimo o 0 si no hay función objetivo.

(17) Se salva el problema en un archivo mps. Servirá por si queremos volver a cargar el problema evitar que volver a leer todo fichero de texto de entrada por índices, así como las restricciones, coeficientes de la función objetivo y las cotas.

(18) Se resuelve la relajación lineal sin tener en cuenta las variables enteras. A continuación, se define cuales de ellas son enteras.

(19) En el caso de que la variable *ccortes*= 1 y se vayan a aplicar los cortes se entrará en este apartado.

(20) Una vez realizados los cortes se vuelcan los resultados al fichero de salida.

(21) Se resuelve el problema entero con el método de Branch and Bound. Para ello se utiliza *sol1.branchAndBound();*.

(22) Para problemas de bastante complejidad se le añade está condición que detiene el problema si encuentra que está inacotado. Sirve de gran ayuda cuando no se logra encontrar la solución para saber detectar el problema.

(23) Otra condición que detiene el problema esta vez si no es capaz de encontrar el óptimo. También muy útil para detectar errores.

(24) Se vuelca la solución entera encontrada en el fichero de salida.

(25) Se cierran los ficheros abiertos durante el programa.

OBSERVACIÓN: *Todos los programas que se han utilizado para resolver los ejemplos planteados junto con sus ficheros de datos iniciales y sus ficheros de salida se encuentran en el CD-R del final.*

Apéndice B

Experiencia Computacional

En los problemas de optimización entera mixta de gran extensión y complejidad un aspecto a tener muy en cuenta es la velocidad de computo y el tiempo que tarda el ordenador en resolver el problema.
En este apéndice se va a estudiar la diferencia de tiempos al introducir cambios en el programa y resolverlo con distintos solvers. Para ello se utilizará el último ejemplo planteado, el CVRP para 10 vértices y 5 vehículos de capacidad 10. Sin ser un problema aparentemente muy complicado a la hora de resolver nos encontramos con 660 variable lo que hacen que resolverlo tarde bastante tiempo.

Se comparan los tiempos de resolución utilizando primero el método de Branch and Bound. Se resuelve la relajación lineal y luego se aplica el método con el solver de COIN-OR. COIN-OR es una iniciativa para estimular el desarrollo de software de código abierto para la comunidad de investigación operativa. Ver[25]. Para ello en el programa se introduce

```
OsiClpSolverInterface sol1
```

Después, se vuelve a resolver el problema añadiendo los distintos cortes que nos ofrece el solver de COIN-OR. Es decir, se añaden los cortes de Gomory, Probing y los cubrimientos Knapsack. Añadir cortes consiste en que a partir de una solución no entera se van construyendo planos de corte, de tal forma que los cortes asociados a los mismos generan de forma iterada la solución entera buscada, si existe. Para ello dentro del programa se cambia la variable ccortes de 0 a 1.

```
const int ccortes=1
```

Al añadir los cortes reducimos el tiempo de resolución del problema entero a 15.152

Por último, volveremos a resolver esta vez utilizando otro solver, esta vez uno de pago y uno de los más potentes del mercado, el CPLEX. Se notará la

rapidez de resolución debido a la robusted de los algoritmos que contiene el solver. Para ello cambiamos en el programa

```
OsiCpxSolverInterface sol1
```

Se presenta tabla comparativa de tiempos totales de cómputo.

MÉTODO	COIN-OR	COIN-OR + Cortes	C-PLEX
TIEMPO	417.554	15.152	0.94

Se observa que, a pesar de no parecer un problema muy complicado, la extensión de este hace que el ordenador utilizando sólo el método de Branch and Bound con el solver de COIN tarde 417.554 segundos. Mucho tiempo si se tiene en cuenta que sólo se están teniendo en cuenta 10 vértices.
Utilizando los diferentes cortes que nos ofrece COIN el ordenador consigue resolverlo en un tiempo razonable de aproximadamente 15 segundos. Para un problema de 10 vértices ya notamos la diferencia y ventaja de añadir cortes pues, el tiempo se reduce bastante.
Por otra parte, el potente solver de C-PLEX lo resuelve en menos de un segundo. Según aumente el tamaño del problema las diferencias de tiempo a favor del solver de C-PLEX frente al de COIN se harán aún mas notables.

Observación: Para resolver este problema se ha cambiado el fichero de entrada de datos en los costes. Mientras que en los anteriores el coste infinito se ponía como e^{20}, en este problema, para que el ordenador sea capaz de resolverlo, se pone como e^{31}. Este es un problema del software pues para resolverlo necesita más eficacia.

Bibliografía

[1] L.Jourdan, M.Basseur, E-G.Talbi: Hybridizing exact methods and metaheuristics: A taxonomy. European Journal of Operations Research vol n 199(3) pp.620-629 (2009)

[2] S-W.Lin, Z-J.Lee, K-C.Ying, C-Y.Lee: Applying hybrid meta-heuristics for capacitated vehicle routing problem. Expert Systems with Applications vol n 36 pp.1505-1512 (2009)

[3] R.Baldacci, P.Toth, D.Vigo: Recent advances in vehicle routing exact algorithms. 4OR vol n 5 pp.269-298 (2007)

[4] G.Laporte: What You Should Know about the Vehicle Routing Problem. Naval Research Logistics vol n 54(8) pp.811-819 (2007)

[5] G.Clarke, J.V.Wright: Scheduling of vehicles from a central depot to a number of delivery points. Operations Research vol n 12 pp.568-581 (1964)

[6] S.Lin, B.W.Kernighan: An Effective Heuristic Algorithm for the Traveling-Salesman Problem. Operations Research vol n 21 pp. 498-516 (1973)

[7] M.L.Fisher: The lagrangian relaxation method for solving integer programming problems. Management Science vol n 27(1) pp.1-18 (1981)

[8] M.Guignard: Lagrangean Relaxation. Sociedad de Estadística e Investigación Operativa Top vol n 11(2) pp.151-228 (2003)

[9] A.Frangioni: About Lagrangian Methods in Integer Optimization. Annals Operations Research vol n 139 pp.163-193 (2005)

[10] S. Ólafsson: Chapter 21: Metaheuristics, En Handbooks in Operations Research and Management. Science, J. Pérez (Ed.), Ciudad(es) de Edición, Editorial, pp.633-654 (2006)

[11] J.Dréo, P.Siarry, A.Pétrowski, E.Taillard: Metaheuristics for Hard Optimization. Primera Edición, Berlin, Springer-Verlag, (2006)

[12] A.S.Fraser: Simulation of genetic systems by automatic digital computers. I. Introduction. Australian Journal of Biological Sciences vol n 10 pp.484-491 (1957)

[13] B.M.Baker, M.A.Ayechew: A genetic algorithm for the vehicle routing problem. Computers & Operations Research vol n 30 pp.787-800 (2003)

[14] C.Prins: A simple and effective evolutionary algorithm for the vehicle routing problem. Computers & Operations Research vol n 31 pp.1985-2002 (2004)

[15] C-H.Wang, J-Z. Lu: A hybrid genetic algorithm that optimizes capacitated vehicle routing problems. Expert Systems with Applications vol n 36 pp.2921-2936 (2009)

[16] J.Brandão: A deterministic tabu search algorithm for the fleet size and mix vehicle routing problem. European Journal of Operational Research vol n 195 pp.716-728 (2009)

[17] Weisstein, Eric W. "Icosian Game". From MathWorld–A Wolfram Web Resource. http://mathworld.wolfram.com/IcosianGame.html

[18] Karl Menger - American Mathematical Society. http://www.ams.org/notices/199605/comm-menger.pdf

[19] Solution of a Large-Scale Traveling-Salesman Problem. Vasek Chvàtal, William J.Cook, George B. Dantzig, Delbert R. Fulkerson, Selmer M. Johnson.

[20] G.B. Dantzig, Application of the simplex method to a transportation problem, Activity Analysis of Production and Allocation (T.C. Koopmans, ed.), Cowles Commission Monograph No. 13. JohnWiley & Sons, Inc., New York, N. Y.; Chapman & Hall, Ltd., London, 1951, pp. 359-373.

[21] Richard ("Dick") Manning Karp United States 1985. http://amturing.acm.org/award_winners/karp_3256708.cfm

[22] The Traveling Salesman Problem: A Computational Study David L. Applegate, Robert E. Bixby, Vasek Chvàtal and William J. Cook. Princeston series in applied mathematics.

[23] Concorde tsp solver. http://www.math.uwaterloo.ca/tsp/concorde.html

[24] TSPLIB. https://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/

[25] COIN-OR: http://www.coin-or.org/