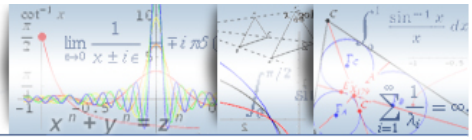
Extensões do Problema do Caixeiro Viajante

André Filipe Maurício de Araújo Oliveira

DEPARTAMENTO DE MATEMÁTICA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA



Extensões do Problema do Caixeiro Viajante

André Filipe Maurício de Araújo Oliveira

Dissertação para a obtenção do Grau de **Mestre em Matemática**

Área de Especialização em **Estatística, Otimização e Matemática Financeira**

Júri

Presidente:	Prof. Dr. João Luís Cardoso Soares
Orientador:	Prof. Dr. José Luis Esteves dos Santos
Vogal:	Prof. Dra. Sílvia Alexandra Alves Barbeiro

Data: 22 Junho de 2015

Resumo

Esta dissertação tem como objetivo o estudo do problema do caixeiro viajante, um problema clássico de otimização combinatória. Apesar da sua simples descrição, o problema do caixeiro viajante é um problema NP-Difícil, não existindo até à data um algoritmo ótimo e eficiente para a sua resolução.

Inicialmente será feito um estudo do problema nas suas variantes simples e múltipla (onde são considerados vários caixeiros). Em seguida será estudado o problema do caixeiro viajante multi-objetivo, também nas suas variantes simples e múltipla. Na otimização multi-objetivo o conceito de solução ótima é substituído pelo conceito de eficiência, deixando de haver um valor ótimo único para o problema. É introduzida a noção de dominância de forma a determinar o conjunto das soluções eficientes sendo este o objetivo na resolução de problemas multi-objetivo.

Para cada um dos problemas abordados, serão referidas diversas variantes e formulações, bem como diversos algoritmos presentes na literatura para a sua resolução. Posteriormente são propostos novos algoritmos com o objetivo de resolver cada um dos problemas abordados. São apresentados resultados computacionais para cada método implementado, de forma a analisar o desempenho dos algoritmos propostos.

Palavras Chave: Problema do Caixeiro Viajante, Otimização Combinatória, Otimização Multi-Objetivo

Abstract

This thesis aims to study the travelling salesman problem, a classical problem of combinatorial optimization. Despite its simple description, the traveling salesman problem is NP-Hard, not existing, so far, an optimum and efficient algorithm to solve it.

We start by studying the problem in its single and multiple variants (where multiple salesman are considered). Furthermore, we will study the multi-objective travelling salesman problem, also in its single and multiple variants. In multi-objective optimization, the concept of optimal solution is replaced by the concept of efficiency, ceasing to be a single optimal value to the problem. We introduce the notion of dominance in order to determine the set of efficient solutions which is the main purpose when solving a multi-objective problem.

For each of the problems addressed, several variants and formulations will be referred, as well as several algorithms in the literature for their resolution. Afterward, new algorithms are proposed with the aim of solving each one of the problems addressed. For each implemented method,

computational results are shown, in order to analyze the performance of the proposed algorithms.

Keywords: Travelling Salesman Problem, Combinatorial Optimization, Multi-Objective Optimization

Agradecimentos

Esta dissertação foi desenvolvida no plano de trabalhos de uma bolsa de investigação para licenciados no contexto do projeto/instituição de I&D Unidade QREN em co-promoção nº 34164 - SmartGeo Portal "Geographic Information Management System", co-financiado pelo QREN, Agência de Inovação, no âmbito do Sistema de Incentivos à Investigação e Desenvolvimento Tecnológico, pelo Programa Operacional Fatores de Competitividade.

Agradeço ao Departamento de Matemática da U.C. pela oportunidade de participação no projeto.

Ao meu orientador, pela enorme disponibilidade ao longo do ano no esclarecimento de dúvidas, troca de ideias e revisão deste trabalho.

Ao Carlos Caçador pelos esclarecimentos e suporte computacional relativos ao projeto.

À minha família e aos meus amigos por todo o apoio.

À Inês pela ajuda e paciência constante.

Conteúdo

Lista de Figuras

Lista de Tabelas

Introdução

O problema do caixeiro viajante (Travelling Salesman/Salesperson Problem - TSP) é um dos problemas mais estudados de otimização combinatória e tem uma enorme variedade de aplicações. A origem do TSP está associada a um caixeiro viajante que pretende passar por um conjunto de cidades uma única vez e por fim voltar à cidade inicial, percorrendo a menor distância possível. Foi formulado matematicamente pela primeira vez em 1932, por Karl Menger [36], mas foi a partir dos anos 50 que começou a ser fortemente estudado. Surgiram diversas propostas para a sua resolução, com especial destaque para Dantzig, Fulkerson e Johnson [18], utilizando técnicas de programação linear pela primeira vez no contexto do TSP. Em 1960, surge o TSP múltiplo, por Miler, Tucker e Zemlin [37] em que são considerados diversos caixeiros viajantes para percorrer todas as cidades. Nos anos 70, a otimização multi-objetivo começou a ganhar maior relevo na otimização combinatória, embora só em 1982 o TSP tenha sido extendido para o caso multi-objetivo, por Fischer e Richter [19]. Nos anos 90 começam a ser resolvidos, até à otimalidade, TSP's com milhares de cidades, sendo em 2008 resolvido o maior TSP até à atualidade, com 85900 cidades [5]. Em 2012, surge o TSP múltiplo multi-objetivo, por Chang e Yen [15] uma variante ainda pouco estudada mas potencialmente muito importante.

Esta dissertação é desenvolvida no plano de trabalhos de uma bolsa de investigação para licenciados no contexto do projeto/instituição de I&D Unidade QREN em co-promoção nº 34164 - SmartGeo Portal "Geographic Information Management System", co-financiado pelo QREN e desenvolvido em co-promoção com a empresa Smartgeo Solutions. O projeto tem como objetivo o desenvolvimento de uma plataforma que permita o cálculo eficiente de rotas em tempo real e tendo em conta diversos critérios em simultâneo. Para tal é necessário o estudo de diversas extensões do TSP e o desenvolvimento de algoritmos para a sua resolução.

Nesse sentido, esta dissertação tem como objetivo o estudo do TSP na sua variante simples e múltipla, prosseguindo para o TSP multi-objetivo. No capítulo 1 será

feita uma introdução ao TSP. O capítulo 2 é dedicado ao TSP múltiplo, em que são considerados diversos caixeiros viajantes. O problema na sua versão multi-objetivo será introduzido no capítulo 3, seguido do TSP múltiplo multi-objetivo no capítulo 4. No final de cada capítulo serão apresentados alguns resultados computacionais relativos a cada um dos problemas em estudo.

Capítulo 1

Problema do Caixeiro Viajante

1.1. Definição e Variantes

O TSP é definido da seguinte forma: dado um conjunto de cidades, qual a melhor forma do caixeiro viajante percorrer todas as cidades, uma única vez e por fim voltar à cidade inicial, percorrendo a menor distância possível? A resposta a esta questão consiste numa ordenação das cidades a percorrer pelo caixeiro. Esta variante é conhecida como o **TSP com retorno** [32]. Caso não seja imposto que o caixeiro volte à cidade inicial, obtemos o **TSP sem retorno** [40].

Uma variante importante do TSP é o **problema do caixeiro viajante múltiplo**, proposto por Miler, Tucker e Zemlin [37], em que são considerados m caixeiros para percorrer todas as cidades. O objetivo é então determinar um conjunto de percursos para os m caixeiros viajantes de forma a percorrer todas as cidades, percorrendo no total a menor distância possível. Esta variante será estudada com detalhe no capítulo 2.

É possível associar a cada cidade um período de tempo no qual esta pode ser visitada, dando origem ao **TSP com janelas temporais (TSP with time windows - TSPTW)** [20]. Este problema pode ser aplicado a situações como planeamento do horário de uma equipa de reparação, em que existem restrições quanto às horas em que poderá atender os seus clientes.

A otimização monocritério é, por vezes, insuficiente para dar resposta às várias exigências que aparecem na vida real. Neste contexto surge a otimização multi-objetivo, onde é possível minimizar, em simultâneo, o custo, a distância e o tempo de viagem, por exemplo. O TSP é então estendido para o **TSP multi-objetivo**, que será estudado com detalhe no capítulo 3.

Associadas a cada variante existem versões do problema, relacionadas com os custos c entre as cidades. O problema diz-se **simétrico** (STSP) se $c(a,b) = c(b,a), \forall a, b$, caso contrário, diz-se **assimétrico** (ATSP). Caso os custos/distâncias entre as cidades cumpram a norma euclidiana, o TSP diz-se **euclidiano**.

Este capítulo terá como objetivo o estudo do TSP com retorno, ou simplesmente TSP, em qualquer uma das suas versões. De uma forma mais genérica, num TSP, referimos-nos às cidades por vértices/nós/pontos e às ligações entre elas por arestas/arcos.

1.1.1. Aplicações

Dada a sua formulação clássica, o TSP é muitas vezes aplicado a problemas de transportes ou de abastecimento de postos. No entanto, também é facilmente aplicável noutras áreas que aparentemente nada têm em comum com a formulação inicial. São descritos em seguida alguns exemplos.

- Cristalografia [11] - No estudo de cristais, são efetuados inúmeros testes com raio X, utilizando diversas intensidades e sob várias perspetivas. O tempo total das experiências pode ser otimizado se a ordem dos procedimentos a efetuar for escolhida de forma adequada. Estas escolhas podem ser obtidas pela resolução do TSP associado ao problema, em que os vértices representam as posições do cristal dentro do dispositivo e as arestas $\{i, j\}$ representam o tempo necessário para mudar da posição i para a posição j.

- Distribuição de encomendas em armazéns [6] - Em armazéns de grandes dimensões, para que a entrada e saída de encomendas decorra de forma eficaz, é necessária uma gestão automatizada da sua distribuição dentro do armazém. A gestão destas encomendas pode ser efetuada através da resolução de problemas do caixeiro viajante. Nestes problemas, os vértices representam os pedidos das encomendas, em que $1 \in V$ representa o pedido a ser realizado no momento, e o arco (i, j) representa a possibilidade de realizar o pedido j a seguir ao pedido i. O custo c_{ij} representa o tempo de viagem da grua sem encomendas, desde o destino da encomenda i à origem da encomenda j. O objetivo é minimizar o tempo total de viagem da grua sem encomendas.

- Sequenciamento de DNA [1] - O TSP pode ser utilizado no mapeamento de cromossomas. Considerando os vértices como as sequências locais e o custo de viagem como uma medida de confiança de que uma sequência local seja imediatamente seguida de outra específica, o TSP é utilizado para reconstruir mapeamentos de radiação híbridos.

- Reposicionamento de Satélites [7] - É utilizado o TSP para otimizar a sequên-

cia de objetos celestiais a serem visionados num programa da NASA. Cada vértice corresponde a uma posição e orientação dos satélites, associadas a uma determinada estrela. Os custos c_{ij} correspondem à quantidade de combustível necessária para o reposicionamento dos dois satélites. O objetivo é minimizar a utilização de combustível necessários à visualização dos objetos celestiais.

Laporte em [32] menciona outras aplicações interessantes do TSP.

1.2. Formulação do Problema

Antes de formular o TSP, comecemos por introduzir alguns conceitos básicos de otimização combinatória.

Definição 1.1 *Um grafo $G = (V, A)$ é um sistema formado pelos conjuntos V e A, em que $V = \{v_1, ..., v_n\}$ é o conjunto dos n vértices/nós do grafo e $A = \{a_1, ..., a_p\} \subset V \times V$ o conjunto das m arestas/arcos do grafo. Quando A é um conjunto de pares ordenados, o grafo diz-se orientado. Caso nenhum dos elementos de A seja um par ordenado, o grafo diz-se não orientado.*

Em grafos não orientados, denotamos por *arestas* os elementos de A, representando cada $a_i \in A$ por $a_i = \{v_j, v_k\}$. Em grafos orientados, denotamos por *arcos* os elementos de A representando cada $a_i \in A$ por $a_i = (v_j, v_k)$. Ao contrário dos arcos, as arestas pode ser percorridas em ambos os sentidos. Uma aresta pode sempre ser representada por dois arcos e consequentemente um grafo não orientado por um grafo orientado. Um grafo diz-se completo se, $\forall v_i, v_j \in V, v_i \neq v_j : (v_i, v_j) \in A$.

Definição 1.2 *O grau de um vértice $v \in V$, $deg(v)$, é dado pelo número de arestas em G a ele ligado. Quando o grafo é orientado, designa-se por grau de entrada, $indeg(v)$, e grau de saída, $outdeg(v)$ ao número de arcos a entrar e sair de v, respetivamente.*

Dado um conjunto de arcos $A' \subset A$, por abuso de linguagem, chamaremos grau de um vértice v ao número de arcos em A' ligados a v.

Definição 1.3 *Um percurso entre dois vértices $v_i, v_j \in V$, é uma sequência de vértices e arcos $< u_1, (u_1, u_2), u_2, ..., u_k >$ verificando:*

(i) $u_i \in V, i \in \{1, ..., k\}$,

(ii) $(u_i, u_{i+1}) \in A, i \in \{1, ..., k-1\}$,

(iii) $u_1 = v_i$ *e* $u_k = v_j$.

Para simplificar, denotamos habitualmente um percurso pela sequência dos vértices $< u_1, u_2, ..., u_k >$. O *comprimento* de um percurso é dado pelo número de arcos contidos nesse percurso.

Definição 1.4 *Chamamos caminho a um percurso em que nenhum vértice (e consequentemente nenhum arco) é percorrido mais do que uma vez exceto, possivelmente, o vértice inicial e terminal que podem coincidir, isto é, se $u_i = u_j$ então $i = j$ ou $i = 1$ e $j = k$. Nesse caso, em que $u_1 = u_k$, temos um ciclo.*

Definição 1.5 *Chamamos ciclo Hamiltoniano a um ciclo que contenha todos os vértices do grafo, isto é, um percurso, de comprimento $|V|$, que passe por todos e cada um dos vértices uma só vez.*

Se considerarmos um grafo G tal que os vértices representam as cidades, os arcos representam as ligações entre elas e a cada arco é associado um custo, resolver o TSP consiste em determinar o menor ciclo Hamiltoniano existente no grafo G.

A primeira formulação do TSP utilizando programação linear inteira foi dada por Dantzig, Fulkerson e Johnson [18] que definiram variáveis binárias x_{ij}, para cada arco (i, j), tais que

$$\begin{cases} x_{ij} = 1 & \text{, se for percorrido o arco da cidade } i \text{ para a cidade } j \\ x_{ij} = 0 & \text{, caso contrário} \end{cases}$$

Considerando c_{ij} a distância (ou o custo) associada ao arco da cidade i para a cidade j, o custo total do percurso é dado por $\sum_{(i,j)\in A} c_{ij}x_{ij}$ (ou $\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij}$, para um grafo completo[1]). Sem perda de generalidade, podemos considerar o grafo como sendo completo (isto é, considerar que $(i, j) \in A, \forall i, j \in V$) acrescentando os arcos em falta com custo infinito. Nesse caso, o problema é formulado da seguinte forma:

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

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

$$\sum_{i=1}^{n} x_{ij} = 1, j = 1, ..., n, \tag{2}$$

$$\sum_{i\in S}\sum_{j\in S} x_{ij} \leq |S| - 1, \forall S \subset V : 1 < |S| < n - 1, \tag{3}$$

$$x_{ij} \in \{0, 1\} \tag{4}$$

[1]Para simplificar a notação, admitiremos a existência de arcos de i para i com custo infinito.

As restrições (1) garantem que do vértice i sai apenas um arco para algum vértice j e as restrições (2) garantem que um vértice j é acedido por apenas um arco vindo de algum vértice i. Temos assim a garantia de que o caixeiro viajante passa por todas as cidades uma e uma só vez. As restrições (3) garantem que a solução não contém vários ciclos disjuntos que cobrem todos os vértices em vez de um único ciclo. De facto, se considerarmos um ciclo de comprimento $m < n$ com o conjunto de vértices $S = \{i_1, ..., i_m\}$, $m < n$, este ciclo tem m cidades, ou seja, $|S| = m$. No entanto, este tipo de soluções não é admissível uma vez que $\sum_{i \in S} \sum_{j \in S} x_{ij} = m > |S| - 1 = m - 1$, o que contradiz as restrições (3). Além disso, as restrições (1) e (2) não permitem ciclos de comprimento 1 ou $n - 1$. Concluímos então que as restrições (3) não permitem que existam ciclos S com menos de n cidades.

As restrições (1) e (2) perfazem um total de $2n$ restrições e as restrições (3) um total de $2^n - 2n - 2$.

Em 1960, Miler, Tucker e Zemlin [37] propuseram uma variante para as restrições (3). Introduzindo as variáveis reais positivas u_i, $i = 2, ..., n$, e as restrições

$$u_i - u_j + nx_{ij} \leq n - 1, i, j = 2, ..., n, i \neq j. \tag{5}$$

Estas restrições não permitem uma solução com mais de um ciclo e perfazem um total de $(n-1)(n-2)$ restrições, um valor bastante inferior às $2^n - 2n - 2$ restrições contidas na condição (3).

1.3. Complexidade e Abordagens

Apesar do seu simples enunciado, o TSP é um problema NP-difícil, o que significa que o problema não pode ser resolvido em tempo polinomial (supondo que $P \neq NP$).

Teorema 1.6 *O Problema do Caixeiro Viajante é NP-difícil.*

Demonstração. Para provar que o TSP é um problema NP-difícil basta provar que é possível reduzir um problema NP-completo no TSP em tempo polinomial. Consideremos o problema do caminho Hamiltoniano, que consiste em, dado um grafo $G = (V, A)$, descobrir se o grafo tem ou não um ciclo Hamiltoniano. Este problema é NP-completo (Garey e Johnson [22]). Consideremos o grafo $G^* = (V^*, A^*)$ tal que $V^* = V$ e $A^* = \{(i, j) : i, j = 1, ..., |V| \wedge i \neq j\}$ e os custos c_{ij} tomam o valor 0 se $(i, j) \in A$ e 1 caso contrário. Então, existe um ciclo Hamiltoniano em G se e só se a solução do TSP associado a G^* for 0. □

O TSP pode ser resolvido através de algoritmos exatos, de forma a obter a solução ótima. No entanto, devido à sua complexidade, é frequente a utilização de heurísticas. Heurísticas são algoritmos que permitem obter boas soluções (não necessariamente ótimas) em tempo consideravelmente inferior. Serão descritos em seguida alguns algoritmos exatos e heurísticas utilizadas para a resolução do TSP.

1.3.1. Algoritmos Exatos

Uma primeira abordagem para o problema do caixeiro viajante consiste no cálculo de todas as possíveis soluções (percursos) e posterior escolha da solução com melhor custo total. Esta abordagem, conhecida na literatura por **método da pesquisa exaustiva** ou **brute force method**, é no entanto pouco interessante pois para um TSP numa rede completa com n cidades, existem $(n-1)!$ (ou $\frac{(n-1)!}{2}$ no caso simétrico) possíveis soluções para o problema. Assim, mesmo para um número reduzido de cidades, este algoritmo deixa de ser exequível em tempo útil.

Uma forma de contornar a complexidade do problema é resolver uma relaxação do mesmo, não sendo consideradas as restrições (3). Esta relaxação, conhecida por *assigment relaxation*, pode ser resolvida em tempo polinomial. Se a solução obtida constituir um só ciclo, então é solução do TSP original. Caso contrário, podemos utilizar um algoritmo do tipo **cutting-planes** e adicionar a restrição do tipo (3) que foi violada ao problema relaxado, tornando a solução atual inválida. O problema é novamente resolvido e este procedimento é repetido até a solução obtida constituir um só ciclo.

Outra abordagem usual para a resolução do TSP é a utilização do método **branch-and-bound**. Este método consiste na divisão do problema em problemas mais pequenos, de forma a obter a solução ótima. O algoritmo utiliza relaxações do problema inicial para obter um *lower bound* (LB) e soluções admissíveis do problema inicial para obter um *upper bound* (UB). Em seguida o problema é dividido em subproblemas disjuntos e são obtidos LB e UB para cada um deles. O UB é atualizado sempre que o UB de algum dos subproblemas é inferior ao UB atual. Isto é feito sequencialmente até ser obtida a solução ótima. Sempre que LB $\geq$ UB, a pesquisa nesse ramo é interrompida. Se LB = UB e a solução encontrada é admissível, temos um candidato a solução ótima. Se LB $>$ UB, não será possível melhorar a solução ótima do problema inicial nesta ramificação.

1.3.2. Heurísticas

As heurísticas mais utilizadas na resolução do TSP dividem-se essencialmente em três grupos: **heurísticas construtivas**, **heurísticas de melhoria iterativa** e **meta-heurísticas**. Como forma de avaliar o comportamento de uma heurística, é por vezes determinado qual o rácio entre a pior solução s que a heurística pode obter e a solução ótima s^*. Utilizaremos a notação $\mathcal{R}(x)$ para indicar que $\frac{s}{s^*} \leq x$. Em seguida serão referidas as vantagens e desvantagens de cada uma destas abordagens, bem como algumas das heurísticas mais presentes na literatura.

Heurísticas Construtivas

Estas são heurísticas simples, em que a construção da solução é construida vértice a vértice (ou aresta a aresta), não precisando de qualquer solução inicial para a sua aplicação.

A primeira destas heurísticas a surgir na literatura é a heurística do **vizinho mais próximo** (Nearest Neighbour - NN), em que, partindo da cidade inicial, a próxima cidade a visitar é, das cidades ainda não visitadas, a cidade mais próxima da cidade atual. No fim é feito o regresso à cidade inicial. A complexidade deste algoritmo é $\mathcal{O}(p)$ em que p é o número total de arcos analisados ($\mathcal{O}(n(n-1)) = \mathcal{O}(n^2)$ para grafos completos). Rosenkrantz, Stearns e Lewis [44] provam que, para um TSP simétrico euclidiano, o NN possui $\mathcal{R}(\frac{1}{2}\lceil\log(n)\rceil + \frac{1}{2})$. No mesmo artigo são propostos **algoritmos de inserção (IA)**, que começam por um caminho constituído por apenas duas cidades e vão inserindo cidades aumentando de forma mínima o custo do caminho (cheapest insertion). Esta heurística, em grafos simétricos euclidianos, possui $\mathcal{R}(2)$, ou seja, produz soluções com custo nunca superior ao dobro do custo da solução ótima, independentemente da dimensão do problema. A sua complexidade é da ordem $\mathcal{O}(n^2 \log(n))$.

Uma abordagem semelhante ao NN é o algoritmo da **ligação mais barata** (cheapest link - CL) em que é adicionada em cada iteração a ligação/aresta mais barata, sem que sejam criados ciclos com menos de n cidades e de forma a que todos os vértices tenham grau inferior ou igual a dois. O algoritmo exige uma ordenação dos arcos A que pode ser feita em $\mathcal{O}(p\log(p))$, em que p é o número de arcos ($\mathcal{O}(n(n-1)\log(n(n-1)) = \mathcal{O}(n^2\log(n))$ para grafos completos). Em seguida analisa os arcos até adicionar n arcos, sem criar ciclos nem vértices com grau superior a 2. Esta análise é de complexidade $\mathcal{O}(p)$ pelo que o algoritmo tem complexidade

$\mathcal{O}(n^2 \log(n))$.

Uma heurística construtiva muito referida na literatura é a **heurística de Christofides (CH)** [16]. Se o TSP for simétrico e euclidiano esta heurística possui $\mathcal{R}(\frac{3}{2})$. O algoritmo começa por calcular a árvore de custo mínimo, T, associada a G. Em seguida, considera o conjunto dos nós de T com vértice ímpar e é obtido o emparelhamento perfeito de custo mínimo M. Desta forma, o conjunto de arcos $R = A(T) \cup M$ permite definir o grafo (V, R), que é constituído por vértices de grau par. Por último, caso todos os vértices tenham grau 2, isto é, $deg(v) = 2, \forall v \in V$, o algoritmo termina; caso contrário, para qualquer vertice v tal que $deg(v) > 2$, $\exists (u,v),(v,w) \in G : R \cup \{(u,w)\} \backslash \{(u,v),(v,w)\}$ permanece um grafo conexo. Como G satisfaz a desigualdade triangular, esta atualização apenas reduz o valor da solução pelo que é efetuada até que todos os vértices tenham grau 2.

Teorema 1.7 *Seja G um grafo não orientado, completo e com custos não negativos a satisfazer a desigualdade triangular. Então a heurística de Christofides produz uma solução com rácio $\mathcal{R}(\frac{3}{2})$.*

Demonstração. Consideremos T uma árvore de custo mínimo de G. Seja $\overline{p}$ a solução ótima do TSP associado a G. Se retirarmos um arco de $\overline{p}$ obtemos uma árvore com custo igual ou superior a T e como os custos de G são não negativos temos que $c(T) \leq c(\overline{p})$. Seja I o conjunto dos vértices de grau ímpar de T. Sabemos que $|I|$ é par. Seja p o caminho que liga os vértices de I pela mesma ordem com que eles aparecem em $\overline{p}$. Como G é euclidiano, qualquer aresta $\{v_i, v_j\}$ de p tem custo inferior ou igual ao sub-caminho em $\overline{p}$ entre v_i e v_j, pelo que $c(p) \leq c(\overline{p})$. Como $|I|$ é par, podemos obter dois emparelhamentos perfeitos disjuntos de p, M_1 e M_2, tais que $c(M_1) \leq c(M_2)$, logo $c(M_1) \leq \frac{1}{2} c(\overline{p})$. A heurística de Christofides obtém o emparelhamento perfeito de custo mínimo M, logo $c(M) \leq c(M_1)$. Em seguida considera o conjunto dos arcos $R = A(T) \cup M$ e, para vértices v com grau superior a 2, ainda substitui arcos (u,v) e (v,w) por (u,w). Como G é euclidiano, esta atualização só melhora o $c(R)$, pelo que, $c(R) \leq c(T) + c(M) \leq \overline{p} + \frac{1}{2}\overline{p} = \frac{3}{2}\overline{p}$.

□

O calculo da árvore de custo mínimo de um grafo por ser feito em $\mathcal{O}(p \log(n))$ ([42], [29]) e o cálculo o emparelhamento perfeito de custo mínimo entre um conjunto de vértices em $\mathcal{O}(n^3)$ ([41]) pelo que CH tem complexidade $\mathcal{O}(n^3)$. Apesar deste algoritmo ser válido apenas para o TSP simétrico, é possível transformar um

problema assimétrico num problema simétrico de maior dimensão. Kumar e Li [30] mostram como fazer esta transformação aumentando o número de nós n para $2n$, possibilitando assim o uso do algoritmo de Christofides.

Heurísticas de Melhoria Iterativa

Estas heurísticas partem de uma solução inicial (fornecida por exemplo por uma heurística construtiva) e efetuam modificações pontuais na tentativa de melhorar a solução. Uma simples heurística deste tipo é a **2-opt**, que funciona para problemas simétricos. Esta heurística analisa todos os pares de arestas não consecutivas $\{u_i, u_j\}$, $\{u_k, u_l\}$ e verifica se a substituição delas pelas arestas $\{u_i, u_k\}$ e $\{u_j, u_l\}$ (sem que a solução deixe de ser conexa) melhora a solução. Este processo é feito até que não hajam alterações na solução.

Figura 1.1: Exemplo de uma iteração do algoritmo 2-opt

Uma extensão da heurística 2-opt é a **heurística de Lin-Kernighan** [28], em que é feita a divisão da solução em duas soluções disjuntas e calculada a melhor forma de unir as duas soluções, removendo e adicionando arestas em cada uma delas. Esta solução é conhecida pelo seu ótimo comportamento empírico, embora não possua nenhum upper bound conhecido relativamente à solução ótima.

Meta-heurísticas

As meta-heurísticas consistem em algoritmos genéricos que podem ser utilizados para resolver diversos problemas de otimização combinatória, em particular o TSP. Não tiram partido de características particulares do problema, mas, também devido à sua complexidade, obtêm em geral boas soluções empíricas.

As heurísticas *tabu search*, introduzida por Glover [25], *simulated annealing*, introduzida por Bonomi e Lutton [13] e algoritmos genéticos, introduzidos por Mühlenbein, Gorges-Schleuter e Krämer [38], são algumas das meta-heurísticas mais abordadas na literatura para este problema.

1.4. Algoritmo de Ligações Prioritárias

Foi sugerida, no trabalho de seminário avançado [39], uma nova heurística construtiva para a resolução do TSP: o **algoritmo de ligações prioritárias** (priority link - PL). Este algoritmo funciona de forma análoga ao CL, adicionando arcos até obter um ciclo hamiltoniano, mas escolhendo, em cada iteração, o arco de maior prioridade. O objetivo passa por evitar a escolha de arcos com custos muito elevados.

Consideremos $O_x = \{(x,k_1),(x,k_2),...,(x,k_p)\}$ o conjunto dos arcos que saem de x e $I_x = \{(l_1,x),(l_2,x),...,(l_q,x)\}$ o conjunto dos arcos que entram em x, ambos ordenados por custos, com $x \in V$. Sejam $DO_i^x = c_{xk_{i+1}} - c_{xk_i}$, com $i \in \{1,...,p-1\}$ e $DO_i^x = \infty$, $i \geq p$, as diferenças de saída de ordem i do vértice x. Analogamente, denotamos $DI_i^x = c_{l_{i+1}x} - c_{l_ix}$, com $i \in \{1,...,q-1\}$ e $DI_i^x = \infty$, $i \geq q$, as diferenças de entrada de ordem i do vértice x. O PL associa, a cada vértice $x \in V$, os vetores $DI^x = (DI_1^x, DI_2^x, ..., DI_n^x)$ e $DO^x = (DO_1^x, DO_2^x, ..., DO_n^x)$, como sendo a prioridade de entrada e saída do vértice x. A comparação entre as prioridades é feita por ordem lexicográfica e é selecionada a maior das prioridades. Seja z o vértice com a maior prioridade. Se esta for uma prioridade de saída, é adicionado ao caminho o arco de menor custo a sair de z, (z,k_1). Se a prioridade for de entrada, é selecionado o arco de menor custo a entrar em z, (l_1,z). Caso o mínimo seja atingido em mais do que um nó, seleciona-se o arco de menor custo associados a esses nós.

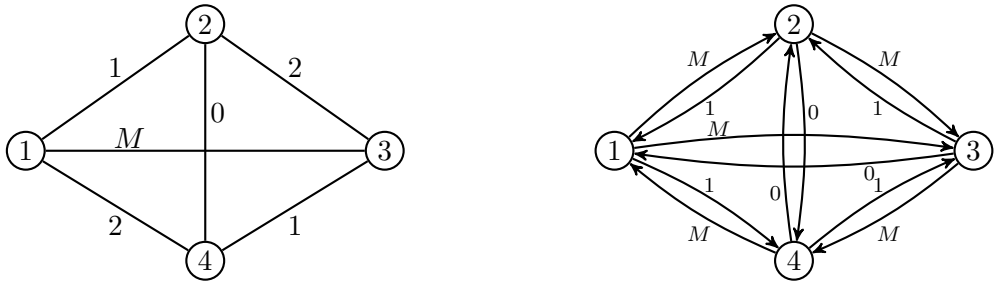
O algoritmo desta heurística pode ser visto em anexo. Para o seu rápido funcionamento, é feita a ordenação dos arcos a sair de cada vértice por ordem de custo, repetindo o mesmo processo para os arcos que chegam a cada vértive. Esta ordenação tem complexidade $\mathcal{O}(k\log(k))$ [2] por cada vértice do grafo, onde k é o número de arcos que sai desse vértice. Para grafos completos, a operação tem complexidade $\mathcal{O}(n(n-1)\log(n-1))$. O cálculo das prioridades de primeira ordem tem complexidade $\mathcal{O}(2n)$ e a sua ordenação é feita numa estrutura *heap* ($\mathcal{O}(\log(6n))$) [3] . O desempate entre cada prioridade é feito em $\mathcal{O}(j \times k)$ passos, em que j é o número de nós com prioridades empatadas e k é o menor número de prioridades associadas a cada uma delas. No pior dos casos temos $\mathcal{O}(2n \times n)$. O algoritmo tem então complexidade $\mathcal{O}(n(n-1)\log(n-1) + n \times (2n + \log(6n) + 2n^2)) = \mathcal{O}(n^3)$. Contudo, este caso é muito improvável pois corresponde a que todas as prioridades de todos os nós em todas as interações sejam iguais.

[2] Ordenação efetuada com a função *sort* da estrutura *list*, em C++.
[3] Utilização da função *make_ heap* da biblioteca *algorithm*, em C++.

1.4.1. Comparação entre heurísticas

O PL resolve qualquer das versões do TSP mencionadas neste trabalho, garantindo uma solução mesmo para grafos assimétricos e não-euclidianos, desde que a rede seja completa. Estudos computacionais mostram que, mesmo em redes não completas, este algoritmo encontra com grande frequência um caminho Hamiltoniano e que o seu valor está próximo do ótimo. O NN, tal como o NN repetitivo, que consiste em aplicar o NN começando por cada um dos vértices e escolhendo a melhor das soluções, garantem igualmente uma solução, podendo no entanto ter um pior comportamento para redes assimétricas e não-euclidianas. O CA precisa que o grafo seja simétrico e, para garantir o rácio $\mathcal{R}(\frac{3}{2})$, precisa que a rede seja euclidiana.

O PL caracteriza-se por escolher menos facilmente arcos maus e por evitar chegar a situações em que não consegue completar o caminho Hamiltoniano. Consideremos os seguintes exemplos, em que M representa um número arbitrariamente grande:



(a) Rede Simétrica (b) Rede Assimétrica

Figura 1.2: Redes não-euclidianas

Na rede 1.2(a) a solução ótima é $p =< 1, 2, 3, 4, 1 >$ com custo 6. No entanto o NN chega à solução $< 1, 2, 4, 3, 1 >$ de custo $2 + M$, independentemente do nó de partida, pelo que esta também será a solução do RNN. O CA começa por fazer a árvore de custo mínimo $T = \{\{1, 2\}, \{2, 4\}, \{4, 3\}\}$ adicionando em seguida a aresta $\{1, 3\}$, obtendo a mesma solução que o RNN. O CL vai escolhendo sempre a aresta mais barata até obter um ciclo Hamiltoniano, logo neste grafo chega à mesma que os algoritmos anteriores. O PL verifica que todas as prioridades de primeira ordem têm valor 1 e calcula as de segunda ordem. Verifica então que a prioridade de entrada (ou saída) do vértice 1 e 3 é $M - 2$, pelo que começa por adicionar a arestas $\{1, 2\}$. O vértice 1 passa a ter prioridade $M - 2$pelo que é adicionada a aresta $\{1, 4\}$. O vértice 2 passa a ser o de maior prioridade, sendo adicionada a aresta $\{2, 3\}$ e por último é adicionada a aresta $\{3, 4\}$, obtendo assim a solução ótima.

Na rede 1.2(b) a solução ótima é $p =< 1, 4, 3, 2, 1 >$ com custo 4 e verificamos

mais uma vez que o PL, ao contrário das outras, chega à solução ótima.

1.4.2. Resultados

Foi realizado um estudo empírico de forma a comparar as heurísticas NN, RNN, CL, PL, CA e IA[4], baseado nos resultados obtidos no seminário avançado. Esse estudo foi alargado para outros tipos de instâncias, tendo sido apresentado no congresso CORS/INFORMS 2015, em Montreal. Relativamente à versão apresentada no seminário, os códigos foram reescritos usando uma nova estrutura que permitiu melhorar os tempos de execução. Apresentamos aqui um breve resumo nas seguintes tabelas. A primeira tabela contém os resultados relativos a problemas simétricos e a segunda relativo a problemas assimétricos. Em cada uma delas é apresentada a média do rácio, $\overline{r}$, entre a solução obtida e a solução ótima relativo a cada instância, bem como o tempo médio de execução, $\overline{t}$, de cada heurística. Os resultados são apresentados na íntegra em anexo.

Heurística	NN	RNN	CL	RCL	PL	CA	IA
$\overline{r}$	1.234	1.152	1.299	1.323	1.203	1.187	1.158
$\overline{t}$	0.039	0.061	0.148	0.131	3.508	0.156	0.334

Tabela 1.1: Resultados em problemas simétricos

Heurística	NN	RNN	CL	RCL	PL	IA
$\overline{r}$	1.358	1.233	1.288	1.191	1.111	1.174
$\overline{t}$	0.015	0.066	0.044	0.039	6.245	0.081

Tabela 1.2: Resultados em problemas assimétricos

Verificamos, tal como no seminário avançado, que o PL é, dos algoritmos implementados, o que produz soluções de melhor qualidade. Relativamente aos resultados apresentados no congresso, foram geradas redes completas e não completas, nas versões simétricas e assimétricas. Concluiu-se que o PL é a heurística com melhores resultados em todas elas. Além disso, para redes não completas, o PL resolve mais de 50% dos problemas e, em redes assimétricas, não completas, produz soluções qualitativamente melhores.

Apesar do seu tempo de execução ser maior do que nos restantes algoritmos, ele ainda é aceitável para problemas de grandes dimensões sendo, por isso, uma boa alternativa como heurística para a resolução do problema do caixeiro viajante.

[4]Os pseudocódigos das heurísticas mencionadas encontram-se no anexo A, secção A.1.

Capítulo 2

Problema do Caixeiro Viajante Múltiplo

2.1. Introdução

O problema do caixeiro viajante múltiplo (multiple traveling salesman problem - mTSP) é uma generalização do problema do caixeiro viajante referido no capítulo anterior. Neste problema são considerados m caixeiros viajantes e o objetivo é determinar m rotas de forma a que os caixeiros viajantes passem por todas as cidades, percorrendo a menor distância possível.

O mTSP foi formulado pela primeira vez em 1960, por Miler, Tucker e Zemlin em [37]. Esta formulação era uma simples generalização do TSP em que o caixeiro viajante teria que passar pela cidade inicial obrigatoriamente m vezes (inclusive o regresso final) e pelas restantes cidades uma única vez. A solução desse problema é constituída por m ciclos, que têm em comum apenas a cidade inicial, correspondentes a m rotas a serem percorridas. O mTSP e as suas versões e variantes foram evoluindo ao longo dos últimos anos, sendo este problema atualmente considerado uma relaxação do problema de roteamento de veículos, em que não são consideradas as restrições de capacidades.

Na literatura são considerados dois tipos de mTSP's: *minsum* mTSP e *minmax* mTSP. Quando o objetivo do problema é minimizar a soma total dos custos dos m caminhos, o problema é referido como *minsum* mTSP [37]. As soluções deste problema podem no entanto não ser úteis em certas aplicações, pois poderão ser obtidas rotas de tamanhos/custos muito díspares. Nesse sentido, pode ser utilizado o *minmax* mTSP ([21]), em que o objetivo é minimizar o caminho mais custoso dos m caminhos. As soluções tendem a ser mais equilibradas, podendo ser aplicadas em situações em que os m caixeiros operam paralelamente.

Neste trabalho, o *minsum* mTSP será simplesmente referido como o mTSP. Outras variantes e versões serão referidas posteriormente.

2.1.1. Aplicações

Uma das maiores aplicações do mTSP é a resolução de problemas relacionados com o cálculo de boas rotas ligadas a problemas de transportes. No entanto, é também muito vulgar a sua utilização em problemas de agendamento de tarefas, armazenamento de produtos ou posicionamento de objetos. São descritas em seguida algumas aplicações referidas na literatura:

- Problema de transportes escolares [3] - Angel, Caudle, Noonan e Whinston aplicam o mTSP para resolver um problema de planeamento de rotas de autocarros escolares, com o objetivo de minimizar a distância total percorrida pelos autocarros e o número de rotas da solução. Neste problema, os vértices representam as paragens por onde os autocarros têm que passar e as arestas representam os caminhos entre essas paragens. O número de autocarros a utilizar é também uma variável do problema.

- Agendamento de entrevistas [24] - O agendamento de entrevistas pode ser tratado como um mTSP com janelas temporais. Neste problema, é necessário planear as entrevistas, de modo a que o conjunto dos compradores (caixeiros viajantes) se reúnam com os proprietários de locais turísticos a serem entrevistados (cidades) em determinados períodos de tempo (janelas temporais).

- Sistema de navegação global por satélite [17] - Estes sistemas permitem a localização geográfica de qualquer local através do uso de satélites artificiais. Para tal, é necessária a colocação de recetores em determinados pontos de referência, que são sujeitos a séries de observações. O problema de achar a melhor ordem para essas sessões decorrerem, dado um conjunto de recetores, pode ser formulado como um mTSP.

- Serviço de segurança noturno [14] - Outro exemplo da aplicação do mTSP com janelas temporais é dado por Calvo e Cardone, que usam o mTSP para agendar os turnos noturnos de um conjunto de guardas, num conjunto de locais a serem vigiados em determinados períodos de tempo.

São descritas por Bektas [8] outras aplicações do problema do caixeiro viajante múltiplo. A enorme variedade das suas aplicações faz do mTSP um problema de grande interesse.

2.2. Variantes e Versões

Tal como o TSP, o mTSP tem diversas variantes que foram surgindo ao longo dos anos. A sua formulação clássica é o **mTSP com armazém único (single depot mTSP)**. Foi inicialmente proposta por Miler, Tucker e Zemlin [37], em que os m caixeiros viajantes iniciam e terminam as suas rotas na mesma cidade (armazém). À exceção do armazém, em cada cidade passa um só caixeiro viajante uma única vez.

Em 1980, Rao [43] generaliza o problema para a variante **mTSP com armazéns múltiplos (multiple depot)**. Nesta variante existem vários armazéns, com um ou vários caixeiros viajantes a partir de cada armazém. No final do percurso pode ser exigido que cada caixeiro viajante volte ao seu armazém (destino fixo) ou pode ser exigido apenas que o caixeiro volte a um armazém (destino não fixo). Nesta última versão é habitualmente exigido que em cada armazém, o número de caixeiros iniciais seja igual ao número de caixeiros finais.

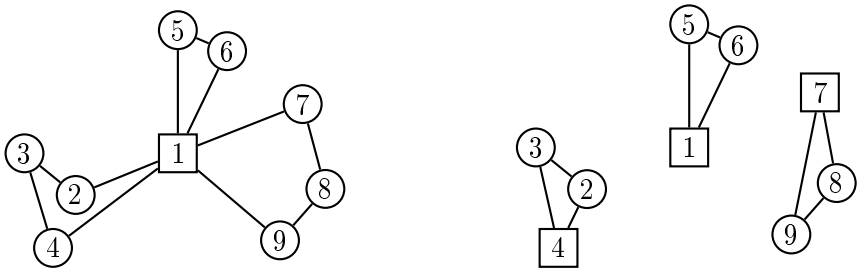


Figura 2.1: Armazém único vs Armazéns múltiplos

No mTSP, é possível que o número de caixeiros m seja **fixo** ou **não fixo**. Em determinadas aplicações é associado um custo a cada caixeiro viajante e não é definido à priori o número de caixeiros viajantes a utilizar. Nesta variante, esse número é mais uma variável do problema e pretende-se minimizar o custo total, considerando o custo do caminho e os custos da utilização de cada caixeiro viajante [33]. Podem também ser adicionadas restrições quanto ao número mínimo e máximo de cidades que cada caixeiro viajante pode visitar. Esta variante é habitualmente designada por **mTSP com limites** [9].

O **mTSP com janelas temporais (mTSP with time windows - mTSPTW)** é uma importante variante do mTSP, com diversas aplicações. Nesta variante, certas cidades têm associadas um período de tempo (janela temporal) no qual podem ser visitadas ou são considerados períodos de tempo em que os caixeiros viajantes podem viajar. Este problema aplica-se a companhias de aviação [45], agendamento

de turnos [14], agendamento de entrevistas [24], etc.

Para cada uma das variantes referidas, existem, tal como no TSP, as versões **euclidiana/não euclidiana** e **simétrica/não simétrica**.

2.3. Formulação do Problema

Seja $G = (V, A)$ o grafo associado ao problema, em que o conjunto de vértices V representa as cidades e o conjunto de arcos A representa os caminhos entre as cidades. Considerando c_{ij} o custo da cidade i para j, o mTSP pode ser formulado como um problema de programação linear inteira. Existem diversas diferenças entre a formulação do mTSP com armazém único e com múltiplos armazéns. Além disso, em qualquer variante podem ser adicionadas restrições associadas por exemplo ao número de vértices a serem visitados por cada caixeiro viajante ou períodos de tempo no qual certas cidades podem ser visitadas.

Serão apresentadas em seguida algumas das formulações presentes na literatura, relativas a cada uma das variantes do problema.

2.3.1. Armazém único

O mTSP na versão single depot é habitualmente formulado de forma análoga ao TSP, diferindo apenas nas restrições associadas ao vértice 1. Como este vértice representa o armazém inicial, haverão m caixeiros viajantes a sair e entrar do mesmo. Considerando as variáveis binárias x_{ij}, tais que

$$\begin{cases} x_{ij} = 1 & \text{, se o arco } (i,j) \text{ pertencer à solução} \\ x_{ij} = 0 & \text{, caso contrário} \end{cases}$$

podemos considerar a seguinte formulação para o mTSP:

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

$$\text{sujeito a} \sum_{j=2}^{n} x_{1j} = \sum_{j=2}^{n} x_{j1} = m, \tag{1}$$

$$\sum_{j=1}^{n} x_{ij} = 1, i = 2, ..., n, \tag{2}$$

$$\sum_{i=1}^{n} x_{ij} = 1, j = 2, ..., n, \tag{3}$$

$$u_i - u_j + nx_{ij} \leq n - 1, i, j = 2, ..., n \tag{4}$$

$$x_{ij} \in \{0, 1\}$$

As restrições (1) garantem que há m arcos a sair e a entrar no vértice 1, correspondentes ao início e fim do caminho de cada um dos m caixeiros viajantes. As restrições (2) e (3) garantem que do vértice i sai e entra um só arco, para $i = 2, ..., n$. As restrições (4) (Miler, Tucker e Zemlin [37]) garantem que a solução é constituída por exatamente m caminhos, um para cada caixeiro.

Em 1980, Laporte e Nobert [33] propõem uma formulação em que o número de caixeiros m é também uma variável do problema. Para este efeito, associaram um custo fixo f a cada caixeiro viajante e consideraram a seguinte função objetivo:

$$\text{minimizar} \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} + fm.$$

Mais recentemente, Bektas e Kara [9] propõem uma formulação com restrições relativas ao número mínimo e máximo de vértices a serem percorridos por cada caixeiro viajante. Para tal, são associadas a cada caixeiro viajante as variáveis u_i, que podem representar, por exemplo, o número de vértices visitados no caminho do caixeiro, desde o início até ao vértice i. Sejam K e L o número mínimo e máximo de vértices que cada caixeiro viajante pode percorrer, com $2 \leq K \leq L$. Temos que $1 \leq u_i \leq L$, $i \geq 2$ e se $x_{i1} = 1$, as desigualdades $K \leq u_i \leq L$ têm que ser satisfeitas. Bektas e Kara proposeram a adição das restrições

$$u_i + (L-2)x_{1i} - x_{i1} \leq L - 1, i = 2, ..., n \tag{5}$$

$$u_i + x_{1i} + (2-K)x_{i1} \geq 2, i = 2, ..., n \tag{6}$$

$$x_{1i} + x_{i1} \leq 1 \tag{7}$$

e a substituição das restrições (4) por

$$u_i - u_j + Lx_{ij} + (L-2)x_{ji} \leq L - 1, 2 \leq i \neq j \leq n \tag{8}$$

As proposições seguintes mostram que estas restrições asseguram uma solução para o mTSP com limites.

Proposição 1 *As restrições* (5) *e* (6) *garantem que o número de cidades visitada por cada caixeiro varia entre* K *e* L.

Demonstração. Pelas restrições (7), não é permitido que $x_{1i} = x_{i1} = 1$, $i = 2, ..., n$, ou seja, não existem caminhos a partir do armazém e a visitar apenas uma cidade.

É assim verificada a restrição $K \geq 2$. Basta então analisar as seguintes situações:

- i **é o segundo vértice do caminho**: neste caso temos que $x_{1i} = 1$ e $x_{i1} = 0$. Das restrições (5) sai que $u_i \leq 1$ e das restrições (6) que $u_i \geq 1$ o que implica que $u_i = 1$.
- i **é o penúltimo vértice do caminho**: neste caso temos que $x_{1i} = 0$ e $x_{i1} = 1$. Das restrições (5) sai que $u_i \leq L$ e das restrições (6) que $u_i \geq K$.
- i **não é segundo nem penúltimo vértice do caminho**: neste caso temos que $x_{1i} = 0$ e $x_{i1} = 0$. Das restrições (5) sai que $u_i \leq L - 1$ e das restrições (6) que $u_i \geq 2$.

□

Proposição 2 *As restrições* (8) *asseguram que não há subcaminhos na solução.*

Demonstração. Consideremos as restrições de Miler, Tucker e Zemlin

$$u_i - u_j + nx_{ij} \leq n - 1, i, j = 2, ..., n.$$

Substituindo n por L, pretendemos adicionar o termo αx_{ji}, o qual só fará diferença quando $x_{ji} = x_{ij} = 1$. Nesse caso, temos que

$$u_i - u_j + L + (L - 2) \leq L - 1 \Leftrightarrow u_j \geq L - 1 + u_i$$

Pelo teorema anterior, sabemos que $u_j \leq L, \forall j$ pelo que a condição só é válida para $u_i = 1$. Mas se $u_i = 1$ então $x_{1i} = 1$ logo x_{ji} não pode ser 1 pelas restrições (3).

□

2.3.2. Armazéns múltiplos

O mTSP com armazéns múltiplos (multidepot mTSP - MmTSP) é uma generalização do mTSP com armazém único. Neste problema, os caixeiros viajantes partem de vários armazéns e o objetivo é achar o melhor conjunto de caminhos, um para cada caixeiro, minimizando o total de distância percorrida. No fim, cada caixeiro viajante tem que voltar ao seu armazém (**destino fixo** [34]) ou a um armazém arbitrário, mas de forma a que cada armazém tenha no fim o mesmo número de caixeiros do que inicialmente (**destino não fixo** [26]).

Para formular o MmTSP *com destino não fixo*, basta considerar dois subconjuntos disjuntos de V, o conjunto dos armazéns, D, e o conjunto das cidades a percorrer,

V'. Temos obviamente que $V = D \cup V'$. Seja $m_i, i \in D$, o número de caixeiros que partem do armazém i. O problema é formulado da seguinte forma:

$$\begin{aligned}
\text{minimizar } & \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \\
\text{sujeito a } & \sum_{j\in V'} x_{ij} = m_i, i \in D && (9)\\
& \sum_{j\in V'} x_{ij} = m_j, j \in D && (10)\\
& \sum_{i\in V} x_{ij} = 1, j \in V' && (11)\\
& \sum_{j\in V} x_{ij} = 1, i \in V' && (12)\\
& \sum_{i\in S}\sum_{j\in S} x_{ij} \leq |S| - 1, \forall S \subset V' : 1 \leq |S| < n-1 && (13)\\
& x_{ij} \in \{0,1\}
\end{aligned}$$

As restrições (9) e (10) garantem que partem e chegam exatamente m_i caixeiros ao armazém i. As restrições (11) e (12) garantem que por cada cidade passa um só caixeiro viajante, uma única vez. As restrições (13) garantem que a solução é composta exatamente por m caminhos, um para cada caixeiro viajante.

Relativamente ao MmTSP com destino fixo, não existem até então formulações simples desta versão como um problema linear inteiro. No entanto, Kara e Bektas [9] apresentam uma transformação da formulação anterior para o MmTSP com destino fixo, através da inserção de variáveis binárias x_{ijk}, descrevendo se o caixeiro k passa ou não pelo arco (i, j) e fazendo os respetivos ajustes.

2.3.3. *minmax* mTSP

No *minmax* mTSP, na versão single depot, pretendemos minimizar o pior dos m caminhos obtidos. Para tal, consideremos as variáveis x_{ijl}, associadas ao caixeiro viajante l, tais que

$$\begin{cases} x_{ijl} = 1 & \text{, se o caixeiro viajante } l \text{ percorrer o arco } (i,j) \\ x_{ijl} = 0 & \text{, caso contrário} \end{cases}$$

A formulação do *minmax* mTSP é então dada por

$$\begin{aligned}
\text{minimizar} \quad & \max_{l\in\{1,\ldots,m\}} \{\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ijl}\} \\
\text{sujeito a} \quad & \sum_{l=1}^{m}\sum_{j=2}^{n} x_{1jl} = \sum_{l=1}^{m}\sum_{j=2}^{n} x_{j1l} = m, && (14)\\
& \sum_{l=1}^{m}\sum_{j=1}^{n} x_{ijl} = 1, i = 2,...,n, && (15)\\
& \sum_{l=1}^{m}\sum_{i=1}^{n} x_{ijl} = 1, j = 2,...,n, && (16)\\
& \sum_{l=1}^{m}\sum_{i\in S}\sum_{j\notin S} x_{ijl} \geq 1, \forall S \subset V && (17)\\
& x_{ijl} \in \{0,1\}
\end{aligned}$$

As restrições 14 garantem que cada caixeiro parte e regressa ao armazém. As restrições (15) e (16) garantem que cada caixeiro passa por cada cidade uma única vez, à exceção do armazém. As restrições (17) garantem que não há dois sub-caminhos disjuntos na solução, associados a um determinado caixeiro. Desta forma temos a garantia de que a solução contém m ciclos, um para cada um dos caixeiros viajantes.

2.4. Métodos de Resolução

De forma análoga ao TSP, o mTSP é um problema NP-difícil pelo que a sua resolução é em geral de grande dificuldade. Existem duas principais formas de resolver o mTSP: **transformar o mTSP num TSP** ou resolver diretamente por meio de **algoritmos exatos** ou **heurísticas**. Serão descritas em seguida cada uma destas abordagens, vantagens e desvantagens e exemplos de algoritmos habitualmente utilizados.

2.4.1. Transformações do mTSP para o TSP

Uma abordagem muito utilizada para a resolução do problema do caixeiro viajante múltiplo é a sua transformação num problema do caixeiro viajante simples. Essa transformação consiste, em geral, num aumento da dimensão do problema, adicionando vértices e arcos ao grafo $G(V, A)$, criando assim um novo grafo $G(V', A')$. Apesar do aumento da dimensão do grafo, esta abordagem permite o uso dos algoritmos/heurísticas do TSP para a resolução do mTSP. Obtida a solução do TSP, é feita a sua transformação para a solução do mTSP, resolvendo assim o problema.

Uma das primeiras transformações do mTSP single depot deve-se a Huckfeldt e Svestka [27], que aumentaram a matriz de custos associada ao problema, adicionando $m-1$ linhas e colunas, cada uma delas cópias da primeira linha e coluna. Em seguida resolveram o TSP através de algoritmos do tipo cutting-planes. Sempre que era violada alguma restrição relativa à quantidade de ciclos da solução, era necessário proceder a alterações da matriz de custo de forma a que a restrição não fosse violada.

Em 1974, Bellmore e Hong [10] propõem uma transformação do *minsum* mTSP com armazém único num TSP, adicionando $m-1$ vértices e alterando a matriz de custos de acordo com o custo de cada caixeiro viajante. Nesta transformação é necessário fixar o número máximo m de caixeiros viajantes a usar embora a solução não utilize necessariamente todos. Esta transformação é particularmente interessante devido à sua simplicidade e eficácia. Consideremos C_i, $i = 0, ..., m-1$, o custo do caixeiro i, tal que os custos C_i estão ordenados da seguinte forma:

$$C_0 \leq C_1 \leq ... \leq C_{m-1}.$$

Seja $G = (V, A)$ o grafo associado ao problema original, em que V é constituído por N vértices, numerados de 0 a $N-1$ e o vértice 0 representa o armazém. O conjunto V' é obtido adicionando $m-1$ vértices, designados por $-1, -2, ..., -(m-1)$. O conjunto A' contém todos os arcos de A, contém os arcos $(-i, j)$ e $(j, -i)$ sempre que existem os arcos $(0, j)$ e $(j, 0)$ em A e contém os arcos $(-i, -(i-1))$, para $i = 1, ..., m-1$. As distâncias d' são dadas por:

$$\begin{cases} d'(i,j) = d(i,j) & , \quad \forall i,j : (i,j) \in A \\ d'(-i,j) = d(0,j) + \frac{1}{2}C_i & , \quad \forall j : (0,j) \in A, i = 0,1,...,(m-1) \\ d'(j,-i) = d(j,0) + \frac{1}{2}C_i & , \quad \forall j : (j,0) \in A, i = 0,1,...,(m-1) \\ d'(-i,-(i-1)) = \frac{1}{2}C_{i-1} - \frac{1}{2}C_i & , \quad i = 1,2,...,(m-1) \end{cases}$$

Os teoremas seguintes mostram que a solução ótima do TSP transformado corresponde à solução ótima do mTSP original.

Teorema 2.1 *Para qualquer solução S do mTSP associado a $G(V, A)$ contendo o caixeiro 0, existe uma solução S' no TSP associado a $G'(V', A')$ de igual custo.*

Demonstração. A solução S é formada por $r \leq m$ ciclos, $Z_{i_1}, ..., Z_{i_r}$, um para cada caixeiro viajante utilizado. O custo da solução é dado por

$$Z = \sum_{k=1}^{r} Z_{i_k} + \sum_{k=1}^{r} C_{i_k}.$$

Admita-se que $i_{r+1} = m$ e consideremos a solução S' construida pelos seguintes passos:

(i) $Z'_0 = Z_0 \backslash \{(i,0)\} \cup \{(i,-(m-1))\}$;

(ii) $< -(i_{k+1}-1), ..., -i_k > \cup Z'_{i_k}$,
em que $Z'_{i_k} = Z_{i_k} \backslash \{(0,i),(j,0)\} \cup \{(-i_k,i),(j,-(i_k-1))\}, k = r, ..., 1$;

(iii) $< -(i_1-1), ..., 0 >$.

O passo (i) acrescenta $\frac{1}{2}C_0 + \frac{1}{2}C_{i_{r+1}-1}$ ao custo da solução. O passo (ii) acrescenta $\frac{1}{2}C_{i_k} - \frac{1}{2}C_{i_{k+1}-1}$ pelo custo do trajeto $< -(i_{k+1}-1), ..., -i_k >$ e $\frac{1}{2}C_{i_k} + \frac{1}{2}C_{i_k-1}$ pela alteração de Z_{i_k} por Z'_{i_k}, para $k = r, ..., 1$. O último passo, acrescenta $\frac{1}{2}C_0 - \frac{1}{2}C_{i_1-1}$ ao custo da solução. Sendo assim, o custo de S' é dado por

$$\sum_{k=1}^{r} Z_{i_k} + \frac{1}{2}C_0 + \frac{1}{2}C_{i_{r+1}-1} + \frac{1}{2}\sum_{k=1}^{r}(\frac{1}{2}C_{i_k} + \frac{1}{2}C_{i_k-1} + \frac{1}{2}C_{i_k} - \frac{1}{2}C_{i_{k+1}-1}) + \frac{1}{2}C_0 - \frac{1}{2}C_{i_1-1} =$$

$$= \sum_{k=1}^{r} Z_{i_k} + \sum_{k=1}^{r} C_{i_k} + \frac{1}{2}\sum_{k=1}^{r+1} C_{i_k-1} - \frac{1}{2}\sum_{k=1}^{r+1} C_{i_k-1} = \sum_{k=1}^{r} Z_{i_k} + \sum_{k=1}^{r} C_{i_k}$$

□

Teorema 2.2 *Para qualquer solução S' do TSP associado a $G'(V', A')$, existe uma solução S no mTSP associado a $G(V, A)$, contendo o caixeiro 0, de igual custo.*

Demonstração. Consideremos o conjunto V' de vértices com índices $-i, i = 0, ..., (m-1)$, presentes na solução S' que são seguidos por um nó de índice positivo. Temos que que $|V'| = r$, com $1 \leq r \leq m$. Substituindo os nós $-i$ na solução S' pelo nó 0, obtemos uma solução S do mTSP constituída por r ciclos com início e fim em 0. Pela construção anterior, facilmente se prova que o custo da solução é dado por

$$\sum_{k=1}^{r} Z_{i_k} + \sum_{k=i_1}^{i_r} C_k,$$

igual ao custo da solução S'.

□

Teorema 2.3 *Se 0 for o caixeiro de custo mínimo, então o mTSP associado a $G(V, A)$ e o TSP associado a $G'(V', A')$ têm a mesma solução ótima.*

Demonstração. Como na solução ótima do mTSP se vai utilizar os r caixeiros mais baratos e $C_0 \leq C_1 \leq ... \leq C_{m-1}$, então o caixeiro 0 será utilizado na solução ótima S. Pelo teorema 2.1 sabemos que $\exists S^* \in G' : c(S^*) = c(S)$. Suponhamos que esta não é solução ótima do TSP associado a $G'(V', A')$. Então, $\exists S' \in G' : c(S') < c(S^*)$. Pelo teorema 2.2, temos que existe $\exists S'' \in G : c(S'') = c(S') \Rightarrow c(S'') < S$, o que contradiz a hipótese. □

Concluímos desta forma que, resolvendo o problema transformado, conseguimos obter a solução ótima do problema original. Ilustremos esta transformação com a seguinte rede simétrica não-euclidiana com 5 nós:

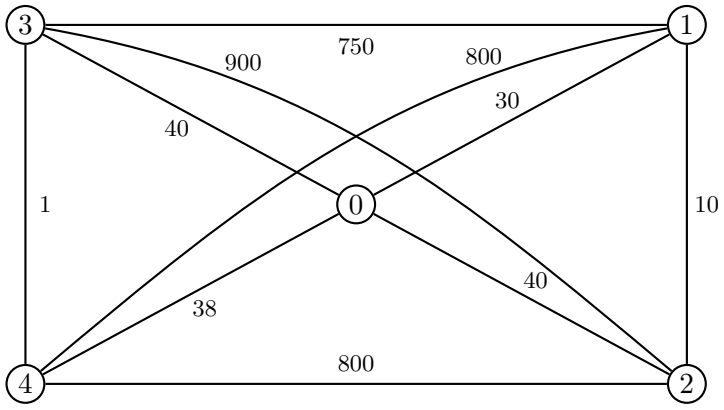


Figura 2.2: Rede inicial

Se pretendermos resolver o mTSP com 2 caixeiros, de custos $C_0 = 20$ e $C_1 = 30$, basta apenas adicionar um vértice à rede. A rede ficaria com a seguinte forma:

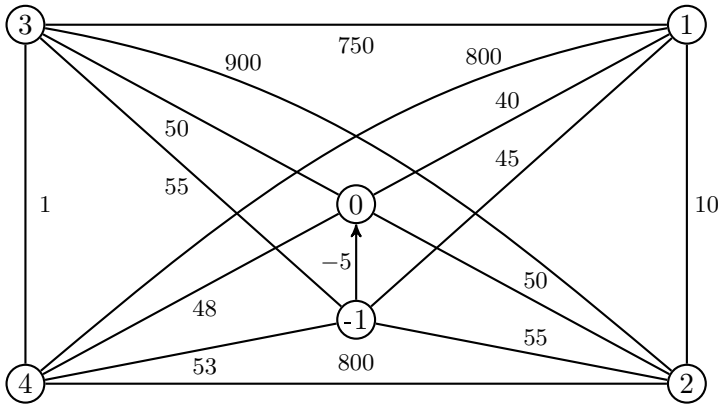


Figura 2.3: Rede transformada

A nova rede é assimétrica e não completa, apesar da rede original ser simétrica. Note-se que isto acontece sempre, mesmo que a rede original seja simétrica e completa, devido às ligações entre os nós $-i, i = 0, 1, 2, ..., (m-1)$. Resolvendo o novo problema, é obtida a solução $(0, 1, 2, -1, 4, 3, 0)$. A solução do caixeiro viajante múltiplo correspondente é dada por $S = \{(0, 1, 2, 0), (0, 4, 3, 0)\}$. Se o problema original fosse resolvido para $m = 3$, uma possível solução seria $(0, 1, 2, -2, -1, 4, 3, 0)$. Neste caso, interpreta-se que o 3º caixeiro (representado pelo nó -2) não efetuará qualquer percurso e a solução é novamente $S = \{(0, 1, 2, 0),\ (0, 4, 3, 0)\}$.

2.4.2. Algoritmos Exatos

Apesar das vantagens práticas que a transformação para o TSP pode trazer à resolução de um mTSP, ela traz também algumas desvantagens, podendo não ser a melhor forma de resolver o problema de forma ótima. O aumento da rede é por vezes significativo. Além disso, em algumas transformações poderão surgir várias soluções ótimas correspondentes à mesma solução, o que dificulta a sua resolução. Neste sentido, pode ser preferível a utilização direta de algoritmos exatos ou heurísticas para a resolução do mTSP.

Os algoritmos exatos utilizados para a resolução do mTSP são semelhantes ao usados na resolução do TSP. Em 1980, Laporte e Nobert [33] foram os primeiros a resolver mTSP's através de algoritmos exatos, utilizando um algoritmo do tipo **cutting-planes**. Através da relaxação inicial de restrições do tipo (4) e adicionando restrições sempre que necessário, resolveram problemas com até 100 cidades. Em 1986, Ali e Kernnington [2], propõem um algoritmo **branch-and-bound**, utilizando a relaxação Lagrangeana para obter limites inferiores para o valor ótimo. No mesmo ano, Gavish e Srikanth [23] propõem um algoritmo semelhante mas relaxando restrições relacionadas com o grau dos vértices. Neste artigo apresentam resultados para problemas com até 500 cidades e 10 caixeiros viajantes.

Devido à dificuldade do mTSP, resolver problemas de grande dimensão através de algoritmos exatos pode ser demasiado custoso em termos de tempo de execução. Surgem mais uma vez de forma natural as heurísticas, que obtêm soluções aproximadas em menos tempo.

2.4.3. Heurísticas

As heurísticas mais populares para a resolução do mTSP são as meta-heurísticas. As mesmas meta-heurísticas que resolviam o TSP resolvem também o mTSP, pois

tratam-se de algoritmos genéricos, facilmente adaptáveis a vários problemas de otimização combinatória. No entanto, estas precisam sempre de uma solução inicial, além de não tirarem partido de algumas características do problema.

As heurísticas construtivas são em geral boas heurísticas a considerar, tanto pela sua rapidez de execução, como pela independência que têm ao não precisarem de qualquer solução inicial.

Uma forma de resolver o mTSP com armazéns múltiplos é aplicar um **algoritmo de duas fases**. Na primeira fase é efetuado um algoritmo de *clustering*, de forma a dividir a rede em m zonas. Na segunda fase são resolvidos m TSP's, um para cada zona, obtendo assim os caminhos disjuntos para os m caixeiros viajantes. Sofge, Schultz e De Jong utilizam em [47] esta estratégia para resolver mTSP's euclidianos, utilizando na segunda fase algoritmos genéticos.

Heurísticas de inserção são também facilmente aplicáveis ao mTSP. Considerando um mTSP com m caixeiros viajantes, o algoritmo começa por criar m caminhos $(1, k_i, 1), i = 1, ..., m$. Os pontos $k_1, ..., k_m$ podem ser escolhidos seguindo vários critérios: aleatoriamente, os pontos mais afastados, os pontos mais próximos, etc. Em seguida vão sendo adicionados nós aos caminhos de acordo com o critério a otimizar. Se o problema for um *minsum* mTSP, é usado o **cheapest insertion**, que escolhe, de todos os caminhos e todos os nós não visitados, qual a melhor insersão de um nó em qualquer caminho. Se o problema for um *minmax* mTSP, é usado o ***insertion by node*** ou ***insertion by cost***. A heurística *insertion by node* insere sequencialmente um nó em cada caminho, de forma a minimizar a diferença de número de nós entre quaisquer caminhos. A heurística *insertion by cost* insere um nó sempre no caminho de menor custo, de forma a minimizar a diferença de custos entre quaiaquer caminhos.

2.5. Resolução do mTSP

Foi realizado um estudo computacional com o objetivo de estudar algumas heurísticas para a resolução do mTSP. Nesta secção serão descritas as estratégias implementadas e apresentados os respectivos resultados.

2.5.1. Algoritmo de Bellmore e Hong

O primeiro algoritmo implementado consiste em transformar o mTSP num TSP, segundo a transformação de Bellmore e Hong e resolver o TSP transformado com as

heurísticas referidas no capítulo 1. Posteriormente é feita a transformação da solução do problema transformado para o problema original, obtendo a solução do mTSP single depot. O algoritmo encontra-se, com detalhe, em anexo.

Foram comparados os resultados obtidos segundo cada uma das heurísticas para problemas entre 2 a 5 caixeiros viajantes. Como a transformação do grafo origina sempre um grafo assimétrico e não completo, não é garantido que as heurísticas obtenham sempre solução. Além disso, devido à assimetria, o algoritmo de Christofides não foi considerado.

Inicialmente foram resolvidos os problemas simétricos da TSPLIB. Numa primeira fase, foi atribuído a cada caixeiro um custo 0. A tabela seguinte mostra a percentagem de problemas resolvidos e a média do rácio entre a solução de cada heurística e a solução mínima obtida, em cada problema ($\frac{s'-min\{s\}}{min\{s\}}$), para cada m.

Heurística	$m=2$		$m=3$		$m=4$		$m=5$	
NN	0.098	100%	0.10	70.27%	0.15	52.70%	0.19	37.84%
RNN	0.034	100%	0.04	100%	0.05	100%	0.05	100%
CL	0.22	100%	0.19	100%	0.26	100%	0.24	100%
RCL	0.23	100%	0.20	100%	0.26	100%	0.24	100%
PL	0.05	100%	0.04	100%	0.10	100%	0.03	100%

Tabela 2.1: Resultados do mTSP por transformação no TSP

Observamos que a transformação do grafo e posterior utilização do NN pode levar a que não seja obtida a solução do problema, mesmo sendo o grafo original completo, simétrico e euclidiano. Além disso, a heurística IA não conseguiu resolver qualquer dos problemas, razão pela qual não foi apresentada. Relativamente aos rácios, observamos que o RNN e o PL foram as heurísticas que mais consistentemente produziram boas soluções, enquanto que as heurísticas CL e RCL obtiveram piores soluções. Na tabela seguinte podemos observar o rácio entre as médias dos valores das soluções obtidas e as médias da solução ótima no problema com um caiseiro $\frac{s}{s^*}$. Observamos também o rácio entre a diferença do custo do pior caminho e o custo do melhor caminho, com o custo da solução.

Os valores indicam que o PL utiliza sempre apenas um caixeiro viajante. Isto deve-se ao facto do custo do arco $(-i, 0)$ ser 0 o que faz com que a prioridade de saida do nó $-i$ tenha tendência a ser elevada. As heurísticas CL e RCL têm um comportamento semelhante. Uma forma de evitar que isto aconteça é adicionar restrições para que cada caixeiro percorra pelo menos uma cidade.

Os resultados mostram, como já era esperado, que a resolução do mTSP na versão

	$m=2$		$m=3$		$m=4$		$m=5$	
Heurística	$\frac{s}{s^*}$	$\frac{c_M-c_m}{s}$	$\frac{s}{s^*}$	$\frac{c_M-c_m}{s}$	$\frac{s}{s^*}$	$\frac{c_M-c_m}{s}$	$\frac{s}{s^*}$	$\frac{c_M-c_m}{s}$
NN	1.24	0.53	1.29	0.66	1.34	0.4	1.43	0.33
RNN	1.18	0.68	1.2	0.62	1.31	0.59	1.31	0.63
CL	1.32	0.96	1.33	0.98	1.41	0.99	1.41	0.99
RCL	1.38	0.97	1.39	0.98	1.48	0.99	1.48	0.99
PL	1.25	1	1.26	1	1.35	1	1.35	1

Tabela 2.2: Rácios relativos à solução e aos caminhos

minsum pode levar a que os custos dos m caminhos sejam muito díspares. Como forma de contornar este problema, vejamos em seguida algoritmos de inserção para a resolução do *minmax* mTSP na versão single depot.

2.5.2. Algoritmos de Inserção

Consideremos o mTSP single depot, cujo armazém corresponde ao vértice v_a. Comecemos por considerar os m vértices mais afastados do armazém, $v_i, i = 1, ..., m$. Em seguida, são criados m caminhos da forma $p_i =< v_a, v_i, v_a >, i = 1, ..., m$. Resolver o minmax TSP consiste em achar o conjunto das m rotas tais que a rota de maior custo tenha o menor custo possível. Uma heurística de inserção que pode ser utilizada consiste em adicionar sempre o vértice seguinte à rota de menor custo. O vértice inserido é aquele com o menor aumento do custo do caminho. Este algoritmo tem como objetivo em cada iteração manter o equilíbrio entre as rotas.

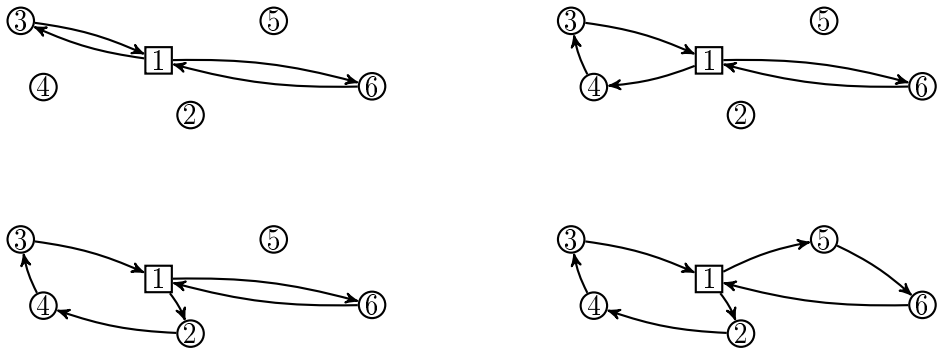


Figura 2.4: Algoritmo de inserção por custo

Outra abordagem consiste em ir adicionando vértices de forma a que os caminhos tenham sempre o mesmo número de vértices, ou nunca mais do que um vértice de diferença. Escolhendo o caminho onde o próximo vértice vai ser inserido, o vértice é escolhido de forma análoga ao caso do TSP.

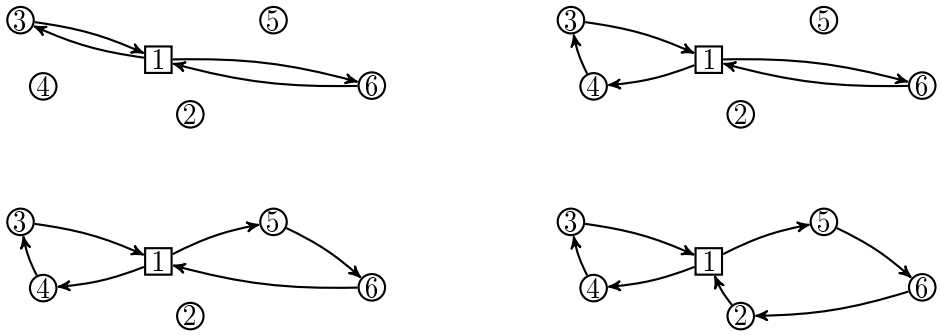


Figura 2.5: Algoritmo de inserção por custo

Por último foi considerado um algoritmo de inserção para resolver o *minsum* mTSP single depot, em que, após a criação dos m caminhos, são adicionados vértices de forma a que o custo total da solução aumente o mínimo possível.

Foram aplicados os três algoritmos aos grafos da TSPLIB simétricos, de forma análoga ao algoritmo anterior. Os resultados encontram-se de forma resumida na tabela seguinte. São apresentados o rácio entre as médias dos valores das soluções obtidas e as médias da solução ótima no problema com um caiseiro $\frac{s}{s^*}$ e o rácio entre a diferença do custo do pior caminho e o custo do melhor caminho, com o custo da solução. Nas duas últimas colunas são reproduzidos novamente os resultados das melhores heurísticas anteriores.

m		By Node	By Cost	Minimum	RNN	PL
2	$\frac{s}{s^*}$	1.061	1.015	0.954	1.178	1.252
	$\frac{c_M - c_m}{s}$	0.042	0.003	0.06	0.682	1
3	$\frac{s}{s^*}$	1.411	1.314	1.196	1.204	1.258
	$\frac{c_M - c_m}{s}$	0.031	0.002	0.166	0.624	1
4	$\frac{s}{s^*}$	1.707	1.614	1.409	1.305	1.347
	$\frac{c_M - c_m}{s}$	0.01	0.005	0.033	0.59	1
5	$\frac{s}{s^*}$	2.018	1.888	1.627	1.308	1.346
	$\frac{c_M - c_m}{s}$	0.005	0.007	0.055	0.63	1

Tabela 2.3: Custos da solução e pior caminho

As soluções obtidas para a versão *minmax* são, como seria de esperar, mais equilibradas do que as obtidas para a versão *minsum* com a transformação. Curiosamente, verifica-se também que as soluções tem um custo global mais reduzido quando $m = 2$, sendo as soluções da inserção mínima melhores em média do que a solução ótima do TSP original. Observamos também que a inserção por custo produz soluções mais equilibradas e de melhor custo global do que a inserção por nó. A inserção mínima, em que em cada iteração adiciona o nó de forma a aumentar o mínimo possível a solução global, produz soluções menos equilibradas, devido à sua natureza *minsum*. Este estudo sugere que estas heurísticas de inserção constituem boas alternativas para a resolução do problema do caixeiro viajante múltiplo.

Capítulo 3

Problema do Caixeiro Viajante Multi-objetivo

3.1. Otimização Multi-objetivo

A otimização multi-objetivo surgiu nos anos 70 e desde então têm havido enormes desenvolvimentos nesta área. É frequente no dia a dia o aparecimento de problemas em que se pretende otimizar vários objetivos em simultâneo, para o qual a otimização mono critério não consegue fornecer boas respostas. No TSP podemos querer minimizar a distância, o tempo de viagem e o seu custo total. No entanto, minimizar a distância pode implicar um aumento do custo (no uso de auto-estradas, por exemplo) e minimizar o custo pode implicar um aumento do tempo de viagem. Neste sentido surge a otimização multi-objetivo, pretendendo dar respostas a problemas deste género.

De forma a ilustrar a utilidade da otimização multi-objetivo, vejamos um exemplo de um problema de decisão.

Exemplo 3.1 *O José pretende comprar um apartamento T3 barato e que se situe próximo do centro de Coimbra, onde ele trabalha. Depois de uma longa procura, o José tem as seguintes opções a considerar:*

Casa	1	2	3	4	5	6	7	8	9	10	11	12
Distância ao centro (km)	23	38	16	12	40	70	35	30	40	16	50	36
Custo (milhares €)	192	110	255	270	185	70	140	170	110	220	90	113

Tabela 3.1: Opções para a compra de casa

Quais as casas que o José deve descartar de imediato? Qual a casa que deve escolher?

A secção seguinte introduz os conceitos necessários para responder da melhor forma possível a estas questões.

3.1.1. Definições

Um problema de otimização multi-objetivo (multiobjective optimization problem - MOP) é constituído por várias funções objetivo, que pretende otimizar em simultâneo. A sua formulação matemática é a seguinte:

$$\begin{aligned} &\max/\min \ \ f_1(x) \\ &\qquad ... \\ &\max/\min \ \ f_k(x) \\ &\qquad \text{s. a} \quad x \in X \subset \mathbb{R}^n \end{aligned}$$

onde $f_i(x) : \mathbb{R}^n \to \mathbb{R}$ é a i-ésima função objetivo, $k \geq 2$ o número de objetivos a otimizar, n a dimensão das variáveis a otimizar e X a região admissível do problema. Definimos $Y = \{f(x) : x \in X\} \subset \mathbb{R}^k$ como sendo a imagem de X pela função objetivo f. A um elemento $x \in X$ chamamos solução admissível e $f(x) = (f_1(x), ..., f_k(x))^T$ o seu vetor de valores objetivos. Consideramos de agora em diante, sem perda de generalidade, que se pretende minimizar cada uma das funções objetivo.

Ao contrário da otimização mono critério, na otimização multi critério o valor ótimo do problema não é, em geral, único. Quando existem vários critérios a serem otimizados em simultâneo, é habitual surgirem objetivos conflituosos, sendo raro o aparecimento de uma solução melhor do que todas as outras em todos os critérios. Nesse sentido, é necessário definir, das soluções possíveis, quais as soluções importantes e quais as que se podem descartar. Surge a definição de *dominância.*

Definição 3.1 *Sejam $z, w \in Y$ dois vetores de dimensão k. Dizemos que z domina w, se e só se*

(i) $\forall i \in \{1, ..., k\} : z_i \leq w_i$,

(ii) $\exists j \in \{1, ..., k\} : z_j < w_j$.

Denotamos esta relação de dominância por $z \prec w$.

Definição 3.2 *Seja X um conjunto de soluções e $x \in X$. Dizemos que x é uma solução eficiente (ou ótimo de Pareto) em X se e só se*

$$\nexists y \in X : f(y) \prec f(x).$$

A imagem de uma solução eficiente x pela função objetivo f é denominada por ponto não dominado.

Definição 3.3 *O conjunto $X_E \subseteq X$ de todas as soluções eficientes de X designa-se por conjunto eficiente e a sua imagem, $Y_E \in Y$, designa-se por conjunto ou frente de Pareto.*

A resolução de um MOP consiste, idealmente, em achar o conjunto eficiente X_E, obtendo assim as soluções pretendidas. Uma heurística pretende achar um conjunto $\hat{X}_E$ o mais próximo possível de X_E. Esse conjunto deve ser em geral diversificado, de forma a constituir uma boa representação de X_E. Além disso, devido à complexidade dos MOP, a heurística deve ser o mais eficiente possível.

Voltemos ao exemplo 3.1. Representemos por x_i a casa i e por c_i o seu vetor de custos, neste caso (distância, custo), com $i = 1, ..., 12$. O gráfico seguinte representa os custos das casas em questão:

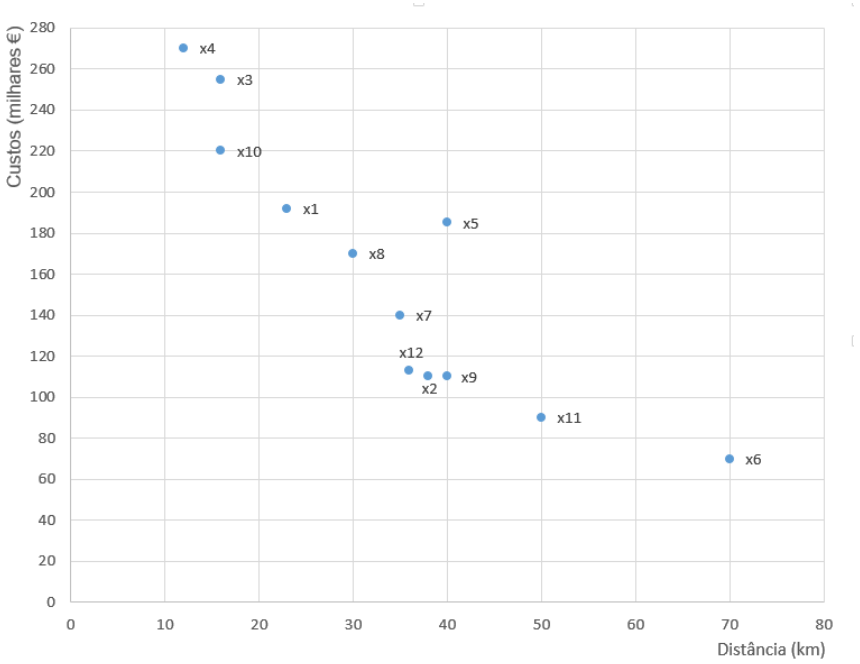


Figura 3.1: Custos das casas x_i apresentados na tabela 3.1

Pretendemos saber quais as soluções eficientes no conjunto das 12 casas. É fácil verificar as seguintes dominâncias:

- $c_{10} \prec c_3$ pois $16 = 16$ e $220 < 255$;
- $c_2 \prec c_9$ pois $38 < 40$ e $110 = 110$;
- $c_7 \prec c_5$ pois têm todos custos inferiores aos de c_5, para qualquer um dos dois critérios.

Sabemos assim que o José pode descartar de imediato as casas x_3, x_5 e x_9, pois são soluções não eficientes. Quanto às restantes casas, nenhum dos seus vetores de custos é dominado. Por exemplo $c_1 \not\prec c_2$ porque $23 < 38$ mas $192 > 110$. Assim sendo, as casas x_1, x_2, x_4, x_6, x_7, x_8, x_{10}, x_{11} e x_{12} são soluções eficientes e o José pode escolher comprar qualquer uma delas. Dentro destas soluções, se ele preferir a mais barata, comprará a casa 6. Se por outro lado quiser ficar o mais próximo possível do trabalho comprará a casa 4. Deste modo o José tem uma clara perceção de quanto irá piorar num dos critérios se quiser melhorar o outro.

3.1.2. Métodos de resolução clássicos

Devido à grande dificuldade associada a estes problemas e ao aparecimento de um número muito elevado de soluções, muitos autores optam por utilizar uma função utilidade real, que permita estabelecer uma relação de ordem total entre as soluções. Desta forma, é possível resolver problemas multi objetivo recorrendo a métodos exatos.

Uma das abordagens habituais consiste em atribuir a cada objetivo um peso λ_i e considera-se como função objetivo a função

$$minf(x) = \sum_{i=1}^{k} \lambda_i f_i(x)$$

em que $0 \leq \lambda_i \leq 1, i = 1, ..., k$ e $\sum_{i=1}^{k} \lambda_i = 1$. Note-se que, dado um conjunto de $\lambda_i, i = 1, ..., k$, a solução x obtida é uma solução eficiente do problema original. Caso contrário, existiria uma solução x' tal que

$$x' \prec x \Rightarrow \sum_{i=1}^{k} \lambda_i f_i(x') \leq \sum_{i=1}^{k} \lambda_i f_i(x),$$

o que seria uma contradição.

Outra relação de ordem entre soluções consiste em minimizar os objetivos por ordem lexicográfica. A solução obtida é mais uma vez eficiente, embora habitualmente seja uma solução extrema. No exemplo anterior, se o José optasse por minimizar em primeiro lugar a distância e em segundo lugar o custo da casa, optaria de imediato pela 4ª casa, que é a mais próximado seu local de trabalho. No entanto é também a casa mais cara, o que pode não ser conveniente.

Neste trabalho não serão utilizadas tais estratégias uma vez que limitam a natureza multi-objetivo do problema e retiram ao utilizador o papel principal da tomada

da decisão. Os objetivos terão igual importância, de forma a descartar apenas pontos dominados, sem que um critério tenha algum privilégio sobre outro, com o objetivo de obter o conjunto X_E. No final serão consideradas estratégias para agrupar as soluções em diversos conjuntos de forma a facilitar o processo de decisão ao utilizador.

3.1.3. Propriedades de dominância

A relação de dominância não é uma relação de ordem total. Consideremos $w, z \in \mathbb{R}^k$ tais que $\exists i, j : w_i < z_i \wedge w_j > z_j$. Então $w \nprec z$ e $z \nprec w$, ou seja, w e z não estão relacionados. A relação de dominância é no entanto uma relação de ordem parcial estrita que goza de certas propriedades. São apresentadas em seguida algumas delas.

Teorema 3.4 *A relação de dominância é uma relação de ordem parcial estrita.*

Demonstração. Para que $\prec$ seja uma relação de ordem parcial estrita basta que seja transitiva e antissimética:

(i) **transitividade**: Sejam $v, w, u \in \mathbb{R}^k$ tais que $v \prec w$ e $w \prec u$. Então $\forall i \in \{1, ..., k\}, v_i \leq w_i \wedge v_i \leq u_i \Rightarrow v_i \leq u_i$. Além disso, como v domina w, $\exists j \in \{1, ..., k\} : v_j < w_j \leq u_j \Rightarrow v_j < u_j$. Então $v \prec w \wedge w \prec u \Rightarrow v \prec u$;

(ii) **antissimetria**: Sejam $v, w \in \mathbb{R}^k$ tais que $v \prec w$. Então $\exists i \in \{1, ..., k\} : w_i > v_i \Rightarrow w \nprec v$.

□

Propriedade 1 *Sejam $v, w, u \in \mathbb{R}^k$ e $m \in \mathbb{R}^+$. A relação de dominância goza das seguintes propriedades:*

(i) $v \prec w \Leftrightarrow v + u \prec w + u$;

(ii) $v \prec w \Leftrightarrow mv \prec mw$;

Demonstração. Basta ter em conta que

(i) Como $v_i \leq w_i \Leftrightarrow v_i + u_i \leq w_i + u_i$ e $v_i < w_i \Leftrightarrow v_i + u_i < w_i + u_i$, então, pela definição de "$\prec$", tem-se de imediato $v \prec w \Leftrightarrow v + u \prec w + u$;

(ii) De forma análoga, $v_i \leq w_i \Leftrightarrow mv_i \leq mw_i$ e $v_i < w_i \Leftrightarrow mv_i < mw_i$, pois $m > 0$, logo $v \prec w \Leftrightarrow mv \prec mw$;

□

3.2. TSP no contexto Multi-objetivo

No TSP multi-objetivo (multiobjective traveling salesman problem - MOTSP), a cada aresta $(i, j) \in A$ do grafo G associado ao problema, está associado não um custo mas um vetor de custos $c_{ij} = (c_{ij}^1, ..., c_{ij}^k)$. O custo de uma solução do MOTSP é dada pelo vetor de custos $s = (s_1, ..., s_k)$, formado pela soma dos custos das suas arestas. Considerando as variáveis binárias x_{ij} de forma análoga ao TSP, o MOTSP tem a seguinte formulação

$$
\begin{aligned}
\text{minimizar } & f_1(x) = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}^1 x_{ij} \\
& ... \\
\text{minimizar } & f_k(x) = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}^k x_{ij} \\
\text{sujeito a } & \sum_{j=1}^{n} x_{ij} = 1, i = 1, ..., n, \\
& \sum_{i=1}^{n} x_{ij} = 1, j = 1, ..., n, \\
& u_i - u_j + n x_{ij} \leq n - 1, i, j = 2, ..., n, i \neq j \\
& x_{ij} \in \{0, 1\}
\end{aligned}
$$

As k funções objetivo, correspondem aos k critérios da rede a otimizar. As restrições não sofrem qualquer alteração relativamente ao TSP.

Resolver o MOTSP consiste em obter o conjunto de soluções (ciclos hamiltonianos) eficientes X_E. No entanto, isso pode levar a um esforço computacional de tal ordem elevado, que é muito habitual na literatura a sua resolução por meio de heurísticas, determinando um conjunto $\hat{X}_E$ que represente da melhor forma possível X_E. Metaheurísticas e heurísticas de melhoria iterativa estão entre as mais populares para a resolução do MOTSP. Lust e Teghem em [35] fazem um resumo minucioso das abordagens até então consideradas, com particular destaque para algoritmos evolutivos e algoritmos de pesquisa local de Pareto.

Para exemplificar, consideremos a seguinte rede assimétrica não euclidiana, completa com $|V| = 5$ e $k = 3$:

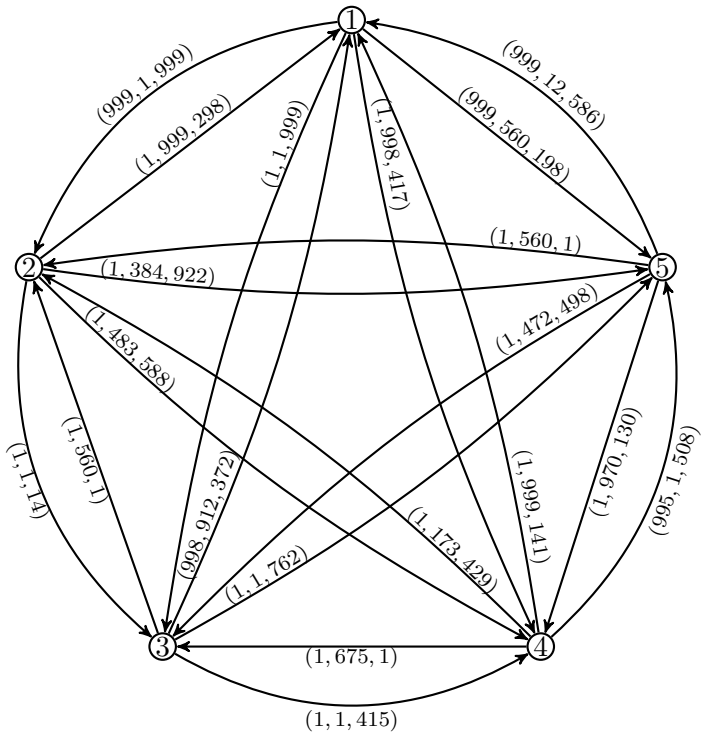


Figura 3.2: Rede completa com 3 critérios

Uma solução deste problema seria o caminho $s_1 =< 1, 2, 3, 4, 5, 1 >$, com custo $c_{s_1} = (999, 1, 999)+(1, 1, 14)+(1, 1, 415)+(995, 1, 508)+(999, 12, 586) = (2995, 16, 2522)$. Consideremos também $s_2 =< 1, 3, 2, 5, 4, 1 >$, com custo $c_{s_2} = (5, 2914, 2193)$ e $s_3 =< 1, 2, 5, 4, 3, 1 >$, com custo $c_{s_3} = (2000, 2942, 2424)$. Verificamos que $c_{s_2} \prec c_{s_3}$, ou seja, s_3 não é uma solução eficiente. Das restantes, s_1 é melhor solução para o segundo critério enquanto que s_2 é melhor para o primeiro e terceiro. Concluímos então que $c_{s_1} \not\prec c_{s_2}$ e $c_{s_2} \not\prec c_{s_1}$. No entanto não temos certezas se $s_1 \in X_E$ ou $s_2 \in X_E$. Na secção seguinte veremos como podemos abordar este problema e algumas formas de o simplificar, obtendo soluções aproximadas.

3.3. Resolução por rótulos

A determinação da frente de Pareto de um MOTSP é um problema bastante difícil. Uma forma de construir o conjunto X_E é a construção de todos os caminhos possíveis e posterior comparação dos seus custos, de forma a verificar quais as soluções eficientes do problema. Nesta secção serão exploradas estratégias para a construção, de forma ótima, do conjunto X_E e de forma aproximada do conjunto $\hat{X}_E$.

Suponhamos que a construção das soluções é feita vértice a vértice, começando

por um vértice inicial v_{ini} arbitrário. A partir de v_{ini} são considerados os seus vértices vizinhos v_i e, para cada um, é criado um rótulo com o percurso $p_i =< v_{ini}, (v_{ini}, v_i), v_i >$. Em seguida, seleciona-se um desses rótulos, e o correspondente vértice final, repetindo-se o processo até que todo os rótulos sejam explorados. Isto é feito sucessivamente até serem obtidos todos os caminhos. No final, dentro das soluções X encontradas, são escolhidas as soluções eficientes, formando assim o conjunto X_E.

Esta abordagem, o **algoritmo de pesquisa exaustiva**, tal como já foi visto para o TSP, é bastante custosa pois para um problema com n nós existem $(n-1)!$ soluções. Além disso, o processo da escolha das soluções eficientes pode implicar a comparação de todas as soluções contra todas, sendo necessárias $\mathcal{O}(k((n-1)!)^2)$ comparações. No entanto existem estratégias que podem tornar esta abordagem mais eficiente.

3.3.1. Algoritmo de Rotulação

No algoritmo anterior, muitos dos ramos da pesquisa podem ser ignorados se assegurarmos que esse ramo não irá gerar uma "boa" solução. Isso pode ser facilmente conseguido se o problema verificar o Princípio de Otimalidade que pode ser enunciado da seguinte forma:

Princípio de Otimalidade: Toda a solução ótima é formada por sub-soluções ótimas.

No contexto da otimização multi-objetivo, o conceito de "óptimo" é entendido como "eficiente". Assim, se restringirmos as comparações a sub-soluções com o mesmo subconjunto de vértices e com o mesmo nó inicial e terminal, obtemos o seguinte resultado.

Teorema (Princípio de Otimalidade) *Se os custos da rede forem não negativos, o MOTSP verifica o Princípio de Otimalidade.*

Demonstração. Sejam $p_1 =< v_i, ..., v_j >$ e $p_2 =< u_i, ..., u_j >$ dois caminhos com o mesmo subconjunto de nós, tal que $v_i = u_i$,$v_j = u_j$ e $c(p_1) \prec c(p_2)$. Suponhamos que existe uma solução eficiente s que contém p_2, isto é, $s =< q, p_2, q' >$. Defina-se $s' =< q, p_1, q' >$ e provemos que s' é uma solução do TSP que domina s. Para tal, basta verificar que

$$c(s') = c(s' \backslash (q \cup q')) + c(p_2) \prec c(s \backslash (q \cup q')) + c(p_1) = c(s),$$

logo s não é uma solução eficiente.

□

Assim sendo, no algoritmo anterior, sempre que são obtidos dois caminhos $p_1 =< v_1, ..., v_j >$ e $p_2 =< u_1, ..., u_j >$ com o mesmo conjunto de nós, tais que $v_1 = u_1$ e $v_j = u_j$ diz-se que p_1 e p_2 são *comparáveis*. Se $c(p_1) \prec c(p_2)$ ou $c(p_2) \prec c(p_1)$, podemos descartar de imediato o caminho dominado, eliminando o rótulo correspondente. Caso não haja dominâncias, nenhum dos caminhos pode ser descartado.

O algoritmo descrito é denominado por algoritmo de rotulação e o esquema seguinte ilustra o seu funcionamento, aplicado no problema da figura 3.2.

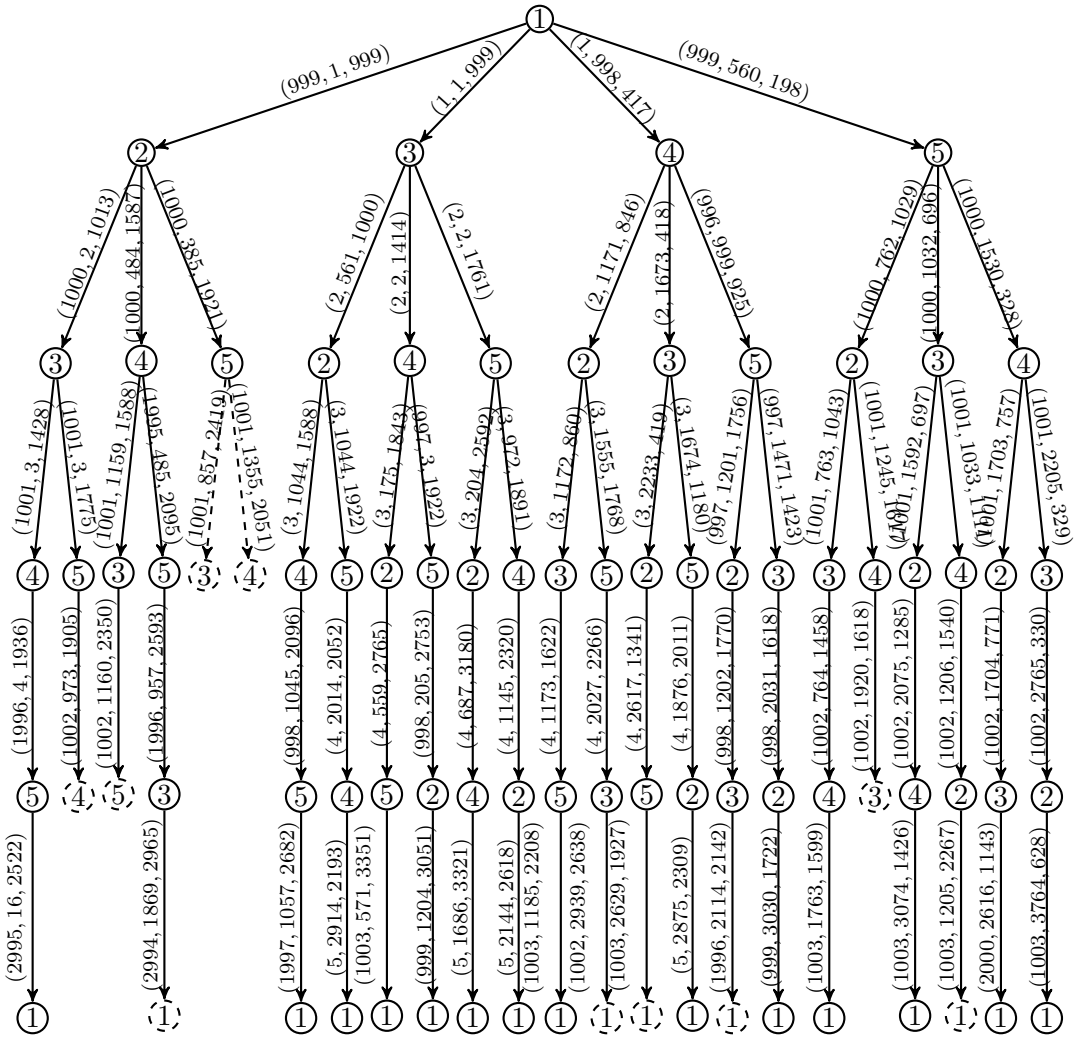


Figura 3.3: Algoritmo de rotulação

Ao fim da terceira iteração do algoritmo é possível comparar alguns dos caminhos obtidos até então. Desta forma, verificamos que o caminho $< 1, 2, 5, 3 >$, de custo $(1001, 857, 2419)$, é dominado pelo caminho $< 1, 5, 2, 3 >$, de custo $(1001, 763, 1043)$

e o caminho $< 1, 2, 5, 4 >$, de custo $(1001, 1355, 2051)$ é dominado pelo caminho $< 1, 5, 2, 4 >$, de custo $(1001, 1245, 1617)$. De forma análoga são dominados os restantes caminhos a tracejado. Concluímos assim que, das 24 soluções deste problema, existem 14 soluções eficientes.

Este algoritmo permite obter todo conjunto X_E. No entanto a sua complexidade é claramente exponencial, mesmo tendo em conta o Princípio da Otimalidade na sua resolução. É assim previsível que apenas funcione para problemas de dimensão consideravelmente baixa. Após a sua implementação[1], o algoritmo foi testado em 20 redes completas com 5, 10 e 15 nós. A tabela seguinte apresenta a média dos tempos de execução do algoritmo e do número de soluções eficientes obtidas, para problemas com 2 critérios e 3 critérios:

k	5		10		15	
	$\overline{t}$	$\overline{n}$	$\overline{t}$	$\overline{n}$	$\overline{t}$	$\overline{n}$
2	0.002	3.1	0.329	14.8	118.025	57.55
3	0.003	6.6	1.612	147.2	2589.906	717.1

Tabela 3.2: Médias dos tempos de execução e nº de soluções eficientes

A diferença entre o número médio de soluções eficientes quando são tidos em conta 2 ou 3 critérios é bastante significativa. Verificamos que para um grafo completo de dimensão 15 com 3 critérios, o algoritmo se torna bastante custoso. Para grafos completos de dimensão 20 o problema já se torna praticamente intratável.

3.3.2. Vizinho mais Próximo Multiobjetivo

De forma análoga ao TSP e mTSP, uma forma de contornar a dificuldade da resolução do MOTSP é recorrer a heurísticas, obtendo não o conjunto X_E mas uma apróximação $\hat{X}_E$. Consideremos o algoritmo do **vizinho mais próximo multiobjetivo** (multiobjective nearest neighbour - MONN), como sendo a extensão do NN referido no capítulo 1. No MOTSP, o caixeiro viajante escolheria, não a cidade vizinha mais próxima, mas sim o conjunto de cidades vizinhas cujos custos sejam não dominados. A junção deste critério de dominância ao algoritmo anterior aumenta o conjunto de soluções dominadas de forma considerável.

Aplicando o MONN no problema anterior, são obtidas 11 soluções. O esquema do algoritmo pode ser visto na figura seguinte. É fácil constatar que, para este problema, $\hat{X}_E \subset X_E$. No entanto o MONN não garante esta propriedade, mesmo quando algumas das soluções de X_E pertençam a $\hat{X}_E$

[1] O pseudocódigo do algoritmo encontra-se no anexo A, secção A.3.

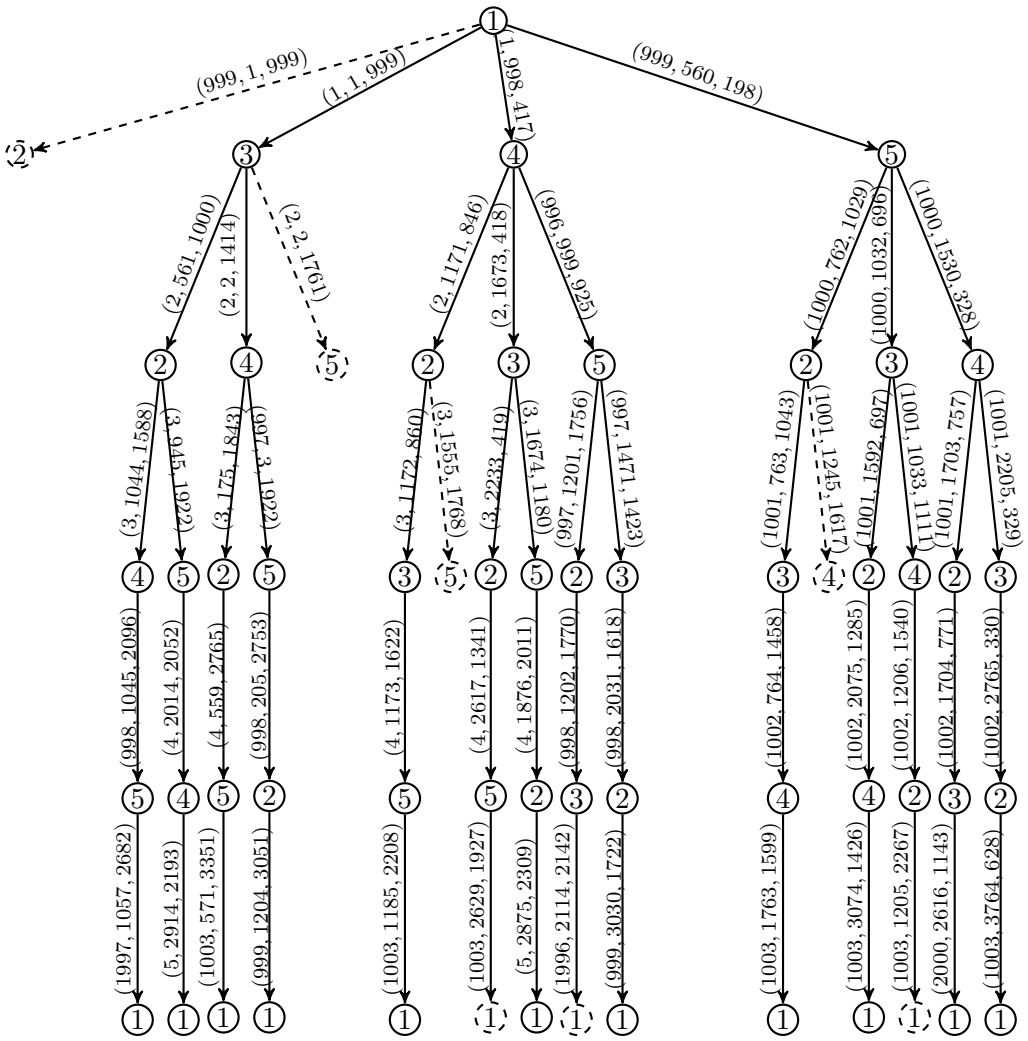


Figura 3.4: MONN por rótulos

Foi testado o MOTSP para redes com 5, 10, 15 e 20 nós. A tabela seguinte apresenta, de forma análoga à anterior, a média dos tempos de execução do algoritmo para problemas com 2 critérios e 3 critérios:

k	5		10		15		20	
	$\overline{t}$	$\overline{n}$	$\overline{t}$	$\overline{n}$	$\overline{t}$	$\overline{n}$	$\overline{t}$	$\overline{n}$
2	0.003	1.75	0.020	7.4	2.99	26.9	766.69	857.10
3	0.004	4.05	0.348	98.45	497.236	512	Inexequível	

Tabela 3.3: Médias dos tempos de execução e nº de soluções eficientes

Utilizando o novo critério de dominância, observamos uma diminuição significativa nos tempos de execução, sendo possível resolver problemas de 2 objetivos com 20 cidades.

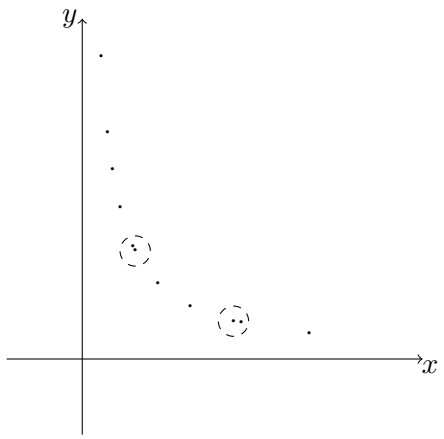
3.3.3. Algoritmo de eliminação por vizinhança

Um dos problemas do algoritmo anterior é a quantidade de rótulos que são guardados durante o processo de construção das soluções. Isto porque, sempre que dois caminhos comparáveis não se dominam, é necessário guardar os rótulos relativos a cada um, o que acontece com muita frequência. Isto pode dar origem a muitas soluções semelhantes.

Uma forma de reduzir o número de rótulos criados, seria, dados dois caminhos comparáveis p_a e p_b, verificar se um deles está contido numa vizinhança do outro. Para tal é considerada a distância euclidiana

$$d(p_1, p_2) = \sqrt{(c_1(p_1) - c_1(p_2))^2 + ... + (c_k(p_1) - c_k(p_2))^2}$$

e dois caminhos são considerados vizinhos se $d(p_1, p_2) < n_n m_A \alpha$, em que m_A representa a soma das médias dos custos dos arcos de G, n_n o número de nós adicionados até ao momento e α um valor para controlar o tamanho da vizinhança. Se dois caminhos se encontram na mesma vizinhança, então apenas um deles é escolhido, mesmo que não haja dominância. A escolha recai sobre a solução de menor norma, ou seja, a solução mais proxima da solução nula. Desta forma obtemos o **algoritmo de eliminação por vizinhança (AEV)**.



Foi feito um estudo análogo para redes com 5, 10, 15 e 20 nós. A tabela seguinte apresenta os resultados anteriores e os resultados para o algoritmo de pesquisa por vizinhança. Para $k = 2$ foi utilizado $\alpha = 0.1$ e para $k = 3$ foi utilizado $\alpha = 0.2$.

	$\|V\| = 5$		$\|V\| = 10$		$\|V\| = 15$		$\|V\| = 20$	
BF	$\overline{t}$	$\overline{n}$	$\overline{t}$	$\overline{n}$	$\overline{t}$	$\overline{n}$	$\overline{t}$	$\overline{n}$
$k = 2$	0.002	3.1	0.329	14.8	118.025	57.55	Inexequível	
$k = 3$	0.003	6.6	1.612	147.2	1294.953	717.1	Inexequível	
MONN	$\overline{t}$	$\overline{n}$	$\overline{t}$	$\overline{n}$	$\overline{t}$	$\overline{n}$	$\overline{t}$	$\overline{n}$
$k = 2$	0.003	1.75	0.020	7.4	2.99	26.9	733.35	52.35
$k = 3$	0.004	4.05	0.348	98.45	497.236	512	Inexequível	
AE	$\overline{t}$	$\overline{n}$	$\overline{t}$	$\overline{n}$	$\overline{t}$	$\overline{n}$	$\overline{t}$	$\overline{n}$
$k = 2$	0.002	1.75	0.016	5	1.502	10.45	198.574	13.15
$k = 3$	0.002	3.85	0.064	12.7	6.201	22.05	515.790	34.5

Tabela 3.4: Médias dos tempos de execução e nº de soluções eficientes

Observamos uma descida bastante considerável dos tempos de execução, sendo possível resolver problemas com 20 nós, mesmo com 3 critérios. O aumento de α implica, como era de esperar, uma diminuição nos tempos de execução e no número de soluções finais.

Para ilustrar a diferença entre o algoritmo de eliminação por vizinhança e o algoritmo ótimo, foi considerada uma rede com 10 cidades e 3 critérios. As duas figuras seguintes mostram os custos das soluções obtidas para cada um dos algoritmos, com $\alpha = 0.1$.

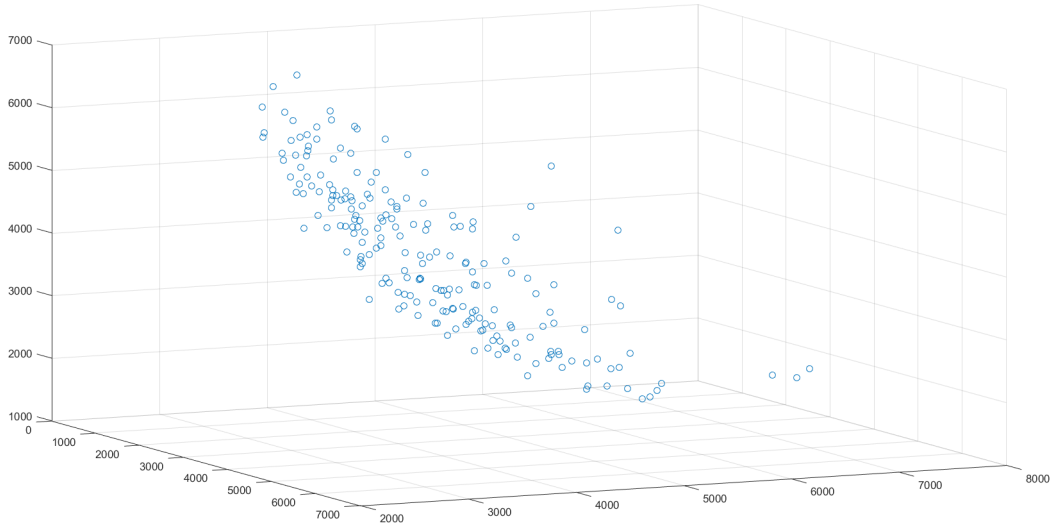


Figura 3.5: Custos das 214 soluções obtidas pelo AR

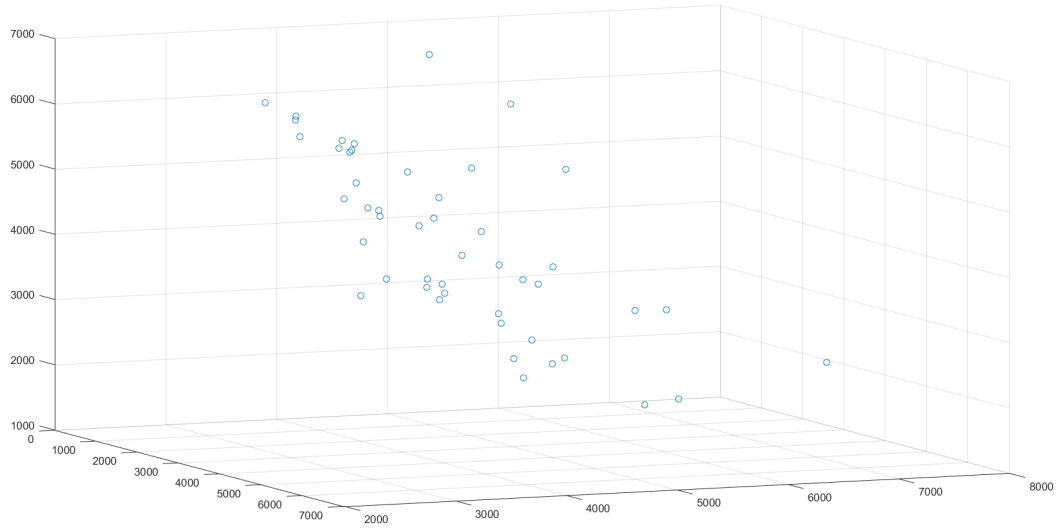


Figura 3.6: Custos das 48 soluções obtidas pelo AEV

Este problema foi resolvido até à otimalidade através do algoritmo de rotulação. O conjunto X_E é constituído por 214 soluções, obtidas em 2.667 segundos. O MONN teve um tempo de execução de 0.76 segundos, enquanto que o AEV um tempo de execução de 0.25 segundos. Até à última iteração, foram criados 53470 rótulos no algoritmo de pesquisa exaustiva, 20444 rótulos no MONN e 7566 rótulos no AEV.

A quantidade de soluções obtidas através da resolução de um MOTSP pode tornar-se demasiado elevada para determinadas aplicações. Caso o utilizador pretenda saber as melhores rotas em termos de tempo e custo, para percorrer um conjunto de locais, não é conveniente a apresentação de dezenas ou centenas de soluções. Nesse sentido, é necessária uma forma de agrupar as soluções em conjuntos, de forma a apresentar um conjunto representativo de X_E (ou mesmo $\widehat{X}_E$). Isto pode ser feito através de um algoritmo de *clustering*, agrupando as soluções através dos seus custos. Em seguida são consideradas soluções representativas em cada conjunto, definida por exemplo pela solução mais próxima da solução média do conjunto. Desta forma, se as soluções forem agrupadas em 3 conjuntos, são apresentadas ao utilizador as 3 opções representativas. Em seguida o utilizador pode optar por escolher de imediato uma dessas soluções ou, se pretender ver mais soluções, abrir o grupo do seu maior interesse. Em seguida, esse grupo é mais uma vez dividido em 3 grupos, e assim sucessivamente. Desta forma é assegurada ao utilizador a possibilidade de escolha.

As estratégias propostas neste capítulo permitem a resolução do MOTSP para pequenas instâncias, mas chegando a conjuntos de soluções bem representativos de X_E, tal como era pretendido.

Capítulo 4

Problema do Caixeiro Viajante Múltiplo Multi-objetivo

4.1. Introdução

O problema do caixeiro viajante múltiplo multi-objetivo (multiobjective multiple travelling salesman problema - MOmTSP) pretende calcular o conjunto de rotas para os m caixeiros viajantes percorrerem, minimizando diversos objetivos em simultâneo.

Este problema foi abordado pela primeira vez em 2012, por Chang e Yen [15], com o objetivo de obter boas rotas a serem percorridas por distribuidores de correio numa cidade. Neste problema foram também adicionadas restrições temporais, obtendo o MOmTSP com janelas temporais (MOmTSPTW). No mesmo ano, Shim [46] apresenta uma formulação do problema em que o objetivo é minimizar em simultâneo o custo total dos m caminhos e o custo do pior dos caminhos obtidos.

Em 2014 Labadie, Melechovsky e Prins[31] abordam o problema do caixeiro viajante múltiplo bi-objetivo com lucros (BOmTSPP), uma generalização do TSP com lucros (TSP with profits - TSPP). Neste problema, cada cliente (cidade), tem associado um lucro por visita e cada cliente é visitado pelo menos uma vez. Pretende-se obter m rotas de forma a maximizar o lucro e minimizar a distância total percorrida. Estes dois objetivos são claramente conflituosos.

Muito recentemente, Bolanosa, Echeverrya e Escobarb [12] apresentam uma variante com dois critérios, distância e tempo, em que o objetivo é minimizar a distância total a ser percorrida e simultaneamente balançar os tempos de viagem dos m caixeiros viajantes.

Este capítulo tem como objetivo introduzir algumas variantes e formulações do MOmTSP, bem como alguns algoritmos para a sua resolução.

4.1.1. Variantes

Existem diversas variantes do MOmTSP, tendo todas elas em comum o cálculo de um conjunto de rotas para os m caixeiros viajantes percorrerem, otimizando diversos objetivos em simultâneo.

Consideremos uma rede $G = (V, A)$ em que, a cada arco $(i, j) \in A$ está associado um vetor de custos $c_{ij} = (c_{ij}^1, ..., c_{ij}^k)$. Uma extensão natural do mTSP para o MOmTSP é determinar o conjunto de rotas para os m caixeiros, de forma a minimizar em simultâneo os k custos do caminho total. Este problema será referido como o **MOmTSP minsum**. O custo de uma solução é dada pelo vetor de custos $s = (s_1, ..., s_k)$, formado pela soma dos custos dos arcos percorridos pelos diversos caixeiros. Se pretendermos minimizar em simultânio os k custos do maior dos m caminhos obtidos, obtemos o **MOmTSP minmax**.

Redes multicritério podem originar outro tipo de variantes. Suponhamos que, a cada arco $(i, j) \in A$, está associado um custo c_{ij} e um tempo t_{ij}. Num determinado problema, podemos estar interessados em minimizar simultaneamente o custo total dos caminhos e o pior dos m tempos dos caixeiros. Desta forma, se o custo representar a gasolina a ser gasta por m camiões, para fazer um conjunto de entregas, o tempo das entregas pode ser considerado como o maior tempo que um dos camiões demora a percorrer a sua rota, uma vez que os camiões operam em simultâneo. Obtemos desta forma um mTSP bi-objetivo. Bolanosa, Echeverrya e Escobarb [12] propõem uma variante semelhante, em que é tida em conta a minimização da diferença entre o tempo de cada caixeiro e o tempo médio.

Para resolver um problema multi-objetivo não é necessário que a rede associada seja multicritério. De facto, consideremos uma rede $G = (V, A)$ monocritério, ou seja, a cada arco $(i, j) \in A$ está associado um só custo c_{ij}. Resolver um MOmTSP neste rede, consiste, por exemplo, em minimizar em simultâneo o custo total dos m caminhos e o custo do maior desses caminhos. Este é um problema bi-objetivo em que um dos critérios é minsum e o outro minmax.

De forma análoga ao MOTSP, resolver o MOmTSP consiste em obter o conjunto de soluções eficientes X_E.

4.2. Formulações

Nesta secção são apresentadas formulações lineares inteiras multi-objetivo de algumas das variantes referidas anteriormente.

O MOmTSP minsum é formulado de forma análoga ao mTSP minsum, com a diferença que pretendemos minimizar diversos custos em simultâneo. Considerando as variáveis binárias

$$\begin{cases} x_{ij} = 1 & \text{, se o arco } (i,j) \text{ pertencer à solução} \\ x_{ij} = 0 & \text{, caso contrário} \end{cases}$$

a sua versão single depot consiste então em

$$\begin{aligned} \text{minimizar } & f_1(x) = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}^1 x_{ij} \\ & \dots \\ \text{minimizar } & f_k(x) = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}^k x_{ij} \\ \text{sujeito a } & \sum_{j=2}^{n} x_{1j} = \sum_{j=2}^{n} x_{j1} = m, \\ & \sum_{j=1}^{n} x_{ij} = 1, i = 2, ..., n, \\ & \sum_{i=1}^{n} x_{ij} = 1, j = 2, ..., n, \\ & u_i - u_j + n x_{ij} \leq n-1, i, j = 2, ..., n \\ & x_{ij} \in \{0,1\} \end{aligned}$$

Esta variante possui k funções objetivo, uma para cada um dos critérios da rede a otimizar. As restrições impostas são as mesmas que as impostas para o mTSP. Para a versão multidepot, basta considerar as restrições respetivas, não sendo alteradas as funções objetivo.

Para formular o MOmTSP minmax, é necessário medir os caminhos de cada caixeiro viajante, de forma a minimizar o maior dos mesmos. Para tal, consideremos as variáveis binárias x_{ijl}, com $l = 1, ..., m$, em que

$$\begin{cases} x_{ijl} = 1 & \text{, se o caixeiro viajante } l \text{ percorrer o arco } (i,j) \\ x_{ijl} = 0 & \text{, caso contrário} \end{cases}$$

A formulação é então dada por

$$\text{minimizar } f_1(x) = \max_{l\in\{1,...,m\}}\{\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}^1 x_{ijl}\}$$

$$...$$

$$\text{minimizar } f_1(x) = \max_{l\in\{1,...,m\}}\{\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}^k x_{ijl}\}$$

$$\text{sujeito a } \sum_{l=1}^{m}\sum_{j=2}^{n} x_{1jl} = \sum_{l=1}^{m}\sum_{j=2}^{n} x_{j1l} = m, \quad (1)$$

$$\sum_{l=1}^{m}\sum_{j=1}^{n} x_{ijl} = 1, i = 2, ..., n, \quad (2)$$

$$\sum_{l=1}^{m}\sum_{i=1}^{n} x_{ijl} = 1, j = 2, ..., n, \quad (3)$$

$$\sum_{l=1}^{m}\sum_{i\in S}\sum_{j\notin S} x_{ijl} \geq 1, \forall S \subset V \quad (4)$$

$$x_{ijl} \in \{0, 1\}$$

Tal como na formulação anterior, o problema tem k funções objetivo, uma para cada um dos critérios da rede, pretendendo cada uma delas minimizar o custo do pior caminho nesse critério. As restrições (1) garantem que cada caixeiro parte e regressa ao armazém. As restrições (2) e (3) garantem que cada caixeiro passa por cada cidade uma única vez e que não passa mais do que um caixeiro por cidade (excepto o armazém inicial). As restrições (4) garantem que não há sub-caminhos na solução.

Consideremos agora um grafo $G = (V, A)$ em que a cada arco (i, j) estão associados dois custos: o custo monetário, c_{ij}, e o tempo que esse arco leva ser percorrido, t_{ij}. Se pretendermos minimizar o custo monetário total e simultaneamente o maior tempo que um caixeiro leva a percorrer a sua rota, obtemos um problema bi-objetivo, cujas funções são

$$\text{minimizar } f_1(x) = \sum_{l=1}^{m}\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij} x_{ijl}$$

$$\text{minimizar } f_2(x) = \max_{l\in\{1,...,m\}}\{t_l\}$$

em que $t_l, l = 1, ..., m$, representa o tempo total do caminho percorrido pelo caixeiro l, dado por

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

Note-se que os custos a otimizar nunca entram nas restrições do problema, pelo que as restrições desta versão são as mesmas da anterior. Para uma rede monocritério em que se pretende resolver o mTSP minsum e minmax em simultâneo, basta considerar as funções

$$\text{minimizar } f_1(x) = \sum_{l=1}^{m}\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ijl}$$

$$\text{minimizar } f_2(x) = \max_{l\in\{1,\dots,m\}}\{\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ijl}\}$$

minimizando desta forma o custo total e o pior dos m custos obtidos.

4.3. Abordagens

Nesta secção são referidas algumas das abordagens consideradas na literatura para a resolução do MOmTSP. Em seguida é proposta uma heurística de inserção para a variante minsum-minmax, de forma a obter soluções equilibradas e simultaneamente com bom custo total.

4.3.1. Métodos existentes na literatura

As poucas abordagens existentes na literatura, para a resolução de um MOmTSP, consistem em achar um conjunto aproximado de soluções eficientes $\hat{X}_E$, através de meta-heurísticas, que vão desde algoritmos de estimação de distribuição a algoritmos evolutivos.

O algoritmo utilizado por Bolanosa, Echeverrya e Escobarb [12] é o algoritmo genético de ordenação não-dominada (non-dominated sorting genetic algorithm), um algoritmo muito utilizado para a resolução de problemas multi-objetivo. O algoritmo começa com um conjunto com N soluções admissíveis, denominado por população P_t (pais). Em seguida são geradas, a partir de P_t, N novas soluções Q_t (descendentes) utilizando operadores de seleção, recombinação e mutação. Cada um destes operadores pode ter em conta um dos objetivos ou vários em simultâneo. Estas operações podem consistir na troca de posição de 2 nós da solução, ou de trocas de arestas obtidas por heurísticas como a 2-opt. Em seguida, é feita uma combinação entre os pais e os descendentes, formando uma nova população de tamanho $2N$. Em seguida as soluções são classificadas em conjuntos de frentes de Pareto, sendo cada um destes conjuntos o ponto de partida para a próxima iteração do algoritmo. O objetivo é que o conjunto das soluções finais sejam o mais variadas possíveis.

4.3.2. Algoritmo de Inserção por rótulos para o MOmTSP

Consideremos o MOmTSP bi-objetivo minsum-minmax, na versão single depot com m fixo, em que se pretende minimizar o custo total da solução e em simultâneo o custo do pior dos m caminhos obtidos. Uma forma de resolver o problema, é aplicar um algoritmo de inserção análogo aos utilizador para o mTSP, em que, sempre que houvessem conflitos de soluções, seria criado um rótulo para cada uma das opções não dominadas. O algoritmo começa por selecionar os vértices $v_i, i = 1, ..., m$ mais distantes do vértice v_a, que representa o armazém. São criados m caminhos da forma $p_i =< v_a, v_i, v_a >, i = 1, ..., m$. Este conjunto de caminhos é guardado no rótulo inicial l_i. Em seguida, é adicionado o vértice cuja inserção seja a de menor custo global e simultaneamente a que minimiza o custo do pior dos m caminhos. Para cada escolha possível não dominada, é criado um rótulo para cada nova sub-solução. Em cada iteração, é feito o mesmo para cada rótulo, até todos os vértices do grafo serem adicionados. Das soluções obtidas, são excluídas as soluções dominadas, obtendo assim o conjunto de soluções $\hat{X}_E$.

Para ilustrar este algoritmo, consideremos a seguinte rede completa e simétrica, com $|V| = 7$:

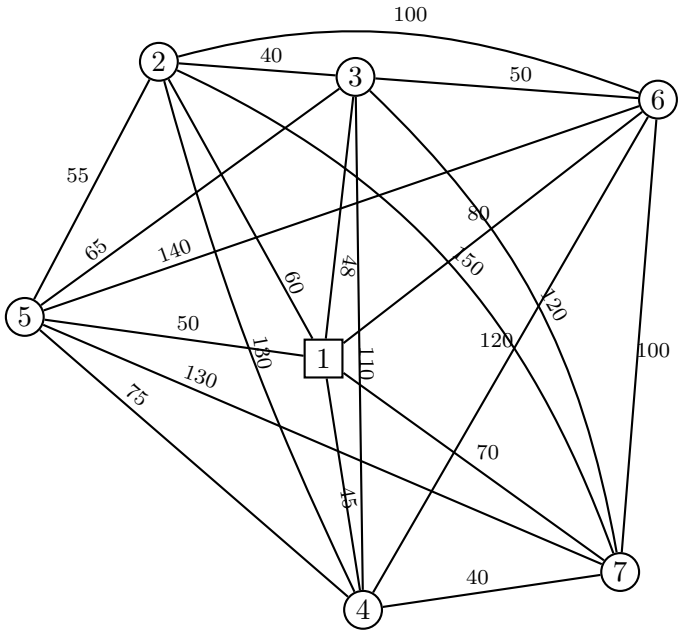


Figura 4.1: Rede com $|V| = 7$

Para resolver o MOmTSP com $m = 2$, é criado um rótulo com 2 caminhos iniciais. Suponhamos que são criados os caminhos $p_1 =< 1, 6, 1 >$ e $p_2 =< 1, 7, 1 >$. Sejam

c_T o custo total da sub-solução e c_{max} o custo do pior dos m caminhos. Sendo r_1 o rótulo inicial, temos que $c_T(r_1) = 300$ e $c_{max}(r_1) = 160$. Por abuso de linguagem, dizemos que $C(r_1) = (300, 160)$.

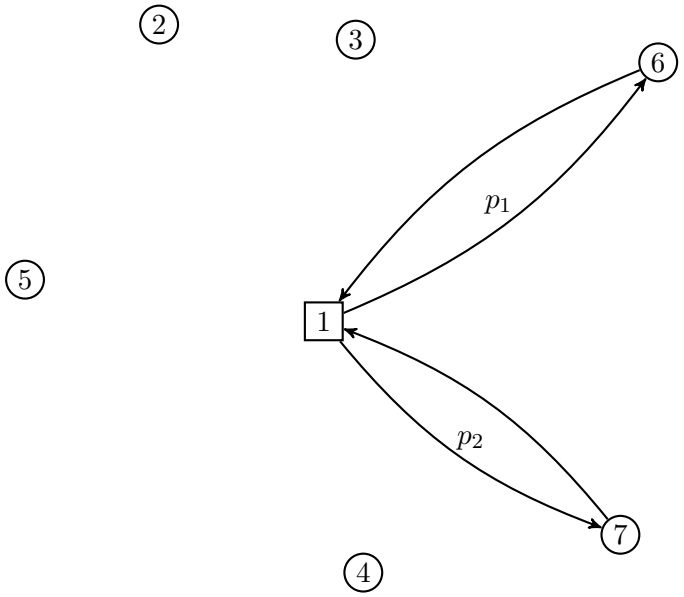


Figura 4.2: Criação do rótulo inicial

Em seguida o algoritmo analisa todas as possibilidades de inserção de um vértice na solução, guardando os custos associados a cada uma. Se adicionarmos o vértice 3 em p_1, temos que $c(p_1) = 178$ e $c(p_2) = 140$, pelo que a solução passa a ter um custo $(318, 178)$. Se adicionarmos o vértice 4 em p_2, temos que $c(p_1) = 160$ e $c(p_2) = 155$, pelo que a solução passa a ter custo $(315, 160)$. Como $(318, 178) \not\prec (315, 160)$ e $(315, 160) \not\prec (318, 178)$, são criados dois rótulos, um para cada caso. Note-se que qualquer outra solução teria custos dominados.

No final, o algoritmo poderia facilmente chegar às soluções

$$s_1 = \{< 1, 7, 4, 5, 1 >, < 1, 6, 3, 2, 1 >\}$$
$$s_2 = \{< 1, 7, 4, 1 >, < 1, 6, 3, 2, 5, 1 >\}.$$

Temos que $c(< 1, 7, 4, 5, 1 >) = 235$, $c(< 1, 6, 3, 2, 1 >) = 230$, $c(< 1, 7, 4, 1 >) = 155$ e $c(< 1, 6, 3, 2, 5, 1 >) = 275$. Desta forma, $C(s_1) = (465, 235)$ e $C(s_2) = (430, 275)$ pelo que ambas as soluções são válidas para entrarem no conjunto $\hat{X}_E$.

Apesar da sua simplicidade, o algoritmo efetua uma quantidade considerável de operações. Para problemas assimétricos, se na iteração i existirem p nós adicionados, há $p(n-p)$ possibilidades de inserção para cada rótulo existente.

Este algoritmo está a ser implementado, fazendo parte dos objetivos do projeto. Com ele espera-se dar resposta ao problema de calcular bons conjuntos de rotas para o MOmTSP, de forma análoga ao MOTSP, e vir a ser aplicado em problemas reais de escalonamento de reparações de avarias em redes de telecomunicações. Deste modo, pretende-se colmatar o estudo teórico desenvolvido ao longo desta dissertação sobre o problema do caixeiro viajante nas suas diversas variantes e completar assim os objetivos do projeto.

Apêndice A

Algoritmos

A.1. Heurísticas do TSP

Algoritmo 1: Vizinho mais Próximo - Nearest Neighbor

input : $G = (V, A)$
output: Caminho p

$v_i \leftarrow$ *vértice inicial*; $p =< v_i >$;
$v_{act} \leftarrow v_i$;
$X \leftarrow V \backslash \{v_i\}$;
while $X \neq \emptyset$ **do**
  $v \leftarrow argmin_{x \in X}\{c_{v_{act},x} : (v_{act}, x) \in A\}$;
  $p \leftarrow p \diamond < v_{act}, v >$;
  $X \leftarrow X \backslash \{v\}$;
  $v_{act} \leftarrow v$;
end
$p \leftarrow p \diamond < v_{act}, v_i >$;

Algoritmo 2: Ligação mais Barata - Cheapest Link

input : $G = (V, A)$
output: Caminho p

$n_{aa} \leftarrow 0$ % número de arcos adicionados
$A_p \leftarrow A$ % arcos que podem ser adicionados
while $n_{aa} \neq |V|$ **do**
  $a^* \leftarrow argmin_{a \in A_p}\{c_a\}$
  $p \leftarrow p \diamond a^*$
  Atualiza A_p
  $n_{aa} \leftarrow n_{aa} + 1$
end

Algoritmo 3: Heurística de Christofides

input : $G = (V, A)$
output: Caminho p

Calcular $T \leftarrow$ árvore de custo mínimo
$I \leftarrow \{v \in V : deg(v)\%2 = 1\}$
$M \leftarrow$ Emparelhamento Perfeito de I
$R \leftarrow T \cup M$
while $\exists v \in M : deg(v) > 2$ **do**
  Encontra $(u, v), (v, w) : R \cup (u, w) \backslash (u, v), (v, w)$ continua conexo
  Actualiza R
end
$p \leftarrow R$

Algoritmo 4: Ligação mais Barata Relativa - Relative Cheapest Link

input : $G = (V, A)$
output: Caminho p

$n_{aa} \leftarrow 0$ % número de arcos adicionados
$A_p \leftarrow A$ % arcos que podem ser adicionados
$c^*_{ij} = \frac{c_{ij}}{\sum_{j \in A} c_{ij}}, \forall (i,j) \in A$
while $n_{aa} \neq |V|$ **do**
  $a^* \leftarrow argmin_{a \in A_p}\{c^*_a\}$
  $p \leftarrow p \diamond a^*$
  Atualiza A_p
  $n_{aa} \leftarrow n_{aa} + 1$
end

Algoritmo 5: Algoritmo de Inserção - Insertion Algorithm

input : $G = (V, A)$
output: Caminho p

v_1,$v_2 \leftarrow argmin_{x,y \in X}\{c_{x,y} + c_{y,x} : (x,y) \in A \wedge (y,x) \in A\}$
$p =< v_1, (v_1, v_2), v_2, (v_2, v_1), v_1 >$
$A_p \leftarrow A$ % arcos que podem ser adicionados
$X \leftarrow V \backslash \{v_1, v_2\}$
while $X \neq \emptyset$ **do**
  $v \leftarrow argmin_{x \in X}\{c_{v_i,x} + c_{x,v_j} - c_{v_i,v_j} : (v_i, x) \in A_p \wedge (x, v_j) \in A_p \wedge (v_i, v_j) \in p\}$
  $p \leftarrow p \diamond < (v_i, v), v, (v, v_j) > \backslash (v_i, v_j)$
  $X \leftarrow X \backslash \{v\}$
  Atualiza A_p
end
$p \leftarrow p \diamond < a >, a \in A_p$ % resta apenas um arco em A_p

Algoritmo 6: Ligação Prioritária - Priority Link

input : $G = (V, A)$
output: Caminho p

Ordena A por custos
$n_{aa} \leftarrow 0$ % número de arcos adicionados
$A_p \leftarrow A$ % arcos que podem ser adicionados
while $n_{aa} \neq |V|$ **do**
  Cálculo de DI^x_1 e DO^x_1, $\forall x \in V$
  $P \leftarrow$ Prioridades
  if *há empate* **then**
    Desempata P com DI^x_i, $DO^x_i, i > 1$
  $a^* \leftarrow argmin_{(x,y) \in A_P}\{D^{x,y}\}$
  $p \leftarrow p \diamond a^*$
  Atualiza A_p
  $n_{aa} \leftarrow n_{aa} + 1$
end

A.2. Heurísticas do mTSP

Algoritmo 7: Algoritmo de Bellmore e Hong

input : $G = (V, A)$, nº máximo de caixeiros m
output: Conjunto de caminhos P

$G' \leftarrow$ transformação(G, m) % cria grafo transformado
$p \leftarrow$ solução de G' % através de uma heurística do TSP
$v_{ini} \leftarrow p[0]$ % o armazém corresponde ao primeiro vértice de p
$i \leftarrow 1$
$sales \leftarrow 10$
$P[sales] \leftarrow P[sales] \diamond v_{ini}$
while $p[i] \neq v_{ini}$ **do**
  if $p[i] > 0$ **then**
    $P[sales] \leftarrow P[sales] \diamond p[i]$
  else
    $P[sales] \leftarrow P[sales] \diamond v_{ini}$
    $sales \leftarrow -p[i]$
    $P[sales] \leftarrow P[sales] \diamond v_{ini}$
  end
end
$P[sales] \leftarrow P[sales] \diamond v_{ini}$

Algoritmo 8: Algoritmo de Inserção por Custo

input : $G = (V, A)$, nº de caixeiros m
output: Conjunto de caminhos P

$v_a \leftarrow depot$
$X \leftarrow V$
while $m > 0$ **do**
  $v \leftarrow argmax_{x \in X}\{c_{v_{act},x} : (v_{act}, x) \in A\}$
  $P[m-1] \leftarrow < v_d, v, v_d >$
  $X \leftarrow X \backslash \{v\}$
  $m--$
end
while $X \neq \emptyset$ **do**
  $c_{min} \leftarrow c(P[0])$
  $i_{min} \leftarrow 0$
  for $P[i], i = 1, ..., m$ **do**
    if $c(P[i]) < c_{min}$ **then**
      $c_{min} \leftarrow c(P[i])$
      $i_{min} \leftarrow i$
  end
  $v \leftarrow argmin_{x \in X}\{c_{v_i,x} + c_{x,v_j} - c_{v_i,v_j} : (v_i, x) \in A_p \wedge (x, v_j) \in A_p \wedge (v_i, v_j) \in P[i_{min}]\}$
  $P[i_{min}] \leftarrow P[i_{min}] \diamond < (v_i, v), v, (v, v_j) > \backslash (v_i, v_j)$
  $X \leftarrow X \backslash \{v\}$
end

Algoritmo 9: Algoritmo de Inserção por Nós

input : $G = (V, A)$, nº de caixeiros m
output: Conjunto de caminhos P

$v_a \leftarrow depot$
$X \leftarrow V$
while $m > 0$ **do**
  $v \leftarrow argmax_{x \in X}\{c_{v_{act},x} : (v_{act}, x) \in A\}$
  $P[m-1] \leftarrow < v_d, v, v_d >$
  $X \leftarrow X \backslash \{v\}$
  $m - -$
end
while $X \neq \emptyset$ **do**
  for $P[i], i = 0, ..., m$ **do**
    $v \leftarrow argmin_{x \in X}\{c_{v_i,x} + c_{x,v_j} - c_{v_i,v_j} : (v_i, x) \in A_p \wedge (x, v_j) \in A_p \wedge (v_i, v_j) \in P[i]\}$
    $P[i] \leftarrow P[i] \diamond < (v_i, v), v, (v, v_j) > \backslash (v_i, v_j)$
    $X \leftarrow X \backslash \{v\}$
    if $p[i] > 0$ **then**
      break
  end
end

Algoritmo 10: Algoritmo de Inserção mais Barata

input : $G = (V, A)$, nº de caixeiros m
output: Conjunto de caminhos P

$v_a \leftarrow depot$
$X \leftarrow V$
while $m > 0$ **do**
  $v \leftarrow argmax_{x \in X}\{c_{v_{act},x} : (v_{act}, x) \in A\}$
  $P[m-1] \leftarrow < v_d, v, v_d >$
  $X \leftarrow X \backslash \{v\}$
  $m - -$
end
while $X \neq \emptyset$ **do**
  $i_{min} \leftarrow 0$
  $v_{min} \leftarrow argmin_{x \in X}\{c_{v_i,x} + c_{x,v_j} - c_{v_i,v_j} : (v_i, x) \in A_p \wedge (x, v_j) \in A_p \wedge (v_i, v_j) \in P[i_{min}]\}$
  $v_{tail} \leftarrow v_i$
  $v_{head} \leftarrow v_j$
  $c_{min} \leftarrow c_{v_{tail},v} + c_{v,v_{head}} - c_{v_{tail},v_{head}}$
  for $P[i], i = 1, ..., m$ **do**
    $v \leftarrow argmin_{x \in X}\{c_{v_i,x} + c_{x,v_j} - c_{v_i,v_j} : (v_i, x) \in A_p \wedge (x, v_j) \in A_p \wedge (v_i, v_j) \in P[i_{min}]\}$
    $c \leftarrow c_{v_i,v} + c_{v,v_j} - c_{v_i,v_j}$
    if $c < c_{min}$ **then**
      $c_{min} \leftarrow c$
      $i_{min} \leftarrow i$
      $v_{min} \leftarrow argmin_{x \in X}\{c_{v_i,x} + c_{x,v_j} - c_{v_i,v_j} : (v_i, x) \in A_p \wedge (x, v_j) \in A_p \wedge (v_i, v_j) \in P[i_{min}]\}$
      $v_{tail} \leftarrow v_i$
      $v_{head} \leftarrow v_j$
  end
  $P[i_{min}] \leftarrow P[i_{min}] \diamond < (v_{tail}, v_{min}), v, (v_{min}, v_{head}) > \backslash (v_{tail}, v_{head})$
  $X \leftarrow X \backslash \{v_{min}\}$
end

A.3. Algoritmos e heurísticas do MOTSP

Algoritmo 11: Algoritmo de Pesquisa Exaustiva com Princípio de Otimalidade

input : $G = (V, A)$
output: Conjunto de soluções não dominadas

$v \leftarrow 1$ % vértice inicial
$n_{vis} \leftarrow (1, 0, ..., 0)$ % vértices até agora visitados
$cost \leftarrow (0, ..., 0)$
$set_n \leftarrow 0$
$ptrPai \leftarrow NULL$
$l_{ini} \leftarrow criaRotulo(v, set_n, n_{vis}, cost, ptrPai)$
$X \leftarrow V \backslash \{v\}$
$lista_a \leftarrow \{l_{ini}\}$
$lista_s \leftarrow \emptyset$
while $X \neq \emptyset$ **do**
  $l \leftarrow lista_a[0]$
  $N \leftarrow getVizinhos(l.v)$
  for ***all*** $v \in N \cup X$ **do**
    $n_{vis} \leftarrow l.n_{vis}$
    $n_{vis} \leftarrow n_{vis}[v]$
    $cost \leftarrow l.cost + c(l.v, v)$
    $set_n \leftarrow 2^{v-2}$
    $ptrPai \leftarrow l^*$
    $l_{novo} \leftarrow criaRotulo(v, set_n, n_{vis}, cost, ptrPai)$
    $d \leftarrow 0$
    while $\exists l_s \in lista_s : l_s$ *e* l_{novo} *são comparáveis* **do**
      $d \leftarrow dominancia(l_{novo}, l_s)$
      if $d < 0$ **then**
        break
      if $d > 0$ **then**
        remove(l_s)
    end
    if $d \geq 0$ **then**
      $lista_s \leftarrow lista_s \cup \{l_{novo}\}$
  end
  $lista_a \leftarrow lista_a \backslash \{l\}$
end

Notas:

- O vetor n_{vis} é um vetor de booleanos, determinando quais os vértices que pertencem ao caminho do rótulo em questão.
- A variável set_n é uma variável *long int* que representa o conjunto de vértices contidos num caminho p. Cada vértice v tem associado o valor 2^{v-2} caso $v \in p$ ou 0 caso contrário. set_n é dado pela soma dos valores associados a todos os vértices.
- O algoritmo **vizinho mais próximo multi-objetivo** difere deste algoritmo apenas nos vértices vizinhos a analisar. Basta então substituir função $getVizinhos()$ por $getVizinhosNaoDominados()$.

Algoritmo 12: Algoritmo de Eliminação por Vizinhança

input : $G = (V, A)$
output: Conjunto de soluções não dominadas

$v \leftarrow 1$ % vértice inicial
$n_{vis} \leftarrow (1, 0, ..., 0)$ % vértices até agora visitados
$cost \leftarrow (0, ..., 0)$
$set_n \leftarrow 0$
$ptrPai \leftarrow NULL$
$l_{ini} \leftarrow criaRotulo(v, set_n, n_{vis}, cost, ptrPai)$
$X \leftarrow V \backslash \{v\}$
$lista_a \leftarrow \{l_{ini}\}$
$lista_s \leftarrow \emptyset$
while $X \neq \emptyset$ **do**
  $l \leftarrow lista_a[0]$
  $N \leftarrow getVizinhosNaoDominados(l.v)$
  for ***all*** $v \in N \cup X$ **do**
    $n_{vis} \leftarrow l.n_{vis}$
    $n_{vis} \leftarrow n_{vis}[v]$
    $cost \leftarrow l.cost + c(l.v, v)$
    $set_n \leftarrow 2^{v-2}$
    $ptrPai \leftarrow l^*$
    $l_{novo} \leftarrow criaRotulo(v, set_n, n_{vis}, cost, ptrPai)$
    $dom \leftarrow 0$
    while $\exists l_s \in lista_s : l_s e l_{novo}$ *são comparáveis* **do**
      if $d(l_s.v, l_{novo}.v) < m_A n_n \alpha$ **then**
        if $\|l_{novo}.cost\| < \|l_s.cost\|$ **then**
          $swap(l_s, l_{novo})$
        $dom \leftarrow -1$
        break
      $dom \leftarrow dominancia(l_{novo}, l_s)$
      if $d < 0$ **then**
        break
      if $d > 0$ **then**
        remove(l_s)
    end
    if $d \geq 0$ **then**
      $lista_s \leftarrow lista_s \cup \{l_{novo}\}$
  end
  $lista_a \leftarrow lista_a \backslash \{l\}$
end

Apêndice B

Resultados

B.1. Tabelas TSP

Soluções STSP

Grafo	$\|V\|$	s^*	NN	RNN	CL	RCL	PL	CA	IA
burma14	14	3323	4048	3841	3952	4335	4007	3939	3359
ulysses16	16	6859	9988	7943	7986	8075	7071	7379	7352
gr17	17	2085	2187	2178	2503	2508	2235	2507	2271
gr21	21	2707	3333	3003	3368	3213	2973	3283	3119
ulysses22	22	7013	10586	8180	8711	8560	7466	8054	7517
gr24	24	1272	1553	1553	1610	1368	1442	1556	1401
fri26	26	937	1112	965	1014	1127	1048	1070	1046
bayg29	29	1610	2005	1935	1931	1971	1816	1802	1846
bays29	29	2020	2258	2134	2484	2253	2359	2580	2257
dantzig42	42	699	956	864	991	917	768	787	794
swiss42	42	1273	1630	1437	1620	1611	1525	1382	1481
att48	48	10628	12861	12012	15160	14925	12234	13731	11488
gr48	48	5046	6098	5840	5830	6800	5569	6248	5816
hk48	48	11461	13181	12137	14012	13605	12327	13274	13015
eil51	51	426	497	489	546	527	453	509	457
berlin52	52	7542	8962	8164	9551	9410	8910	9341	8980
brazil58	58	25395	30774	27384	29349	29827	30022	31021	29126
st70	70	675	797	787	900	839	776	792	750
eil76	76	538	662	598	710	689	637	657	600
pr76	76	108159	153442	130903	132897	144731	121657	129185	125253
gr96	96	55209	70916	63945	74805	72392	69759	62744	69675
rat99	99	1211	1530	1443	1649	1614	1380	1406	1413
kroA100	100	21282	27772	24659	26908	29646	28885	28115	25259
kroB100	100	22141	29136	25830	29773	27205	27658	25524	25149
kroC100	100	20749	26281	23520	27056	27123	27830	24424	25361
kroD100	100	21294	26906	24810	28719	29300	27994	25236	25367
rd100	100	7910	9889	9380	9999	10256	9727	10097	9097
eil101	101	629	796	750	835	844	661	707	690
lin105	105	14379	20341	16914	22691	20609	18123	17585	16885
pr107	107	44303	46657	46657	55615	53666	57539	49011	51422
gr120	120	6942	9351	8438	9881	9334	8244	9190	7851
pr124	124	59030	69282	67039	74900	78885	75346	67530	65916
bier127	127	118282	135713	133936	153683	152754	144548	143916	140746
ch130	130	6110	7511	7126	8014	7788	7144	7180	7034
pr136	136	96772	120760	114538	132063	134106	112270	113379	112966
gr137	137	69853	93912	84406	94528	94393	76997	79231	84718
pr144	144	58537	61633	60946	73267	75700	76622	74658	73019
ch150	150	6528	8259	7053	8106	8405	8313	7365	7696
kroA150	150	26524	34064	30566	34983	37539	31549	35257	30132
kroB150	150	26130	32767	31543	35042	35196	33551	32694	31261
u159	159	42080	54637	48558	55318	56919	56364	49514	50505
si175	175	21407	22263	22000	22187	22404	22449	22268	22130
rat195	195	2323	2661	2579	3754	3303	2628	2648	2692
d198	198	15780	18163	17545	19875	19876	19699	17774	17569

Grafo	$\|V\|$	s^*	NN	RNN	CL	RCL	PL	CA	IA
kroA200	200	29368	35715	35030	38831	40514	38503	35176	35030
kroB200	200	29437	36509	34767	37923	42042	37271	37091	35931
gr202	202	40160	49336	47060	51504	53384	44439	45746	46600
ts225	225	126643	152493	140484	190525	183884	137913	156342	169721
tsp225	225	3916	4648	4496	4726	5374	4564	4443	4406
pr226	226	80369	94672	92540	109167	103552	96453	96064	91173
pr264	264	49135	58015	54483	57955	72133	57026	60286	57576
a280	280	2579	3203	2954	3160	3714	3131	2894	3096
pr299	299	48191	59833	58222	67045	73730	58688	56662	57773
rd400	400	15281	18968	18061	19661	20767	19300	18800	18281
fl417	417	11861	14843	13719	16041	15681	15978	15892	14207
gr431	431	171414	210069	207189	229470	234387	228962	201929	198746
pr439	439	107217	131211	127156	141210	163216	144016	135436	130857
pcb442	442	50778	61926	58896	71986	75006	59461	59647	61123
d493	493	35002	43244	40168	44054	46386	42496	41072	39853
att532	532	27686	35516	33387	39084	39379	37735	32744	32351
ali535	535	202339	253127	240828	250672	287410	262828	251377	242309
si535	535	48450	50144	50036	50015	50585	50690	51337	49663
pa561	561	2763	3422	3279	3794	3856	3105	3332	3461
u574	574	36905	46604	45139	47777	52363	51119	42490	43685
rat575	575	6773	8431	8038	9768	9323	8017	7654	7866
d657	657	48912	61289	60763	67528	67203	61448	56881	57706
gr666	666	294358	366962	350243	376130	412358	386820	338940	354458
u724	724	41910	52539	50381	52957	61117	54913	49717	50313
rat783	783	8806	10867	10272	12775	12437	10948	10072	9931
pr1002	1002	259045	315346	311987	344306	375122	343330	307627	302736

Tabela B.1: Resultados das heurísticas aplicadas ao STSP

Tempos de Execução STSP

Grafo	$\|V\|$	NN	RNN	CL	RCL	PL	CA	IA
burma14	14	0	0	0	0	0	0	0
ulysses16	16	0	0	0	0.015	0	0	0
gr17	17	0	0	0	0	0	0	0
gr21	21	0	0	0	0	0	0	0
ulysses22	22	0	0	0.016	0	0	0	0
gr24	24	0	0	0	0	0.016	0	0
fri26	26	0	0	0	0	0	0	0
bayg29	29	0	0	0	0	0.016	0	0
bays29	29	0	0	0	0	0	0	0
dantzig42	42	0	0	0	0	0.016	0	0
swiss42	42	0.016	0	0	0	0	0	0.015
att48	48	0	0	0	0	0	0	0
gr48	48	0.015	0	0	0	0.016	0	0
hk48	48	0	0.016	0	0	0	0	0
eil51	51	0	0	0	0	0	0.015	0
berlin52	52	0.015	0	0	0	0.016	0	0
brazil58	58	0	0	0	0	0.016	0	0
st70	70	0	0	0	0	0.016	0	0
eil76	76	0	0	0	0	0.031	0.016	0
pr76	76	0	0	0.016	0	0.031	0	0.016
gr96	96	0.016	0	0.016	0.015	0.047	0.016	0
rat99	99	0	0.016	0.016	0	0.047	0.016	0.015

Grafo	$\|V\|$	NN	RNN	CL	RCL	PL	CA	IA
kroA100	100	0.016	0	0.015	0.016	0.047	0.015	0.016
kroB100	100	0	0	0.016	0.016	0.046	0.016	0
kroC100	100	0	0.015	0.016	0.015	0.047	0.016	0.016
kroD100	100	0.016	0	0.015	0.016	0.047	0	0
rd100	100	0	0.016	0.015	0	0.047	0.016	0
eil101	101	0	0.016	0.016	0	0.063	0.015	0.016
lin105	105	0	0.015	0.016	0	0.047	0	0
pr107	107	0	0.016	0.015	0.016	0.125	0.015	0.016
gr120	120	0	0	0.016	0.016	0.078	0.016	0.016
pr124	124	0	0.016	0.016	0.016	0.094	0.015	0.016
bier127	127	0	0	0.031	0.015	0.094	0.031	0.016
ch130	130	0	0	0.016	0.031	0.11	0.015	0.032
pr136	136	0.015	0.016	0.031	0.016	0.234	0.015	0.016
gr137	137	0	0	0.015	0.031	0.125	0.016	0.015
pr144	144	0	0	0.016	0.015	0.157	0.016	0.031
ch150	150	0	0	0.031	0.031	0.156	0.032	0.031
kroA150	150	0.016	0.016	0.031	0.016	0.172	0.031	0.031
kroB150	150	0	0.015	0.031	0.032	0.156	0.031	0.031
u159	159	0	0.016	0.031	0.015	0.203	0.016	0.031
si175	175	0.016	0.015	0.031	0.031	0.281	0.031	0.032
rat195	195	0	0.016	0.047	0.031	0.375	0.047	0.047
d198	198	0.016	0.016	0.062	0.031	0.375	0.063	0.047
kroA200	200	0.015	0.015	0.047	0.046	0.313	0.047	0.047
kroB200	200	0.016	0.031	0.063	0.046	0.344	0.062	0.047
gr202	202	0.015	0.031	0.063	0.047	0.344	0.047	0.063
ts225	225	0.031	0.015	0.062	0.047	0.781	0.047	0.062
tsp225	225	0.016	0.032	0.063	0.047	0.672	0.047	0.078
pr226	226	0.016	0.031	0.063	0.046	0.844	0.063	0.078
pr264	264	0.016	0.031	0.078	0.062	3.203	0.063	0.11
a280	280	0.031	0.031	0.109	0.078	1.078	0.094	0.14
pr299	299	0.032	0.047	0.109	0.093	1.125	0.109	0.172
rd400	400	0.062	0.109	0.234	0.203	2.718	0.219	0.406
fl417	417	0.094	0.109	0.234	0.235	3.641	0.234	0.406
gr431	431	0.062	0.109	0.265	0.235	3.156	0.25	0.438
pr439	439	0.063	0.109	0.265	0.234	4.313	0.25	0.485
pcb442	442	0.078	0.109	0.266	0.25	3.985	0.25	0.5
d493	493	0.094	0.187	0.422	0.36	5.359	0.36	0.688
att532	532	0.109	0.188	0.453	0.391	7.031	0.531	1.063
ali535	535	0.125	0.219	0.484	0.563	6.375	0.547	0.969
si535	535	0.11	0.172	0.438	0.344	11.984	0.422	0.875
pa561	561	0.14	0.203	0.438	0.39	16.828	0.484	0.969
u574	574	0.125	0.204	0.468	0.422	8.641	0.484	1.015
rat575	575	0.125	0.188	0.485	0.407	15.11	0.516	1.109
d657	657	0.188	0.281	0.687	0.641	12.813	0.688	1.547
gr666	666	0.172	0.281	0.657	0.578	11.25	0.75	1.562
u724	724	0.203	0.328	0.766	0.703	19.891	0.844	2.156
rat783	783	0.25	0.39	0.984	0.875	53.234	1.062	2.563
pr1002	1002	0.375	0.61	1.578	1.406	47.172	1.922	5.328

Tabela B.2: Tempos de execução das heurísticas aplicadas ao STSP

Soluções ATSP

Grafo	$\|V\|$	s^*	NN	RNN	CL	RCL	PL	CA
br17	17	39	92	56	97	79	39	39
ftv33	34	1286	1683	1590	1627	1506	1591	1515
ftv35	36	1473	1791	1667	1825	1795	1765	1634
ftv38	39	1530	1778	1759	1908	1888	1889	1787
ft53	53	6905	9514	8584	12272	8325	7840	8842
p43	43	5620	5768	5684	5796	5656	5663	5664
ftv44	45	1613	2014	1844	1916	1799	1720	1755
ftv47	48	1776	2374	2173	2238	2248	1966	2125
ry48p	48	14422	16757	15575	19116	15646	15792	15990
ftv55	56	1608	2012	1948	1971	1802	1725	1968
ftv64	65	1839	2639	2202	2327	2276	2029	2259
ft70	70	38673	43186	41815	44411	40901	40747	41868
ftv70	71	1950	2571	2287	2500	2424	2059	2378
ftv170	171	2755	3923	3582	3662	3526	3292	3994
kro124p	100	36230	47506	43316	43841	43446	42947	41615
rbg323	323	1326	1734	1702	1447	1443	1443	1672
rbg358	358	1163	1812	1747	1255	1228	1290	1549
rbg403	403	2465	3535	3497	2488	2486	2660	2735
rbg443	443	2720	3922	3858	2741	2740	2940	3005

Tabela B.3: Resultados das heurísticas aplicadas ao ATSP

Tempos de Execução ATSP

Grafo	$\|V\|$	RNN	CL	RCL	PL	CA
br17	0	0.015	0	0	0	0
ftv33	0	0	0	0	0.015	0
ftv35	0	0	0	0	0.016	0
ftv38	0	0	0	0.016	0	0
ft53	0	0	0	0	0.016	0
p43	0.015	0	0	0	0.016	0
ftv44	0	0	0	0	0	0
ftv47	0	0	0	0	0.016	0
ry48p	0	0	0	0	0.016	0
ftv55	0	0	0.015	0	0	0
ftv64	0	0	0	0	0.016	0
ft70	0	0	0.016	0	0.015	0
ftv70	0	0.015	0	0	0.016	0
ftv170	0.016	0.016	0.032	0.031	0.219	0.031
kro124p	0.016	0	0.016	0.015	0.047	0.015
rbg323	0.046	0.14	0.14	0.125	12.063	0.234
rbg358	0.047	0.14	0.172	0.156	16.218	0.281
rbg403	0.062	0.39	0.203	0.188	33.813	0.422
rbg443	0.078	0.531	0.25	0.218	56.156	0.547

Tabela B.4: Tempos de execução das heurísticas aplicadas ao ATSP

B.2. Tabelas mTSP

Soluções Algoritmos de Inserção com 2 caixeiros

Grafo	$\|V\|$	By Node	p_1	p_2	By Cost	p_1	p_2	Min	p_1	p_2
burma14	14	4536	2628	1908	4606	2599	2007	4280	2708	1572
ulysses16	16	9408	6072	3336	9089	4628	4461	8546	6419	2127
gr17	17	3307	1694	1613	3290	1595	1695	2825	1812	1013
gr21	21	3717	1363	2354	3257	1613	1644	3192	1606	1586
ulysses22	22	9697	6387	3310	9589	4628	4961	8688	6554	2134
gr24	24	1824	910	914	1647	841	806	1456	937	519
fri26	26	1452	760	692	1508	743	765	1351	440	911
bayg29	29	2214	1123	1091	1973	988	985	1944	1034	910
bays29	29	2653	1237	1416	2550	1313	1237	2363	1201	1162
dantzig42	42	887	502	385	838	419	419	823	385	438
swiss42	42	1785	866	919	1792	841	951	1510	578	932
att48	48	14172	6616	7556	14800	7324	7476	12742	7231	5511
gr48	48	6410	3492	2918	6610	3390	3220	5581	4165	1416
hk48	48	14603	7374	7229	14336	7357	6979	14396	8456	5940
eil51	51	472	255	217	493	250	243	436	346	90
berlin52	52	10658	5399	5259	9148	4691	4457	9134	3818	5316
brazil58	58	38183	18168	20015	34248	16507	17741	32354	21920	10434
st70	70	757	413	344	800	399	401	709	297	412
eil76	76	558	236	322	491	243	248	497	231	266
pr76	76	144839	73061	71778	141764	72859	68905	134210	76371	57839
gr96	96	54262	30331	23931	52006	26843	25163	51929	27309	24620
rat99	99	1244	595	649	1213	612	601	1243	652	591
kroA100	100	25757	13927	11830	24455	12064	12391	23637	8872	14765
kroB100	100	22998	11448	11550	23147	11573	11574	22628	15085	7543
kroC100	100	30090	15654	14436	26507	12870	13637	23854	18070	5784
kroD100	100	26090	12787	13303	28178	13709	14469	23476	9319	14157
rd100	100	9050	4538	4512	9186	4621	4565	8572	4372	4200
eil101	101	554	237	317	552	276	276	543	290	253
lin105	105	19912	10121	9791	19373	9694	9679	18674	12622	6052
pr107	107	73176	38932	34244	72888	34305	38583	68949	34409	34540
gr120	120	680	4154	-3474	6895	3371	3524	7112	4655	2457
pr124	124	92517	45048	47469	83723	43704	40019	71885	31033	40852
bier127	127	129387	66815	62572	136439	74193	62246	127560	78667	48893
ch130	130	6967	3270	3697	7219	3606	3613	6740	5252	1488
pr136	136	112349	54659	57690	114414	58540	55874	110342	60651	49691
gr137	137	77958	34581	43377	78708	39321	39387	71372	40091	31281
pr144	144	77930	41118	36812	81792	40853	40939	72826	40729	32097
ch150	150	6677	3422	3255	6538	3262	3276	6516	2948	3568
kroA150	150	29177	14150	15027	29177	14150	15027	27155	9831	17324
kroB150	150	29231	15165	14066	28962	14579	14383	25458	8537	16921
pr152	152	104240	50497	53743	107173	53672	53501	101006	52071	48935
u159	159	47642	24134	23508	39852	19803	20049	45777	25797	19980
si175	175	22658	11237	11421	22666	11386	11280	21958	5146	16812
rat195	195	1881	898	983	2023	985	1038	1833	954	879
d198	198	24774	12647	12127	24399	12226	12173	24285	15347	8938

Tabela B.5: Soluções Algoritmos de Inserção com 2 caixeiros

Apêndice B Resultados

Grafo	$\|V\|$	By Node	p_1	p_2	By Cost	p_1	p_2	Min	p_1	p_2
kroA200	200	36051	17722	18329	31162	15572	15590	31180	19383	11797
kroB200	200	30572	14962	15610	33658	16879	16779	29258	15196	14062
gr202	202	36952	15575	21377	39029	20262	18767	38481	28474	10007
ts225	225	131762	63462	68300	135619	67687	67932	127216	64474	62742
tsp225	225	3856	1823	2033	3240	1646	1594	3774	1765	2009
pr226	226	118543	48290	70253	114861	53505	61356	115996	37447	78549
gr229	229	129174	63965	65209	125821	63651	62170	116468	71663	44805
gil262	262	1914	1048	866	2070	1016	1054	1662	851	811
pr264	264	68780	35658	33122	70448	35461	34987	64484	31505	32979
a280	280	1902	981	921	1923	959	964	1860	904	956
pr299	299	48968	22212	26756	47765	23887	23878	49202	34506	14696
rd400	400	16177	8471	7706	14868	7385	7483	14639	7873	6766
fl417	417	17074	8238	8836	17184	8607	8577	15208	5808	9400
gr431	431	108701	43032	65669	92126	45908	46218	85647	46749	38898
pr439	439	129839	65880	63959	135714	68736	66978	116651	63496	53155
pcb442	442	44433	20309	24124	45083	22378	22705	44798	21580	23218
d493	493	32410	17060	15350	32999	16077	16922	32317	10991	21326
att532	532	27283	13067	14216	28309	14173	14136	27946	20099	7847
ali535	535	172388	98954	73434	168096	83089	85007	166124	64703	101421
si535	535	50000	24140	25860	50183	25070	25113	48877	47251	1626
pa561	561	2661	1485	1176	2763	1404	1359	2591	2297	294
u574	574	33030	16873	16157	31716	15796	15920	31032	13752	17280
rat575	575	4668	2282	2386	4537	2332	2205	4763	1814	2949
p654	654	49838	22730	27108	51816	26072	25744	43479	23352	20127
d657	657	44031	24167	19864	43945	21283	22662	40705	11962	28743
gr666	666	178917	40083	138834	82656	40213	42443	72112	40238	31874
u724	724	35356	16116	19240	31850	15869	15981	27393	12816	14577
rat783	783	6290	2494	3796	5942	2970	2972	5554	2747	2807

B.3. Tabelas MOTSP

	BF			MONN			EV		
Instance	nº Sol	t	Rótulos	nº Sol	t	Rótulos	nº Sol	t	Rótulos
seed1.txt	4	0	52	3	0	10	3	0	10
seed2.txt	2	0	47	1	0	8	1	0	8
seed3.txt	5	0	51	4	0	25	4	0	25
seed4.txt	1	0	43	1	0.015	8	1	0	8
seed5.txt	3	0.016	46	3	0	13	3	0	13
seed6.txt	2	0	42	1	0	11	1	0	10
seed7.txt	3	0	55	3	0	12	3	0.015	12
seed8.txt	2	0.015	49	1	0	5	1	0	5
seed9.txt	7	0	53	2	0.016	12	2	0	12
seed10.txt	4	0	53	1	0	5	1	0	5
seed11.txt	4	0.016	50	1	0	5	1	0.016	5
seed12.txt	2	0	44	2	0	10	2	0	10
seed13.txt	2	0	44	1	0	11	1	0	11
seed14.txt	3	0	47	2	0	14	2	0	14
seed15.txt	3	0	47	1	0.016	8	1	0	8
seed16.txt	2	0	44	2	0	7	2	0	7
seed17.txt	5	0	52	1	0	5	1	0	5
seed18.txt	2	0	48	2	0.016	10	2	0.015	10
seed19.txt	4	0	54	2	0	7	2	0	7
seed20.txt	2	0	48	1	0	9	1	0	9

Tabela B.6: Soluções MOTSP para $|V| = 5$, $k = 2$, $\alpha = 0.1$

	BF			MONN			EV		
Instance	nº Sol	t	Rótulos	nº Sol	t	Rótulos	nº Sol	t	Rótulos
seed1.txt	14	0.234	12012	5	0.015	257	4	0	227
seed2.txt	12	0.359	16402	4	0.016	612	5	0.016	481
seed3.txt	16	0.375	17278	5	0.016	472	4	0	340
seed4.txt	10	0.344	16524	6	0.016	473	5	0.016	371
seed5.txt	20	0.375	17203	8	0.016	971	6	0.031	720
seed6.txt	17	0.328	15054	7	0.015	947	5	0.031	718
seed7.txt	15	0.406	18799	9	0.015	423	4	0.016	352
seed8.txt	13	0.297	14224	4	0.016	393	3	0	340
seed9.txt	13	0.328	14695	6	0.015	602	4	0.016	509
seed10.txt	22	0.344	15813	7	0.016	275	4	0.015	217
seed11.txt	16	0.343	16287	4	0.015	526	3	0.016	417
seed12.txt	9	0.281	13976	7	0.031	1184	6	0.016	977
seed13.txt	13	0.297	13938	9	0.032	846	5	0.015	691
seed14.txt	9	0.344	15971	6	0.031	1244	5	0.031	1048
seed15.txt	11	0.297	14936	7	0.016	928	7	0.016	740
seed16.txt	7	0.281	13632	5	0.016	469	4	0.015	410
seed17.txt	18	0.328	15283	15	0.047	2213	5	0.031	1478
seed18.txt	21	0.359	16589	16	0.031	1634	11	0.015	1232
seed19.txt	14	0.359	16800	7	0.015	579	5	0.016	436
seed20.txt	26	0.297	14629	11	0.016	769	5	0.015	598

Tabela B.7: Soluções MOTSP para $|V| = 10$, $k = 2$, $\alpha = 0.1$

	BF			MONN			EV		
Instance	nº Sol	t	Rótulos	nº Sol	t	Rótulos	nº Sol	t	Rótulos
seed1.txt	81	106.86	2100974	17	1.906	89677	9	1.125	42864
seed2.txt	62	176.021	2856822	37	2.516	107766	13	1.328	47952
seed3.txt	55	93.967	1931038	27	1	48582	13	0.625	25001
seed4.txt	59	129.278	2408468	31	2.547	111368	11	1.391	50237
seed5.txt	77	161.317	2686384	43	9.944	344611	15	4.162	128181
seed6.txt	37	113.42	2254357	16	3.066	136263	8	1.782	64120
seed7.txt	91	97.213	1970747	23	1.594	71936	15	0.828	29987
seed8.txt	28	111.411	2106483	25	0.562	29163	5	0.359	14729
seed9.txt	39	115.307	2136795	20	3.25	132940	9	1.481	52788
seed10.txt	43	125.211	2301168	25	3.365	136163	13	1.766	61901
seed11.txt	60	171.419	2818521	42	9.439	341079	15	4.281	130570
seed12.txt	44	109.21	2041983	29	2.078	93809	12	1.094	41422
seed13.txt	53	137.62	2450931	29	3.594	143313	12	1.705	57951
seed14.txt	53	134.503	2371156	40	4.805	196852	10	2.437	85931
seed15.txt	45	78.72	1675396	13	0.609	31893	5	0.407	17850
seed16.txt	58	101.236	2007250	23	2.047	92980	10	1.155	43403
seed17.txt	58	116.057	2167163	25	2.25	100037	10	1.187	43836
seed18.txt	49	86.647	1817997	12	2.151	100019	5	1.318	49687
seed19.txt	69	101.983	2010958	25	1.421	65637	8	0.753	28691
seed20.txt	90	93.091	1911111	36	1.656	76400	11	0.86	34205

Tabela B.8: Soluções MOTSP para $|V| = 15$, $k = 2$, $\alpha = 0.1$

	MONN			EA		
Instance	nº Sol	t	Rótulos	nº Sol	t	Rótulos
seed1.txt	63	1372.49	25739671	12	282.69	5619376
seed2.txt	62	1659.45	29598246	18	367.758	6906883
seed3.txt	44	705.196	15639062	9	180.825	3867410
seed4.txt	37	179.176	4949546	7	59.741	1599159
seed5.txt	32	643.906	13455788	12	166.48	3426370
seed6.txt	55	1142.5	23905735	13	339.71	6478876
seed7.txt	50	645.412	14190299	16	189.841	3882087
seed8.txt	63	399.914	9295651	16	105.003	2224887
seed9.txt	41	193.606	4924032	8	56.891	1329852
seed10.txt	46	574.279	12901364	17	168.067	3699665
seed11.txt	61	394.295	8960989	12	90.979	2094408
seed12.txt	62	889.919	18343165	16	225.152	4626897
seed13.txt	48	596.227	12666391	16	172.631	3645666
seed14.txt	61	413.972	8960864	12	93.291	2093261
seed15.txt	56	946.532	18331210	14	234.307	4609960
seed16.txt	58	440.343	10061811	10	121.686	2682494
seed17.txt	43	784.804	16503620	13	218.244	4389691
seed18.txt	49	935.626	20050140	13	299.415	5619286
seed19.txt	57	865.316	18133071	14	232.787	4514218
seed20.txt	59	1550.92	29243977	15	365.981	7006439

Tabela B.9: Soluções MOTSP para $|V| = 20$, $k = 2$, $\alpha = 0.1$

	BF			MONN			EV		
Instance	nº Sol	t	Rótulos	nº Sol	t	Rótulos	nº Sol	t	Rótulos
seed1.txt	14	0	59	11	0	40	10	0.016	39
seed2.txt	6	0	58	2	0.016	15	3	0	15
seed3.txt	7	0	51	3	0.015	17	3	0	17
seed4.txt	5	0	56	5	0.016	27	5	0	26
seed5.txt	8	0.016	52	5	0	23	5	0	22
seed6.txt	6	0	51	3	0	20	4	0	20
seed7.txt	11	0	60	3	0.016	20	3	0	19
seed8.txt	3	0	56	3	0.015	23	2	0.015	21
seed9.txt	8	0	57	4	0	29	4	0	29
seed10.txt	9	0.015	54	6	0	31	5	0	31
seed11.txt	7	0.016	56	6	0	20	5	0	19
seed12.txt	3	0	49	2	0	9	2	0	9
seed13.txt	4	0.015	52	3	0	44	3	0	42
seed14.txt	4	0	55	3	0	21	3	0	20
seed15.txt	4	0	51	2	0	15	2	0	15
seed16.txt	7	0	57	4	0	18	3	0	17
seed17.txt	7	0	59	6	0	30	5	0	28
seed18.txt	6	0	58	2	0	10	2	0	10
seed19.txt	8	0	58	5	0	22	5	0	23
seed20.txt	5	0	57	3	0	17	3	0	17

Tabela B.10: Soluções MOTSP para $|V| = 5$, $k = 3$, $\alpha = 0.2$

	BF			MONN			EV		
Instance	nº Sol	t	Rótulos	nº Sol	t	Rótulos	nº Sol	t	Rótulos
seed1.txt	116	0.89	29408	71	0.141	6418	12	0.047	1625
seed2.txt	249	2.594	54657	134	0.422	14472	20	0.078	2694
seed3.txt	158	1.548	41877	65	0.14	6267	15	0.047	1602
seed4.txt	147	1.766	44759	86	0.156	6980	14	0.046	1727
seed5.txt	214	2.59	56086	206	0.722	20554	16	0.093	3189
seed6.txt	157	1.73	43857	95	0.313	11200	12	0.063	2097
seed7.txt	210	2.062	48171	160	0.563	17079	14	0.078	2742
seed8.txt	101	1.234	38077	52	0.19	7809	10	0.047	1644
seed9.txt	145	1.875	46226	106	0.656	21376	15	0.094	3483
seed10.txt	178	1.594	42500	98	0.281	10538	12	0.062	2315
seed11.txt	123	1.312	38336	63	0.094	4264	8	0.015	967
seed12.txt	94	1.109	34603	51	0.187	9035	7	0.047	2125
seed13.txt	99	0.968	32774	65	0.172	7761	8	0.062	2115
seed14.txt	228	3.412	62731	179	1.109	26823	22	0.109	4013
seed15.txt	35	0.797	28770	28	0.109	6385	10	0.047	1872
seed16.txt	131	1.375	39725	87	0.218	9621	13	0.063	2490
seed17.txt	186	1.729	44754	162	0.531	17632	13	0.078	2902
seed18.txt	133	1.391	39966	106	0.484	18247	12	0.094	3347
seed19.txt	162	1.491	40121	102	0.407	14872	14	0.094	3008
seed20.txt	78	0.781	28169	53	0.063	3236	7	0.016	1076

Tabela B.11: Soluções MOTSP para $|V| = 10$, $k = 3$, $\alpha = 0.2$

	BF			MONN			EV		
Instance	nº Sol	t	Rótulos	nº Sol	t	Rótulos	nº Sol	t	Rótulos
seed1.txt	626	2587.97	11579143	473	681.085	4688173	17	8.515	184772
seed2.txt	957	3452.43	12697345	732	653.882	4114094	24	6.512	143646
seed3.txt	497	1406.18	8559007	297	110.286	1411775	15	2.893	73087
seed4.txt	879	3076.71	12824892	618	548.64	4198493	29	7.511	162373
seed5.txt	493	2586.21	11780935	393	655.344	4692011	23	8.527	173754
seed6.txt	1130	4514.91	14051581	790	1007.98	5687088	31	8.448	163079
seed7.txt	503	1050.84	7182303	388	168.803	2029285	15	3.963	89426
seed8.txt	845	2858.24	12152675	666	639.025	4573234	29	6.593	147257
seed9.txt	576	2215.75	10534998	410	353.096	3097051	15	4.579	106462
seed10.txt	815	3141.08	12507449	590	587.388	4317100	28	7.302	160525
seed11.txt	832	4233.09	14034336	645	1136.33	6106144	20	9.264	188811
seed12.txt	566	1790.47	8912901	379	278.257	2798666	14	5.381	127940
seed13.txt	1180	4005.75	14156669	821	832.503	4918383	29	8.535	182687
seed14.txt	648	2494.34	11291332	404	401.847	3252449	27	5.347	130141
seed15.txt	653	2015.88	9894579	458	202.334	1881950	14	3.087	78308
seed16.txt	786	2464.71	11403301	611	645.289	4530440	25	6.977	156737
seed17.txt	785	2658.25	11724688	483	329.794	2722590	21	5.113	122585
seed18.txt	355	1707.16	9777328	261	198.558	2350696	21	5.453	132243
seed19.txt	649	1968.73	10203013	420	239.362	2389176	21	3.75	89375
seed20.txt	567	1569.41	9176027	401	274.908	2780732	23	6.267	146816

Tabela B.12: Soluções MOTSP para $|V| = 15$, $k = 3$, $\alpha = 0.2$

	EA		
Instance	nº Sol	t	Rótulos
seed1.txt	33	470.257	6650529
seed2.txt	37	639.983	8567759
seed3.txt	33	447.283	6588877
seed4.txt	32	362.461	5583777
seed5.txt	23	272.723	4383328
seed6.txt	40	622.542	8492594
seed7.txt	38	476.732	6379199
seed8.txt	22	441.029	6572453
seed9.txt	33	269.567	4173486
seed10.txt	46	615.873	8600681
seed11.txt	34	381.778	6007815
seed12.txt	32	573.831	8012379
seed13.txt	47	619.077	8676964
seed14.txt	29	376.65	5931422
seed15.txt	40	575.109	8024352
seed16.txt	35	435.716	6042940
seed17.txt	39	616.938	7999692
seed18.txt	28	809.861	9701027
seed19.txt	37	659.661	8306765
seed20.txt	32	648.73	8186915

Tabela B.13: Soluções MOTSP para $|V| = 20$, $k = 3$, $\alpha = 0.2$

Bibliografia

[1] Agarwala,R., Applegate,D.L., Maglott,D., Schuler,G.D. and Schaffer A.A, 2000 A fast and scalable radiation hybrid map construction and integration strategy. *Genome Research 10* 350–364.

[2] Ali, A.I., Kennington, J.L., 1986. The asymmetric m-traveling salesmen problem: a duality based branch-and-bound algorithm. *Discrete Applied Mathematics 13* 259–276.

[3] Angel, R. D., Caudle, W. L., Noonan, R., Whinston, A., 1972. Computed-Assisted School Bus Scheduling *Management Science Vol. 18* 279–288.

[4] Applegate, D.L., Bixby, R.E., Chvátal, V., Cook, W., 2007 The Traveling Salesman Problem: A Computational Study. *Princeton University Press, Princeton, NJ*

[5] Applegate, D.L., Bixby, R.E., Chvátal, V., Cook, W., Espinoza, D.G., Goycoolea, M., Helsgaun, K., 2009 Certification of an optimal TSP tour through 85,900 cities. *Operations Research Letters 37* 11–15

[6] Ascheuer, N., Grötschel, M., Hamid, A., 1981 Order Picking in an Automatic Warehouse: Solving Online Asymmetric TSPs *Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB), Takustr. 7, D–14195 Berlin Dahlem.*

[7] Bailey, C., McLain, T., Beard, R., 2000. Fuel saving strategies for separated spacecraft interferometry *Proceedings of the AIAA Guidance, Navigation, and Control Conference, Denver*

[8] Bektas, T., 2005. The multiple traveling salesman problem: an overview of formulations and solution procedures. *Omega 34* 209–219.

[9] Bektas, T., Kara, I., 2005. Integer linear programming formulations of multiple salesman problems and its variations. *European Journal of the Operations Research 174* 1449–1458.

[10] Bellmore, M., Hong, S., 1974. Transformation of multisalesmen problem to the standard traveling salesman problem. *Journal of Association for Computing Machinery 21* 500–504.

[11] Bland, R.G., Shallcross, D.F. 1989. Large traveling salesman problems arising experiments in X-ray crystollography: A preliminary report on computation *Operations Research Letters 8* 125–128.

[12] Bolanosa, R.I., Echeverrya, M.G., Escobarb, J.W., 2015. A multiobjective non-dominated sorting genetic algorithm (NSGA-II) for the Multiple Traveling Salesman Problem. *Decision Science Letters 4*.

[13] Bonomi, E., Lutton, J.L., 1984. The N-city travelling salesman problem: Statistical mechanics an the Metropolis algorithm. *SIAM Review 26* 551–568.

[14] Calvo, R.W., Cordone, R., 2002. A heuristic approach to the overnight security service problem. *Computers & Operations Research 30* 1269–1287.

[15] Chang, T.S., Yen, H.M., 2012. City-courier routing and scheduling problems. *European Journal of Operational Research 223* 489–498.

[16] Christofides, N., 1976. Worst-Case Analysis of a New Heuristic for the Travelling Salesman Problem. *Report 388, Graduate School of Industrial Administration, Carnegie-Mellon University.*

[17] Chelouah, R., Saleh, H.A., 2003. The design of the global navigation satellite system surveying networks using genetic algorithms. *Engineering Applications of Artificial Intelligence 17* 111–122.

[18] Dantzig, G., Fulkerson, R., Johnson, S., 1954. Solution of a large-scale traveling-salesman problem. *Operations Research 2* 393–410.

[19] Fischer, R., Richter, k., 1982. Solving a multiobjective traveling salesman problem by dynamic programming. *Mathematische Operationsforschung und Statistik, Series Optimization 13* 247–252.

[20] Fox, K., Gavish, B., Graves, S.C., 1980. An n-constraint formulation of the (time-dependant) traveling salesman problem. *Operations Research 28* 1019–1021.

[21] Frederickson, G.N., Hecht, M.S., Kim, C.E., 1978. Approximation algorithms for some routing problems. *SIAM Journal on Computing 7* 178–193.

[22] Garey, M.R., and Johnson, D.S., 1979. Computers and intractability: A guide to the theory of NP-completeness. *Freeman, San Francisco.*

[23] Gavish, B., Srikanth, K., 1986. An optimal solution method for large-scale multiple traveling salesman problems. *Operations Research 34* 698–717.

[24] Gilbert, K.C., Hofstra, R.B., 1992. A New Multiperiod Multiple Traveling Salesman Problem with Heuristic and Application to a Scheduling Problem. *Decision Sciences 23* 250–259.

[25] Glover, F., 1989. Tabu Search – Part I. *ORSA Journal on Computing 1* 190–206.

[26] GuoXing, Y., 1995. Transformation of multidepot multisalesmen problem to the standard travelling salesman problem. *European Journal of Operational Research 81* 557–560.

[27] Huckfeldt, V.E., Svestka, J.A, 1973. Computational experience with an m-salesman traveling salesman algorithm *Management Science 19* 790–799.

[28] Kernighan, B.W., Lin, S., 1973. An effective heuristic algorithm for the traveling salesman problem. *Operations Research 21* 498–516.

[29] Kruskal, J.B., 1996. On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem. *Proceedings of the American Mathematical Society 7* 48–50.

[30] Kumar, R., Li, H., 1996. On Asymmetric TSP: Transformation to Symmetric TSP and Performance Bound. *Department of Electrical Engineering, University of Kentucky*

[31] Labadie, N., Melechovsky, J., Prins, C., 2014. An Evolutionary Algorithm with Path Relinking for a Bi-objective Multiple Traveling Salesman Problem with Profits. *Applications of Multi-Criteria and Game Theory Approaches* 195–223.

[32] Laporte, G., 1991. The Traveling Salesman Problem: An overview of exact and approximate algorithms. *European Journal of Operational Research 59* 231–247.

[33] Laporte, G., Nobert, Y., 1980. A cutting planes algorithm for the m-salesman problem. *Journal of the Operations Research Society 31* 1017–1023.

[34] Laporte, G., Nobert, Y., Taillefer, S., 1988. Solving a family of multi-depot vehicle routing and location-routing problems. *Transportation Science 22* 161–172.

[35] Lust, T., Teghem, J., 2010. The Multiobjective Traveling Salesman Problem: A Survey and a New Approach. *Studies in Computational Intelligence 272* 119–141.

[36] Menger, K., 1932. Das botenproblem. *Ergebnisse eines Mathematischen Kolloquiums 2* 11–12.

[37] Miler, C.E., Tucker, A.W., Zemlin, R.A., 1960. Integer programming formulations and traveling salesman problems. *Journal of the Association for Computing Machinery 7* 326–329.

[38] Mühlenbein, H., Gorges-Schleuter, M., Krämer, O., 1988. Evolution algorithms in combinatorial optimization. *Parallel Computing 7* 65–85.

[39] Oliveira, A., 2015. O Problema do Caixeiro Viajante. *Seminário em Estatística, Otimização e Matemática Financeira.*

[40] Papadimitriou, C.H., 1977. The euclidean travelling salesman problem is NP-Complete. *Theoretical Computer Science 4* 237–244.

[41] Papadimitriou, C.H., Steiglitz, K., 1982. Combinatorial optimization: algorithms and complexity. *Prentice-Hall, Inc. Upper Saddle River, NJ.*

[42] Prim, R.C., 1957. Shortest Connection Networks And Some Generalizations. *Bell System Technical Journal 36* 1389–1401.

[43] Rao, M.R., 1980. A note on Multiple Travelling Salesmen Problem. *Operations Research 28-3* 628–632.

[44] Rosenkrantz, D.J., Stearns, R.E., Lewis, P.M., 1977. An analyses of several heuristics for the traveling salesman problem. *SIAM Journal on Computing 6* 563–581.

[45] Ryan, J.L., Bailey, T.G., Moore, J.T., Carlton, W.B., 1998. Reactive Tabu search in unmanned aerial reconnaissance simulations. *Simulation Conference Proceedings, Winter* 873–880.

[46] Shim, V. A., Tan, K. C., Tan, K. K., 2012. A hybrid estimation of distribution algorithm for solving the multi-objective multiple traveling salesman problem. *Evolutionary Computation (CEC)* 1–8.

[47] Sofge, D., Schultz, A., De Jong, K., 2002. Evolutionary Computational Approaches to Solving the Multiple Traveling Salesman Problem Using a Neighborhood Attractor Schema. *Lecture notes in computer science 2279* 51–60.