USC
UNIVERSIDADE DE SANTIAGO DE COMPOSTELA
FACULTADE DE MATEMÁTICAS

Traballo Fin de Grao

Problemas de rutas de vehículos e algoritmos de aforro: o algoritmo de Clarke and Wright

Elena Blanco González

2018/2019

UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

GRAO DE MATEMÁTICAS

Traballo Fin de Grao

Problemas de rutas de vehículos e algoritmos de aforro: o algoritmo de Clarke and Wright

Elena Blanco González

Xullo, 2019

UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

Traballo proposto

Área de Coñecemento: Estatística e Investigación Operativa
Título: Problemas de rutas de vehículos e algoritmos de aforro: o algoritmo de Clarke and Wright
Breve descrición do contido
Os problemas de rutas de vehículos foron introducidos por Dantzig e Ramser (1959). O interese nesta área de investigación é teórico e aplicado. A motivación académica para resolver estes problemas é que non é posible construír un algoritmo de tempo polinomial que resolva calquera instancia do problema. Os denominados algoritmos heurísticos constitúen ferramentas efectivas para resolver problemas de gran complexidade. Entre eles, o proposto por Clarke and Wright (1964) que tivo grande interese na literatura e serviu como punto de partida para modificacións aplicables a modelos máis xerais. Neste traballo preténdese facer unha revisión bibliográfica de diferentes modelos de problemas de rutas de vehículos, comezando polo máis básico e pasando por outros como, por exemplo, o modelo con vehículos con diferentes compartimentos, con entregas divididas ou modelos estocásticos. Aprenderemos a modelalos con linguaxe AMPL para, a continuación, estudar o algoritmo de aforros de Clarke and Wright. O estudo destes modelos e algoritmos acompañaranse da ilustración dos mesmos por medio de exemplos e datos da vida real.

Índice xeral

Resumo

Os Problemas de Rutas de Vehículos (VRP) son un exemplo de problemas para os que moitas veces non é posible aplicar algoritmos que os resolvan en tempo polinomial, principalmente debido ao seu tamaño e á súa complexidade. Habitualmente para resolver exemplos deste tipo de problemas usaranse algoritmos heurísticos. Neste traballo presentamos o Problema do Viaxante de Comercio (TSP) para continuar cos Problemas de Rutas de Vehículos. A continuación enunciaremos o algoritmo heurístico proposto por Clarke and Wright para o VRP con restricións de capacidade. Finalmente contaremos cos datos dun problema de rutas da vida real que usaremos para comparar as solucións obtidas usando unha implementación en R do algoritmo de Clarke and Wright coa solución obtida a dito problema mediante a súa modelación con linguaxe AMPL.

Abstract

The Vehicle Routing Problems (VRP) are an example of problems where it can not be possible to apply an exact method to solve them in a polynomial time in many times due to their complexity or size. Frequently to solve examples of this kind of problems heuristic algorithms will be used. In this project we will first present the Travelling Salesman Problem (TSP) to continue with the Vehicle Routing Problems. Next, we will show the heuristic algorithm proposed by Clarke and Wright for Capacited VRP. Finally, based on a real-world problem, we will compare the solutions obtained by an R implementation of Clarke and Wright algorithm versus the results returned by an AMPL model used together with an exact optimization method.

Introdución

As raíces da investigación de operacións remóntanse moitas décadas atrás cando se fixeron os primeiros intentos de empregar o enfoque científico na administración dunha empresa. Porén, o inicio da actividade chamada *investigación de operacións* adoita atribuírse á Segunda Guerra Mundial, cando había unha gran necesidade de administrar os recursos loxísticos e as liñas de subministración. A Forza Aérea Británica formou o primeiro grupo que desenvolvería métodos cuantitativos para resolver estes problemas operacionais, pouco despois, as Forzas Armadas dos Estados Unidos formaron un grupo similar, composto por científicos de distintos campos, cinco dos cales foron logo laureados co premio Nobel.

O primeiro traballo no eido da investigación de operacións atribúese ao matemático e economista ruso L. V. Kantorovich en 1939, aínda que dito traballo non viu a luz ata 1959. Por iso é o matemático e físico G. B. Dantzig quen é considerado o pai da programación linear tal e como a coñecemos na actualidade, debido aos seus traballos en 1947 como asesor das Forzas Aéreas dos Estados Unidos no desenvolvemento dunha ferramenta para a automatización da planificación de actividades como despregue de tropas, adestramentos e demais loxística das tropas aliadas.

A programación linear é unha parte importante da investigación operativa, adicada á optimización (maximización ou minimización) de funcións lineais, suxeitas a un conxunto de restricións tamén lineais de igualdade ou desigualdade. En 1949 Dantzig publicou o método símplex para resolver problemas de programación linear e, dende entón, é o algoritmo de referencia para resolver este tipo de problemas, independentemente do seu tamaño (aínda que para problemas moi grandes existen outros algoritmos que poden ofrecer un mellor rendemento dependendo da estrutura concreta do problema en estudo). Este método baséase en ir saltando de punto extremo en punto extremo da rexión factible ata atopar un punto extremo óptimo.

Estimulados polo evidente éxito da investigación de operacións en temas militares, na industria comezaron a interesarse de novo por este eido. Os primeiros esforzos adicáronse a desenvolver modelos apropiados e procedementos correspondentes para solucionar

problemas que xurdían en áreas tales como a programación de refinerías de petróleo, a distribución de produtos, a planificación de produción, o estudo de mercados e a planificación de inversións.

Aínda que a programación linear permite modelar e resolver problemas de optimización en multitude de campos e aplicacións, existen moitos problemas para os que as solucións fraccionarias poden non ser realistas e deberanse impoñer ao problema de programación linear restricións de integralidade das variables que se contemplan de forma explícita. Este tipo de problemas coñécense como *problemas de programación linear enteira*, ou como *programación enteira mixta* cando só algunhas das variables están restrinxidas a valores enteiros. As técnicas de optimización non linear son habitualmente moito máis custosas dende o punto de vista computacional e poden non garantir a converxencia a óptimos globais do problema.

Un algoritmo é un procedemento matemático, paso a paso, descrito a través de instrucións totalmente inequívocas, que empeza nunhas condicións iniciais especificadas e, eventualmente, finaliza devolvendo o resultado desexado ou unha aproximación do mesmo. Á hora de traballar con algoritmos, xorde de forma natural a necesidade de saber como de bo é un algoritmo dado. Este tipo de preguntas non comezaron a estudarse formalmente ata os primeiros anos da década dos 70 e deron lugar a un dos campos máis activos en matemáticas e ciencias computacionais: a teoría da complexidade computacional.

Tipicamente, o enfoque utilizado consiste en acoutar a cantidade de operacións que o algoritmo necesitará en función do tamaño dos datos necesarios para especificar o problema. Diremos que un algoritmo ten unha velocidade $O(f(n))$ se existen constantes k e n_0 tales que o número de operacións efectuadas polo algoritmo para resolver calquera problema de tamaño $n \geq n_0$ é, ao sumo, $kf(n)$. Por exemplo, un algoritmo que necesita $\log(n)+100n+n^2+0{,}001n^3$ operacións para resolver un problema de tamaño n, ten unha velocidade $O(n^3)$ é dicir, $f(n) = n^3$ pois basta tomar $k > 0{,}001$ e entón, se eliximos n_0 suficientemente grande teremos que $kn^3 > \log(n)+100n+n^2+0{,}001n^3$. Na práctica interesa atopar algoritmos cuxo tempo de computación sexa, como moito, polinomial, xa que os algoritmos non polinomiais poden ser moi pouco útiles na práctica para estudar problemas grandes.

Dado un problema, dicimos que pertence á clase de complexidade P se existe algún algoritmo que resolva calquera exemplo de dito problema nun tempo polinomial e dicimos que pertence á clase de complexidade NP se existe algún algoritmo que pode verificar calquera solución dun problema de dita clase en tempo polinomial. É importante notar que $P \subset NP$, pois calquera problema fácil de resolver será fácil de verificar. A gran pregunta da teoría de complexidade computacional é se estas dúas clases coinciden, se se cumpre $P = NP$, é dicir, se calquera problema fácil de verificar será tamén fácil de resolver. Aínda

que parece que ambas clases son diferentes, ata o momento ninguén conseguiu probalo e, en caso de que ambas clases fosen iguais, tería unha gran repercusión en multitude de campos pois, por exemplo, isto significaría que existe un algoritmo polinomial para factorizar en números primos, o cal está detrás de moitos protocolos de seguridade.

No estudo das distintas clases de complexidade, resulta de especial utilidade transformar uns problemas noutros. Supoñamos que un problema P_1 é tal que calquera exemplo en dita clase se pode transformar en tempo polinomial nun exemplo da clase P_2. Neste caso, dicimos que P_1 se reduce a P_2 en tempo polinomial. Se a clase P_2 pertence a P, entón tamén a clase P_1 pertence a P pois dado un elemento de P_1 poderémolo resolver en tempo polinomial sen máis que transformalo previamente (en tempo polinomial) nun elemento de P_2.

Este enfoque permite definir a clase de problemas NP-completos como os máis difíciles de todos os de NP. Un problema $P_1 \in NP$ é NP-completo se calquera problema en NP se pode reducir a P_1 en tempo polinomial. Por outra parte, un problema é NP- duro se calquera problema da clase NP se pode reducir a él en tempo polinomial. A diferenza entre as clases NP-completo e NP-duro é que un problema pode ser NP-duro sen pertencer á clase NP.

Dous problemas moi coñecidos da clase NP-duro son o problema do viaxante de comercio (TSP) e os problemas de rutas de vehículos (VRP). Tras esta introdución á Investigación Operativa -que se fixo seguindo os apuntamentos de González Díaz (2018) e as publicacións de Hillier e Lieberman (1997), e Mathur e Solow (1996)-, no primeiro capítulo do presente traballo trataremos o TSP, a evolución histórica dende o seu nacemento ata a actualidade, a definición do problema coa súa correspondente formulación e métodos de resolución do mesmo. Nun segundo capítulo presentaremos o VRP, diferentes variantes da súa formulación e algunhas das variantes do VRP.

Habitualmente, ante un problema que non se pode resolver en tempo polinomial, adoita renunciarse a ter unha solución exacta, debido ao esforzo e tempo computacional necesario para a súa obtención en exemplos moi grandes, e facer uso dos algoritmos aproximados e heurísticos que, nun tempo razoable proporcionan solucións cuasi-óptimas. Para a resolución da versión máis básica do VRP presentaremos no terceiro capítulo o algoritmo heurístico de Clarke and Wright (1964), o cal implementaremos en R e, partindo dos datos dun problema do mundo real, faremos unha comparación entre a solución proporcionada polo mesmo e a solución exacta usando a linguaxe AMPL (modelado do problema) xunto cun método de optimización exacta como Gurobi (para resolver o problema modelado) nun cuarto capítulo.

Finalmente faremos un pequeno apartado de conclusións, enumeraremos as fontes bi-

bliográficas utilizadas para a redacción do TFG e contaremos con catro apéndices. O primeiro apéndice é unha revisión de conceptos de teoría de grafos necesarios para seguir o traballo, e nos seguintes presentaremos táboas con datos dos tempos de execución dos diferentes algoritmos, así como os códigos de AMPL e R usados.

Capítulo 1

Problema do viaxante de comercio

Neste primeiro capítulo facemos unha breve introdución histórica das orixes da teoría de grafos así como do TSP e, a continuación, presentamos unha formulación para o TSP e enunciamos algúns métodos de resolución do mesmo. Para a súa elaboración utilizamos principalmente o libro de Biggs et al. (1976), o artigo de Chauhan et al. (2012), os apuntamentos de González Díaz (2018), o capítulo de Matai et al. (2010) e a páxina web da facultade de matemáticas na Universidade de Waterloo(2019) entre outras.

1.1. Introdución histórica

As orixes da teoría de grafos foron modestas pois, mentres os matemáticos do século XVIII trataban de resolver problemas de cálculo, movemento e medida, os problemas que encamiñaban cara o desenvolvemento da teoría de grafos eran considerados como quebracabezas. Porén, a pesar da aparente trivialidade destes quebracabezas, lograron captar o interese dalgúns matemáticos, o que fixo que a teoría de grafos se convertese nunha área rica en resultados teóricos de gran variedade e profundidade.

Na figura 1.1 podemos ver un mapa de Königsberg co seu río Pregel que flúe a través da cidade. Pódese ver que o río rodea a unha illa e, na parte dereita do mapa, se separa en dúas ramas. Para facilitar que os habitantes de Königsberg poidan viaxar facilmente dunha parte da cidade a outra, construíronse sete pontes cruzando o río. Dise que a xente da cidade adoitaba entreterse tratando de deseñar unha ruta a través da cidade de forma que, saíndo dun punto calquera, se cruce cada unha das sete pontes unha única vez e se retorne ao punto de saída. Estes intentos foron sempre errados, é por iso que moitos comezaban a pensar que non era posible deseñar unha ruta con tales características, o cal foi probado por Leonhard Euler (1736) tras tratar o problema dende un punto de vista

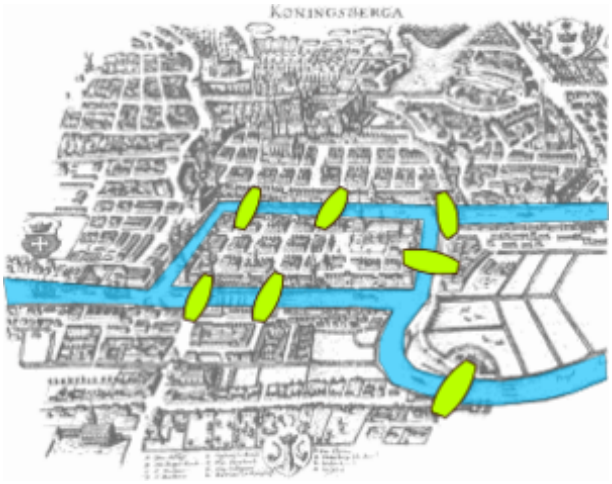

Figura 1.1: Mapa de Königsberg.

matemático representándoo mediante o grafo da figura 1.2 onde cada parte da cidade está representada por un nodo e as pontes están representadas polas aristas. Euler escribiu un artigo no que non só trataba este problema concreto, senón que probou que calquera problema deste tipo ten solución se e só se o número de aristas que inciden en cada vértice é par. Historicamente, atribúese a dito artigo a orixe da teoría de grafos.

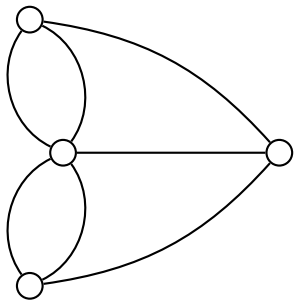

Figura 1.2: Grafo pontes de Königsberg.

En 1771 o matemático francés Alexandre-Theophile Vandermonde publicou un artigo coñecido como 'ruta do cabalo nun taboleiro de xadrez'. Como sabemos, o cabalo móvese dúas casas paralelas a un lado do taboleiro e logo outra perpendicular; o problema consiste en atopar unha ruta de forma que o cabalo visite todas as casas do taboleiro unha única vez e retorne á casa na que comezou. Este problema é un caso especial dos que buscan unha ruta que pase por cada vértice unha única vez. No caso particular da ruta do cabalo o grafo correspondente ten 64 vértices, un por cada casa do taboleiro, e dous vértices están unidos por unha arista sempre que o cabalo poida pasar dun ao outro mediante un movemento

legal.

Podemos observar que, ao contrario do problema das pontes de Königsberg que busca pasar por todas as aristas, agora o que se require é pasar por todos os vértices, o que se achega máis ao problema do viaxante de comercio. A diferenza das cadeas de Euler, para o tipo de grafos deste problema non temos condicións suficientes e necesarias para a existencia.

En 1832, en Alemaña publicouse "O viaxante de comercio: como debe ser e que debe facer para conseguir comisións e triunfar no seu negocio. Por un viaxante de comercio veterano" (Ein alter Commis-Voyageur (1832)), un libro que, aínda que se centra principalmente noutros aspectos da profesión, no último capítulo define o problema do viaxante de comercio aínda que non dá unha formulación matemática. Esta adoita considerarse como a primeira referencia bibliográfica ao TSP. Sen dúbida é un documento de grande importancia pois o problema é definido por un vendedor case un século antes de que a comunidade científica comece a estudar este tipo de problemas. Ademais, o libro inclúe cinco rutas que percorren rexións de Alemaña e Suiza, unha das cales é efectivamente a solución óptima do TSP.

A primeira discusión xeral do problema do viaxante de comercio foi dada por Thomas Penyngton Kirkman. Un dos intereses matemáticos de Kirkman, era o estudo de poliedros. A superficie dun poliedro contén as liñas de intersección das súas caras e os puntos de intersección destas liñas, polo tanto, a cada poliedro correspóndelle un grafo cuxos vértices e aristas se corresponden con estes puntos e liñas. Unha forma de describir estes poliedros é mediante o seu grafo así na figura 1.3 podemos ver, por exemplo o grafo dun cubo e o dunha pirámide pentagonal.

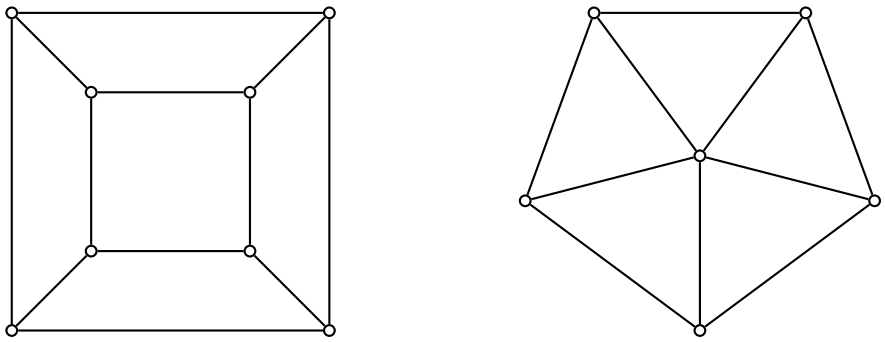

Figura 1.3: Grafos correspondentes a un cubo e a unha pirámide pentagonal.

Nun artigo enviado á Royal Society en 1855, Kirkman formulaba a seguinte pregunta: "Dado o grafo dun poliedro, pódese atopar sempre un circuíto que pase por cada vértice

unha e só unha vez?" Kirkman probou que, se un poliedro ten un número impar de vértices e cada cara ten un número par de lados, non existe un circuíto que pase por todos os vértices.

Ao mesmo tempo que Kirkman escribía sobre isto, William Rowan Hamilton creaba un quebracabezas chamado Xogo Icosian cun taboleiro como o da figura 1.4 onde o obxectivo era atopar camiños e circuítos do grafo do dodecaedro cunhas determinadas condicións. En particular, o primeiro problema era atopar un circuíto que pasase unha e só unha vez por cada vértice do grafo.

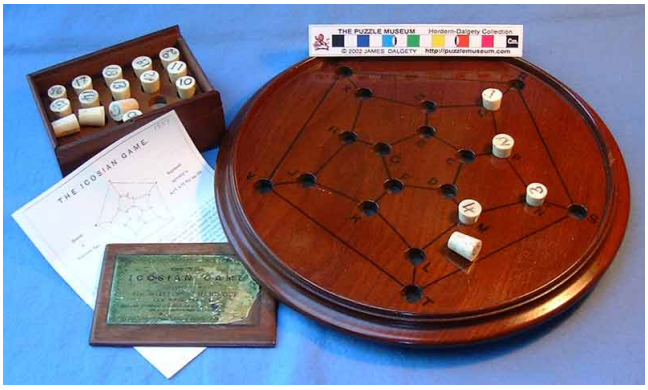

Figura 1.4: Xogo Icosian.

Existe outra versión do xogo de Hamilton coñecido como 'O dodecaedro do viaxante' coa base dun dodecaedro en lugar do seu grafo. Neste xogo, os vértices representaban vinte cidades importantes: Bruselas, Canton, Delhi,... rematando con Zanzibar. Cada vértice estaba marcado cunha chincheta e cun fío uníanse as chinchetas para indicar un camiño ou un circuíto. Un circuíto completo pasando por todas as chinchetas unha única vez, recibía o nome de 'viaxe arredor do mundo'.

Como podemos ver, a mesma idea matemática foi plantexada por Kirkman e Hamilton independentemente pero ao mesmo tempo e, aínda que na actualidade sexan máis coñecidos os resultados de Hamilton, podemos dicir que ambos estudaron no século XIX por primeira vez o problema do viaxante de comercio.

As orixes do TSP non están claras, non se sabe quen introduciu o termo TSP no círculo matemático. Porén parece estar claro que Merrill Flood foi quen máis o publicitou e o responsable de que hoxe se coñeza. Flood afirma que escoitou falar del a A. W. Tucker en 1937 quen, á súa vez, di que non pode confirmar nen desmentir que escoitou a Hassel Whitney falar sobre o tema na Universidade de Princenton, pero que, en caso de ser certo, tería sido entre os anos 1931 e 1932. En 1948, John Williams insistiu para que Flood popularizase o TSP na Corporación RAND motivado polo propósito de crear retos

intelectuais para modelos fóra da teoría de grafos. A reputación e autoridade de RAND fixo eco da mensaxe de Flood popularizando o TSP entre os círculos científicos.

Na década dos 50 e os 60 o problema comezou a ser máis popular tamén fóra do mundo científico grazas, entre outras cousas, á campaña publicitaria que Procter and Gamble lanzou en 1962 baseada nos personaxes da serie 'Car 54, Where are you?'. Era un concurso no cal os participantes debían pensar acerca da ruta máis rápida que permitise aos protagonistas Toody e Muldoon viaxar arredor dos Estados Unidos visitando as cidades marcadas na figura 1.5.

Figura 1.5: Cartel do concurso Car 54.

Dantzig et al. (1954) publicaron o artigo "Solution of a large-scale traveling-salesman problem" onde resolven o TSP para 49 cidades, unha por cada estado de EEUU e Washington. En liña con este artigo, moitos autores comezaron a desenvolver algoritmos que fosen aplicables a problemas cun número cada vez máis grande de cidades. Resumimos no cadro 1.1 algúns dos avances máis destacados ata a actualidade.

Ano	Autores	Tamaño
1954	G. Dantzig, R. Fulkerson, S. Johnson	49 cidades
1971	M. Held, R. M. Karp	64 cidades
1975	P. M. Camerini, L. Fralta, F. Maffioli	67 cidades
1977	M. Grötschel	120 cidades
1980	H. Crowder, M. W. Padberg	318 cidades
1987	M. Padberg, G. Rinaldi	532 cidades
1987	M. Grötschel, O. Holland	666 cidades
1987	M. Padberg, G. Rinaldi	2 392 cidades
1994	D. Applegate, R. Bixby, V. Chvátal, W. Cook	7 397 cidades
1998	D. Applegate, R. Bixby, V. Chvátal, W. Cook	13 509 cidades
2001	D. Applegate, R. Bixby, V. Chvátal, W. Cook	15 112 cidades
2004	D. Applegate, R. Bixby, V. Chvátal, W. Cook, K. Helsgaun	24 978 cidades
2006	D. Applegate, R. Bixby, V. Chvátal, W. Cook, D. Espinoza, M. Goycoolea, K. Helsgaun	85 900 cidades

Cadro 1.1: Avances na resolución do TSP.

Na actualidade están abertos numerosos retos referentes ao TSP. Un exemplo é o reto *TSP da Mona Lisa*, un TSP que consta de 100.000 nodos distribuídos no espazo de tal forma que, se se unen mediante unha liña continua dan lugar á famosa obra de Leonardo da Vinci como podemos observar na figura 1.6. Este problema foi enunciado en 2009 por Robert Bosch e atopar unha solución suporía unha nova marca na resolución do TSP.

Figura 1.6: TSP da Mona Lisa.

1.2. Definición do problema

Dado un conxunto de cidades e os custos do desprazamento (ou distancia) entre cada par, o TSP consiste en atopar a mellor ruta que as visite todas e volva á de inicio que minimice os custos (ou distancia) da viaxe.

O TSP pode ser descrito segundo a teoría de grafos da seguinte maneira: Sexa $G = (V, A)$ un grafo completo onde $V = \{1, ..., n\}$ é o conxunto de nodos e A o conxunto de aristas. Consideramos a cidade 1 como cidade de orixe e á que debe volver. A cada arista (i, j) asóciaselle un valor non negativo c_{ij} que representa o custo de viaxar do vértice i ao j. O uso de aristas (i, i) non está permitido, polo que se asigna $c_{ii} = \infty$ para todo $i \in V$. Se G é un grafo dirixido , a matriz de custos C é asimétrica, mentres que, se $c_{ij} = c_{ji}$ para todo $(i, j) \in A$, C será simétrica e neste caso o conxunto A substitúese por un conxunto E de aristas non dirixidas (i, j) tales que $i < j$.

O obxectivo do problema do viaxante de comercio é atopar unha ruta que, comezando e rematando nunha mesma cidade (neste caso a cidade 1), pase unha única vez por cada unha das cidades restantes, é dicir, un circuíto hamiltoniano, e minimice a distancia total percorrida.

1.3. Formulación

Existen múltiples formulacións distintas do TSP. Neste traballo presentaremos a máis popular, expresando o problema como un caso de programación linear enteira. Sexa

$$x_{ij} = \begin{cases} 1 & \text{, se o comerciante vai directamente de } i \text{ a } j \\ 0 & \text{, noutro caso} \end{cases}$$

e c_{ij} a distancia de i a j. Entón o TSP pode formularse como

$$\text{minimizar} \qquad \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{1.1}$$

$$\text{suxeito a} \qquad \sum_{j=1}^{n} x_{ij} = 1 \qquad i = 1, ..., n \tag{1.2}$$

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad j = 1, ..., n. \tag{1.3}$$

(1.2) e (1.3) garanten que unha única cidade sexa visitada directamente despois de calquera cidade e que a cada unha se chegue directamente dende só unha cidade. Estas restricións son necesarias, pero non suficientes para a resolución do TSP pois permiten subcircuítos. Por exemplo, se consideramos n=4 e as rutas segundo a figura 1.7

Figura 1.7: Subcircuítos no TSP.

$x_{12} = x_{21} = x_{34} = x_{43} = 1$ polo que cumpre todas as condicións expostas anteriormente pero presenta subcircuítos. Debemos entón engadir algunha restrición máis que evite os subcircuítos. Existen numerosas formas de facelo, veremos a continuación a que fai uso da programación linear enteira.

Sexa S un subconxunto non baleiro de $V = \{1, ..., n\}$ con $S \neq V$ e $|S|$ o número de elementos de S, a restrición

$$\sum_{i \in S} \sum_{j \in S} x_{ij} \leq |S| - 1 \tag{1.4}$$

ou de forma equivalente,

$$\sum_{i \in S} \sum_{j \notin S} x_{ij} \geq 1 \tag{1.5}$$

indica que para todo subconxunto de nodos debe haber, polo menos, unha arista que "saia" do conxunto. O número de operacións necesarias para comprobar as restricións (1.4) e (1.5) aumenta exponencialmente ante o incremento no número de nodos polo tanto, non son prácticas para a obtención de resultados. Miller et al. (1960) deron unha solución a este problema introducindo, en lugar das 2^n restricións requiridas para o anterior procedemento, $O(n^2)$ novas restricións e n novas variables: u_i, $\forall i \in V$, que representa o lugar do circuíto no que está o nodo i. Para o primeiro nodo $u_1 = 1$ e para o resto de vértices estas variables toman un valor entre 2 e n. Para construír o problema de programación linear deberemos engadir a (1.2) e (1.3) a seguinte restrición,

$$u_i - u_j + n x_{ij} \leq n - 1 \qquad\qquad i, j = 2, ..., n. \tag{1.6}$$

Estes autores usaron este método tamén para a extensión do TSP na que o viaxante de comercio visita as cidades en diferentes subcircuítos, todos eles comezando e rematando na primeira cidade. Logo (1.6) pode ser escrita como

$$u_i - u_j + px_{ij} \leq p - 1 \qquad i, j = 2, ..., n \tag{1.7}$$

e, dado que a primeira cidade pode ser visitada máis dunha vez, (1.2) e (1.3) deben modificarse tamén eliminando as restricións correspondentes a $\sum_{j=1}^{n} x_{1j}$ e $\sum_{i=1}^{n} x_{i1}$.

1.4. Métodos de resolución

1.4.1. Algoritmos para a solución exacta

Existen dous grupos de algoritmos para obter solucións exactas para un TSP. Un deles resolve relaxacións da formulación de programación linear para o TSP e usa métodos como ramificación e acotación, planos de corte, ou ramificación e corte. E o outro utiliza programación dinámica. Para ambos grupos a característica principal é que garanten unha solución óptima aínda que o tempo de execución e o espazo necesario non sexan óptimos.

Método de ramificación e acotación

Como xa vimos, o TSP pódese formular como un problema de programación linear enteira, é dicir, un problema de programación linear ao que se lle impoñen unhas restricións adicionais de integralidade. Para resolvelo como un problema de programación linear bastará con tomar o problema de programación linear sen estas restricións e resolvelo como tal. Entón a solución obtida non é unha solución do problema inicial senón unha acotación, así, por exemplo, se é un problema de minimización, será unha cota inferior da solución.

A idea dos algoritmos de ramificación e acotación, tamén coñecidos como *branch and bound*, consiste en, apoiándose nas observacións anteriores, subdividir sucesivamente a rexión factible do problema de partida e resolver as versións relaxadas destes subproblemas o cal permitirá ir mellorando sucesivamente as cotas inferiores e superiores ata que se atope unha solución óptima do problema enteiro.

Método de planos de corte

O método de planos de corte é un proceso iterativo cuxa idea é resolver o problema de programación linear sen as restricións de integralidade; en caso de que a solución atopada sexa enteira trátase da solución do problema; se a solución atopada non é enteira, engádese unha nova restrición que corte á solución non enteira pero non corte ningún outro punto da rexión factible do problema e vólvese resolver o problema con esta nova restrición. Repítese este proceso ata atopar unha solución que sexa enteira.

Programación dinámica

A programación dinámica trata de dividir o problema inicial en sub-problemas de forma recursiva, resolver cada un destes problemas unha única vez e almacenar as súas solucións de forma que, se o mesmo sub-problema volve aparecer, se use a solución xa obtida para o mesmo en lugar de volver a calculala.

1.4.2. Algoritmos para unha solución non-exacta

Este tipo de algoritmos non nos ofrecen unha solución exacta, pero os tempos de resolución son moito menores que os dos algoritmos vistos na sección 1.4.1. Este tipo de algoritmos á súa vez poden dividirse en dous grupos:

- Algoritmos de aproximación que presentan un factor de aproximación da solución atopada.
- Algoritmos heurísticos que tan só aseguran unha solución factible.

A continuación exporemos brevemente algúns dos algoritmos pertencentes a este tipo. Como exemplo de algoritmo de aproximación explicaremos o algoritmo de Christofides como algoritmo de aproximación e, como algoritmos heurísticos, o algoritmo do veciño máis próximo, os heurísticos de inserción e o algoritmo heurístico voraz.

Algoritmo de Christofides

Foi definido por Christofides en 1976 e é o mellor algoritmo aproximado que se coñece na actualidade para o TSP. É un algoritmo 1.5-aproximado o que garante que a solución proporcionada por este algoritmo dista da exacta como máximo un 50 %.

O obxectivo do algoritmo de Christofides de complexidade $O(n^3)$ é atopar unha solución para os casos do TSP nos que os custos c_{ij} satisfagan a desigualdade triangular seguindo os seguintes pasos:

1. Atopar unha árbore de expansión mínima T.
2. Atopar E, o emparellamento de mínimo custo entre os nodos de grado impar de T. Definir $G' = (V, A')$, onde A' está formado polas aristas de T e E.
3. Atopar unha cadea euleriana en G e percorrela para obter o circuíto hamiltoniano solución do problema de partida.

Algoritmo do veciño máis próximo

Trátase dun algoritmo rápido pois ten complexidade $O(n^2)$ pero pode funcionar mal debido á súa "miopía" xa que, cando chega a un nodo, simplemente busca o máis próximo sen ter en conta os efectos de facelo e as distancias que quedarán ao percorrer os outros a continuación. Procede da seguinte forma:

1: Seleccionar unha cidade aleatoria,
2: **while** queden cidades por visitar **do**
3: atopar a cidade máis próxima e desprazarse cara ela
4: **end while**
5: Retornar á cidade inicial.

Algoritmos heurísticos de inserción

Existe unha gran cantidade de variantes deste tipo de algoritmos con complexidade $O(n^2)$. A idea é comezar cun circuíto que una as cidades dun subconxunto das totais e logo ir engadindo o resto a través dalgún método heurístico. Habitualmente este circuíto inicial é un triángulo e, para engadir as demais cidades, procédese da seguinte forma:

1: Seleccionar unha cidade que non estea presente no circuíto e cuxa distancia a algunha das xa presentes sexa mínima,
2: **while** queden cidades fóra do circuíto **do**
3: atopar unha arista no circuíto tal que o custo de enxerir a cidade elixida entre as dúas cidades unidas por dita arista sexa mínima
4: **end while**

Algoritmo heurístico voraz

Este algoritmo de complexidade $O(n^2 log_2(n))$ consiste en construír un circuíto seleccionando repetidamente a arista máis curta e engadíndoa, sempre e cando non se creen subcircuítos ou o grado dalgún nodo pase a ser maior que 2. Procede da seguinte forma:

1: Ordenar as aristas segundo a súa lonxitude,
2: **while** haxa cidades fóra do circuíto **do**
3: seleccionar a arista máis curta e engadila ao circuíto sempre que non se incumpra ningunha das restricións
4: **end while**

Capítulo 2

Problemas de rutas de vehículos

Neste capítulo exporemos os problemas de rutas de vehículos (VRP) e as súas variacións centrándonos especialmente na súa versión máis básica, o VRP con restricións de capacidade (CVRP). Para a súa elaboración utilizamos principalmente os libros de Lawler et al. (1986) e de Toth e Vigo (2002).

2.1. Introdución

O *problema de rutas de vehículos* (VRP polas súas siglas en inglés *Vehicle Routing Problem*) é un dos problemas de optimización combinatoria máis estudados. Foi proposto por primeira vez por Dantzig e Ramser en 1959 en *The truck dispatching problem*, onde estudaban a aplicación na distribución de carburante para gasolineiras e propoñían unha solución baseada nunha formulación de programación linear que daba lugar a unha solución cuasi-óptima. Dende entón, publicáronse centos de artigos sobre a solución exacta e aproximacións da mesma de moitas das variantes deste problema. O grande interese no VRP, igual que no TSP, está relacionado coas numerosas aplicacións en problemas do mundo real así como a súa considerable dificultade.

Consideramos o problema de distribución no cal uns vehículos con base nun almacén deben visitar durante un período de tempo dado, uns clientes que se atopan xeograficamente dispersos co fin de satisfacer unhas demandas coñecidas. Este problema aparece en numerosas situacións prácticas que involucran a distribución de produtos e é o que se coñece como VRP. Naquelas situacións non necesariamente relacionadas coa entrega de produtos como, por exemplo, a recollida do diñeiro das máquinas expendedoras, as visitas a pacientes dun médico, etc; os conceptos 'vehículo' e 'demandas dos clientes' toman diferentes significados, algúns dos cales poden incluso non ter unha natureza física. En vista da enorme variedade de situacións prácticas que se poden dar, debemos centrarnos en estudar

o problema VRP con restricións de capacidade (CVRP) que é o modelo máis básico e o máis estudado, estando no núcleo de todos os demais.

2.2. Problema de rutas de vehículos con restricións de capacidade

No CVRP todos os clientes e súas correspondentes demandas se coñecen con anterioridade e non poden ser divididas, é dicir, a totalidade da demanda dun cliente debe ser abastecida por un único vehículo. Os vehículos son idénticos, e cada un ten a súa base nun único almacén e só existen restricións de capacidade.

O CVRP consiste en describir unha ruta que comece e remate no almacén para cada un dos diferentes vehículos, de forma que todos os clientes sexan visitados e as súas demandas abastecidas; cada vehículo faga unha única ruta e o custo total da viaxe sexa mínimo. A figura 2.1 mostra a forma dunha solución do VRP.

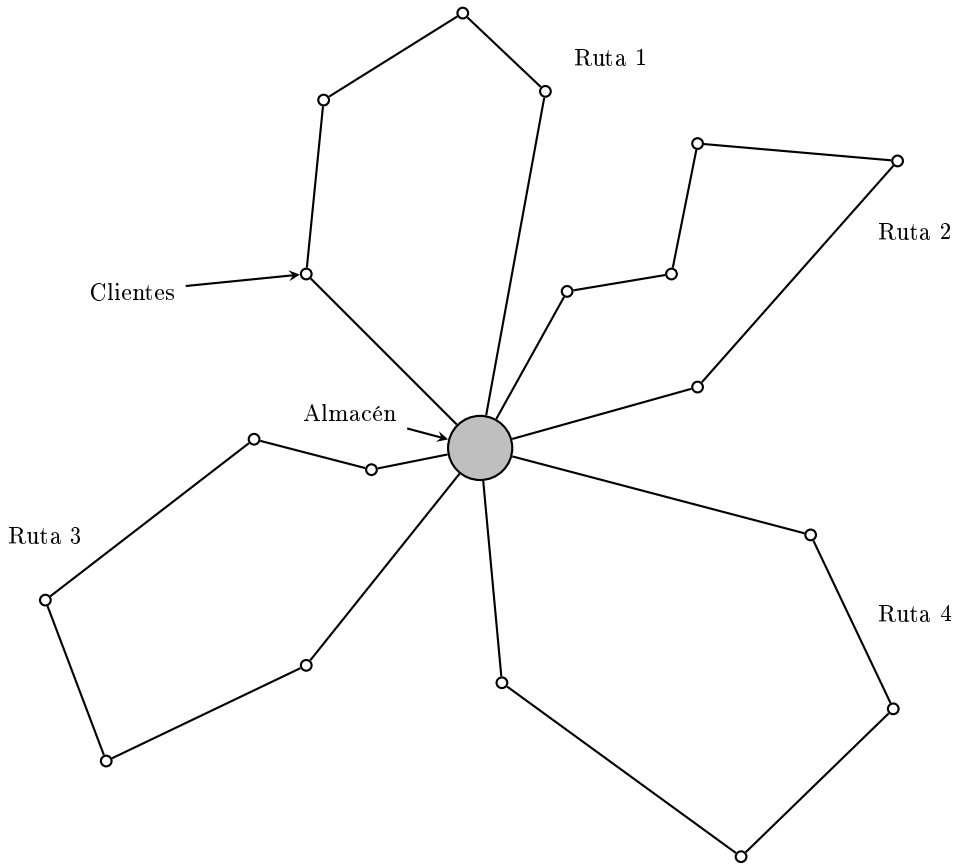

Figura 2.1: Forma da solución do VRP.

Notación

Deixemos clara algunha notación necesaria para describir o modelo do problema:

Conxuntos

I O conxunto de clientes;

K O conxunto de vehículos.

Índices

$i = 2, ..., n$ Os índices dos clientes;

$i = 1$ O almacén;

$k = 1, ..., m$ Os índices dos vehículos.

Parámetros

$q_i \geq 0$ A demanda do cliente i (asumimos $q_1 = 0$);

$c_{ij} \geq 0$ O custo do traxecto entre os clientes i e j;

$Q_k \geq 0$ A capacidade do vehículo k.

Asuncións

Todos os clientes e vehículos están ordenados de tal forma que $q_2 \geq q_3 \geq ... \geq q_n$ e $Q_1 \geq Q_2 \geq ... \geq Q_m$ respectivamente.

Dado que no CVRP os vehículos se consideran idénticos em canto á súa capacidade, $Q_k = Q$, $\forall k \in K$. Para asegurar que o problema teña unha solución factible é necesario que $q_i \leq Q$, $\forall i \in I$.

Consideraremos durante o resto do traballo esta notación enunciada en Lawer et al. (1986) aínda que outros autores como Toth et al. (2002) optan por considerar o problema como un grafo completo e fortemente conexo, $G = (V, A)$, con $V = \{0, ..., n\}$ o conxunto de vértices e A o conxunto de aristas.

2.2.1. Formulacións

Formulación de Fisher & Jaikumar

Fisher e Jaikumar (1981) formularon o TSP como segue.

Sexa

$$x_{ijk} = \begin{cases} 1, & \text{se o vehículo } k \text{ visita ao cliente } j \text{ inmediatamente} \\ & \text{despois do cliente } i, \\ 0, & \text{noutro caso.} \end{cases} \tag{2.1}$$

$$y_{ik} = \begin{cases} 1, & \text{se o vehículo } k \text{ visita ao cliente } i, \\ 0, & \text{noutro caso.} \end{cases} \tag{2.2}$$

Logo o VRP básico consistirá en minimizar

$$\sum_{i,j} c_{ij} \sum_{k} x_{ijk} \tag{2.3}$$

suxeito a

$$\sum_{k} y_{ik} = \begin{cases} 1, & i = 2, ..., n, \\ m, & i = 1, \end{cases} \tag{2.4}$$

$$\sum_{i} q_i y_{ik} \leq Q_k, \qquad k = 1, ..., m, \tag{2.5}$$

$$\sum_{j} x_{ijk} = \sum_{j} x_{jik} = y_{ik}, \qquad i = 1, ..., n, \qquad k = 1, ..., m, \tag{2.6}$$

$$\sum_{i,j \in S} x_{ijk} \leq |S| - 1, \qquad \text{para todo } S \subseteq \{2, ..., n\}, \qquad k = 1, ..., m, \tag{2.7}$$

$$y_{ik} \in \{0, 1\}, \qquad i = 1, ..., n, \qquad k = 1, ..., m, \tag{2.8}$$

$$x_{ijk} \in \{0, 1\}, \qquad i, j = 1, ..., n, \qquad k = 1, ..., m. \tag{2.9}$$

Onde (2.4) obriga a que cada nodo correspondente a un cliente sexa visitado por un único vehículo e o nodo correspondente ao almacén sexa visitado por todos eles. (2.5) garante que non se exceda a capacidade de cada vehículo. (2.6) asegura que cada vehículo que chega a un nodo correspondente cun cliente saia tamén de dito nodo. (2.7) evita que se formen subcircuítos e (2.8) e (2.9) esixen que y_{ik} e x_{ijk} sexan variables binarias.

Formulación de Christofides, Mingozi & Toth

Vexamos agora a formulación para o VRP proposta por Christofides et al. (1981), unha formulación de programación dinámica do VRP que permite que as capacidades dos vehículos sexan diferentes. Sexa $N = \{2, ..., n\}$ o conxunto de clientes. Para calquera $T \subseteq N$, definimos $f(k, T)$ como o custo mínimo de abastecer os clientes en T usando só os vehículos $1, ..., k$, $v(T)$ como o mínimo custo da solución do TSP definido polo almacén e os clientes en T e $q(T) = \sum_{i \in T} q_i$. Iniciamos o proceso para $k = 1$ con $f(1, T) = v(T)$ e definimos para $k \geq 2$

$$f(k, T) = \min_{S \subset T} \{f(k - 1, T \setminus S) + v(S)\} \text{ con } v(\emptyset) = 0 \tag{2.10}$$

suxeito a

$$q(T) - \sum_{h=1}^{k-1} Q_h \leq q(S) \leq Q_k. \tag{2.11}$$

Os conxuntos $T \subseteq N$ considerados deberán satisfacer

$$q(N) - \sum_{h=k+1}^{m} Q_h \leq q(T) \leq \sum_{h=1}^{k} Q_h. \tag{2.12}$$

As restricións sobre S e T evitan que se calculen as funcións f e v para conxuntos que conducen a cargas infactibles. O lado dereito de 2.11 é a restrición de capacidade do vehículo k, mentres que o lado esquerdo é a restrición das capacidades dos primeiros $k-1$ vehículos.

2.3. Outros problemas de rutas de vehículos

O CVRP tal como foi enunciado na sección 2.2 ignora unha gran variedade de restricións e extensións que están presentes na maioría dos problemas do mundo real como por exemplo:

- Dúas rutas diferentes poden compartir vehículo sempre que o tempo total empregado nesas rutas sexa menor que un determinado tempo T correspondente co tempo de funcionamento de dito vehículo. Poder engadir esta restrición implica coñecer os tempos de viaxe entre cada par de clientes (t_{ij}).
- Cada cliente deberá ser visitado unicamente durante un horario determinado coñecido como Split Delivery VRP ou SDVRP.
- Un mesmo problema pode comprender entregas e devolucións dos seus clientes. Neste caso poderá ser posible mesturar entregas e recollidas ou pode ser necesario que un vehículo remate primeiro todas as entregas antes de comezar coas recollidas.
- Os vehículos (en concreto os seus condutores) terán un horario de traballo, logo dito vehículo só pode operar nese intervalo de tempo.
- Aparte dos xa mencionados tempos de viaxe (t_{ij}) débense ter en conta outros tempos como os de descarga (ou carga en caso de recollida) no lugar de entrega, os tempos de carga no propio almacén, os tempos de espera, etc.

Porén, todas estas consideracións non cambian a esencia do VRP e poden ser incorporadas en moitos métodos heurísticos para a resolución do problema. Pola contra, existen algunhas outras consideracións prácticas que aparecen con frecuencia que non encaixan realmente na estrutura do CVRP como as que se exporán a continuación:

Varios almacéns

Existen compañías que teñen máis dun almacén, habitualmente estes almacéns funcionan autonomamente, é dicir, cada un ten a súa propia flota de vehículos e unha determinada área xeográfica á que debe abastecer. Pero noutros casos os almacéns están relacionados e un vehículo poderá, por exemplo, rematar a súa ruta nun almacén distinto ao de inicio ou incluso parar a reencherse con nova mercancía nun almacén intermedio. Neste caso cada almacén non pode ser considerado independentemente.

Nivel de servizo aos clientes

O intervalo de tempo no cal a demanda de cada un dos clientes debe ser abastecida é un dos parámetros máis importantes nun VRP e é unha medida do nivel de servizo. Como o pedido dos clientes é un proceso dinámico e non periódico, as planificacións das rutas de vehículos para un período determinado serán unha aproximación ou unha orde arbitrariamente imposta. Algunhas das aproximacións que podemos considerar son as seguintes:

- *Período típico*: Considera o caso cando os clientes están fixados e se asume que as demandas serán semellantes nun período dado. Neste caso un cliente que se espera que pida mercancía cada t días deberá ser visitado T/t veces nun intervalo de T días e as visitas deben estar separadas $t \pm \varepsilon$ días para un ε pequeno. Neste caso resolverase o TSP para o período T e farase pública a ruta decidida, así os clientes saben cando chegará a súa mercancía.

- *Tempo limitado*: Unha forma de traballar bastante frecuente é establecer unha data límite para os pedidos, é dicir, os pedidos feitos durante T días serán entregados nos T días seguintes. Porén, nun sistema así pode que un pedido recibido durante o período actual sexa ignorado ata o seguinte, aínda que puidese ser atendido no actual. Nestes problemas poden xurdir situacións de inviabilidade en determinados períodos que habitualmente se resolverán mediante a contratación de máis vehículos.

- *Prioridades dos clientes*: Unha alternativa usada habitualmente para definir os períodos de reparto é asignando unha prioridade a cada cliente de acordo co intervalo de tempo restante ata a data na que o cliente deba ser visitado (por exemplo, T días despois de recibir o pedido). A menor intervalo de tempo, maior prioridade. Logo o VRP terá en conta os custos de ruta e as prioridades dos clientes, intentando que o servizo teña un atraso máximo de T días.

Múltiples mercancías

Nalgúns VRP, os vehículos posúen compartimentos de forma que diferentes mercancías irán en diferentes compartimentos. Cada cliente pedirá certa cantidade de cada tipo de produtos. Este tipo de problemas aparecen na distribución de gasolinas, comidas que precisan ou non refrixeración, etc.

Obxectivos diferentes

Nalgunhas ocasións non será posible resolver o problema tal e como está dado. Na práctica esta imposibilidade resolverase contratando máis vehículos e/ou pospoñendo o servizo a algúns clientes. Nestes casos o obxectivo nun VRP será unha combinación linear dos seguintes factores:

- minimizar o número de vehículos extra contratados ,
- minimizar o número (ou suma de pesos) de clientes que non son servidos no período actual,
- minimizar a distancia (ou tempo) de viaxe total.

Tendo en conta algunhas das características enunciadas podemos atoparnos con diferentes variacións do VPR, algunhas das cales explicaremos brevemente a continuación como o VRP con recollidas (VRPB polas súas siglas en inglés, *Vehicle Routing Problem with Backhauls*), o VRP con recollidas e entregas (VRPPD polas súas siglas en inglés *Vehicle Routing Problem with Pick-ups and Deliveries*), o CVRP con restricións de distancia para cada vehículo (DCVRP polas súas siglas en inglés *Distance-constrained Capacited Vehicle Routing Problem*), o VRP con ventás de tempos (VRPTW polas súas siglas en inglés, *Vehicle Routing Problem with Time Windows*), o VRP con entregas divididas (SDVRP polas súas siglas en inglés, *Split Delivery Vehicle Routing Problem*), o VRP con vehículos con compartimentos (VRPC polas súas siglas en inglés, *Vehicle Routing Problem with Compartments*) ou o VRP estocástico (SVRP polas súas siglas en inglés, *Stochastic Vehicle Routing Problem*).

2.3.1. VRP con recollidas

O VRPB é unha variante do VRP que considera puntos que deben ser abastecidos e puntos nos que se recolle mercancía. A suposición fundamental é que as demandas deben ser abastecidas antes de comezar a visitar os clientes nos que se deben recoller produtos. Isto débese a que no momento no que se carga o camión no almacén se fará de forma estratéxica

tendo en conta a orde na que se visitarán os clientes e reordenar esta mercancía ao longo da ruta non é economicamente factible. As cantidades que se deberán tanto deixar como recoller coñécense de antemán e a flota de vehículos considérase homoxénea. Polo tanto unha solución factible para o problema consiste nun conxunto de rutas onde en cada unha se completan antes as demandas que as recollidas sen sobrepasar a capacidade do vehículo asociado a dita ruta. O obxectivo é atopar un deses conxuntos de rutas que minimice a distancia percorrida.

2.3.2. VRP con recollidas e entregas

Na versión máis básica do VRPPD a cada cliente i asócianselle dúas cantidades, d_i e p_i, que representan demanda de bens a ser entregados e recollidos respectivamente no cliente i. Para cada cliente i, O_i denota o nodo do cal proceden os produtos que se entregan no mesmo e D_i o nodo ao que van os que se recollen no nodo i. Asúmese que en cada nodo as entregas se fan antes que as recollidas, así, a carga dun vehículo antes de chegar ao lugar do cliente i está definida pola carga inicial, menos os produtos xa entregados, máis os produtos recollidos nos nodos anteriores.

O VRPPD consiste en atopar exactamente m rutas simples [1] con custo mínimo e tales que cada ruta visite o almacén; cada cliente sexa visitado por exactamente un vehículo; a carga do vehículo ao longo da ruta sexa non negativa e non superior á capacidade do vehículo; para cada nodo i, os clientes O_i e D_i (cando sexan diferente do almacén) deberán pertencer á mesma ruta que o cliente i e estar antes e despois respectivamente que o cliente i.

Habitualmente a orixe ou o destino das demandas é o mesmo para todo $i \in I$ (o almacén no caso do CVRP e o VRPB), en dito caso non é necesario especificalos.

2.3.3. CVRP con restricións de distancia

Segundo Alvina et al. (2008), dados un conxunto de almacéns e outro de clientes xeograficamente definidos, o obxectivo do DCVRP é atopar a ruta con menor custo tendo en conta restricións de capacidade e tempos. Neste problema os custos están relacionados tanto co prezo da viaxe como co de utilización do vehículo. Logo, igual que nos problemas anteriores cada cliente poderá ser visitado só unha vez, o vehículo deberá volver ao almacén no que iniciou a súa ruta, a ruta de cada vehículo pode pasar por un mesmo depósito exactamente unha vez, a cada cliente asóciaselle unha demanda non negativa e a suma das demandas dos clientes que forman parte dunha mesma ruta non debe exceder a capacidade

[1] Chamaremos *ruta simple* a aquela que percorre un único vehículo.

do vehículo que a percorre e, neste caso, o tempo total consistente en tempo de cargar o vehículo no almacén, tempo de viaxe entre clientes e tempo de descargar a mercancía no lugar do cliente non pode exceder o límite de horas que ese camión pode estar de servizo.

2.3.4. VRP con ventás de tempos

O VRPTW é a extensión do CVRP na cal cada cliente i terá asociado un intervalo $[a_i, b_i]$ durante o cal poderá ser visitado chamado ventá de tempo. Para definir este problema daranse tamén o instante cando o vehículo sae do almacén, o tempo de viaxe t_{ij} para cada arista (i, j), e un tempo de servizo adicional s_i para cada cliente. O servizo de cada cliente debe comezar dentro da súa ventá de tempo, e o vehículo deberá parar no lugar de cada cliente i s_i unidades de tempo. De feito, no caso de que o vehículo chegue ao lugar do cliente i antes de tempo, deberá espera ata o instante de tempo a_i para efectuar o servizo.

Habitualmente, a matriz de custos e a de tempo de viaxe coinciden e a de ventás de tempos é definida asumindo que todos os vehículos saen do depósito no instante de tempo $t = 0$.

O VRPTW consiste en atopar unha solución con exactamente m rutas simples con custo mínimo e tales que cada ruta visite o almacén; cada cliente sexa visitado por unha e só unha ruta; a suma das demandas dos clientes pertencentes á mesma ruta non exceda a capacidade do vehículo e, para cada cliente i, o servizo comece na ventá de tempo $[a_i, b_i]$ e o vehículo pare no seu lugar s_i unidades de tempo.

2.3.5. VRP con entregas divididas

No SDVRP unha flota de vehículos con capacidade idéntica abastece a un conxunto de clientes, pero neste caso, a diferenza dos enunciados ata o momento, cada cliente pode ser visitado por máis dun vehículo. Este tipo de problemas pode estar motivado porque a demanda dun ou varios clientes excede a capacidade dos vehículos aínda que non é imprescindible que se dea esta condición pois probouse que considerar un VRP sen a restrición de que un cliente pode ser visitado por un único vehículo, conduce a aforros importantes, superiores ao 50 % nalgúns casos.

2.3.6. VRP con vehículos con compartimentos

O VRPC é unha extensión do VRP na cal os vehículos contan con compartimentos que permiten transportar diferentes tipos de mercancías no mesmo vehículo sen seren mesturadas. Consecuentemente ademais das restricións de capacidade do propio vehículo, existirán tamén restricións de capacidade dos diferentes compartimentos que poden comportarse

de forma diferente, así nalgunhas aplicacións a separación entre compartimentos pode ser axustada de forma que a capacidade dun compartimento poida variar entre 0 e a capacidade total do vehículo, mentres que noutros casos os compartimentos están fixados e non poden ser modificados.

Esta extensión do VRP é menos estudada que outras e a maioría das publicacións acerca da mesma tratan distribucións de combustible, é dicir, os compartimentos son depósitos que conteñen diferentes tipos de combustibles. En Muyldermans et al. (2010) podemos ver un exemplo de VRPC que usa como solución inicial a dada pola versión paralela do algoritmo de Clarke and Wright que presentaremos na sección 3.2.

2.3.7. VRP estocástico

No SVRP algúns dos parámetros como as demandas dos clientes, os propios clientes, o tempo de servizo ou o tempo de viaxe poden ser aleatorios ou non coñecidos con antelación, en Berhan et al (2014) podemos ver algúns exemplos de problemas deste tipo . Este tipo de problemas abórdanse a través dos VRP clásicos, formulando unha solución planificada a priori e a continuación facendo accións correctivas.

Segundo Aykagan e Erera (2007), a importancia deste tipo de problemas radica no feito de que nos problemas reais de transporte ou de distribución loxística, algúns parámetros do modelo tales como demanda, tempo ou distancia entre nodos son estocásticos por natureza aínda que adoitan ser simplificados e tratados como deterministas.

O estudo do SVRP é unha área de investigación relativamente nova e en rápido crecemento, aínda así, estes problemas están bastante subdesenvolvidos en comparación cos casos deterministas o cal dificulta dar resposta á necesidade dun sistema de orientación en tempo real de vehículos para proporcionar instrucións actualizadas aos condutores.

Dror et al. (1989) estenden o algoritmo de Clarke & Wright de forma inmediata para o SVRP considerando en lugar dos custos reais das rutas, os custos esperados no peor dos casos de proporcionar un servizo de emerxencia a través de entregas individuais.

Capítulo 3

Algoritmos de solución

Neste capítulo facemos unha breve revisión dos procedementos de resolución de problemas de rutas de vehículos por medio de algoritmos heurísticos. Para a súa elaboración utilizamos principalmente os libros de Lawler et al. (1986) e de Toth e Vigo (2002). Ademais presentamos con detalle un destes algoritmos heurísticos: o proposto por Clarke and Wright(1964).

3.1. Introdución

Como xa dixemos anteriormente, o VRP é un problema de tipo NP-duro, o que implica que, para determinados modelos complexos e de gran tamaño, non se desenvolveu ningún algoritmo capaz de atopar unha solución con garantía de optimalidade nun tempo polinomial. Os métodos deterministas tratan de avaliar todas as solucións posibles e quedarse coa de menor custo, non é factible para modelos VRP con moitas cidades.

Os métodos heurísticos dan lugar a solucións cuasi-óptimas en tempos razoables, é dicir, poden atopar solucións de boa calidade cun esforzo computacional accesible. Para o VRP propuxéronse diferentes familias de métodos heurísticos. En termos xerais estes pódense clasificar en *heurísticos clásicos*, maiormente desenvoltos entre 1960 e 1990 que habitualmente se constrúen para un problema determinado, e *metaheurísticos*, estudados nas últimas décadas que son métodos heurísticos que se poden aplicar a toda clase de problemas de optimización, sen ter en conta as súas particularidades. A maioría das construcións estándar e dos procedementos de mellora que se usan na actualidade pertencen á primeira clase. Estes métodos adoitan obter solucións de boa calidade con tempos de computación modestos. Ademais, a maioría deles poden ser estendidos facilmente ás diversas restricións dos contextos da vida real. Pola súa parte, nos métodos metaheurísticos o énfase está en realizar unha exploración do espazo de busca de maneira global e de forma aleatoria, tra-

tando o problema de optimización como se fose unha caixa negra. Habitualmente, este tipo de métodos combinan regras de busca de veciñanzas, estruturas de memoria e recombinación das solucións. A calidade das solucións atopadas por estes métodos é moito maior que a das obtidas por métodos heurísticos clásicos, pero o tempo de computación é tamén maior.

Os métodos heurísticos clásicos para o VRP poden clasificarse en tres categorías:

- Os *heurísticos construtivos* que van engadindo elementos ata completar unha solución de tal forma que, para cada iteración, se engade o elemento coa mellor avaliación mentres se focalizan no custo da solución pero non teñen unha fase de mellora per se.

- Os *heurísticos de dúas fases*, nos cales o problema se descompón nas súas dúas compoñentes naturais: a agrupación de nodos en rutas factibles e a construción dunha ruta real con posible retroalimentación entre dúas etapas. Este tipo de métodos poden á súa vez, dividirse en dúas clases: os que primeiro organizan os nodos en agrupacións factibles e logo trazan unha ruta entre eles e os que o fan ao revés, primeiro trazan rutas entre todos os nodos e a continuación segméntase en rutas factibles.

- Os *métodos de mellora* que intentan mellorar calquera solución factible permitindo a unha secuencia de nodos ou aristas intercambios entre elas ou incluso entre diferentes rutas. A diferenza entre os métodos construtivos e os de mellora non sempre é clara pois a maioría dos algoritmos construtivos incorporan pasos de mellora en varias etapas.

Aínda que o número de métodos e variables existentes é moi amplo, centrarémonos neste traballo no *método heurístico construtivo de Clarke and Wright*. A maioría dos heurísticos desenvoltos para o VRP aplícanse directamente a CVRPs e habitualmente poden ser estendidos aos DCVRPs aínda que non sempre estea explicitado na descrición do algoritmo. As dúas técnicas principais para construír solucións do VRP son unir rutas existentes usando criterios de aforro e asignar nodos ás rutas de vehículos gradualmente usando custos de introdución.

3.2. Algoritmo de aforro de Clarke and Wright

O coñecido como *algoritmo de aforro de Clarke and Wright*, que de aquí en diante denotaremos por CW, é un dos primeiros métodos heurísticos e, sen dúbida, o máis coñecido para VRP. Data de 1964 e o seu propósito inicial era resolver dunha forma efectiva o problema que presentaran cinco anos antes Dantzig e Ramser (1959) sobre o reparto de

carburante que mencionamos no capítulo anterior. Este método aplícase para problemas onde o número de vehículos non está definido é dicir, é unha variable de decisión.

Deberemos calcular o aforro, en termos de distancia, que supón a unión de dous puntos que non pertencen á mesma ruta. Se partimos das rutas $(1, ..., i, 1)$ e $(1, j, ..., 1)$ o aforro que supón unilas nunha mesma ruta da forma ilustrada na figura 3.1 é $s_{ij} = c_{i1} + c_{1j} - c_{ij}$. O algoritmo procede da seguinte maneira:

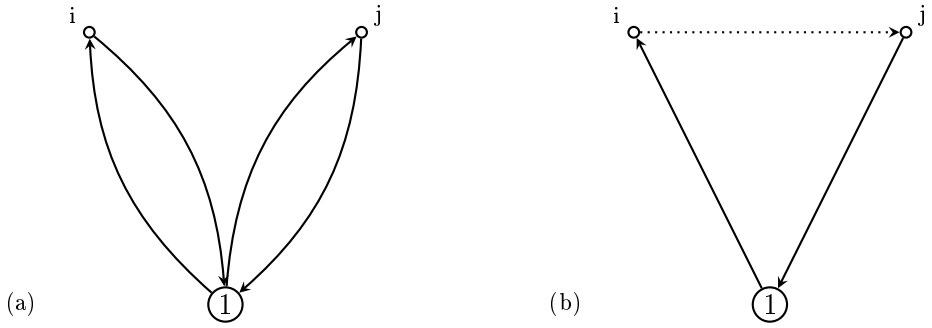

Figura 3.1: Aforro unindo os nodos i e j. (a) Estado inicial. (b) Estado tras a unión.

Paso 1 Comezamos considerando como solución inicial aquela na que cada un dos clientes é visitado nunha ruta que só pasa por el e o almacén.

Paso 2 Para cada par de nodos i, j $i \neq j$ calculamos os aforros s_{ij}.

Paso 3 Ordenamos os aforros calculados no paso anterior en orde descendente e as unións correspondentes con cada aforro.

Versión paralela

Paso 4 Considerando a unión $[i, j]$ do inicio da lista construída no Paso 3 únense dúas rutas separadas usando $[i, j]$ se

(i) os nodos i e j pertencen a rutas diferentes,

(ii) i e j son o primeiro ou último nodo nas rutas ás que pertencen tras saír do almacén ou antes de volver ao mesmo,

(iii) a capacidade máxima do vehículo non se excede.

Paso 5 Repetimos o Paso 4 ata que a lista de aforros remate ou as capacidades dos vehículos non permitan seguir engadindo clientes.

Versión secuencial

Paso 4 Buscar na lista a primeira unión factible que poida ser usada para estender un dos dous extremos da ruta actual.

Paso 5 Eliminar da lista a unión considerada no Paso 4 así como as que conteñan aos clientes que agora son nodos intermedios da lista e as que relacionen clientes que xa pertencen á mesma ruta.

Paso 6 Se a ruta non se pode estender máis, remátase a ruta e búscase a seguinte unión factible para comezar unha nova ruta.

Tanto na versión paralela como na secuencial, é necesario comprobar que a solución obtida en cada paso sexa factible. En caso contrario, poderemos chegar ao final cunha solución para a cal os vehículos dispoñibles non poidan percorrer as rutas formadas. Debemos notar tamén que a ruta inicial formada no Paso 1, na cal cada cliente está nunha ruta individual, non é factible. Porén, si existe a posibilidade de que na solución final apareza algunha ruta cun único cliente.

c_{ij}	1	2	3	4	5	6	7	8	9	10
1	-									
2	12	-								
3	11	8	-							
4	7	5	9	-						
5	10	9	15	7	-					
6	10	12	17	9	3	-				
7	9	14	8	11	17	18	-			
8	8	16	18	12	7	6	16	-		
9	6	17	14	12	15	15	8	11	-	
10	12	22	22	17	18	15	16	11	10	-

Cadro 3.1: Matriz de distancias do VRP.

Exemplo 3.1. Vexamos como aplicar ambas versións do algoritmo de Clarke and Wright ao VRP que ten o depósito no nodo 1 e nove clientes nos nodos 2,..,10. A matriz de

distancias simétrica móstrase no cadro 3.1. A capacidade dos vehículos é $Q = 40$ o vector de demandas dos clientes vén dado por $(q_2, ..., q_{10}) = (10, 15, 18, 17, 3, 5, 9, 4, 6)$. [1]

Paso 1 Creamos nove rutas de forma que o vehículo saia do almacén, visite un único cliente e volva ao mesmo para iniciar o proceso. O custo de abastecer aos clientes segundo estas rutas é $2c_{21} + 2c_{31} + 2c_{41} + 2c_{51} + 2c_{61} + 2c_{71} + 2c_{81} + 2c_{91} + 2c_{101} = 170$.

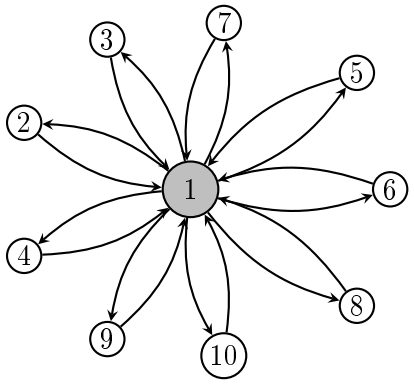

Figura 3.2: Paso 1.

Paso 2: Calculamos todos os aforros. $s_{ij} = s_{ji} = c_{j1} + c_{i1} - c_{ij}$.

Observamos que, dado que a matriz de distancias do problema é simétrica, tamén o será a matriz de aforros.

Paso 3 Ordenamos agora os aforros recollidos no cadro 3.2 en orde descendente:

$s_{65} \geq s_{32} \geq s_{42} \geq s_{52} \geq s_{73} \geq s_{86} \geq s_{85} \geq s_{62} \geq s_{54} \geq s_{43} \geq s_{108} \geq s_{64} \geq s_{109} \geq s_{72} \geq s_{106} \geq s_{97} \geq s_{53} \geq s_{74} \geq s_{107} \geq s_{81} \geq s_{63} \geq s_{105} \geq s_{93} \geq s_{84} \geq s_{98} \geq s_{102} \geq s_{104} \geq s_{75} \geq s_{92} \geq s_{83} \geq s_{103} \geq s_{94} \geq s_{95} \geq s_{76} \geq s_{96} \geq s_{87}$.

E facemos unha lista das unións asociadas a ditos aforros:

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

[1] Os datos deste exemplo foron tomados do seguinte enlace de internet correspondente a material docente da LUT University, unha universidade tecnolóxica finlandesa: http://www.mafy.lut.fi/study/DiscreteOpt/CH6.pdf consultado a 15.06.2019

s_{ij}	1	2	3	4	5	6	7	8	9	10
1	-									
2	0	-								
3	0	15	-							
4	0	14	9	-						
5	0	13	6	10	-					
6	0	10	4	8	17	-				
7	0	7	12	5	2	1	-			
8	0	4	1	3	11	12	1	-		
9	0	1	3	1	1	1	7	3	-	
10	0	2	1	2	4	7	5	9	8	-

Cadro 3.2: Matriz de aforros.

Versión paralela

Paso 4.1 Tomamos a primeira unión da lista, $[6, 5]$, e miramos se cumpre as condicións necesarias,

(i) actualmente o nodo 6 está na ruta $[1, 6, 1]$ e o 5, na $[1, 5, 1]$ polo que, efectivamente son rutas diferentes,

(ii) 5 e 6 son o único nodo distinto do almacén nas súas respectivas rutas polo que se cumpre trivialmente,

(iii) $q_6 + q_5 = 3 + 17 = 20 < Q$.

Dado que cumpre todas as condicións, esta unión de rutas é factible, resultando **[1,6,5,1]**.

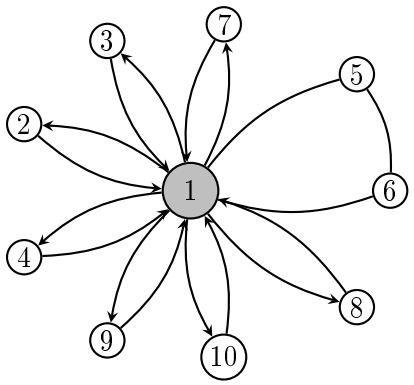

Figura 3.3: Paso 4.1.

Paso 4.2 Tomamos a segunda unión da lista, $[3, 2]$, e miramos se cumpre as condicións necesarias,

(i) actualmente o nodo 3 está na ruta $[1, 3, 1]$ e o 2, na $[1, 2, 1]$ polo que, efectivamente son rutas diferentes,

(ii) 3 e 2 son o único nodo distinto do almacén nas súas respectivas rutas polo que se cumpre trivialmente,

(iii) $q_3 + q_2 = 10 + 15 = 25 < Q$.

Dado que cumpre todas as condicións, esta unión de rutas é factible, resultando **[1,3,2,1]**.

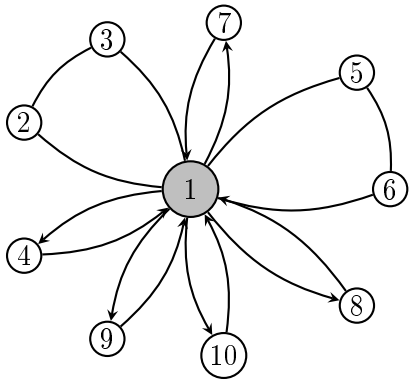

Figura 3.4: Paso 4.2.

Paso 4.3 Tomamos a terceira unión da lista, $[4, 2]$, e miramos se cumpre as condicións necesarias,

(i) actualmente o nodo 4 está na ruta $[1, 4, 1]$ e o 2, na $[1, 3, 2, 1]$ polo que, efectivamente son rutas diferentes,

(ii) 5 é o único nodo distinto do almacén na súa ruta e 2 é o último que se visita antes de volver ao almacén logo cúmprese esta condición,

(iii) $q_4 + q_3 + q_2 = 25 + 18 = 43 \nleq Q$.

Como a unión das rutas existentes mediante $[4, 2]$ non cumpre (iii), non é factible, desbotámola e pasamos á seguinte da lista.

Paso 4.4 Tomamos a cuarta unión da lista, $[5, 2]$, e miramos se cumpre as condicións necesarias,

(i) actualmente o nodo 5 está na ruta $[1, 6, 5, 1]$ e o 2, na $[1, 3, 2, 1]$ polo que, efectivamente son rutas diferentes,

(ii) ambos nodos, 5 e 2, son os últimos antes de volver ao almacén pero dado que estamos ante un problema con matriz de custos simétrica as rutas non están orientadas e podemos considerar o 2 como o primeiro cliente da ruta $[1, 2, 3, 1]$ para poder facer a unión en caso de que sexa posible,

(iii) $q_6 + q_5 + q_2 + q_3 = 25 + 20 = 45 \nleq Q$.

Como a unión das rutas existentes mediante $[5, 2]$ non cumpre (iii), non é factible, desbotámola e pasamos á seguinte da lista.

Paso 4.5 Tomamos a quinta unión da lista, $[7, 3]$, e miramos se cumpre as condicións necesarias,

(i) actualmente o nodo 7 está na ruta $[1, 7, 1]$ e o 3, na $[1, 3, 2, 1]$ polo que, efectivamente son rutas diferentes,

(ii) o nodo 7 é o único cliente da súa ruta e na ruta do nodo 3, este é o primeiro logo cúmprese a condición,

(iii) $q_7 + q_3 + q_2 = 5 + 25 = 30 < Q$.

Dado que cumpre todas as condicións, esta unión de rutas é factible, resultando **[1,7,3,2,1]**.

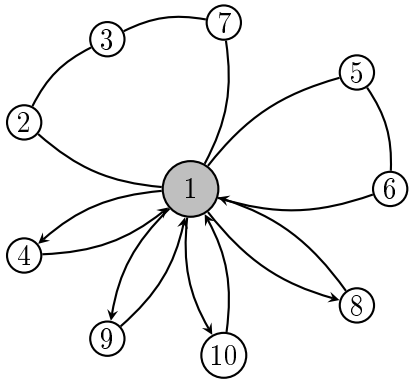

Figura 3.5: Paso 4.5.

Paso 4.6 Tomamos a sexta unión da lista, $[8, 6]$, e miramos se cumpre as condicións necesarias,

(i) actualmente o nodo 8 está na ruta $[1, 8, 1]$ e o 6, na $[1, 6, 5, 1]$ polo que, efectivamente son rutas diferentes,

(ii) o nodo 8 é o único cliente da súa ruta e na ruta do nodo 6, este é o primeiro logo cúmprese a condición,

(iii) $q_8 + q_6 + q_5 = 20 + 9 = 29 < Q$.

Dado que cumpre todas as condicións, esta unión de rutas é factible, resultando **[1,8,6,5,1]**.

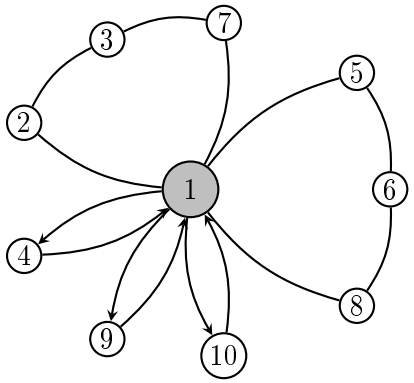

Figura 3.6: Paso 4.6.

Paso 4.7 Tomamos a sétima unión da lista, $[8, 5]$, e miramos se cumpre as condicións necesarias,

(i) actualmente o nodo 8 e o 5 xa están na mesma ruta, $[1, 8, 6, 5, 1]$.

Como a unión das rutas existentes mediante $[8, 5]$ non cumpre (i), non é factible, desbotámola e pasamos á seguinte da lista.

Paso 4.8 Tomamos a oitava unión da lista, $[6, 2]$, e miramos se cumpre as condicións necesarias,

(i) actualmente o nodo 6 está na ruta $[1, 8, 6, 5, 1]$ e o 2, na $[1, 7, 3, 2, 1]$ polo que, efectivamente son rutas diferentes,

(ii) aínda que o nodo 2 si é o último cliente da súa ruta, o 6 non o é na súa logo non se cumpre esta condición.

Como a unión das rutas existentes mediante $[6, 2]$ non cumpre (ii), non é factible, desbotámola e pasamos á seguinte da lista.

Paso 4.9 Tomamos a novena unión da lista, $[5, 4]$, e miramos se cumpre as condicións necesarias,

(i) actualmente o nodo 5 está na ruta $[1, 8, 6, 5, 1]$ e o 4, na $[1, 4, 1]$ polo que, efectivamente son rutas diferentes,

(ii) o nodo 4 é o único cliente da súa ruta e na ruta do nodo 8, este é o primeiro logo cúmprese a condición,

(iii) $q_8 + q_6 + q_5 + q_4 = 29 + 18 = 47 \nleq Q$.

Como a unión das rutas existentes mediante $[5, 4]$ non cumpre (iii), non é factible, desbotámola e pasamos á seguinte da lista.

Paso 4.10 Tomamos a décima unión da lista, $[4, 3]$, e miramos se cumpre as condicións necesarias,

(i) actualmente o nodo 4 está na ruta $[1, 4, 1]$ e o 3, na $[1, 7, 3, 2, 1]$ polo que, efectivamente son rutas diferentes,

(ii) o nodo 4 é o único cliente da súa ruta, pero na outra ruta o nodo 3 é un nodo intermedio polo que non cumpre a condición.

Como a unión das rutas existentes mediante $[4, 3]$ non cumpre (ii), non é factible, desbotámola e pasamos á seguinte da lista.

Paso 4.11 Tomamos a décimo primeira unión da lista, $[10, 8]$, e miramos se cumpre as condicións necesarias,

(i) actualmente o nodo 8 está na ruta $[1, 8, 6, 5, 1]$ e o 10, na $[1, 10, 1]$ polo que, efectivamente son rutas diferentes,

(ii) o nodo 10 é o único cliente da súa ruta e na ruta do nodo 8, este é o primeiro logo cúmprese a condición,

(iii) $q_{10} + q_8 + q_6 + q_5 = 6 + 29 = 35 < Q$.

Dado que cumpre todas as condicións, esta unión de rutas é factible, resultando **[1,10,8,6,5,1]**.

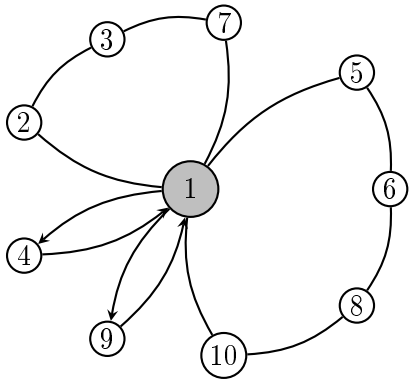

Figura 3.7: Paso 4.11.

Paso 4.12 Tomamos a décimo segunda unión da lista, $[6, 4]$, e miramos se cumpre as condicións necesarias,

(i) actualmente o nodo 6 está na ruta $[1, 8, 6, 5, 1]$ e o 4, na $[1, 4, 1]$ polo que, efectivamente son rutas diferentes,

(ii) o nodo 4 é o único cliente da súa ruta, pero na outra ruta o nodo 6 é un nodo intermedio polo que non cumpre a condición.

Como a unión das rutas existentes mediante $[6, 4]$ non cumpre (ii), non é factible, desbotámola e pasamos á seguinte da lista.

Paso 4.13 Tomamos a décimo terceira unión da lista, $[10, 9]$, e miramos se cumpre as condicións necesarias,

(i) actualmente o nodo 10 está na ruta $[1, 10, 8, 6, 5, 1]$ e o 9, na $[1, 9, 1]$ polo que, efectivamente son rutas diferentes,

(ii) o nodo 9 é o único cliente da súa ruta e na ruta do nodo 10, este é o primeiro logo cúmprese a condición,

(iii) $q_9 + q_{10} + q_8 + q_6 + q_5 = 4 + 35 = 39 < Q$.

Dado que cumpre todas as condicións, esta unión de rutas é factible, resultando **[1,9,10,8,6,5,1]**.

Chegados a este punto, temos todos os clientes agás o 4 cunha ruta asignada. Podemos ver que ambas rutas non se poden unir debido á restrición de carga dos vehículos e que o cliente 4 non pode ser engadido a ningunha das dúas rutas xa formadas polo mesmo motivo,

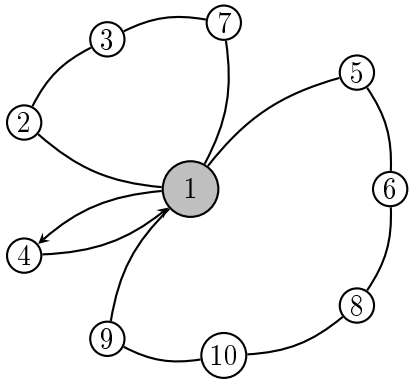

Figura 3.8: Paso 4.13.

o que implica que este será visitado na ruta **[1,4,1]**. O algoritmo conclúe proporcionando a solución que consta das seguintes rutas:

- Ruta 1: $[1, 7, 3, 2, 1]$ con custo $c_{17} + c_{73} + c_{32} + c_{21} = 9 + 8 + 8 + 12 = 37$
- Ruta 2: $[1, 9, 10, 8, 6, 5, 1]$ con custo $c_{19} + c_{910} + c_{108} + c_{86} + c_{65} + c_{51} = 6 + 10 + 11 + 6 + 3 + 10 = 46$
- Ruta 3: $[1, 4, 1]$ con custo $2 \cdot c_{14} = 2 \cdot 7 = 14$

Polo que o custo total será a suma dos custos de cada unha das tres rutas, é dicir 97. Tendo en conta que o custo de abastecer a cada cliente individualmente era 170, podemos observar que o aforro é do 42.94 %.

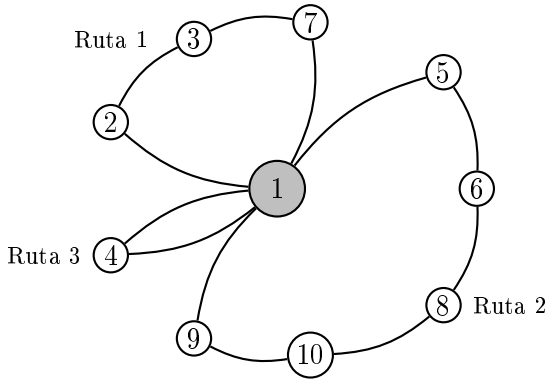

Figura 3.9: Solución versión paralela.

Versión secuencial

Ruta 1

Paso 4.1 Consideramos a primeira unión da lista do Paso 3, $[6,5]$, que induce á ruta **[1,6,5,1]** para a cal $q_6 + q_5 = 3 + 17 = 20 < Q$ polo que é factible.

Paso 5.1 Eliminamos da lista esta unión quedando: $[3,2]$, $[4,2]$, $[5,2]$, $[7,3]$, $[8,6]$, $[8,5]$, $[6,2]$, $[5,4]$, $[4,3]$, $[10,8]$, $[6,4]$, $[10,9]$, $[7,2]$, $[10,6]$, $[9,7]$, $[5,3]$, $[7,4]$, $[10,7]$, $[8,2]$, $[6,3]$, $[10,5]$, $[9,3]$, $[8,4]$, $[9,8]$, $[10,2]$, $[10,4]$, $[7,5]$, $[9,2]$, $[8,3]$, $[10,3]$, $[9,4]$, $[9,5]$, $[7,6]$, $[9,6]$, $[8,7]$.

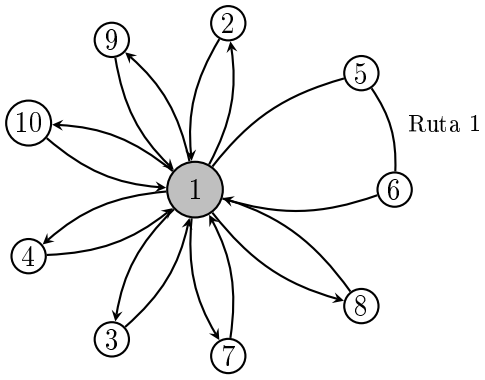

Figura 3.10: Ruta 1, Paso 4.1.

Paso 4.2 Consideramos a primeira unión da lista do Paso 5.1 que conteña ao cliente 5 ou ao 6, $[5,2]$, que induce á ruta **[1,6,5,2,1]** para a cal $q_6 + q_5 + q_2 = 20 + 10 = 30 < Q$ polo que é factible.

Paso 5.2 Eliminamos da lista esta unión, as que conteñan ao cliente 5 e as que relacionen a clientes da ruta entre si quedando $[3,2]$, $[4,2]$, $[7,3]$, $[8,6]$, $[4,3]$, $[10,8]$, $[6,4]$, $[10,9]$, $[7,2]$, $[10,6]$, $[9,7]$, $[7,4]$, $[10,7]$, $[8,2]$, $[6,3]$, $[9,3]$, $[8,4]$, $[9,8]$, $[10,2]$, $[10,4]$, $[9,2]$, $[8,3]$, $[10,3]$, $[9,4]$, $[7,6]$, $[9,6]$, $[8,7]$.

Paso 4.3 Consideramos a primeira unión da lista do Paso 5.2 que conteña ao cliente 2 ou ao 6, $[3,2]$, que induce á ruta $[1,6,5,2,3,1]$ para a cal $q_6 + q_5 + q_2 + q_3 = 30 + 15 = 45 \nleq Q$ polo que non é factible, desbotámola.

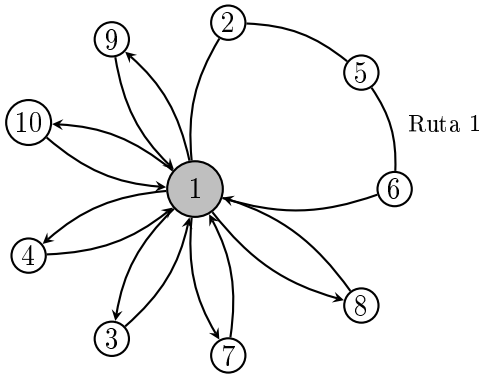

Figura 3.11: Ruta 1, Paso 4.2.

Paso 5.3 Eliminamos da lista esta unión quedando $[4,2]$, $[7,3]$, $[8,6]$, $[4,3]$, $[10,8]$, $[6,4]$, $[10,9]$, $[7,2]$, $[10,6]$, $[9,7]$, $[7,4]$, $[10,7]$, $[8,2]$, $[6,3]$, $[9,3]$, $[8,4]$, $[9,8]$, $[10,2]$, $[10,4]$, $[9,2]$, $[8,3]$, $[10,3]$, $[9,4]$, $[7,6]$, $[9,6]$, $[8,7]$.

Paso 4.4 Consideramos a primeira unión da lista do Paso 5.3 que conteña ao cliente 2 ou ao 6, $[4,2]$, que induce á ruta $[1,6,5,2,4,1]$ para a cal $q_6+q_5+q_2+q_4 = 30+18 = 48 \nleq Q$ polo que non é factible, desbotámola.

Paso 5.4 Eliminamos da lista esta unión quedando $[7,3]$, $[8,6]$, $[4,3]$, $[10,8]$, $[6,4]$, $[10,9]$, $[7,2]$, $[10,6]$, $[9,7]$, $[7,4]$, $[10,7]$, $[8,2]$, $[6,3]$, $[9,3]$, $[8,4]$, $[9,8]$, $[10,2]$, $[10,4]$, $[9,2]$, $[8,3]$, $[10,3]$, $[9,4]$, $[7,6]$, $[9,6]$, $[8,7]$.

Paso 4.5 Consideramos a primeira unión da lista do Paso 5.4 que conteña ao cliente 2 ou ao 6, $[8,6]$, que induce á ruta $[1,8,6,5,2,1]$ para a cal $q_8+q_6+q_5+q_2 = 9+30 = 39 < Q$ polo que é factible.

Paso 5.5 Eliminamos da lista esta unión, as que conteñan ao cliente 6 e as que relacionen a clientes da ruta entre si quedando quedando $[7,3]$, $[4,3]$, $[10,8]$, $[10,9]$, $[7,2]$, $[9,7]$, $[7,4]$, $[10,7]$, $[9,3]$, $[8,4]$, $[9,8]$, $[10,2]$, $[10,4]$, $[9,2]$, $[8,3]$, $[10,3]$, $[9,4]$, $[8,7]$.

Paso 6 Dado que non existe cliente con demanda menor ou igual a 1 que é o máximo que se pode engadir a esta ruta rematamos a Ruta 1 e eliminamos da lista de unións todas as que conteñan algún dos clientes presentes na mesma quedando $[7,3]$, $[4,3]$, $[10,9]$, $[9,7]$, $[7,4]$, $[10,7]$, $[9,3]$, $[10,4]$, $[10,3]$, $[9,4]$.

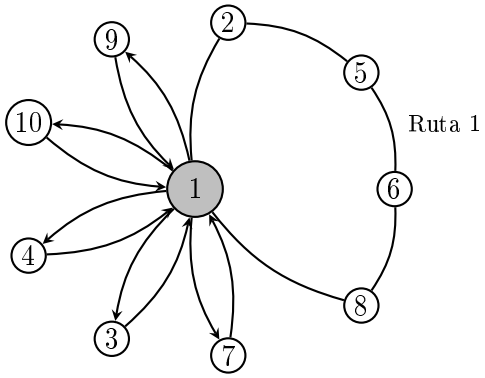

Figura 3.12: Ruta 1, Paso 4.5.

Ruta 2

Paso 4.1 Consideramos a primeira unión da lista do Paso 6 da Ruta 1, $[7,3]$, que induce á ruta **[1,7,3,1]** para a cal $q_7 + q_3 = 5 + 15 = 20 < Q$ polo que é factible.

Paso 5.1 Eliminamos da lista esta unión quedando $[4,3]$, $[10,9]$, $[9,7]$, $[7,4]$, $[10,7]$, $[9,3]$, $[10,4]$, $[10,3]$, $[9,4]$.

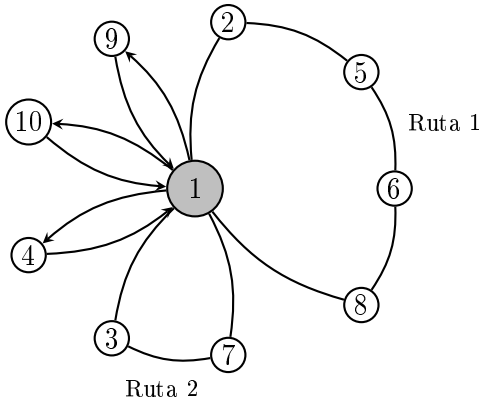

Figura 3.13: Ruta 2, Paso 4.1.

Paso 4.2 Consideramos a primeira unión da lista do Paso 5.1 que conteña ao cliente 7 ou ao 3, $[4,3]$, que induce á ruta **[1,7,3,4,1]** para a cal $q_7 + q_3 + q_4 = 20 + 18 = 38 < Q$ polo que é factible.

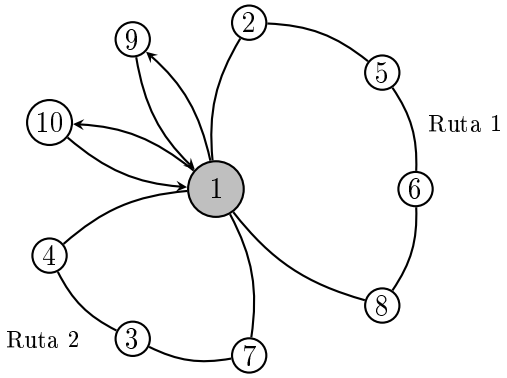

Figura 3.14: Ruta 2, Paso 4.2.

Paso 5.2 Eliminamos da lista esta unión, as que conteñan ao cliente 3 e as que relacionen a clientes da ruta entre si quedando $[10, 9]$, $[9, 7]$, $[10, 7]$, $[10, 4]$, $[9, 4]$.

Paso 6 Dado que non existe cliente con demanda menor ou igual a 2 que é o máximo que se pode engadir a esta ruta rematamos a Ruta 2 e eliminamos da lista de unións todas as que conteñan algún dos clientes presentes na mesma quedando unicamente a $[10, 9]$.

Ruta 3

Paso 4.1 Consideramos a única unión da lista restante, $[10, 9]$, que induce á ruta $[1, 10, 9, 1]$ para a cal $q_{10} + q_9 = 6 + 4 = 10 < Q$ polo que é factible.

Paso 5.1 Ao eliminar esta unión da lista chegamos ao final do procedemento logo temos a solución buscada.

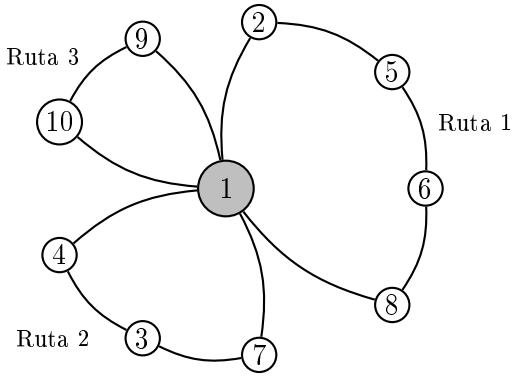

Figura 3.15: Solución versión secuencial.

A solución propón visitar aos clientes seguindo as rutas:

- Ruta 1: $[1, 8, 6, 5, 2, 1]$ con custo $c_{18} + c_{86} + c_{65} + c_{52} + c_{21} = 8 + 6 + 3 + 9 + 12 = 38$.
- Ruta 2: $[1, 7, 3, 4, 1]$ con custo $c_{17} + c_{73} + c_{34} + c_{41} = 9 + 8 + 9 + 7 = 33$.
- Ruta 3: $[1, 10, 9, 1]$ con custo $c_{110} + c_{109} + c_{91} = 12 + 10 + 6 = 28$.

Polo que o custo total será a suma dos custos de cada unha das tres rutas, é dicir 99. O cal é un 2.06 % maior que o obtido usando a versión paralela do algoritmo, pero igualmente un 41.76 % menor que o custo de visitar cada cliente individualmente.

Como podemos ver, a versión paralela e a secuencial non proporcionan a mesma solución. Toth e Vigo (2002) compararon ambas versións para os catorce casos de Christofides et al. (1979) usando distancias reais como podemos ver no cadro 3.3 e observaron que a versión paralela domina claramente á secuencial, o que coincide cos resultados obtidos no exemplo.

Problema	Versión secuencial	Versión paralela	Mellor solución coñecida
E051-05e	625.56	584.64	524.61[1]
E076-10e	1005.25	900.26	835.26[1]
E101-08e	982.48	886.83	826.14[1]
E101-10c	939.99	833.51	819.56[1]
E121-07c	1291.33	1071.07	1042.11[1]
E151-12c	1299.39	1133.43	1028.42[1]
E200-17c	1708.00	1395.74	1291.45[1]
D051-06c	670.01	618.40	555.43[1]
D076-11c	989.42	975.46	909.68[1]
D101-09c	1054.70	973.94	865.94[1]
D101-11c	952.53	875.75	866.37[1]
D121-11c	1646.60	1596.72	1541.14[2]
D151-14c	1383.87	1287.64	1162.55[2]
D200-18c	1671.29	1538.66	1395.85[1]

[1] Taillard (1993).

[2] Rochat and Taillard (1995).

Cadro 3.3: Datos comparación das versións de CW en Toth e Vigo (2002).

Capítulo 4

Estudo dun caso do mundo real

4.1. Introdución

Nos capítulos anteriores fixemos unha revisión do TSP e o VRP así como as súas formulacións e presentamos o método heurístico de Clarke and Wright. Neste capítulo, partindo dos datos dun problema real, faremos unha comparación entre a solución obtida co modelador AMPL (2019) e o solver Gurobi (2019) e a que nos proporciona a implementación de Clarke and Wright en R (2019).

AMPL (A Mathematical Programming Language) é unha linguaxe e un sistema de modelado que permite resolver, usando diferentes solvers de optimización incluídos na súa distribución problemas de programación matemática como o VRP, é dicir, con AMPL podemos representar problemas de minimización ou maximización, escribindo facilmente a súa función obxectivo e as súas restricións, e resolvelo con terceiras ferramentas incluídas como Gurobi, unha ferramenta de optimización para resolver problemas de programación linear enteira e enteira-mixta.

AMPL usa unha representación alxébrica de alto nivel que describe os modelos de optimización tal e como pensamos sobre eles. Para obter os datos presentados neste traballo, usamos AMPL con licencia académica [1] así como NEOS Server (2019), un servizo de internet gratuíto para a resolución de problemas numéricos de optimización con base no Wisconsin Institute for Discovery na Universidade de Wisconsin en Madison e que ten acceso a numerosos solvers que se executan en computadoras de alta capacidade distribuídas en varias universidades de todo o mundo. Como solver utilizamos Gurobi na súa versión para problemas de programación linear enteira. [2]

[1] Salvo que se indique o contrario, todos os datos que figuren como obtidos con Gurobi será usando esta opción.

[2] Para facer as simulacións usaremos un PC con unha Intel(R) Core(TM) i5-5200U CPU @ 2.20 GHz,

Para traballar co servidor gratuíto NEOS basta entrar na súa páxina web e elixir a opción de "submit a job to NEOS", entón entraremos nunha páxina onde se mostran diferentes tipos de problemas e os diferentes solvers dos que se dispón para cada un deles. Como o VRP é un problema de programación enteira mixta deberemos buscar os solvers compatibles con este tipo de problemas, elixiremos Gurobi e seleccionamos a opción "AMPL Input". Para traballar bastará enviar 3 ficheiros de texto co modelo, os datos e os comandos de execución e recíbense os resultados por correo electrónico.

Como xa vimos na sección 3.1 debido a que o VRP é un problema NP-duro, adoitan usarse algoritmos heurísticos para obter solucións cuasi-óptimas en tempos razoables. Como método heurístico para aplicar ao problema implementamos a versión paralela de Clarke & Wright en R que procede sistematicamente como mostra o seguinte pseudocódigo:[3]

1: Crear R a matriz de rutas coas rutas iniciais de ir e volver a cada cliente (a fila i-ésima de R será $(0, i, 0)$)
2: Calcular a matriz de aforros S
3: **while** existan aforros maiores que 0 **do**
4: escoller $\max_{i,j \in N} s[i, j]$
5: **while** $R[i, 3] = 0$, $R[j, 1] = 0$ e a suma das demandas de todos os clientes pertencentes á ruta sexa menor ou igual que a capacidade do vehículo **do**
6: $R[i, 3] = j$ e $R[j, 1] = i$
7: **end while**
8: **end while**

4.2. Aplicación a un problema real

Contamos cos datos dunha cooperativa galega, distribuidora de catro tipo de pensos para animais que conta con aproximadamente 1500 clientes distribuídos en 60 municipios adxacentes. Cada cliente adoita facer entre un e dous pedidos ao mes, habitualmente do mesmo tipo de penso. Ditos pedidos poden ser urxentes (en cuxo caso se incrementará o seu prezo) ou ter un período de abastecemento específico. Esta cooperativa posúe unha flota heteroxénea onde cada camión conta con varios compartimentos con diferentes capacidades e nos cales non poden mesturarse diferentes tipos de pensos nin pedidos de clientes diferentes. Os camións contan con restricións en canto ao tempo de viaxe diario, á súa carga e aos camiños polos que poden pasar, isto implica que o acceso a cada cliente pode estar restrinxido a un determinado tipo de vehículos. O condutor do camión cobra segundo

2.20GHz e 4GB de RAM.

[3] O código completo pode consultarse no Apéndice D.

a distancia viaxada e a carga transportada.

Dada a complexidade do problema real, para o obxectivo deste capítulo de comparar a solución dun VRP coa aproximación obtida por CW reducimos este problema a un CVRP supoñendo unha flota homoxénea con $Q = 15300$ Kg e tomando os pedidos recibidos durante un determinado día sen ter en conta que os pensos poden ser de diferentes tipos. Estes pedidos están representados na figura 4.1 onde no eixe de abscisas se representan os clientes (considerando a cooperativa como cliente 1) e no eixe de ordenadas as demandas de cada un en quilogramos. Consideramos tamén as distancias entre os clientes que fixeron pedidos durante o día estudado en quilómetros.

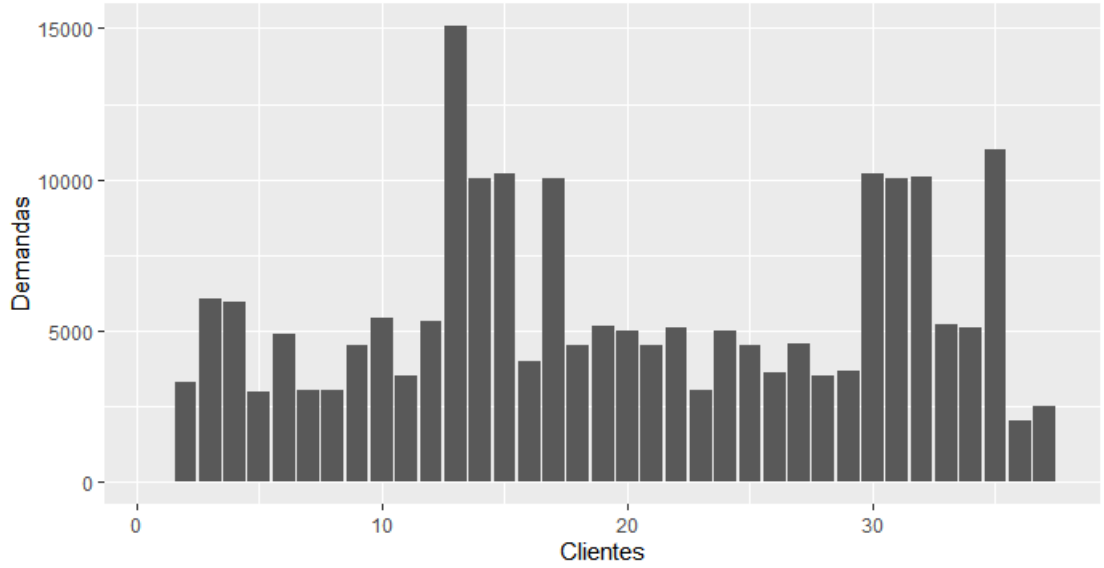

Figura 4.1: Demandas en Kg dos clientes da cooperativa durante un determinado día.

Exemplo 4.1. Comparamos as solucións obtidas con AMPL e o solver Gurobi e os tempos de obtención das mesmas co tempo e as aproximacións á solución que nos proporciona a implementación en R do algoritmo heurístico de Clarke & Wright. Usaremos para iso o comando *_total_solve_elapsed_time* en AMPL e *system.time* en R.

Comezamos tomando o subproblema que conta cos primeiros **5 clientes** do problema orixinal. Neste caso o vector demandas en quilogramos vén dado por $(q_2, ..., q_6) = (3300, 6041, 5959, 2951, 4885)$ e a matriz de distancias simétrica en quilómetros móstranse no cadro 4.1 estando no nodo 1 a cooperativa.

Neste caso, a solución de Gurobi e a do heurístico coinciden. As rutas da solución preséntanse na figura 4.2 e o custo de percorrelas é 177 km. Por outra parte, os tempos de

c_{ij}	1	2	3	4	5	6
1	-					
2	21	-				
3	20	4	-			
4	17	6	4	-		
5	65	60	59	57	-	
6	63	58	56	54	3	-

Cadro 4.1: Matriz de distancias con 5 clientes.

obtención de dita solución son diferentes pois mentres Gurobi precisa aproximadamente 0.24 segundos, a implementación de CW dá a solución en 0.015 segundos.

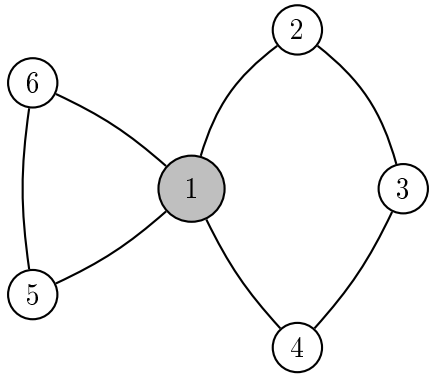

Figura 4.2: Rutas solución para 5 clientes.

Tomando o subproblema cos primeiros **7 clientes** o vector de demandas en quilogramos é $(q_2, ..., q_8) = (3300, 6041, 5959, 2951, 4885, 3003, 3016)$ e a matriz simétrica de distancias en quilómetros vén dada no cadro 4.2 estando no nodo 1 a cooperativa.

c_{ij}	1	2	3	4	5	6	7	8
1	-							
2	21	-						
3	20	4	-					
4	17	6	4	-				
5	65	60	59	57	-			
6	63	58	56	54	3	-		
7	60	55	53	52	7	4	-	
8	19	15	13	11	66	64	61	-

Cadro 4.2: Matriz de distancias con 7 clientes.

As rutas acadadas por Gurobi e por CW son diferentes como se pode ver na figura 4.3, pero o custo de percorrelas é o mesmo, 198 km. Gurobi tarda aproximadamente 1.247 segundos mentres que a CW bástanlle 0.033 segundos.

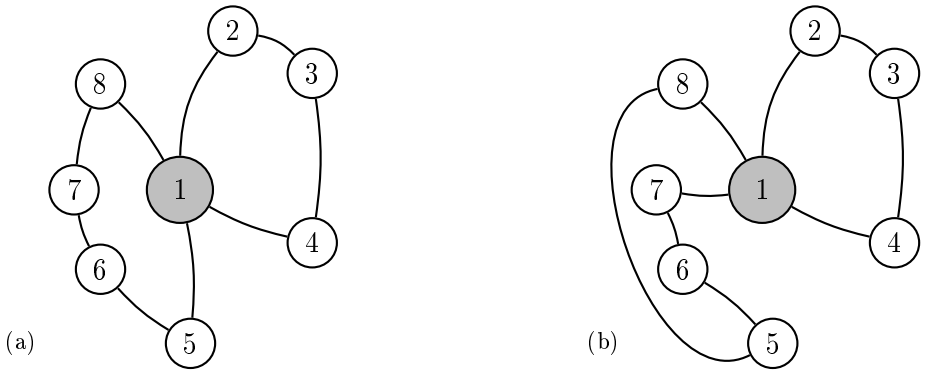

Figura 4.3: Rutas solución para 7 clientes. (a) Usando CW. (b) Usando Gurobi.

Se en vez de 7, tomamos os primeiros **10 clientes**, o vector de demandas en quilogramos é $(q_2, ..., q_{11}) = (3300, 6041, 5959, 2951, 4885, 3003, 3016, 4478, 5413, 3490)$ e a matriz simétrica de distancias en quilómetros vén dada no cadro 4.3 estando no nodo 1 a cooperativa.

c_{ij}	1	2	3	4	5	6	7	8	9	10	11
1	-										
2	21	-									
3	20	4	-								
4	17	6	4	-							
5	65	60	59	57	-						
6	63	58	56	54	3	-					
7	60	55	53	52	7	4	-				
8	19	15	13	11	66	64	61	-			
9	22	18	8	1	69	66	64	3	-		
10	24	20	12	16	71	69	66	5	7	-	
11	24	8	4	9	63	60	58	8	9	12	-

Cadro 4.3: Matriz de distancias con 10 clientes.

Na figura 4.4 podemos ver representadas as rutas propostas por ambos métodos que, ao contrario que no caso anterior, non coinciden, dando CW un resultado un 9.67 % peor.

O custo de levar a cabo as rutas propostas por CW é 272 km mentres que se seguimos as de Gurobi será 248 km. Á súa vez mentres que o algoritmo implementado en R dá a solución en pouco máis de 0.05 segundos, Gurobi precisa máis de 68 segundos.

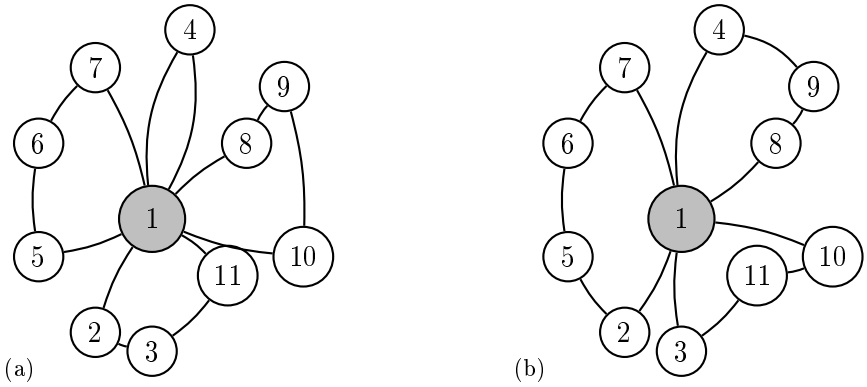

Figura 4.4: Rutas solución para 10 clientes. (a) Usando CW. (b) Usando Gurobi.

Se tomamos agora o subproblema cos primeiros **13 clientes** o vector de demandas é $(q_2, ..., q_{14}) = (3300, 6041, 5959, 2951, 4885, 3003, 3016, 4478, 5413, 3490, 5283, 15078, 10017)$ expresadas en quilogramos e a matriz simétrica de distancias, expresadas en quilómetros, dáse no cadro 4.4 estando no nodo 1 a cooperativa.

c_{ij}	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	-													
2	21	-												
3	20	4	-											
4	17	6	4	-										
5	65	60	59	57	-									
6	63	58	56	54	3	-								
7	60	55	53	52	7	4	-							
8	19	15	13	11	66	64	61	-						
9	22	18	8	1	69	66	64	3	-					
10	24	20	12	16	71	69	66	5	7	-				
11	24	8	4	9	63	60	58	8	9	12	-			
12	27	11	10	12	58	55	52	17	15	18	6	-		
13	23	7	6	7	62	59	57	9	10	14	2	4	-	
14	5	20	18	1	71	68	66	18	20	23	23	26	21	-

Cadro 4.4: Matriz de distancias con 13 clientes.

Neste caso, as rutas obtidas coa implementación de CW coinciden novamente coa solución proporcionada por Gurobi que se representa na figura 4.5 con custo 342 km. Novamente os tempos necesarios para chegar a esta solución son moi diferentes pois mentres con R se pode obter en menos de 0.1 segundos, Gurobi precisa aproximadamente 2785 segundos, é dicir, máis de 46 minutos.

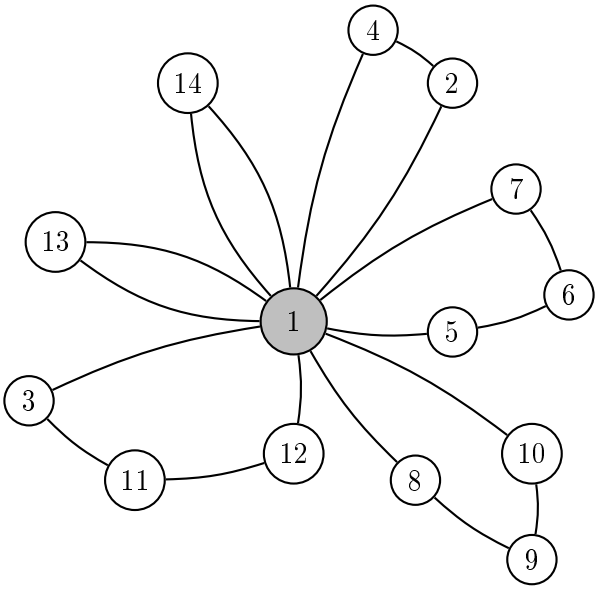

Figura 4.5: Rutas solución para 13 clientes.

Usando o servidor gratuíto NEOS, para subproblemas con máis de 10 clientes devólvenos un erro conforme se supera o tempo máximo permitido para a resolución dun problema e coa licencia académica de AMPL este subproblema de 13 clientes é o máximo que conseguimos resolver pois, en 25 horas Gurobi non foi capaz de acadar unha solución óptima para **14 clientes**. Pola contra, o método heurístico permítenos seguir resolvendo problemas con un número de clientes moito maiores e en tempos pequenos. Así por exemplo, o problema orixinal de 36 clientes resólveo en pouco máis de 1 segundo proporcionando as rutas da figura 4.6.[4]

[4]Debido ás súas dimensións, a matriz de distancias non se inclúe, pero pode consultarse coa autora do TFG.

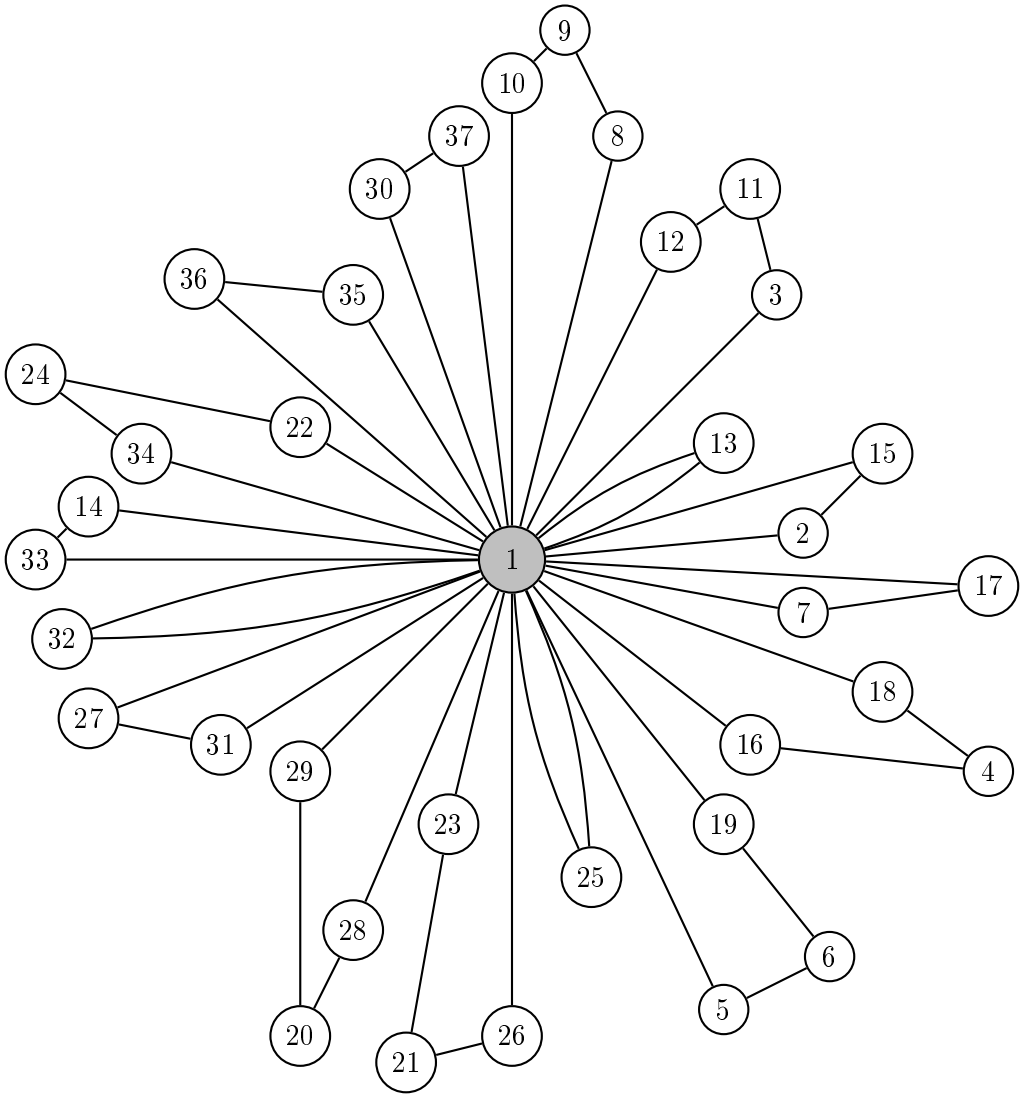

Figura 4.6: Rutas solución para 36 clientes.

Para obter as solucións dos diferentes problemas, tanto con CW como con Gurobi con 10 ou menos clientes, os tempos son pequenos e moi susceptibles a cambios polo tanto, co fin de ter unha maior exactitude, executamos os algoritmos varias veces. Pódense ver os tempos de cada execución así como a media e a varianza no apéndice B. Ao longo deste capítulo usouse a media dos tempos para as diferentes comparacións. Na táboa 4.5 podemos ver dunha forma esquematizada todas as comparacións feitas durante este capítulo.

Número de clientes	**Gurobi**		**CW**	
	Custo	Tempo	Custo	Tempo
5	177	0.241s	177	0.015s
7	198	1.247s	198	0.033s
10	248	68.101s	272	0.0527s
13	342	2785s	342	0.095s
15	-	>25h	379	0.124s
36	-	-	756	1.033s

Cadro 4.5: Comparación custos e tempos con Gurobi e CW.

Aínda que os métodos heurísticos non sempre nos ofrezan a solución óptima, como vimos neste exemplo para o caso de 10 clientes, cando si o fan, proporciónannos dita solución nun tempo notablemente menor, véxase por exemplo o subproblema con 13 clientes. Pero a maior das vantaxes deste tipo de métodos cando tratamos de atopar unha solución a un problema NP-duro como é neste caso o VRP é que, aínda cando o algoritmo exacto (AMPL neste caso) deixa de funcionar, o heurístico continúa ofrecéndonos solucións cuasi-óptimas en tempos razoables.

Conclusións

Para a elaboración deste traballo comezamos cunha revisión bibliográfica do TSP: as súas orixes e a súa evolución, pois dende que en 1954 Dantzig et al. resolveron por primeira vez un TSP para 49 cidades, non se fixo máis que avanzar e no 2006 conseguiuse resolver un TSP con 85900 cidades; a súa formulación; algunhas variacións do mesmo e métodos de resolución. A complexidade computacional do TSP -que vimos que é un problema que, para rutas moi grandes non se pode atopar unha solución óptima en tempo polinomial usando métodos de optimización exactos- levounos a falar dos métodos heurísticos para este tipo de problemas que, aínda que non garanten que a solución que nos proporcionan sexa óptima, o esforzo computacional dos mesmos son notablemente menores que os dos algoritmos exactos.

No segundo capítulo fixemos unha revisión bibliográfica de diferentes variacións dos problemas de rutas de vehículos introducidos por Dantzig e Ramser en 1959, comezando polo modelo máis básico e introducindo outros como por exemplo o problema con vehículos con diferentes compartimentos, con entregas divididas ou problemas estocásticos. Este tipo de situacións implican tratar de minimizar ou maximizar unha función obxectivo de varias variables definidas nun conxunto discreto. A motivación académica para resolver estes problemas é que, igual que para o TSP, non é posible construír un algoritmo de tempo polinomial que resolva calquera instancia do problema. Presentamos entón no terceiro capítulo o algoritmo heurístico proposto en 1964 por CW o cal tivo grande interese na literatura e serviu como punto de partida para modificacións aplicables a modelos máis xerais.

No cuarto capítulo, modelamos un problema CVRP con linguaxe AMPL e, usando o solver Gurobi, obtivemos solucións para subproblemas con pequenos números de clientes. Por outra parte, fixemos unha implementación en R do algoritmo heurístico de CW que se estudara no capítulo anterior e executámolo para os mesmos subproblemas. Comparamos os datos obtidos por cada un dos métodos e puidemos observar que, aínda que os métodos heurísticos non sempre nos ofrezan a solución óptima, como por exemplo pasou para un subproblema de 10 clientes, cando si o fan, proporciónannos dita solución nun tempo

notablemente menor. Por outra parte puidemos observar que, para subproblemas con máis de 13 clientes, o algoritmo exacto deixaba de proporcionarnos unha solución nun tempo de computación razoable, mentres que o heurístico non só o facía para estes subproblemas senón que chegaba a resolver o problema total con 36 clientes en pouco máis dun segundo. Comprobamos entón que os algoritmos heurísticos, aínda que dende o punto de vista de obtención da solución óptima nos poidan resultar insatisfactorios, poden ser realmente útiles na práctica debido á súa velocidade.

Este traballo preséntase como unha continuación da materia de *Programación linear e enteira* do Grao en Matemáticas da USC onde se estudan diferentes problemas entre os que se atopan o TSP, as clases de complexidade de algoritmos e algoritmos de solución para problemas de programación linear e de programación linear enteira. O interese na área de investigación dos VRP é teórico e aplicado pois este tipo de problemas posúen un gran carácter interdisciplinar e de potencial transferencia á industria pois úsanse modelos matemáticos para problemas pertencentes ao ámbito da organización e da loxística usando como ferramenta a informática. Este TFG é unha introdución a este tipo de problemas e poderíase continuar considerando diferentes tipos de algoritmos, outros modelos de VRP máis complexos ou incluso propoñendo novos algoritmos de solución.

Bibliografía

[1] Alvina, K., Ruey, C. and Quiang, M. (2008) *Distance-constrained capacited vehicle routing problema with flexible assignment of start and end depots.* ScienceDirect. Singapore.

[2] AMPL web: https://ampl.com/ (Visitada o 26 de xuño de 2019)

[3] Aykagan, A. and Erera, A. L. (2007) *A paired-vehicle recourse strategy for the vehicle-routing problem with stochastic demands.* Transportation Science,41. Páxinas 222-237.

[4] Berhan, E., Beshah, B., Kitaw, D. and Abraham, A.(2014) *Stochastic vehicle routing problem: a literature survey.* Journal of Information & Knowledge Management, 13.

[5] Biggs, N. L., Lloyd, E. K. and Wilson, R. J. (1976) *Graph theory 1736-1936.* Clarendon Press. Oxford (Great Britain). Capítulos 1 e 2.

[6] Chauhan, C., Gupta, R. and Pathak, K. (2012) *Methods of solving TSP along with its implementation using dynamic programming approach.* International Journal of Computer Applications, 52(4). Páxinas 12-19.

[7] Clarke, G. and Wright, J. (1964) *Scheduling of vehicles from a central depot to a number of delivery points.* Operations Research, 12 (4). Páxinas 568-581.

[8] Christofides, N., Mingozzi, A. and Toth, P. *The vehicle routing problem.* In Christofides, N., Mingozzi, A., Toth, P. and Sandi, C. (Ed) (1979) *Combinatorial optimization.* Wiley, Chichester, UK.

[9] Christofides, N., Mingozzi, A. and Toth, P. (1981) *State-space relaxation procedures for the computation of bounds to routing problems.* Networks, 11. Páxinas 145-164.

[10] Dantzig, G. B., Fulkerson, D. R. and Johnson, S. M. (1954) *Solution of a large-scale traveling-salesman problem.* Operations Research, 2. Páxinas 393–410.

[11] Dantzig, G. B. and Ramser, J. H. (1959) *The truck dispatching problem.* Management Science, 6. Páxinas 80-91.

[12] Dror, M., Laporte, G. and Trudean, P. (1989) *Vehicle routing with stochastic demands: properties and solution frameworks.* Transportation Science, 23. Páxinas 166-176.

[13] *Ein alter Commis-Voyageur.* (1832) *Der Handlungsreisende-wie er sein soll und was er zu thun hat, um Auftr age zu erhalten und eines gl ucklichen Erfolgs in seinen Gesch aften gewiss zu sein.* B. Fr. Voigt, Ilmenau.

[14] Euler, L. (1736) *Solutio problemats ad geometriam situs pertinentis.* Commemtarii Academia Scientiarum Imperialis Petropolitanae, 8. Páxinas 128-140.

[15] Fisher, M. L. and Jaikumar, R. (1981) *A generalized assigment heuristic for vehicle routing.* Networks, 11. Páxinas 109-124.

[16] González Díaz, J. *Apuntamentos da materia Programación Linear e Enteira.* Curso 2018-2019. Universidade de Santiago de Compostela.

[17] Gurobi web: https://www.gurobi.com/es/ (Visitada o 26 de xuño de 2019)

[18] Hillier, F. S. and Lieberman G. J. (1997) *Introducción a la Investigación de Operaciones.* McGraw-Hill.

[19] Lawler, E., Lenstra, J., Rinnooy Kan, A. and Shmoys, D. (1986) *The Traveling Salesman Problem.* Wiley-Interscience series in discrete mathematics and optimization. Great Britain.

[20] Matai, R., Singh, S. P. and Mittal, M. L.*Traveling salesman problem: an overview of applications, formulations, and solution approaches.* In Davendra D. (Ed) (2010) *Traveling salesman problem, theory and applications.* InTech. Rijeka (Croacia).

[21] Mathur, K. and Solow, D. (1996) *Investigación de operaciones: el arte de la toma de decisiones.* Prentice-Hall Hispanoamericana. Capítulo 1.

[22] Miller, C. E., Tucker, A. W. and Zemlin, R. A. (1960) *Integer programming formulations and traveling salesman problems.* Journal of the ACM, 7. Páxinas 326-329.

[23] Muyldermans, L. and Pang, G. (2010) *On the benefits of co-collection: experiments with a multi-compartment vehicle routing algorithm.* European Journal of Operational Research, 206. Páxinas 93-103.

[24] NEOS web: https://neos-server.org/ (Visitada o 26 de xuno de 2019)

[25] R web: https://www.r-project.org/ (Visitada o 6 de xullo de 2019)

[26] Rochat, Y. and Taillard, E. D. (1995) *Probabilistic diversification and intensification in local search for vehicle routing.* Journal of Heuristics, 1. Páxinas 147-167.

[27] Taillard, E. D. (1993) *Parallel iterative search methods for vehicle routing problems.* Networks, 23. Páxinas 661-673.

[28] Toth, P. and Vigo, D. (2002) *The Vehicle Routing Problem.* SIAM monographs on discrete mathematics and applications. Bolonia (Italia).

[29] Vandermonde, A. T. (1974) *L'Historie de l'Acadèmie des Sciences for 1771.*

[30] Waterloo University web: http://www.math.uwaterloo.ca/tsp/ (Visitada o 17 de xuño de 2019)

Apéndice A

Revisión de conceptos de teoría de grafos

A continuación revisamos algúns conceptos de teoría de grafos empregados no traballo.

- **Grafo:** Un grafo G é un par (V, A) consistente nun conxunto V de elementos chamados *nodos* ou *vértices* e un conxunto A cuxos elementos representan os *arcos* ou *aristas*.
- **Grafo dirixido:** Un grafo dirixido ou orientado é aquel no que $A \subset V \times V$, é dicir, as aristas son pares ordenados: a arista (i, j) comeza no nodo i e remata no j.
- **Grafo non dirixido:** Nun grafo non dirixido, A está composto por subconxuntos de V de dous elementos. Neste caso $\{i, j\}$ e $\{j, i\}$ representan o mesmo conxunto e, polo tanto, a mesma arista
- **Grafo completo:** Un grafo é completo se todas as aristas posibles están presentes.
- **Subgrafo:** Un subgrafo de G é un grafo $G' = (V', A')$ que ten todos os seus vértices e aristas en G, é dicir, $V' \subseteq V$ e $M' \subseteq M$.
- **Subgrafo de expansión:** Un subgrafo de expansión de G é un subgrafo $G' = (V', A')$ tal que $V' = V$.
- **Grado:** O grado dun nodo calquera dun grafo G é o número de aristas incidentes con el.
- **Cadea:** Sexa G un grafo non dirixido e sexa $(a_1, a_2, ..., a_r)$ unha secuencia de aristas en G. Se existen vértices $(v_0, v_1, ..., v_r)$ tales que, para $l \in \{1, 2, ..., r\}$, $a_l = (v_{l-1}, v_l)$, dicimos que a secuencia é unha cadea.

- **Cadea pechada:** Unha cadea pechada é aquela na que $v_0 = v_r$.
- **Camiño:** Un camiño é unha cadea na que todos os vértices son distintos.
- **Circuíto:** Un circuíto ou ciclo é unha cadea pechada na que non hai máis nodos coincidentes que o primeiro e o último.
- **Grafo conexo:** Un grafo dise que é conexo se para cada par de vértices existe unha cadea non dirixida que os une.
- **Grafo fortemente conexo:** Un grafo dise que é fortemente conexo se para cada par de vértices existe unha cadea dirixida que os une.
- **Árbore:** Un grafo dise que é unha árbore se é conexo e non contén ciclos (non dirixidos).
- **Árbore de expansión:** Unha árbore de expansión dun grafo G é unha árbore que é un subgrafo de expansión de G.
- **Circuíto hamiltoniano:** Un circuíto hamiltoniano nun grafo G é un circuíto que contén a todos os vértices de G.
- **Grafo hamiltoniano:** Chamamos grafo hamiltoniano a aquel que conteña un circuíto hamiltoniano.
- **Cadea de Euler:** Unha cadea de Euler nun grafo G non dirixido é unha cadea pechada que contén a todos oso nodos polo menos unha vez e a todas as aristas unha única vez.
- **Grafo euleriano:** Chamamos grafo euleriano a aquel que contén unha cadea de Euler.

Apéndice B

Tempos para a obtención de solucións con Gurobi e CW

Nas seguintes táboas recóllense os datos temporais resultado de executar 30 veces o código de AMPL co solver Gurobi e a implementación de CW en R respectivamente para diferentes subproblemas con distinto números de clientes e cos datos do problema da vida real presentado na sección 4.2.

B.1. Tempos de execución de Gurobi

Clientes	Tempos de execución (segundos)					Media tempos	Varianza tempos
5	0.250	0.234	0.391	0.234	0.234	0.241	0.001
	0.235	0.250	0.234	0.250	0.219		
	0.250	0.234	0.219	0.234	0.219		
	0.219	0.234	0.234	0.235	0.219		
	0.250	0.234	0.219	0.250	0.281		
	0.234	0.250	0.234	0.234	0.219		
7	1.313	1.344	1.218	1.250	1.203	1.247	0.003
	1.235	1.234	1.219	1.234	1.282		
	1.266	1.250	1.203	1.234	1.219		
	1.250	1.203	1.219	1.281	1.250		
	1.219	1.219	1.234	1.469	1.235		
	1.234	1.219	1.218	1.234	1.235		
10	69.093	63.782	75.36	68.109	76.984	68.101	23.318
	63.719	61.891	66.891	69.625	73.515		
	65.219	63.625	67.140	70.093	69.360		
	64.407	62.078	66.172	68.562	66.813		
	63.062	64.407	71.422	66.813	66.562		
	62.188	82.953	72.969	71.203	69.015		

B.2. Tempos de execución da implementación en R de CW

Clientes	Tempos de execución (s)					Media tempos	Varianza tempos
5	0.00	0.01	0.01	0.02	0.00	0.0150	0.0731
	0.02	0.02	0.02	0.03	0.01		
	0.01	0.01	0.02	0.02	0.01		
	0.02	0.02	0.02	0.03	0.01		
	0.02	0.02	0.01	0.01	0.01		
	0.01	0.01	0.02	0.01	0.02		
7	0.09	0.04	0.03	0.03	0.09	0.0333	0.0003
	0.03	0.03	0.05	0.03	0.03		
	0.03	0.01	0.03	0.03	0.04		
	0.03	0.03	0.03	0.03	0.02		
	0.03	0.03	0.03	0.01	0.03		
	0.03	0.03	0.02	0.03	0.03		
10	0.04	0.07	0.09	0.07	0.03	0.0527	0.0191
	0.05	0.05	0.06	0.04	0.05		
	0.05	0.06	0.05	0.12	0.03		
	0.07	0.03	0.04	0.07	0.06		
	0.05	0.06	0.04	0.04	0.05		
	0.03	0.05	0.04	0.04	0.05		
13	0.14	0.10	0.07	0.11	0.10	0.0947	0.0003
	0.10	0.08	0.08	0.08	0.09		
	0.09	0.07	0.10	0.08	0.09		
	0.10	0.14	0.08	0.11	0.10		
	0.09	0.08	0.08	0.09	0.09		
	0.11	0.10	0.10	0.10	0.09		
15	0.12	0.13	0.12	0.13	0.12	0.1243	0.0133
	0.14	0.11	0.11	0.13	0.13		
	0.16	0.12	0.12	0.11	0.11		
	0.13	0.12	0.11	0.12	0.13		
	0.11	0.14	0.12	0.16	0.11		
	0.12	0.13	0.13	0.11	0.13		
36	1.01	1.00	1.10	1.00	1.03	1.0330	0.0321
	1.04	1.00	1.11	1.05	1.00		
	1.02	1.02	1.04	1.00	1.03		
	1.04	1.02	1.01	1.01	1.12		
	1.02	1.06	0.99	1.04	1.05		
	1.02	1.03	1.03	1.04	1.06		

Apéndice C

Código AMPL

```
# Parámetros

param n;                          # Número de clientes incluíndo a cooperativa (1).
param t;                          # Número de camións.
param d{i in 1 ..n, j in 1 ..n};  # Distancia entre os clientes.
param c{k in 1 .. t};             # Capacidade dos camións.
param p{i in 1 ..n};              # Pedidos dos clientes.


# Conxuntos

set index_N ordered := {1..((2**(n-1))-1)};
set POW_N {k in index_N} ordered := {i in 2 ..n: (k div 2**(ord(i) - 1)) mod 2 = 1};


# Variables
var x{i in 1 ..n, j in 1 ..n, k in 1 ..t} binary;   # Vale 1 se o camión k vai do cliente i ao j.


# Obxectivo

minimize Ruta: sum{i in 1 ..n, j in 1 ..n, k in 1 ..t}d[i,j]*x[i,j,k];
# Minimizamos a distancia percorrida.
```

```
#Restricións

subject to entrasale {i in 1 ..n, k in 1 ..t}:
    sum{j in 1 ..n: j!=i}x[i,j,k]=sum{j in 1 ..n: j!=i}x[j,i,k];
# Todo camión que chega a un cliente deberá saír do mesmo.

subject to va {i in 2 ..n}:
    sum{j in 1 ..n, k in 1 ..t}x[i,j,k]=1;
# Visitamos todos os clientes.

subject to Subtour {k in 1 ..t, z in index_N: card(POW_N[z]) >= 2}:
    sum{i in POW_N[z], j in POW_N[z]} x[i,j,k] <= card(POW_N[z]) - 1;
# Evitamos subrutas que non conteñan á cooperativa.

subject to capacidad {k in 1 ..t}:
    sum{i in 2 ..n, j in 1 ..n: i!=j}x[i,j,k]*p[i]<=c[k];
# Non poderemos exceder a capacidade dos camións.

subject to entregar:
    sum{i in 2 ..n, j in 1 ..n, k in 1 ..t: i!=j}x[i,j,k]*p[i]-sum{i in 1 ..n}p[i]=0;
# Debemos abastecer as demandas de todos os clientes.
```

Apéndice D

Código da implementación de CW en R

```
# FUNCTION Clarke and Wright
# Inputs:
#       (1) vector.demandas
#       (2) matriz.distancia
#       (3) capacidade.vehiculo
# Outputs:
#       (1) vector.solution
#       (2) cost
CW<-function(vector.demandas,matriz.distancias,capacidade.vehiculo){
  n<-dim(matriz.distancias)[1,] #numero de clientes mais a cooperativa
  c<-numeric(n) #vector custos das rutas
  R<-matrix(0,nrow=n,ncol=3) #matriz de rutas
  iter = 0


  ##########Paso 1: calcular os ciclos iniciales (rutas ir e volver).
  c<-matriz.distancias[1,]*2 #Custo de ir dende a cooperativa ata
                             # o cliente i e volver
  ctotal<-sum(c) #Custo total de ir dende a cooperativa a cada cliente
  R[,2]<-1:n #Xeramos as rutas (0,i,0) onde 0 e a cooperativa
  R[1,2]<-0
```

```
############Paso 2: calcular a matriz de aforros
S<-matrixS(matriz.distancias,n)

###########Paso 3: optimizar rutas
indicar<-1
Sm<-1 #Valores de entrada do primeiro while
while(Sm>0){ #Mentras existan aforros maiores que cero,
             #buscamos rutas factibles
  Sm<-max(S) #Escollemos o valor maximo da matriz de aforros
  if(Sm>0){

    #Coordenadas de Sm en S
    pos = positionSm(S,Sm,n)

    #Demandas dos clientes i e j de Sm
    CargaT<-vector.demandas[pos$Positionfilas]+
      vector.demandas[pos$Positioncolumnas]

    #Indicamos a que cliente visitamos antes de ir a i
    #e despois de ir a j
    if(R[pos$Positionfilas,3]==0 && R[pos$Positioncolumnas,1]==0
       && CargaT<=capacidade.vehiculo){

      newPositionfilas<-pos$Positionfilas
      newPositioncolumnas<-pos$Positioncolumnas
      x<-0 #Evitamos ciclos

      while(R[newPositionfilas,1]!=0){ #Sumamos a carga dos
                                       #clientes anteriores a i
        CargaT<-CargaT+vector.demandas[R[newPositionfilas,1]]
        newPositionfilas<-R[newPositionfilas,1]
        if(newPositionfilas==pos$Positioncolumnas) x<-x+1
      }

      while(R[newPositioncolumnas,3]!=0){ #Sumamos a carga dos
                                          #clientes posteriores a j
```

```
      CargaT<-CargaT+vector.demandas[R[newPositioncolumnas,3]]
      newPositioncolumnas<-R[newPositioncolumnas,3]
      if(newPositioncolumnas==pos$Positionfilas) x<-x+1
    }
    if(CargaT<=capacidade.vehiculo && x==0){ #Engadimos a ruta
                                             #se e factible
      R[pos$Positionfilas,3]<-pos$Positioncolumnas
      R[pos$Positioncolumnas,1]<-pos$Positionfilas
    }


    S[pos$Positionfilas,pos$Positioncolumnas]<-0
    S[pos$Positioncolumnas,pos$Positionfilas]<-0 #Borramos os
                          #aforros utilizados para evitar ciclos
  }

  S[pos$Positionfilas,pos$Positioncolumnas]<-0 #Se non e factible
                                               #tamen o borramos
 }
 iter=iter+1
} #Fin do while

rutas<-numeric() #Vector de rutas FINAL
rutas[1]<-0 #Comezamos na cooperativa
indicador<-2 #Movemonos polo vector de rutas

#Creamos a ruta final
for(i in 2:n){
 if(R[i,1]==0){
   rutas[indicador]<-i
   while(rutas[indicador]!=0){
     rutas[indicador+1]<-R[rutas[indicador],3]
     indicador<-indicador+1
   }
   indicador<-indicador+1
 }
```

```
  }
  rutas[which(rutas==0)]<-1
  custo.total<-0
  for(i in 1:(length(rutas)-1)){
    custo.total<-custo.total+matriz.distancias[rutas[i],rutas[i+1]]
  }
  rutas<-rutas-1
  # Output
  result=list()
  result$rutas = rutas
  result$coste = custo.total
  return(result)
} #Fin da funcion

#positionSm: Calcula a posicion de Sm en S
positionSm<-function(S, Sm, n){

  if(order(S,decreasing=TRUE)[1] %%n==0){
    Positionfilas<-n
    Positioncolumnas<-order(S,decreasing=TRUE)[1] %/% n
  }
  else{
    Positionfilas<-order(S,decreasing=TRUE)[1] %%n
    Positioncolumnas<-order(S,decreasing=TRUE)[1] %/% n + 1
  }

  pos=list()
  pos$Positionfilas = Positionfilas
  pos$Positioncolumnas = Positioncolumnas

  return(pos)
}

#matrixS: Calcula a matriz de aforros
matrixS<-function(matriz.distancias,n){
```

```
  S<-matrix(0,nrow=n,ncol=n) #Matriz de aforros
  for(i in 2:n){
    for(j in 2:n){
      {
        if(i!=j){
          S[i,j]<-matriz.distancias[1,i]+matriz.distancias[1,j]
                  -matriz.distancias[i,j]
        }
      }
    }
  }
  return(S)
}
```