UNIVERSIDADE ABERTA

O Problema do Mercador Viajante

João Carlos Rodrigues Marques

Dissertação para obtenção de Grau de Mestre em

Tecnologias e Sistemas Informáticos Web

orientada pelo
Professor Doutor José Pedro Fernandes da Silva Coelho

2016

Resumo

O propósito deste trabalho é de dar a conhecer o Problema do Mercador Viajante (Travelling Purchaser Problem – TPP), apresentado-o e resolvendo-o. O TPP tem uma lista de mercados e uma lista de produtos. As distâncias entre mercados são dadas, assim como os preços de cada produto em cada mercado. Pretende-se encontrar um trajeto onde seja possível adquirir todos os produtos, sendo esse trajeto um conjunto ordenado de mercados. O objetivo é encontrar o trajeto mais curto e ao mesmo tempo mais barato, por outras palavras, onde os produtos se adquirem aos seus menores preços.

Existem duas grandes dificuldades. A primeira é o facto de o TPP se tratar de um problema pertencente a *NP-Hard*. A segunda dificuldade reside no facto de se pretender minimizar dois objetivos: a distância e o preço. Esses objetivos entram em conflito um com o outro. Em vez de se procurar uma solução ótima, deve procurar-se uma fronteira de eficiência ótima. Esse tipo de problemas é designado por bi-objetivo, pois ao melhorar-se um objetivo, quase sempre se piora o outro. Sendo assim, mantém-se um conjunto de soluções ótimas, onde existem algumas com boas distâncias e outras com bons preços, e apresenta-se esse conjunto como uma fronteira não dominada de soluções.

É usado um algoritmo *branch and bound* para encontrar as fronteiras de soluções ótimas. Aplicam-se alguns cortes básicos no espaço de resultados durante a execução do algoritmo exato para evitar percorrer soluções desnecessárias.

É ainda criada uma base de dados de soluções obtidas a partir do algoritmo exato truncado e de um algoritmo de escalada do monte.

Palavras-chave: Problema do Mercador Viajante, Complexidade Exponencial, Algoritmos, Otimização, Bi-objetivo

Abstract

The purpose of this work is to present and solve the Travelling Purchaser Problem. The TPP has a list of markets and a list of products. The distances between markets are given, and the prices of each product in each market as well. The aim is to find a path where it is possible to buy all the products, and that path being a set of ordered markets. The goal is to find the shortest path and, at the same time, the cheapest one, where the products can be bought at their minimum price among all the markets.

There are two major issues. The first one is the fact that the TPP is an *NP-Hard* problem. The second issue lies in the fact that both objectives should be minimized: the distance and the price. These objectives are in conflict with each other. Instead of searching for an optimal solution, we must search for an optimal efficient front. This kind of problems are referred as bi-objective, because improving one objective, almost always lowers the other. So, we keep a set of optimal solutions, where there are some with good distances and others with good prices, and we present this set as a non dominated front.

We use a *branch and bound* algorithm in order to find the non dominated fronts of optimal solutions. We apply some basic cuts in the search space during the exact algorithm, in order to avoid visiting unnecessary solutions.

We create a database of solutions obtained by the truncated exact algorithm and a Hill Climbing algorithm.

Keywords: Travelling Purchaser Problem, Exponential Complexity, Algorithms, Optimization, Bi-objective

Agradecimentos

Uma dissertação é um projeto muito pessoal que exige total dedicação. No entanto, apenas se conseguem alcançar certos objetivos se tivermos determinadas pessoas do nosso lado, cuja colaboração, ajuda e apoio são essenciais para alcançar o sucesso. Assim agradeço:

Ao meu orientador, Professor Doutor José Pedro Fernandes da Silva Coelho, que sempre me incentivou a abraçar este projeto desafiante. Esteve sempre disponível para me aconselhar, ensinar e orientar com uma enorme dedicação permitindo-me manter as expectativas altas.

Ao Professor Doutor Jorge Riera-Ledesma, que já escreveu bastantes artigos sobre o mesmo tema desta dissertação, e que me forneceu informações e dados dos seus trabalhos, por diversas vezes.

Aos professores e *staff* da Universidade Aberta que tanto me ensinaram e apoiaram em questões físicas durante provas presenciais.

À minha mãe, a mulher mais corajosa do Mundo, que dedicou a sua vida a cuidar de mim. A minha confidente, o meu pilar em bons e maus momentos. A sua força e determinação foram essenciais para fazer de mim a pessoa que sou. Quando passo uns minutos a conversar com ela, fico apto para mais umas boas horas de trabalho.

Ao meu pai, cuja paciência e carinho não têm limites, que se levanta de madrugada para me deitar após as minhas longas horas de trabalho no silêncio da noite, pois é quando me sinto mais produtivo. A sua enorme calma para lidar com dificuldades inerentes as minhas limitações físicas é inigualável.

Ao meu irmão, que, apesar de mais novo, já me ultrapassou em muitos aspetos da vida, e para o qual eu olho com orgulho, encarando-o como um exemplo a seguir. Ele sabe sempre como me ajudar, tanto em aspetos académicos, profissionais ou pessoais.

À minha doce avó Graça que passa os seus dias a poucos metros de mim, ajudando-me em tudo, o que preciso, desde necessidades básicas até digitalizar-me as folhas de estudo para eu ler confortavelmente no meu computador.

Ao meu tio Paulo, um elemento chave no meu crescimento enquanto homem e enquanto estudante. É uma referência em termos académicos e pessoais, pois é o irmão mais velho que nunca tive. Graças aos seus conselhos e motivação, ganhei mais ambição pelo estudo e pude chegar aqui hoje.

À minha amiga Lurdes Breda, que me indicou a Universidade Aberta como opção para estudar em regime de e-learning, abrindo-me as portas para anos de estudo nesta universidade.

Ao meu colega e grande amigo David Fernandes que foi meu companheiro de estudo inúmeras vezes, que está sempre disponível para me ajudar com toda a paciência e dedicação.

Aos meus tios Helena e Ramiro, assim como aos primos Sandra, Liliana, Luís, Helena, Carlos e João, por estarem sempre por perto apoiando-me e motivando-me a perseguir os meus sonhos.

Aos primos José e Irene, que são como avós para mim. À Rosa e ao Jeff, que tantas vezes me acompanharam em provas presenciais, e sem os quais esses dias teriam sido muito mais complicados.

Aos primos Saudade e José que fizeram parte do meu crescimento e que participaram ativamente nos meus primeiros anos escolares. Aos meus primos Toni e Joana, por serem os meus amigos de infância e ainda hoje me apoiarem em tudo.

Aos meus priminhos Hugo, Lara, Gabriel, Luísa e Patrícia que me alegram muito e para os quais quero ser um exemplo a seguir.

Obrigado a todos.

Índice de Conteúdos

Índice de Tabelas

Índice de Figuras

Lista de abreviaturas

B&B	Branch and bound
HC	Hill Climbing
MOND	Market ordering by nearest to depot (ordenação de mercados pelo mais próximo em relação ao depósito)
MONN	Market ordering by nearest neighbor (ordenação de mercados pelo vizinho mais próximo)
STMMT	Sequenciamento de trabalhos em máquinas multi-tarefas
TPP	Travelling Purchaser Problem (Problema do Mercador Viajante)
TSP	Travelling Salesman Problem (Problema do Caixeiro Viajante)
VRP	Vehicle Routing Problem (Problema de roteamento de veículos)

1. Introdução

O Problema do Caixeiro Viajante (TSP) é muito utilizado em logística e em tantas outras aplicações. No entanto, este tem sido amplamente estudado e trata-se de um modelo muito simplista das situações reais de transporte, onde entram mais fatores em jogo além da distância. Indo ao encontro desses problemas com múltiplos objetivos, decidi enfrentar o Problema do Mercador Viajante (TPP) que acrescenta uma variável ao TSP, tornando-se num problema de otimização bi-objetivo.

Existem inúmeros trabalhos onde o TPP é transformado num problema de uma variável, representando a soma da distância e do preço, regredindo-se assim a um problema de um objetivo, com uma solução única. Esta dissertação, tem como objetivo o estudo das duas variáveis separadamente, gerando, para cada instância, um conjunto de soluções ótimas.

Os principais objetivos deste trabalho são: desenvolver uma solução exata para este problema, que lide com a questão bi-objetivo; estudar formas de efetuar cortes simples, de modo a simplificar e acelerar o algoritmo exato; conseguir resolver até à otimalidade instâncias de reduzida dimensão; ponderar a utilização de abordagens híbridas, do algoritmo exato com uma meta-heurística.

As questões de investigação, essenciais são as seguintes: Até que dimensão de um problema é possível obter a fronteira de eficiência ótima? É possível melhorar a eficiência de uma meta-heurística com a utilização de um algoritmo exato?

No segundo capítulo deste trabalho, é feito um enquadramento teórico com algumas noções e definições matemáticas aplicadas nos capítulos subsequentes. É brevemente revisitada a combinatória e as suas definições mais comuns. Há também uma curta descrição de problemas de otimização multi-critério, onde constam noções importantíssimas para este trabalho. São ainda abordadas algumas meta-heurísticas e diversas heurísticas.

No capítulo 3, é descrito e formulado o TPP ao qual este trabalho se dedica. São clarificadas algumas noções do problema usando um exemplo ilustrativo e é descrito o estado de arte do TPP. Por fim, são enumeradas algumas aplicações deste problema na vida real.

No capítulo 4, é feita descrição detalhada da estrutura de dados e principais funções implementadas nestes algoritmos. São explicadas as variáveis de instância, a representação da solução, e ainda os métodos e funções mais importantes.

No quinto capítulo descreve-se o algoritmo exato implementado, os pré-processamentos realizados, algumas noções heurísticas aplicadas ao mesmo e os cortes básicos usados. Descreve-se detalhadamente a técnica de *branch and bound* usada no algoritmo exato, ilustrando com um exemplo.

O capítulo 6 apresenta e descreve os algoritmos aproximados desenvolvidos. Descreve-se o algoritmo exato truncado e as razões das decisões tomadas nos detalhes deste algoritmo. Além disso, mostra-se uma implementação de *Hill Climbing*, um algoritmo aproximado bastante eficaz, e descrevem-se os pormenores todos da execução deste algoritmo.

No sétimo capítulo, são mostrados os resultados dos algoritmos desenvolvidos. São descritos todos os passos do desenvolvimento, de modo a mostrar as melhorias obtidas com cada nova funcionalidade. Analisam-se os resultados das execuções dos algoritmos desenvolvidos e efetuam-se comparações pormenorizadas.

Por fim, o capítulo 8 aborda as conclusões tiradas e as potencialidades do uso deste trabalho para outros fins, tanto práticos como de continuação de investigação.

2. Enquadramento teórico

Este capítulo é dedicado à introdução de vários conceitos teóricos sobre os quais assentam os algoritmos desta dissertação. São apresentadas algumas noções de contagem e combinatória, para proporcionar uma melhor perceção da dimensão do problema tratado. É abordado também o tema da otimização multi-critério, pois essa é uma das grandes novidades deste trabalho em relação à literatura existente sobre este tema.

2.1. Combinatória

A combinatória é um ramo da matemática que estuda as diferentes formas de contar objetos de um determinado conjunto.

Exemplo 2.1: O João tem 4 camisolas e 2 pares de calças. De quantas formas pode vestir-se? Atribuem-se nomes às camisolas (A, B, C e D) e às calças (X e Y). Juntando uma camisola a um par de calças, obtém-se as seguintes indumentárias: {AX, BX, CX, DX, AY, BY, CY, DY}. Assim, existem 8 formas possíveis de o João escolher a roupa que vai vestir.

***Definição 2.1**:* O princípio fundamental da contagem estipula que um evento que ocorre em n situações independentes e sucessivas, tendo a primeira situação ocorrendo de m_1 maneiras, a segunda situação ocorrendo de m_2 maneiras e assim sucessivamente até a n-ésima situação ocorrendo de m_n maneiras, o número total de ocorrências é dado pelo produto $m_1 \times m_2 \times ... \times m_n$

Exemplo 2.2: Quantos caminhos existem entre 3 cidades denominadas A, B e C? Enumerando todos os caminhos possíveis {ABC, ACB, BAC, BCA, CAB, CBA} e verifica-se que são 6. Começa-se por escolher a primeira cidade a visitar. Existem 3 hipóteses de escolha: A, B ou C. De seguida, escolhe-se a segunda cidade a visitar de entre as duas que restaram, e por fim escolhe-se a última. Obtém-se assim o resultado: $3 \times 2 \times 1 = 6$

Definição 2.2: Uma permutação é uma bijeção de um conjunto S finito nele próprio. Se S tem n elementos, então tem $P(n) = n! = n \times (n-1) \times (n-2) \times ... \times 2 \times 1$ permutações.

Exemplo 2.3: Quantos caminhos de 2 cidades existem entre 4 cidades denominadas A, B, C e D? À semelhança do exemplo anterior, enumeram-se os casos todos: {AB, AC, AD, BA, BC, BD, CA, CB, CD, DA, DB, DC}. Existem 4 hipóteses de escolha para a primeira cidade a visitar e 3

hipóteses para a segunda cidade, logo o resultado é: $4 \times 3 = 12$

Definição 2.3: Um arranjo simples de *n* elementos tomados *p* a *p* é dado pelo número de subconjuntos ordenados de *p* elementos que se pode formar a partir do conjunto inicial de *n* elementos. Um arranjo é calculado da seguinte forma:

$$P(n,k) = n \times (n-1) \times (n-2) \times \ldots \times (n-k+1) = \frac{n!}{(n-k)!}$$

Exemplo 2.4: Quantos conjuntos de 2 cidades existem entre 4 cidades denominadas A, B, C e D? À semelhança do exemplo anterior, enumeram-se os casos todos: {AB, AC, AD, BC, BD, CD}. No entanto, constata-se que não é necessário contar os subconjuntos repetidos (escolher AB é o mesmo que escolher BA). Existem então 6 hipóteses de escolha em vez das 12 do exemplo anterior, pois dividiu-se por 2 (eliminando as repetições).

Definição 2.4: Uma combinação de *n* elementos tomados *p* a *p* é dada pelo número de subconjuntos não ordenados de *p* elementos que se pode formar a partir do conjunto inicial de *n* elementos. Uma combinação é calculada da seguinte forma:

$$\binom{n}{k} = \frac{n!}{k! \times (n-k)!}$$

2.2. Otimização multi-critério

A otimização de um problema consiste em procurar a sua melhor solução, dados determinados critérios. Segundo Riera (2002), a otimização pode ser vista como a disciplina que cobre todo o processo interativo de análise e desenho resultando num sistema ótimo.

Num problema de otimização multi-objetivo, há, normalmente, mais do que um objetivo que deve ser otimizado. Na maioria das vezes, esses objetivos entram em conflito entre si.

Exemplo 2.5: Temos 4 trabalhadores a pintar uma casa e prevê-se que a obra esteja terminada em 10 dias. Se aumentarmos o número de trabalhadores para 8, a previsão de fim dos trabalhos é de 5 dias. Como se pode verificar, existem objetivos em conflito. Tendo em conta que melhorar o tempo da obra implica aumentar o número de trabalhadores.

Por vezes, os objetivos podem ser transformados num único objetivo, simplificando um pouco o problema.

Exemplo 2.6: A cada trabalhador do exemplo 2.5, paga-se 50€ por dia. O custo da obra é de

1000€ aos quais são subtraídos 50€ por cada dia que a obra demora a ser concluída. Temos assim o problema expresso só com um custo, tornando-o mais simples, embora com menos sentido em relação à pergunta inicial.

Na maioria dos problemas, os objetivos não podem ser transformados num único. Então surge a necessidade de representar a solução ótima de outra forma.

Exemplo 2.7: Num concessionário, há 5 marcas de carros e pretende-se escolher o carro mais barato que tem menor consumo de combustível. Abaixo seguem a tabela dos preços (em Euros) e do consumo em $\frac{l}{100\,km}$ e o gráfico onde se pode verificar que a marca B é a mais barata, e a marca C é a que consome menos. No entanto, a marca E tem melhor preço que a B, e melhor consumo que a C. Assim, a marca E deve ser guardada como uma solução válida, pois tem algumas variáveis melhores que as marcas B e C. As marcas A e D nunca serão escolhidas, pois existem outras marcas com melhores preços e consumo de combustível em simultâneo.

Marca	**Preço**	**Consumo**
A	14400	6,8
B	11500	6,5
C	9800	8,4
D	13000	7,5
E	11000	7,9

Tabela 2.1: Preços e consumos em diferentes marcas de carros

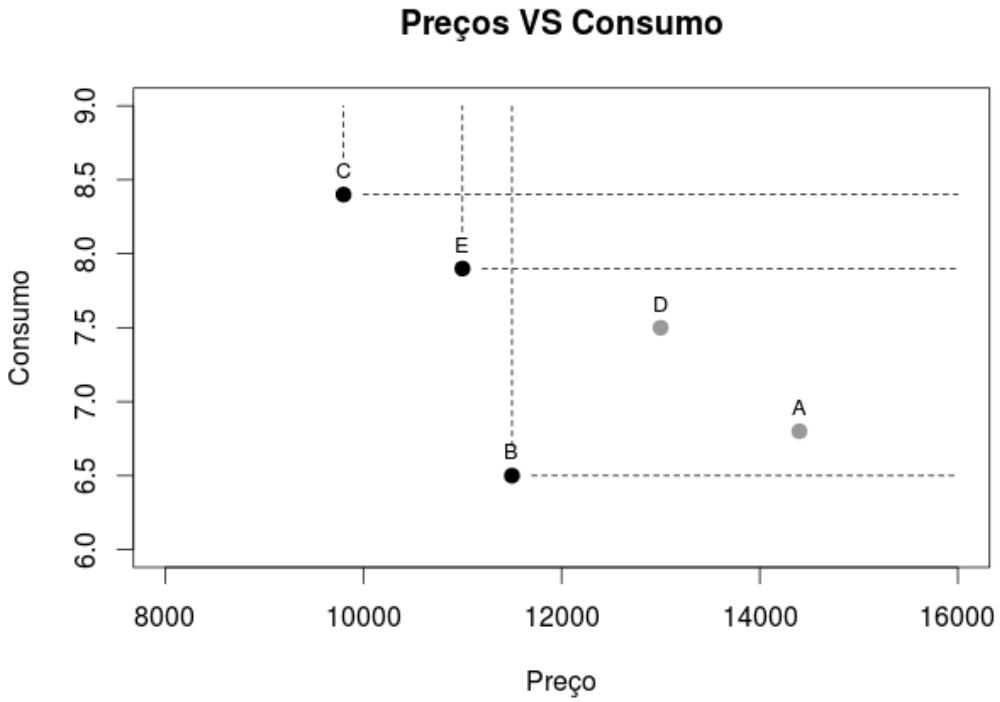

Figura 2.1: Gráfico de preços e consumos em diferentes marcas de carros

Tendo em conta os objetivos conflituosos, haverá soluções boas para um certo objetivo mas más para outro. Então, opta-se por representar o resultado da otimização usando uma fronteira não dominada de soluções.

Segundo Ehrgott (2006), a primeira referência a este tipo de situações com objetivos conflituosos é atribuída a Vilfredo Pareto, em 1896:

Os membros de um grupo auferem de um ganho máximo numa certa situação quando é impossível encontrar uma alteração que aumente o ganho global do grupo sem diminuir o ganho individual de nenhum dos elementos do grupo (Ehrgott, 2006 apud Pareto, 1896).

Definição: 2.5: Uma melhoria de Pareto é alcançada quando se consegue melhorar uma variável de uma solução sem piorar nenhuma das outras variáveis.

Definição: 2.6: Um ótimo de Pareto ocorre, quando não dá para melhorar nenhuma das variáveis sem piorar pelo menos uma outra variável. Por outras palavras, um ótimo de Pareto ocorre quando não se podem efetuar mais melhorias de Pareto.

Definição 2.7: Uma fronteira não dominada é um conjunto de soluções que são ótimos de Pareto.

No caso do exemplo 2.7, a fronteira não dominada de soluções é {B, C, E}, pois em qualquer uma das soluções, não dá para melhorar o preço sem piorar o consumo, e vice-versa.

2.3. Algoritmos exatos

Os problemas de otimização mais desafiantes são, na maioria, de complexidade exponencial. Estes são habitualmente resolvidos por algoritmos aproximados visto que o espaço de resultados é enorme, tornando muito difícil a análise de cada solução. No entanto, a busca por soluções exatas é extremamente importante. Face a essa necessidade, criaram-se alguns algoritmos que calculam soluções exatas para estes problemas.

A forma mais fiável de procurar uma solução exata de um determinado problema é percorrendo todas as soluções. Na maioria dos problemas, tal é impossível devido à enorme dimensão do espaço de resultados. Um algoritmo exato oferece a garantia de que todas as soluções são percorridas direta ou indiretamente. No entanto, para certos problemas, é possível desenhar algoritmos que são significativamente mais rápidos do que uma busca exaustiva (Woeginger, 2003).

Segundo Sörensen (2015), algoritmos exatos devem examinar cada solução do espaço de resultados de forma exaustiva, a menos que possam determinar explicitamente que uma certa solução (ou um conjunto delas) não necessitam de ser examinadas.

2.3.1. Branch and bound

O *branch and bound* (B&B) é um algoritmo exato de procura muito utilizado em problemas de otimização. Este consiste em percorrer o espaço de resultados como se de uma árvore se tratasse. É um algoritmo de profundidade que percorre as soluções todas usando exploração indireta na maioria dos casos.

Começa-se por escolher o tipo de ramificação e dividir a árvore de procura nos ramos desejados. Este passo é denominado de *branching* e é muito importante para proporcionar um bom equilíbrio entre os vários ramos. Convém que cada ramo tenha mais ou menos o mesmo número de soluções.

De seguida, é aplicada uma função de *bounding* que avalia a solução de uma forma mais relaxada, obtendo-se assim uma avaliação mais baixa do que a solução mínima encontrada neste ramo (no caso da função de avaliação estar a minimizar o problema). Essa solução relaxada é comparada com a melhor solução obtida até ao momento. Se for melhor, prossegue-se a procura

nesse ramo. Se for pior ou igual, todo esse ramo é descartado. O passo de descartar um ramo inteiro é chamado de *pruning*.

A função de *bounding* deve retornar uma aproximação quanto mais precisa melhor da solução atual, garantindo que não há soluções com valor mais baixo que esse, pois só assim se consegue efetuar um bom *pruning*, e por consequente, tornar o algoritmo mais eficiente.

Serão mostrados exemplos específicos de *branch and bound* aquando da descrição do algoritmo exato usado neste trabalho.

2.4. Meta-heurísticas

Uma meta-heurística é um método algorítmico, independente do problema a tratar, que providencia um conjunto de diretrizes ou estratégias para desenvolver algoritmos de otimização heurística (Glover, 1986). Não se trata de um algoritmo nem de uma sequência de ações que devem ser seguidas. Uma meta-heurística é um conjunto de ideias e conceitos consistentes, que podem ser usados para desenhar algoritmos de otimização heurística (Sörensen, 2015).

Por outro lado, uma heurística designa um método algorítmico específico ao problema a tratar.

Usando a metáfora de Sörensen (2015), pode-se dizer que uma meta-heurística é um "estilo culinário" e não uma receita específica. Uma heurística já se assemelha mais a uma receita em particular.

2.4.1. Escalada do Monte

A escalada do monte (*hill climbing*) é uma das meta-heurísticas mais simples, pois trata-se de uma técnica que percorre as soluções de forma iterativa na região da última solução visitada e adota as novas soluções se estas forem melhores que as anteriores. Este procedimento permite "escalar o monte" e alcançar um ótimo local (Sean, 2015).

Em termos de algoritmo, a escalada do monte é um ciclo que começa com uma solução arbitrária (ou escolhida a propósito) e tenta melhorá-la a cada iteração, através da sua vizinhança (soluções da região). O algoritmo é interrompido quando cumpre um certo critério de paragem ou quando nenhum vizinho é melhor do que a solução atual.

Este último critério de paragem faz com que a escalada do monte fique presa em ótimos locais e possa nunca alcançar a solução ótima. Não se desce em busca de um "pico" melhor e não se

guardam as soluções já visitadas.

Esta meta-heurística parece a busca do topo do Monte Evereste, em um dia de muito nevoeiro, por alguém que sofre de amnésia (Russel e Norvig, 2010).

No entanto, a sua simplicidade permite um desenvolvimento rápido e oferece resultados bastante satisfatórios.

2.4.2. Arrefecimento simulado

Um algoritmo como a escalada do monte que só "sobe" e nunca "desce", não explorando caminhos alternativos, está condenado a falhar quase sempre na busca da solução ótima.

Essencialmente, o arrefecimento simulado (*simulated annealing*) difere da escalada do monte na medida em que se a nova solução é pior que a atual, pode-se adotar a nova na mesma mediante uma certa probabilidade (Sean, 2015).

A introdução da aleatoriedade nesta técnica permite escapar dos locais ótimos e explorar caminhos que parecem maus, mas podem conter soluções boas mais à frente.

Esta meta-heurística surgiu nos anos 1980 sendo descrita por vários autores (Kirkpatrick et al., 1983).

O arrefecimento simulado fundamenta-se numa analogia com a termodinâmica. Basicamente, é usada uma metáfora de um processo térmico usado nas metalúrgicas, que consiste em duas etapas:

- a temperatura do metal é aumentada para uma temperatura muito alta, até este derreter;
- a temperatura é reduzida de forma controlada, proporcionando flexibilidade ao metal derretido e assim permitir que este seja moldado.

Se se deixar o metal arrefecer demasiado depressa, os seus átomos não conseguem tomar novas e boas posições, logo congelam em configurações instáveis. Se se arrefecer o metal lentamente, é dado aos átomos o tempo suficiente para estes se organizarem e formarem um sólido resistente (Sean, 2015).

2.4.3. Pesquisa tabu

A pesquisa tabu é uma meta-heurística de procura local criada por Fred Glover em 1986

(Glover, 1986) e formalizada pelo mesmo em (Glover, 1989) e (Glover, 1990).

É importante realçar que Glover não via a pesquisa tabu como uma heurística, mas sim como uma meta-heurística, uma estratégia geral para guiar e controlar heurísticas internas, talhadas especificamente para os problemas tratados (Glover & Kochenberger, 2003).

A pesquisa tabu fornece soluções muito próximas das soluções ótimas e está entre as meta-heurísticas mais eficientes, ou até a melhor para enfrentar os problemas mais complexos (Glover & Kochenberger, 2003).

Esta meta-heurística usa uma abordagem diferente das anteriores na medida em que mantém um histórico dos movimentos de vizinhança aplicados, evitando assim perder tempo a explorar soluções más, não ficando presa em ótimos locais (Glover & Kochenberger, 2003). O algoritmo recusa-se a visitar essas soluções a menos que estas já tenham sido visitadas há muito tempo (Sean, 2015). Esse histórico é denominado de lista tabu e pode ter vários níveis. Pode-se ter uma lista com memória de curta duração, que é limpa com alguma frequência, permitindo explorar novamente movimentos de vizinhança que pareceram ser maus no passado. Uma memória de longa duração também é útil, pois permite guardar as piores vizinhanças, aquelas que certamente não conduzem à solução ótima.

Quando se adota uma nova solução, esta é guardada na lista tabu. Se a lista estiver cheia, remove-se a solução mais antiga que pode ser visitada novamente (Sean, 2015).

A pesquisa tabu pode ser vista como uma técnica de escalada do monte que tenta evitar bloqueios em ótimo locais ao permitir a exploração de soluções piores do que a atual (Sels et al., 2015).

3. O Problema do Mercador Viajante

O Problema do Mercador Viajante, conhecido por "*Travelling Purchaser Problem*", ou simplesmente por TPP, foi proposto por Ramesh (1981) e é um problema combinatório, pertencente ao grupo *NP-Hard*.

O TPP é uma generalização do famoso Problema do Caixeiro Viajante ("*Travelling Salesman Problem*" – TSP).

No TSP, é dada uma lista de cidades e as distâncias entre elas. O problema consiste em percorrer todas as cidades numa determinada ordem e, por fim, regressar à cidade inicial, usando o caminho mais curto. Trata-se de minimizar o valor da função objetivo que dá a distância total.

O TPP é um pouco mais complexo na medida em que é dada uma lista de mercados, as distâncias entre os mercados, uma lista de produtos e os seus respetivos custos em cada um dos mercados. O problema consiste em adquirir todos os produtos de uma determinada lista percorrendo um subconjunto dos mercados, com uma distância mínima e um preço mínimo dos produtos. Trata-se de minimizar duas funções objetivo: a que dá a distância e a que dá o preço.

3.1. Formulação matemática

Usando notação idêntica à de Ramesh (1981), Singh & van Oudheusden (1997), e Riera & Salazar (2005a), define-se o TPP da seguinte forma.

Considera-se um conjunto de mercados $M=\{1,2,\ldots,m\}$, um mercado $0\notin M$ ao qual chamaremos depósito, pois este não vende produtos, e um conjunto de produtos $P=\{1,2,\ldots,p\}$. Também são dadas as distâncias entre cada par de mercados *i* e *j*, da seguinte forma d_{ij}, onde $0\le i\le m$, $0\le j\le m$. Ainda são dados os preços (custos) de cada produto *k* em cada mercado *i* na forma c_{ik}, onde $1\le k\le p$.

Para formalizar a notação, são necessários três matrizes lógicas *X* e *Y* onde:

- X_{ij} é igual a 1 se o mercado *i* antecede o mercado *j*, e é igual a 0 em caso contrário;
- Y_{ik} é igual a 1 se o produto *k* é adquirido no mercado *i*, e é igual a 0 em caso

contrário.

- Y^*_{ik} é igual a 1 se o produto *k* é adquirido ao seu menor preço no mercado *i*, e é igual a 0 em caso contrário.

É de lembrar que:

- é obrigatório adquirir todos os produtos;
- não é obrigatório percorrer todos os mercados.

Este problema tem imensas variantes, no entanto, neste trabalho será apenas abordada a mais simples, onde:

- as distâncias entre dois mercados *i* e *j* são iguais, independentemente do sentido da viagem;

$$d_{ij} = d_{ji} \tag{1}$$

- a lista de produtos exige a aquisição de apenas uma unidade por produto;

$$\sum_{i=1}^{m} Y_{ik} = 1, k = 1, 2, \ldots, p \tag{2}$$

- em cada mercado, estão disponíveis todos os produtos

$$\forall i \in M, \forall k \in P, \exists c_{ik} \geq 0 \tag{3}$$

- nenhum mercado vende todos os produtos ao preço mais baixo.

$$\sum_{i=1}^{m} Y^*_{ik} < p, k = 1, 2, \ldots, p \tag{4}$$

A restrição (1) indica que estamos perante o TPP simétrico e a condição (2) garante que se trata do TPP capacitado e unitário. É necessário visitar vários mercados para encontrar bons preços. Ao tentar-se minimizar o preço, aumenta-se a distância, e vice-versa. Portanto, estamos perante um problema de otimização bi-objetivo. O TPP não tem uma solução única, mas sim um conjunto de soluções ótimas, sob forma de uma fronteira não dominada.

Formalizemos então as funções objetivo do TPP. Seja f_d a função objetivo das distâncias e seja f_p a função objetivo dos preços, temos para uma solução s :

$$f_d(s) = \sum_{i=0}^{m} \sum_{j=0}^{m} d_{ij} X_{ij} \tag{5}$$

$$f_p(s) = \sum_{i=0}^{m} \sum_{k=1}^{p} c_{ik} Y_{ik} \quad (6)$$

A fronteira de soluções não dominadas é calculada através da minimização das funções objetivo f_d e f_p em simultâneo.

3.2. Exemplo ilustrativo

O TPP é um problema complexo e nada melhor do que um pequeno exemplo para ilustrar o que se pretende alcançar neste trabalho.

Foi criado um exemplo com três mercados e três produtos. Para evitar confusões com números, os mercados foram designados por A, B e C, onde X é o depósito. Os produtos são denominados de J, K e L.

Um caminho possível seria XACX, ou seja, partir do depósito, ir ao mercado A, ir ao mercado C e finalmente regressar ao depósito. Todos os caminhos começam e terminam no depósito, portanto facilmente se pode anotar o caminho anterior apenas com AC, pois está implícito que começa e termina em X.

Relembra-se que estamos perante o TPP simétrico, logo os caminhos AB e BA são iguais, podendo-se descartar o último. Assim, é fácil enumerar todos os caminhos possíveis: A, B, C, AB, AC, BC, ABC, ACB e BAC.

Abaixo seguem um grafo e uma tabela que representam a distância d_{ij} entre mercados, assim como uma tabela de preços simbolizando os valores c_{ik} .

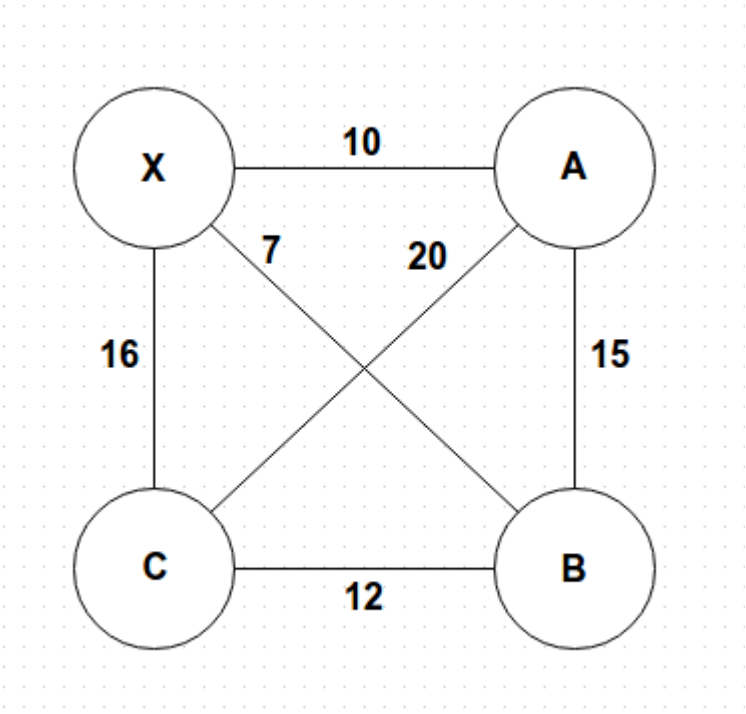

Figura 3.1: Distâncias entre os mercados e depósito do exemplo ilustrativo

Mercados	(X)	A	B	C
(X)	-	10	7	16
A	10	-	15	20
B	7	15	-	12
C	16	20	12	-

Tabela 3.1: Distâncias do exemplo ilustrativo

Mercados \ Produtos	J	K	L
A	3	7	1
B	5	2	4
C	1	2	6

Tabela 3.2: Preços do exemplo ilustrativo

Por exemplo, para o caminho AB, temos uma distância total de 10 + 15 + 7 = 32. No cálculo dos preços, é importante realçar novamente que, dentro dos mercados do caminho, cada produto é adquirido no mercado onde o seu preço é o mais baixo. No caso do produto J, este tem um custo de 3 no mercado A e um custo de 5 no mercado B. Assim, adquire-se o produto J no mercado A a um custo de 3. O produto K é adquirido no mercado B por um preço de 2 e o produto L é comprado em

A por 1 unidade monetária. Temos então um preço final de 3 + 2 + 1 = 6. A avaliação da solução AB é então de (32; 6).

Abaixo segue uma tabela com a avaliação de todas as soluções deste exemplo.

Caminho	Distância	Mercados onde cada produto foi adquirido ao seu menor preço	Preço
A	20	A: J(3), K(7), L(1)	11
B	14	B: J(5), K(2), L(4)	11
C	32	C: J(1), K(2), L(6)	9
AB	32	A: J(3), B: K(2), A: L(1)	6
AC	46	C: J(1), C: K(2), A: L(1)	4
BC	35	C: J(1), B/C: K(2), B: L(4)	7
ABC	53	C: J(1), B/C: K(2), A: L(1)	4
ACB	49	C: J(1), B/C: K(2), A: L(1)	4
BAC	58	C: J(1), B/C: K(2), A: L(1)	4

Tabela 3.3: Avaliação de todas as soluções do exemplo ilustrativo

Facilmente se percebe que as melhores soluções são B, AB e AC. A solução B tem apenas 14 de distância e mais nenhuma outra solução tem distância menor. Apesar do seu preço alto, trata-se de uma solução não dominada. Na solução AC, a situação é idêntica, mas desta vez temos uma distância alta e um preço baixíssimo não dominado. A solução AB é não dominada, pois tem melhor preço que B (mesmo tendo pior distância) e tem melhor distância que AC (mesmo tendo pior preço). A presença da solução AB na fronteira de soluções não dominadas mostra que algumas soluções representam um equilíbrio entre distância e preço.

A solução ACB é descartada pois a adição do mercado B não melhora nenhum dos preços. Após a passagem pelos mercados A e C, temos os preços (1, 2, 1). Os preços de B são (5, 2, 4). Nota-se que nenhum deles melhora os preços correntes, portanto, não é necessário passar pelo mercado B.

Na figura abaixo, temos uma representação gráfica destas soluções e nota-se perfeitamente que as três soluções indicadas são, de facto as melhores.

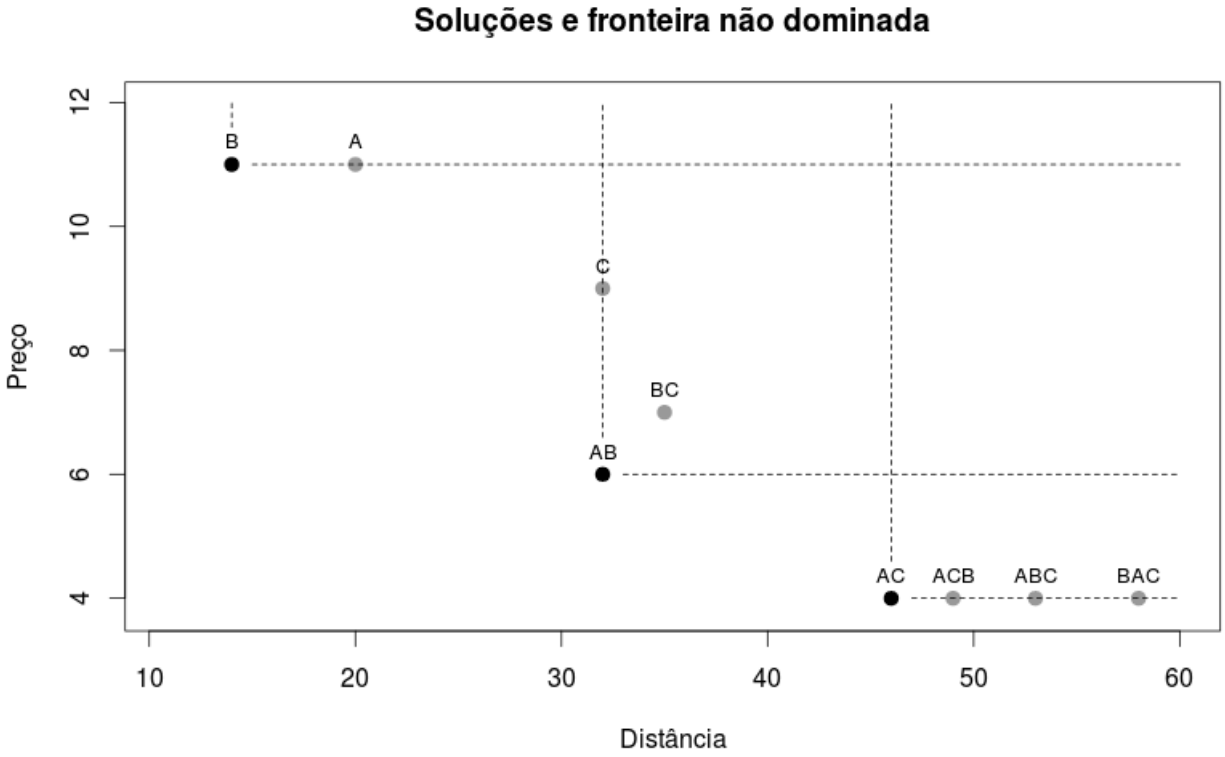

Figura 3.2: Representação gráfica das soluções do exemplo ilustrativo

Este exemplo ilustrativo será usado mais à frente na descrição dos algoritmos desenvolvidos.

3.3. Estado de Arte

Já em 1966 e 1971, foram analisados problemas com estrutura idêntica ao TPP, por Burstall e por Buzacott e Dutta, respetivamente. Tratavam-se de problemas de sequenciamento de trabalhos em máquinas multi-tarefas (Burstall, 1966) (Buzacott & Dutta, 1971), processo esse que será descrito mais adiante nas aplicações do TPP.

O TPP foi apresentado formalmente, com este nome, por Ramesh (1981) e tem sido alvo de imensos algoritmos heurísticos ao longo dos anos. A revisão literária permitiu-nos encontrar poucos algoritmos exatos. No entanto, a vasta maioria dos trabalhos efetuados sobre este TPP encaram-no como um problema de um objetivo: minimizar a função da soma das distâncias com os preços.

$$f_{d+p}(s)=\sum_{i=0}^{m}\sum_{j=0}^{m} d_{ij}X_{ik}+\sum_{i=0}^{m}\sum_{k=1}^{p} c_{ik}X_{ik} \quad (7)$$

Ramesh desenvolveu um algoritmo exato baseado numa procura léxica, onde cada solução é representada por um conjunto de símbolos e a busca pela solução ótima assemelha-se à procura de uma palavra num dicionário. Usando algumas técnicas de *lower bound*, este conseguiu apenas

resolver instâncias até 12 mercados e 10 produtos (Ramesh, 1981).

A abordagem de Singh & van Oudheusden já considera um algoritmo de *branch and bound* mais poderoso. Dividem o espaço de resultados em pequenos conjuntos usando a ramificação (*branching*) da seguinte forma. O primeiro ramo é composto por todas as soluções contendo um certo par de mercados (i, j), sendo o outro ramo composto por todas as soluções que não contêm o par de mercados (i, j). De seguida, o primeiro ramo é, por sua vez, ramificado da mesma forma, em dois novos ramos: um com as soluções que contêm o par de mercados (k, l) e o outro com as soluções sem esse par de mercados. O *bounding* é calculado usando um relaxamento nas posições dos mercados da solução. O algoritmo resolve instâncias até 25 mercados e 100 produtos (Singh & van Oudheusden, 1997).

Laporte, Riera e Salazar. propõem um algoritmo exato baseado numa formulação em programação linear inteira, ao qual é aplicado um algoritmo de *branch and cut*, onde o objetivo é ainda a minimização da soma das distâncias com os preços. No entanto, mostram bons resultados pois conseguem resolver instâncias até 200 mercados e 200 produtos (Laporte et al., 2003).

Passando para os artigos baseados em heurísticas, temos Pearn e Chien com uma melhoria do algoritmo de Ramesh (Pearn & Chien, 1998), uma heurística de perturbação (Boctor et al., 2003) e uma otimização por via de um algoritmo de colónias de formigas (Bontoux & Feillet, 2008).

Riera e Salazar continuam a fazer tentativas de melhorias através de procura local e de heurísticas onde a vizinhança é calculada com uma variedade de métodos diferentes. Assim, conseguem gerar soluções para problemas contendo até 350 mercados e 200 produtos (Riera & Salazar, 2005b).

El-Dean (2008) e Voß (1996) atacam o problema com a pesquisa tabu, que prova ser uma ferramenta muito poderosa neste tipo de problemas. Voß chega a obter bons resultados para instâncias de 60 mercados e 120 produtos.

Os algoritmos genéticos são mais uma abordagem muito usada na otimização do TPP. Existe um algoritmo genético que resolve instâncias contendo até 250 mercados e 200 produtos (Goldbarg et al., 2009). Ainda antes desse, houve um algoritmo genético e paralelo que abordou instâncias até 500 mercados e 500 produtos (Ochi et al., 1997).

Na vasta maioria dos casos, o problema é resolvido como se se tratasse da otimização de uma variável (distância + preço) em vez de se tentar encontrar um conjunto de boas soluções tanto a nível de distância como de preço (problema bi-objetivo).

Em 1998, Ravi e Salman abordam o TPP bi-critério, mas apenas com algoritmos de aproximação.

Riera foi o primeiro a realizar sérias tentativas de resolver o problema bi-objetivo, tanto na sua tese (Riera, 2002) como, em conjunto com Salazar, em (Riera & Salazar, 2005a). Estes obtiveram soluções ótimas para instâncias até 100 mercados e 200 produtos, fornecendo um enorme contributo para o avanço na investigação do TPP. No entanto, não foi possível encontrar resultados detalhados, tais como fronteiras não dominadas de soluções de cada instância.

Esses trabalhos de Riera datam de há cerca de 10 anos, e desde então têm continuado a surgir imensos artigos sobre aproximações ao TPP, sem tentativas de melhorar o trabalho deste autor.

3.4. Problemas relacionados

O TPP tem várias aplicações práticas, principalmente quando se consegue adaptá-lo para que este se assemelhe a outros problemas conhecidos.

3.4.1. Sequenciamento de trabalhos em máquinas multi-tarefas

Como já foi mencionado anteriormente, Burstall e depois Buzacott e Dutta (Burstall, 1966) (Buzacott & Dutta, 1971), (Riera & Salazar, 2005a) abordaram o problema de sequenciamento de trabalhos em máquinas multi-tarefas (STMMT). Este problema consiste no seguinte.

Seja *n* o número de trabalhos e *m* o número de máquinas multi-tarefas. Cada trabalho é tratado por uma máquina que tem de ser configurada para um estado capaz de o tratar. Temos uma lista de tempos d_{ij} que representa o tempo de mudança de uma certa máquina de um estado *i* para um estado *j*. Temos ainda uma lista p_k que representa o tempo de conclusão do trabalho *k*.

O problema consiste em encontrar uma ordem para os trabalhos que minimize o tempo total gasto pelas máquinas a correr os trabalhos.

Consideremos que o número de máquinas é o número de mercados do TPP, o número de trabalhos é o número de produtos, a lista de tempos d_{ij} é a distância entre dois mercados e a lista p_k é o preço de cada produto.

Temos uma correspondência praticamente exata com a semântica do TPP. Apenas é de notar que no STMMT temos dois objetivos temporais que podem, e devem ser somados. Portanto,

estamos perante o caso do TPP de um objetivo, onde se somam a distância e o preço finais.

3.4.2. Armazenamento em múltiplos locais

O problema de armazenamento (*warehousing*) consiste numa encomenda de um certo número de itens, que devem ser recolhidos num armazém e colocados na área de entrega (por exemplo, à porta do armazém). O objetivo é o de minimizar a distância percorrida para ir buscar e entregar os itens.

Numa situação onde os itens estão guardados numa única zona do armazém, este problema já foi reconhecido como sendo uma variante do TSP (Singh & van Oudheusden, 1997).

No entanto, se um item estiver guardado em várias zonas do armazém, o problema passa a ser semelhante ao TPP, na medida em que temos *m* zonas no armazém, temos *n* itens, temos uma matriz $D=\{d_{ij}\}$ que consiste nas distâncias entre as várias zonas do armazém, e uma matriz $P=\{p_{ik}\}\in\{0,+\infty\}$ onde p_{ik} é 0 se o item *k* estiver guardado na zona *i*, e é $+\infty$ em caso contrário (Singh & van Oudheusden, 1997).

3.4.3. Problema de roteamento do autocarro escolar

O Problema de roteamento de veículos, VRP (Vehicle Routing Problem), é também um problema de otimização que consiste no atendimento de clientes através de uma frota de *q* veículos. O objetivo é atender todos os clientes usando a frota de veículos, partindo de um depósito comum a todos e minimizando o custo dos caminhos a percorrer.

Uma variante deste é o problema de roteamento do autocarro escolar, que consiste em *m* paragens e *n* crianças a recolher. Temos ainda uma lista de tempos d_{ij} que representa o tempo de viagem entre as paragens *i* e *j*, e temos uma lista p_k que representa o tempo de viajem a pé que cada criança *k* demora desde a sua casa até a uma paragem.

O problema consiste em atribuir uma paragem a cada criança de modo a minimizar os dois tempos: tempo de viagem do autocarro entre paragens e tempo de viagem de cada criança entre a sua casa e a paragem designada.

Isto é claramente a estrutura do TPP onde cada paragem é considerada como um mercado e cada criança é representada por um produto.

Riera e Salazar abordam este problema usando múltiplos autocarros escolares e uma

generalização do TPP que envolve múltiplos veículos a partirem do mesmo depósito (Riera & Salazar, 2012).

3.5. Instâncias de teste

No âmbito desta dissertação, o termo “instância” designa um conjunto de dados iniciais, sobre os quais vão ser aplicados os algoritmo.

Neste trabalho, são usadas as instâncias de Singh e van Oudheusden (Singh & van Oudheusden, 1997) usadas também em (Riera, 2002), (Laporte et al., 2003) e (Riera & Salazar, 2005a). Assim, estas instâncias são adequadas para a comparação com outras obras sobre o TPP.

Este conjunto de instâncias contém 33 mercados com distâncias simétricas. O primeiro “mercado” é na verdade o depósito, que não vende nenhum produto. Os restantes 32 mercados são comuns e vendem os produtos de forma normal. Os preços dos produtos são gerados entre nove intervalos, nomeadamente de 1 até 5, 10, 50, 100, 500, 1000, 5000, 10000 ou 50000, de acordo com uma distribuição uniforme discreta. O número de produtos é de 50, 100, 150, 200, 250, 300, 350, 400, 450 e 500 e existem 5 instâncias de cada tipo. No total, temos um leque de 450 instâncias.

Por exemplo, na instância denominada “Singh33_2.33.50.1.5.tpp”, o prefixo “Singh33_2” é o nome do *dataset*. A estrutura do nome pode ser mapeada no seguinte “ Singh33_2.**M.P.X.Y**.tpp”, onde:

- M → número de mercados (incluindo o depósito);
- P → número de mercados;
- X → número da instância (visto que são geradas 5 de cada tipo (M, P e Y);
- Y → valor máximo para o preço de um produto.

3.6. Metodologia de investigação

Este trabalho tem uma forte componente prática, no sentido em que se procura implementar algoritmos que forneçam fronteiras de eficiência exatas para o TPP bi-objetivo. É necessário verificar se as soluções obtidas estão corretas, validando assim os algoritmos implementados.

A metodologia de investigação é o método que se utiliza para validar as hipóteses de investigação. Implementaram-se vários algoritmos, e efetuaram-se testes empíricos num conjunto

de instâncias de modo a obter dados minimamente generalizáveis.

A primeira questão é: como provar que uma determinada fronteira é, de facto, a fronteira de eficiência? Para enfrentar e ultrapassar esse problema, decidiu-se usar um pequeno exemplo ilustrativo (ver secção 3.2), resolvê-lo manualmente e usá-lo como base para verificar se as primeiras versões dos algoritmos estavam corretas. Assim que se criou uma primeira versão do algoritmo exato com *branch and bound*, foi possível testá-lo no exemplo ilustrativo, validá-lo e calcular fronteiras de eficiência para instâncias de dimensão um pouco maior.

Para certificar que a fronteira era de facto ótima, decidiu-se efetuar uma validação gráfica para uma instância de 7 mercados. Obtiveram-se centenas de soluções e representaram-se no plano cartesiano, na forma de pontos. A abcissa foi o valor da distância total, e a ordenada foi o valor do preço total. Os pontos vermelhos (borda inferior esquerda) são as solução da fronteira ótima.

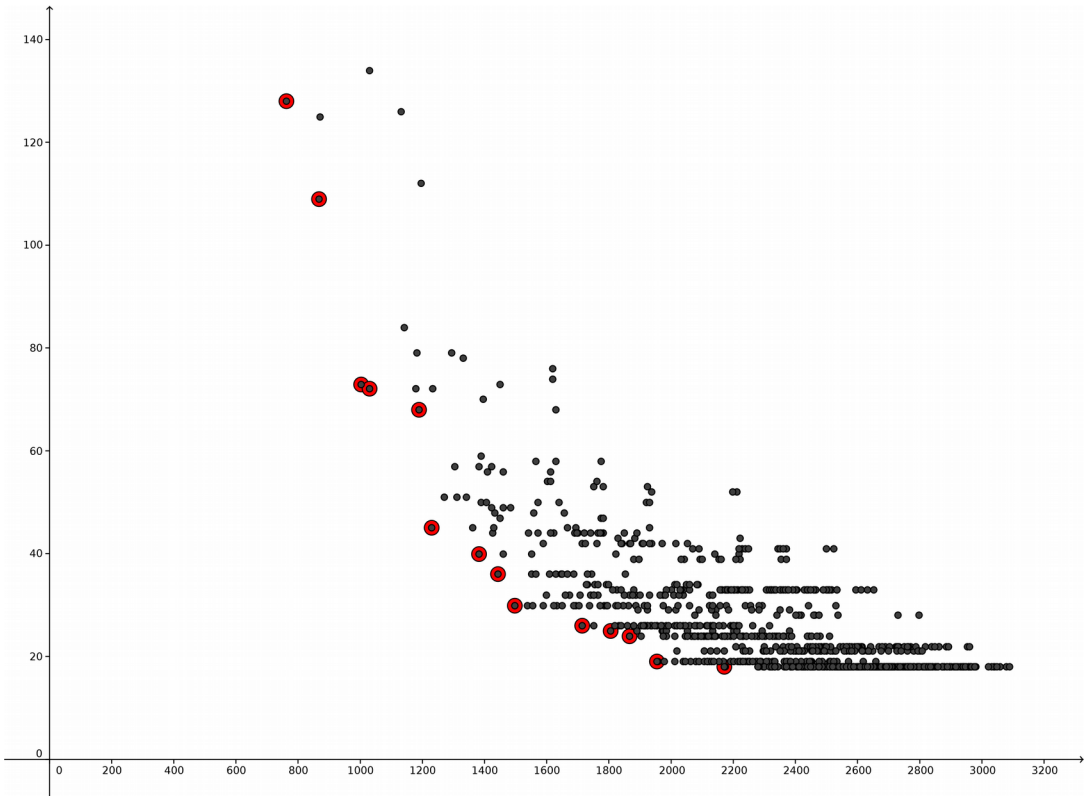

Figura 3.3: Gráfico de uma instância com 7 mercados

A partir daí, essas fronteiras de eficiência foram usadas como base de validação para as restantes versões dos algoritmos, pois procura-se generalizar. Procura-se criar algoritmos que produzam resultados válidos para múltiplas instâncias.

4. Desenho técnico e implementação

O presente capítulo tem como fim uma descrição detalhada da estrutura de dados e principais funções implementadas nestes algoritmos. A implementação destes algoritmos é em C++ e começa com a definição da classe TPP, que contêm todas as informações de uma determinada instância. É feita uma definição das variáveis de instância, seguida de uma descrição detalhada da representação da solução. É explicado o método de geração de soluções aleatórias, assim como as razões que levaram à sua escolha. É feita uma descrição da função de avaliação e da forma como as soluções são guardadas na fronteira. Os detalhes dos algoritmos exato e aproximados serão discutidos nos seus respetivos capítulos.

4.1. Variáveis de instância

Ao iniciar-se qualquer um dos algoritmos implementados, é necessário carregar uma instância a partir de um ficheiro e guardá-la numa estrutura de dados o mais intuitiva e simples possível.

Para manipular as instâncias, são necessárias as seguintes variáveis numéricas:

- um inteiro m que guarda o número de mercados;
- um inteiro p que guarda o número de produtos.

Ainda são necessárias duas variáveis vetoriais, descritas de seguida.

4.1.1. Matriz das distâncias

A cada dois mercados corresponde uma única distância (ver secção 3.1), portanto estes valores poderão ser guardados numa matriz diagonal, ou seja, num vetor de vetores em C++.

A disposição em forma de triângulo deve-se à disposição triangular dos dados no ficheiro da instância. Por simplicidade de importação, decidiu-se guardar os dados no mesmo formato.

Baseando-nos no exemplo ilustrativo da secção 3.2, mais precisamente na Tabela 3.1, podemos construir o seguinte modelo da matriz das distâncias.

Mercado de chegada (i_b) / Mercado de partida (i_a)	0	1	2
0	10	7	16
1	15	20	
2	12		

Tabela 4.1: Matriz das distâncias (valores)

As distâncias são colocadas no início de cada linha, obrigando a maiores cuidados com os índices, pois estes não representam os mercados.

Por exemplo, o valor `matriz[0][2]` representa a distância entre os mercados 0 (depósito) e 3. Assim, é devolvido o valor 16. A tabela abaixo mostra o significado de cada valor.

Índice do mercado de chegada (i_b) / Índice do mercado de partida (i_a)	0	1	2
0	dist(0, 1)	dist(0, 2)	dist(0, 3)
1	dist(1, 2)	dist(1, 3)	
2	dist(2, 3)		

Tabela 4.2: Matriz das distâncias (significado)

Analisando a Tabela 4.2 e os exemplos dados acima pode concluir-se que o mercado de partida a corresponde efetivamente ao seu índice de partida i_a. Quanto ao mercado de chegada, é necessária uma análise um pouco mais aprofundada.

O mercado de chegada é igual à soma dos índices adicionada de uma unidade $b = i_a + i_b + 1$. Assim, é possível escrever os índices em função dos mercados de partida a e de chegada b.

- $i_a = a$
- $i_b = b - i_a - 1 \Leftrightarrow i_b = b - a - 1$

Pode-se concluir que `matriz[a][b-a-1]` devolve a distância do mercado a ao mercado b. Para obter a distância do mercado b ao mercado a, basta trocar a ordem das letras: `matriz[b][a-b-1]`.

4.1.2. Matriz dos preços

Cada mercado oferece os seus próprios preços para cada um dos produtos. Esses também são

armazenados em forma de matriz. Numeram-se os mercados usando os índices vetoriais. O depósito será o mercado 0. Este não tem produtos à venda, portanto esse mercado tem os preços colocados a "-1".

Baseando-nos no exemplo ilustrativo da secção 3.2, mais precisamente na Tabela 3.2, podemos construir o seguinte modelo da matriz dos preços.

Mercados \ Produtos	0	1	2
0	-1	-1	-1
1	3	7	1
2	5	2	4
3	1	2	6

Tabela 4.3: Matriz dos preços

4.2. Representação da solução

Para se chegar a uma solução, são necessários bastantes cálculos e algumas variáveis para guardar informações importantes. O objetivo deste trabalho é obter-se soluções com distância mínima e preço mínimo, logo é necessário ir armazenando os melhores preços para cada produto, assim como os mercados visitados, de modo a poder-se calcular a distância e o preço finais.

No entanto, algumas das variáveis enumeradas anteriormente são apenas auxiliares e não são todas obrigatórias para definir uma solução.

Olhando para o problema de uma forma simples, podemos reparar que uma listagem dos mercados visitados é suficiente, pois mostra claramente os mercados percorridos, assim como a ordem de visita, A partir dessa lista, pode-se calcular a distância total, o preço e os melhores preços para cada produto. O percurso será efetuado na ordem em que os mercados estão nessa lista, adquirindo os produtos mais baratos desses mercados.

Resumindo, uma solução é definida por um conjunto de mercados (exceto o depósito), numa certa ordem. Pode tratar-se de uma permutação do total de mercados, mas também pode ser apenas um subconjunto dos mercados todos. O que importa é que a solução é definida por uma lista de mercados, onde não constam o mercado 0 (depósito) nem mercados repetidos.

4.3. Geração de soluções aleatórias

Nos algoritmos aproximados, serão necessárias soluções aleatórias. Ora, a quantidade soluções de grande dimensão (muitos mercados) é muito maior do que a quantidade de soluções de pequena dimensão (poucos mercados).

O número de soluções com *k* mercados, de entre *p* mercados, pode ser calculado usando um arranjo simples (ver secção 2.1). Vejam-se os seguintes exemplos, considerando um total de 32 mercados (tirando o depósito):

- Número de soluções com 3 mercados: $P(32,3)=\frac{32!}{(32-3)!}=\frac{32!}{29!}=29760$

- Número de soluções com 25 mercados: $P(32,25)=\frac{32!}{(32-25)!}=\frac{32!}{7!}\simeq 52\times 10^{30}$

Como se pode verificar, é muito mais provável obter-se uma solução aleatória de grande dimensão. No entanto, as de pequena dimensão são extremamente importantes. É necessário um método que gere soluções grandes e pequenas com uma probabilidade bastante próxima.

O algoritmo de geração de soluções aleatórias tem uma primeira fase onde é gerado um percurso de máxima dimensão (por exemplo, 32 mercados) e a ordem dos mercados é baralhada, de modo a obter-se um percurso aleatório.

A segunda fase determina o número de mercados que a solução deve conter. Para tal, começa-se com o número de mercados no seu valor máximo e gera-se um valor aleatório. Se esse número for maior do que 80%, então o número de mercados permanece como está. Se esse número for menor que 80%, o ciclo continua, removendo um mercado da solução e gerando um novo valor aleatório. A cada ciclo, há 20% de probabilidade do número de mercados se manter como está e 80% de probabilidade de o ciclo continuar.

Na última iteração do ciclo, há 20% de probabilidade de se ficar com 2 mercados e há 80% de probabilidade de se ficar com um mercado. Esse parece incoerente, mas foi decidido que ficaria assim, pois desta forma é possível alcançar soluções pequenas mais facilmente.

Geraram-se 10 milhões de soluções aleatórias e compararam-se as frequências absolutas do número de mercados. Com este método, a probabilidade de geração de uma solução com 32

mercados é de 18,97%, já a probabilidade de se gerar uma solução com um mercado é de 0,15%. A diferença entre probabilidade já é muito mais aceitável, agora.

4.4. Função de avaliação

A função de avaliação consiste em calcular a distância entre os mercados (na ordem em que estes estão indicados) e o preço mínimo de compra dos produtos.

4.4.1. Cálculo da distância

A distância de uma solução é calculada percorrendo os mercados e somando as distâncias parciais entre mercados seguidos. Como a variável de solução não contêm o depósito, tem de se adicionar ainda os seguintes valores:

- distância entre o depósito e o primeiro mercado da solução;
- distância entre o último mercado da solução e o depósito.

4.4.2. Cálculo do preço

Cada produto é adquirido no mercado onde o preço é menor. Portanto, para efetuar esse cálculo, percorrem-se todos os mercados. Para cada mercado, percorrem-se todos os preços e procura-se o menor. Os preços mínimos são adicionados e esse valor será o preço mínimo (total) da solução em análise.

4.5. Armazenar soluções na fronteira de soluções não dominadas

Na secção 3.1, é referido que o TPP tem um conjunto de soluções ótimas, sob forma de uma fronteira não dominada. Essa fronteira é representada por um vetor de instâncias da classe TPP.

A otimização dessas soluções é um processo onde ao minimizar-se a distância, o preço aumenta quase sempre, e vice-versa. Sendo assim, existirão soluções com distâncias baixas e preços altos, outras com distâncias altas e preços baixos e ainda outras mais equilibradas. Cada uma das soluções da fronteira é melhor do que todas as outras em pelo menos um dos dois fatores : distância ou preço.

A fronteira vai aumentando consoante se vão otimizando soluções, portanto são necessárias

algumas regras de armazenamento de uma solução na fronteira.

Quando chega uma nova solução, percorre-se a fronteira em busca de duas coisas:

- um local onde inserir a nova solução, caso esta deva mesmo ser inserida;
- soluções piores que a solução atual, de modo a removê-las da fronteira;

A fronteira está ordenada por distâncias crescentes, portanto o local de inserção é definido quando se encontra a primeira distância maior que a atual. Porém, não se insere já a nova solução. Ainda ao longo desse ciclo, são removidas todas as soluções piores que a atual.

Por fim, caso a solução atual deva ser inserida, procede-se à inserção, deixando a fronteira atualizada, por ordem crescente de distâncias e por ordem decrescente de preços.

5. Algoritmo exato

Perante a pouca variedade de algoritmos exatos para o TPP e o seu número ainda menor para o TPP bi-objetivo, decidiu-se apresentar um novo algoritmo neste trabalho. Este servirá para resolver algumas instâncias de forma exata, proporcionando assim uma base de dados de soluções úteis para futuros trabalhos nesta área.

5.1. Pré-processamento

Alguns cálculos são efetuados em pré-processamento, pois só devem ser realizados uma vez, antes de se dar início ao algoritmo.

5.1.1. Calcular a melhor ordem de expansão dos mercados

Uma solução do TPP é construída adicionando um mercado ao caminho de forma recursiva, num estilo de “profundidade primeiro”.

Por exemplo: Começa-se com a solução A, avalia-se a mesma e segue-se para a solução seguinte AB, depois ABC, e por aí adiante até se esgotarem todos os mercados. Segue-se com a solução B, a seguinte é BC, depois é BCD, e continuando desta forma, percorrem-se todas as soluções.

Suponhamos que as melhores soluções começam com o mercado D. Se iniciarmos a procura nas soluções começadas por A, B ou C, estamos a percorrer imensas soluções que vão dar resultados razoáveis, mas não ótimos. Assim, não é possível cortar ramos da árvore de procura. Quando chegarmos às soluções começadas com o mercado D, muitas das outras serão descartadas. Já, se iniciarmos a procura em D, as melhores soluções vão aparecer nos primeiros passos da procura e será possível efetuar cortes significativos nas soluções menos satisfatórias.

Portanto, é essencial que a ordem de procura pelos mercados seja bem escolhida. Trata-se de encontrar uma boa heurística de reordenação dos mercados

5.1.1.1. Reordenação pelo mercado mais próximo em relação ao anterior (MONN)

Numa primeira tentativa de heurística de reordenação, tentou-se organizar os mercados usando a técnica do vizinho mais próximo. A cada cálculo do vizinho mais próximo, são ignorados

os mercados já selecionados para a reordenação.

Por exemplo, dados os mercados A, B, C, D e o depósito X, teríamos o caminho CADB, onde C é o vizinho mais próximo do depósito, A é o vizinho mais próximo de C, D é o vizinho mais próximo de A, e B é o mais próximo de D.

No entanto, esta heurística não trouxe resultados muito bons e decidiu-se investir na busca de uma alternativa.

5.1.1.2. Reordenação pelo mercado mais próximo em relação ao depósito (MOND)

Outra heurística bem mais simples, e surpreendentemente mais eficaz, é a reordenação em relação à distância entre o mercado e o depósito.

Por exemplo, dados os mercados A, B, C, D e o depósito X, teríamos o caminho DCAB, onde D é o mercado mais próximo do depósito, C é o 2º mais próximo, A é o 3º e B é o mas afastado do depósito.

Esta heurística trouxe melhorias significativas no algoritmo, que aumentam com o número de mercados. Quantos mais mercados tiver a instância, melhores são os resultados em comparação com antes de se reordenar os mercados.

5.1.2. Recalcular distâncias

Em 3.5, já foram descritas as instâncias de teste usadas neste trabalho, no entanto, as distâncias destas instâncias não respeitam a desigualdade triangular, isto é, o caminho ABC pode ser mais curto do que o caminho AC.

Ora se é mais rápido passarmos por B, podemos atribuir a AC a distância de ABC, tendo o cuidado de “apenas atravessar” o mercado B sem adquirir nenhum produto.

Com esta correção, temos agora distâncias que respeitam a desigualdade triangular e podemos usar critérios como: “Adicionando um mercado ao percurso, a distância vai ser igual ou maior que a anterior.” Critérios deste género são extremamente úteis no âmbito do algoritmo de *branch and bound* descrito mais à frente.

Nos primeiros testes efetuados depois deste recálculo das distâncias, obtiveram-se resultados diferentes dos anteriores e decidiu-se olhar mais de perto para a situação, de modo a bem entender a causa do aparecimento de soluções diferentes.

Para melhor ilustrar a situação, usemos uma instância de Singh e van Oudheusden simplificada. Da instância “Singh33_2.33.50.1.5.tpp”, retiraram-se mercados, ficando esta apenas com 5 (1 depósito e 4 mercados reais). Chamemos-lhe “ Singh33_2.5.50.1.5.tpp”.

Apresentam-se tabelas simplificadas das distâncias “antes” e “depois” do recálculo.

Mercados	**1**	**2**	**3**	**4**
0	670	540	380	288
1		449	310	215
2			292	195
3				184

Tabela 5.1: Distâncias da instância “Singh33_2.33.50.1.5” antes do recálculo de distâncias

Mercados	**1**	**2**	**3**	**4**
0	503	483	380	288
1		410	310	215
2			292	195
3				184

Tabela 5.2: Distâncias da instância “Singh33_2.33.50.1.5” depois do recálculo de distâncias

Para resumir as alterações que o recálculo causou, temos a tabela seguinte.

Antes		**Depois**	
Caminho	***Distância***	***Caminho***	***Distância***
0-1	670	**0-4-1**	503
0-2	540	**0-4-2**	483
1-2	449	**1-4-2**	410

Tabela 5.3: Resumo das alterações que o recálculo de distâncias causou

O caminho 0-1 passou a ser o caminho 0-4-1, atravessando o mercado 4 sem se adquirir nenhum produto. Aconteceu uma situação semelhante com os caminhos 0-2 e 1-2.

Antes			Depois		
Distância	**Preço**	**Solução**	**Distância**	**Preço**	**Solução**
576	126	4	576	126	4
760	109	3	760	109	3
852	73	4 3	852	73	4 3
…	…	…	1006	70	4 1
1155	52	4 2 3	1155	52	4 2 3
1193	42	4 1 3	1193	42	4 1 3
1622	28	4 2 1 3	1583	28	4 2 1 3

Tabela 5.4: Comparação dos testes antes e depois do recálculo de distâncias

A solução “ *1583 ; 28 4 2 1 3*” é na verdade “0 4 2 1 3 0”. É diferente de antes do recálculo porque a distância (***1***-***2***) foi recalculada de 449 para 410 (**1**-**4**-**2**), pois o caminho que passa pelo mercado 4 é mais curto. Passando de 1622 para 1583, a diferença é 39, que é precisamente a redução obtida de 1 para 2 (ver Tabela 5.1 e Tabela 5.2).

A existência da nova solução “*1006 ; 70 4 1*” parece causar confusão, pois se é ótima, deveria haver uma solução antes do recálculo com os mesmos valores, mas um caminho diferente. A solução é na verdade “0 4 1 0”. A distância (**0**-**1**) foi recalculada de 670 para 503 (**0**-**4**-**1**). A distância desta nova solução calcula-se da seguinte forma.

Caminho (partes)	Distâncias (partes)
0-4	288
4-1	215
1-0	503 (após recálculo)
0-4-1-0	1006 (TOTAL)

Tabela 5.5: Detalhes do cálculo da nova solução (4-1)

Antes do recálculo, a solução equivalente seria “0 1 0”, pois o (0-1) foi substituído por (0-4-1) já que é mais rápido passar por 4 para chegar ao mercado 1, a partir do 0. A solução “0 1 0” não pertencia à fronteira antes do recálculo, pois a sua distância teria sido 1340 (670 + 670) e o seu preço certamente teria sido alto demais impedindo a solução de ser não dominada.

Com esta reflexão, pretende-se entender e mostrar que a solução "*1006 ; 70 4 1*" é válida, mesmo sem ter aparecido antes do recálculo. Portanto, podemos considerá-la e prosseguir com as novas fronteiras exatas.

A passagem por um mercado é apenas para efeitos de distância, mas fazendo-se em pré-processamento, não vai pesar mais o algoritmo.

5.2. Cortes básicos

Para poder aplicar o algoritmo exato com mais eficácia, é necessário efetuar alguns cortes básicos, ou seja, evitar percorrer algumas soluções que sabemos não serem úteis.

Na Figura 5.1 segue uma expansão em árvore do exemplo ilustrativo descrito na secção 3.2. Esta figura ilustra bem os cortes que irão ser descritos de seguida.

5.2.1. Evitar mercados repetidos

Já foi mencionado que uma solução é o conjunto de todos os mercados ou um subconjunto destes. Portanto, evita-se passar duas vezes pelo mesmo mercado, pois é algo completamente inútil, visto que numa passagem por um mercado é possível adquirir todos os produtos que desejarmos obter dele.

Nota: Evitam-se mercados repetidos no sentido de "paragem" nesses mercados para aquisição de produtos. No sentido de "passagem" por mercados repetidos, a situação é coberta pelo recálculo das distâncias, mencionado em 5.1.2.

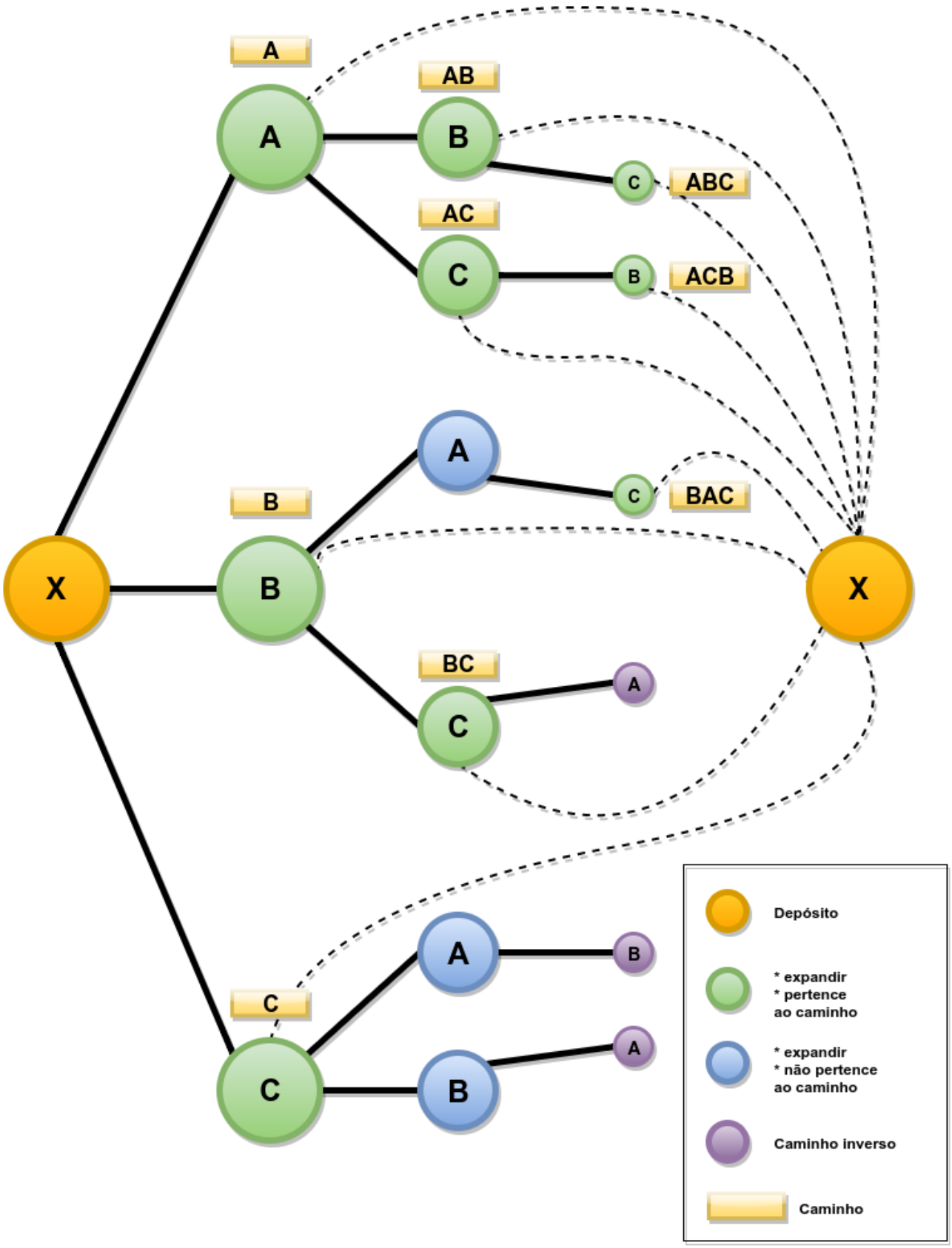

Figura 5.1: Expansão em árvore do exemplo ilustrativo

5.2.2. Evitar caminhos inversos

Relembra-se que na secção 3.2 foi dito que estamos perante o TPP simétrico, logo os caminhos AB e BA são iguais, podendo-se descartar o último.

Enumerando todos os caminhos possíveis A, B, C, AB, AC, BC, ABC, ACB e BAC, percebe-se que o último mercado é sempre maior (no que diz respeito à ordem dos mercados; neste caso: ABC) do que o primeiro.

Portanto, assume-se que, se num dado percurso tivermos o primeiro mercado maior que o último, então esse percurso é logo descartado.

No entanto, podemos expandi-lo, pois este pode ter sucessores que não são percursos inversos.

Por exemplo, considera-se a ordem dos mercados DACB. O caminho DC é válido, pois D < C na ordem DACB. Já o caminho AD é inverso, pois A > D, e também já passámos pelo caminho igual a esse em DA. O caminho ADC é válido, pois A < C. É de notar aqui que apesar de AD não ser válido, deve-se continuar a expandir, pois pode conter sucessores válidos, tais como ADC. Na Figura 5.1, os nós azuis são precisamente aqueles que são caminhos inversos, mas devem ainda ser expandidos.

5.2.3. Não expandir o último mercado da ordenação

Seguindo a ideia da secção anterior, devem descartar-se caminhos onde o último mercado é menor que o primeiro, na ordenação.

Na Figura 5.1, podemos facilmente verificar que não adianta expandir o caminho C, pois este só pode gerar sucessores com caminhos inversos.

Enumerando os seus sucessores CA, CAB, CB, CBA, vemos claramente que o último mercado é sempre menor que o primeiro e isso faz todo o sentido, pois o mercado C é o último da ordenação ABC escolhida nesta figura. Logo, necessariamente, todos os restantes serão menores e pode-se parar as expansões em C.

É de notar que este corte oferece uma eficiência significativa no algoritmo. Consideremos, de forma geral, uma instância de *n* mercados. Segundo a árvore de procura da Figura 5.1, temos *n*

ramificações logo no início. Ao efetuarmos este corte, descartamos aproximadamente $\frac{1}{n}$ soluções. Assim, obtemos uma melhoria de perto de $\frac{1}{n} \times 100\,\%$.

Por exemplo:

- numa instância com 4 mercados, a melhoria é de $\frac{1}{4} \times 100\,\% = 25\,\%$;
- numa instância com 12 mercados, a melhoria é de $\frac{1}{12} \times 100\,\% = 8{,}33\,\%$;
- numa instância com 32 mercados, a melhoria é de $\frac{1}{32} \times 100\,\% = 3{,}125\,\%$.

A melhoria vai diminuindo à medida que se aumenta o número de mercados, mas mesmo assim é significativa, tendo em conta o número astronómico de soluções para instâncias com muitos mercados.

5.2.4. Não expandir quando todos os maiores mercados já estão na solução

O corte anterior pode ser generalizado e assim oferecer mais alguma eficiência ao algoritmo. Vimos que na Figura 5.1, não é necessário expandir o caminho C, pois essa expansão só vai gerar caminhos inversos. Isso deve-se ao facto de que o maior mercado já se encontra na solução.

Vejamos o que acontece com o caminho BC. Ao expandi-lo, ficamos com BCA que é um caminho inverso, pois B > A. Nota-se que os dois mercados mais altos (B e C) já se encontram na solução, logo qualquer expansão iria introduzir um mercado menor do que o primeiro.

Assim, generaliza-se o corte da secção 5.2.3 da seguinte forma: Se uma solução já contém os *k* maiores mercados (relativamente à ordenação escolhida), então essa solução não deve ser expandida, pois todos os seus sucessores serão caminhos inversos.

Por exemplo, considera-se a ordem dos mercados DACB, como na secção 5.2.3. Se a solução atual for BC, esta não se expande, pois um dos sucessores iria ser BCA, que é o inverso de ACB já analisado antes na procura. Ambos D e A são menores que B, logo qualquer expansão de BC daria origem a caminhos inversos. Assim, deve efetuar-se o corte aqui.

5.3. Algoritmo de *branch and bound*

Já foi referido na secção 2.3.1 que o algoritmo de *branch and bound* é um dos mais utilizados e mais eficazes em problemas de otimização combinatória. Este percorre as soluções ramificando o espaço de soluções e corta ramos dependendo do resultado da aplicação de um *lower bound*.

5.3.1. Branching

Em primeiro lugar, escolhe-se a forma como se vai efetuar a ramificação (*branching*). O algoritmo de *branch and bound* é aplicado através de uma função recursiva que segue a abordagem de profundidade primeiro. Assim, a ramificação é feita através do primeiro mercado do caminho.

Considera-se uma instância com 5 mercados. O *branch and bound* vai gerar 5 ramos principais, um para cada subconjunto de soluções começada por esse mercado. Portanto, o número de soluções de cada ramo está equilibrado.

5.3.2. Lower bounds

Como estamos a tratar o TPP bi-objetivo, temos de pensar em duas funções de *lower bound*: uma para a distância e outra para o preço.

5.3.2.1. Lower bound da distância

O recálculo de distâncias foi implementado para que os caminhos gerados respeitem a desigualdade triangular. Ao adicionarmos um mercado a uma solução, a sua distância só pode aumentar. Tendo em conta essa propriedade importante, o *lower bound* da distância podia ser apenas a distância do caminho atual. No entanto, esta abordagem não efetua cortes tão significativos como o método seguinte.

O *lower bound* da distância é calculado usando um relaxamento na posição dos mercados ainda não adicionados ao caminho.

Exemplificando com os mercados A, B, C e D (X é o depósito) organizados nesta ordem, temos uma solução atual com percurso AC. Sabemos que a distância total é calculada da seguinte forma: XA + AC + CX.

O *lower bound* é calculado da seguinte forma: adiciona-se o mercado B e calcula-se a soma das distâncias parciais

$$d_1 = d_{parcial}(XA) + d_{parcial}(AC) + d_{parcial}(CB) + d_{parcial}(YX) \quad (8)$$

onde Y é o vizinho mais próximo do depósito de entre os mercados ainda não colocados (neste caso só pode ser B ou D).

Repare-se que não estamos a ligar B diretamente a X no fim do percurso. Estamos a colocar a hipótese de DX ser menor que BX, obtendo assim um valor de distância inferior ou igual ao que seria de esperar. Logo trata-se de um *lower bound*.

Segue-se o mesmo procedimento para todos os mercados restantes, incluindo a adição de múltiplos mercados

$$d_2 = d_{parcial}(XA) + d_{parcial}(AC) + d_{parcial}(CD) + d_{parcial}(YX) \quad (9)$$

$$d_3 = d_{parcial}(XA) + d_{parcial}(AC) + d_{parcial}(CB) + d_{parcial}(BZ) + d_{parcial}(YX) \quad (10)$$

onde Z é o vizinho mais próximo de B de entre os mercados ainda não colocados (neste caso só pode ser D).

Por fim, escolhe-se o menor destes três valores como *lower bound* final:

$$d_{lower\ bound} = min(d_1, d_2, d_3) \quad (11)$$

Este *lower bound* mostrou enormes melhorias em relação ao algoritmo cujo *lower bound* seria a distância atual. Os resultados serão descritos ao longo do capítulo 6.

5.3.2.2. Lower bound do preço

O *lower bound* do preço foi muito difícil de definir, pois o algoritmo exato constrói cada solução aos poucos, adicionando mercados a cada passo. A adição de um mercado apenas pode manter o preço igual ou melhorá-lo. O preço final de uma solução nunca piora se lhe adicionarmos um mercado. Tendo em conta essa característica, torna-se bastante difícil encontrar um *lower bound* do preço eficaz.

Uma tentativa baseou-se em usar o conjunto de mercados da solução atual (M1) e o conjunto de mercados ainda não visitados (M2), deixando de parte um eventual conjunto de mercados descartados (D). A ideia tem potencial para produzir bons *lower bounds*, no entanto, é extremamente complicado definir quais os mercados que podem ser descartados.

Pensou-se em descartar os mercados pelos quais já se passou “dada uma certa ordem”. Por exemplo, se a ordem de mercados for “4 3 2 1” e a solução atual for “3 1”, consideram-se os

seguintes conjuntos: M1 = {1, 3}; M2 = {2} e D = {4}. O mercado 4 seria descartado, pois por esta altura já se teria percorrido todos os caminhos começados por 4. No entanto, usar esta estratégia como *lower bound* do preço não resulta, pois eliminam-se soluções válidas, tais como "3 1 4".

Sendo assim, o *lower bound* do preço é apenas o preço dos produtos se visitássemos todos os mercados, ou seja o menor preço possível. Trata-se de um *lower bound* pouco eficiente, mas completo, no sentido em que permite percorrer todas as soluções válidas.

5.3.3. Aplicação dos lower bounds

Os *lower bounds* são aplicados quando uma determinada solução não é aceite na fronteira não dominada. É criada uma solução fictícia com a distância e preço obtidos nos *lower bounds* e tenta-se introduzir essa solução na fronteira. Podem acontecer duas coisas:

- a solução fictícia é aceite na fronteira, mas não é adicionada, visto que não é uma solução verdadeira, logo o algoritmo continua a expandir essa solução, pois certamente ela tem sucessores que podem pertencer à fronteira;
- a solução fictícia não é aceite na fronteira, logo efetua-se um corte e essa solução já não é expandida.

5.4. Aplicação do algoritmo exato no exemplo ilustrativo

Usando a Tabela 3.1 e a Tabela 3.2, gerou-se a tabela de aplicação do algoritmo exato no exemplo ilustrativo, que inclui os estados mais importantes, os *lower bounds* (quando necessários) e o conteúdo da fronteira não dominada em cada um desses estados.

A Figura 5.2 descreve detalhadamente a aplicação do algoritmo exato no exemplo ilustrativo apresentado na secção 3.2.

Supondo que o pré-processamento retorna a ordem de mercados BAC, procede-se agora à explicação de cada estado designado por uma letra:

(a) é explorada a solução B e adicionada à fronteira de soluções não dominadas;

(b) é explorada a solução BA e também é adicionada à fronteira;

(c) idem para a solução BAC;

(d) prossegue-se agora para a solução BC, mas não é aceite na fronteira;

(e) é explorada a solução BCA, que é adicionada à fronteira dominando a solução BAC, que é removida da fronteira;

(f) segue-se para a exploração da solução A, que não é aceite na fronteira;

(g) explora-se a solução AB que não é aceite na fronteira por ser o simétrico a BA (já explorado antes). No entanto, pelo seu *lower bound* e constata-se que se pode efetuar um corte aqui, descartando de imediato a solução ABC (não vai ser explorada, sequer);

(h) explora-se a solução AC, que é adicionada à fronteira dominando a solução BCA, que é removida da fronteira;

(i) segue-se para a solução ACB, que é descartada por ser o simétrico de BCA;

(j) explora-se a solução C e verifica-se que esta não entra na fronteira nem qualquer dos seus descendentes, pois C é o maior mercado da ordenação escolhida (ver 5.2.3);

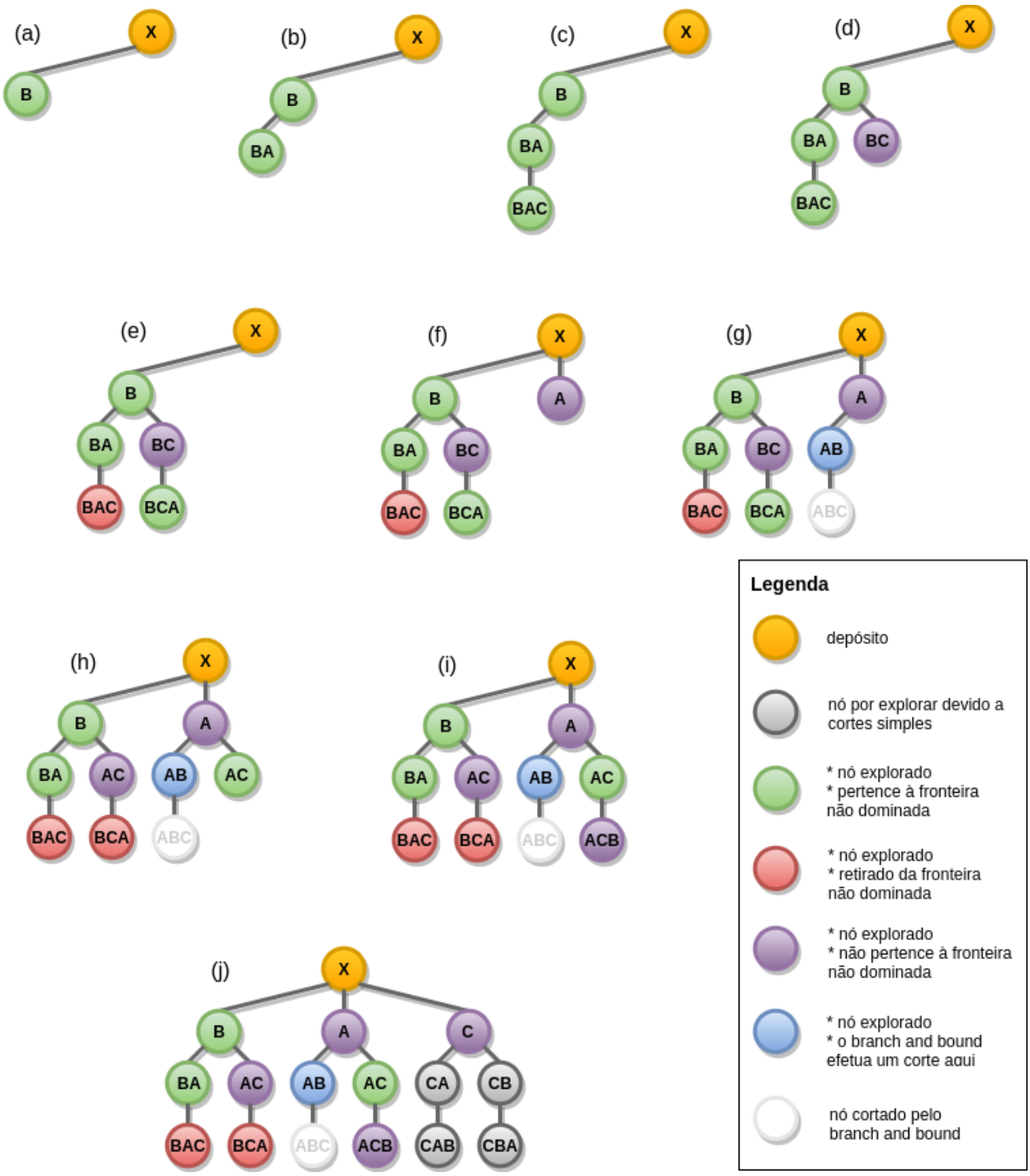

Figura 5.2: Aplicação do algoritmo exato no exemplo ilustrativo

6. Algoritmos aproximados

O algoritmo exato é muito demorado e por vezes é preferível obter-se soluções sub-ótimas, ganhando no tempo de computação. Portanto, é extremamente importante desenvolver-se também algoritmos aproximados. Estes podem conter elementos do algoritmo exato e heurísticas que irão acelerar o processo de busca de soluções sub-ótimas, ou podem basear-se em meta-heurísticas conhecidas.

6.1. Algoritmo exato truncado

Uma ideia simples para o algoritmo híbrido seria apenas truncar o algoritmo exato, ou seja impor-lhe um critério de paragem.

6.1.1. Critério de paragem

Existem várias estratégias que se poderiam usar como critério de paragem. Na estratégia do limite de tempo, o algoritmo exato interrompe a sua execução após se esgotar o tempo determinado. Dependendo do estado da máquina, podem ocorrer variações nas soluções obtidas. Essas variações poderiam ser evitadas com a imposição de um limite no número de gerações, em vez do limite de tempo. No entanto, sendo este um algoritmo aproximado, a variação das soluções de corrida para corrida não é um problema.

Usando uma instância com 32 mercados, efetuaram-se corridas para 10, 30 e 60 segundos, registando-se melhorias significativas com tempos maiores, como seria de esperar.

A Tabela 6.1 mostra uma comparação entre execuções do algoritmo truncado para 10 e 60 segundos. Notam-se melhorias nas 10 últimas soluções. Portanto, optou-se por impor um limite de tempo de 60 segundos.

10 segundos	60 segundos
(692,103) - 23	(692,103) - 23
(1108,72) - 23 32	(1108,72) - 23 32
(1358,60) - 23 24	(1358,60) - 23 24
(1380,49) - 23 32 8	(1380,49) - 23 32 8
(1610,47) - 23 32 13	(1610,47) - 23 32 13
(1806,43) - 23 32 2	(1806,43) - 23 32 2
(1836,41) - 23 32 24 31	(1836,41) - 23 32 24 31
(2020,38) - 23 32 24 8	(2020,38) - 23 32 24 8
(2063,37) - 23 32 24 27	(2063,37) - 23 32 24 27
(2182,36) - 23 32 24 25	(2182,36) - 23 32 24 25
(2244,34) - 23 32 24 2	(2244,34) - 23 32 24 2
(2250,32) - 23 32 24 13	(2250,32) - 23 32 24 13
(2372,26) - 23 32 24 8 29	(2372,26) - 23 32 24 8 29
(2522,24) - 23 32 24 8 13	(2522,24) - 23 32 24 8 13
(2718,23) - 23 32 24 8 2	(2718,23) - 23 32 24 8 2
(2729,22) - 23 32 24 8 29 31	(2729,22) - 23 32 24 8 29 31
(2803,21) - 23 32 24 8 29 27	(2803,21) - 23 32 24 8 29 27
(2840,20) - 23 32 24 8 29 30	(2840,20) - 23 32 24 8 29 30
(3137,18) - 23 32 24 8 29 2	(3137,18) - 23 32 24 8 29 2
(3140,17) - 23 32 24 8 29 22	(3140,17) - 23 32 24 8 29 22
(3390,16) - 23 32 24 8 29 25 27	(3390,16) - 23 32 24 8 29 25 27
(3430,15) - 23 32 24 8 29 25 30	(3430,15) - 23 32 24 8 29 25 30
(3635,13) - 23 32 24 8 29 25 31 27	(3635,13) - 23 32 24 8 29 25 31 27
(3711,12) - 23 32 24 8 29 25 31 30	(3711,12) - 23 32 24 8 29 25 31 30
(3959,10) - 23 32 24 8 29 25 31 30 27	(3851,10) - 23 32 24 8 29 25 31 27 30
(4007,9) - 23 32 24 8 29 25 31 30 1	(4007,9) - 23 32 24 8 29 25 31 30 1
(4433,7) - 23 32 24 8 29 25 31 30 27 7	(4147,8) - 23 32 24 8 29 25 31 27 30 1
(4761,6) - 23 32 24 8 29 25 31 30 7 28	(4325,7) - 23 32 24 8 29 25 31 27 30 7
(4897,5) - 23 32 24 8 29 25 31 30 27 6 26	(4583,6) - 23 32 24 8 29 25 31 30 1 28
(5009,4) - 23 32 24 8 29 25 31 30 27 7 28	(4723,5) - 23 32 24 8 29 25 31 27 30 1 28
(5376,3) - 23 32 24 8 29 25 31 30 27 6 26 7	(4901,4) - 23 32 24 8 29 25 31 27 30 7 28
(5603,2) - 23 32 24 8 29 25 31 30 27 6 7 3	(5163,3) - 23 32 24 8 29 25 31 27 30 1 6 26
(6124,1) - 23 32 24 8 29 25 31 30 27 6 26 1 22	(5495,2) - 23 32 24 8 29 25 31 30 17 1 7 28
(6545,0) - 23 32 24 8 29 25 31 30 27 6 26 28 1 22	(5856,1) - 23 32 24 8 29 25 31 27 6 26 1 30 22
	(6220,0) - 23 32 24 8 29 25 31 30 1 13 6 26 22

Tabela 6.1: Comparação entre execuções do algoritmo truncado por limites de tempo diferentes

6.1.2. Impacto da ordenação de mercados

A Tabela 6.1 mostrou também que as fronteiras são idênticas até às 10 últimas soluções. São muito parecidas, pois analisam a mesma zona do espaço de resultados. Portanto há necessidade de incluir uma fator de diversificação.

A reordenação de mercados mostrou-se extremamente eficaz na melhoria do algoritmo exato. Portanto também terá um forte impacto neste algoritmo aproximado visto que se trata do mesmo, apenas truncado.

A reordenação aleatória e por mercado mais barato foram testadas, mas trouxeram resultados muito fracos comparados com outros tipos de reordenação de mercados, tais como:

- pelo mercado mais próximo em relação ao anterior (ver secção 5.1.1.1)
- pelo mercado mais próximo em relação ao depósito (ver secção 5.1.1.2)

A Tabela 6.2 mostra uma comparação de resultados usando os tipos de ordenação acima mencionados.

Usando a ordenação por mercado mais próximo do depósito (MOND), obtém-se uma fronteira com 35 soluções. Já, com a ordenação por mercado mais próximo do anterior (MONN), a dimensão da fronteira é apenas de 25. As soluções mais curtas, com menores distâncias são idênticas em ambas as estratégias. As soluções começam a ser diferentes a partir da 6ª (aproximadamente, pois cada execução traz resultados ligeiramente diferentes). Dentro das soluções diferentes, as do MONN superam as do MOND.

Assim, o MONN parece ser a estratégia de reordenação de mercados mais adequada para o algoritmo truncado.

Mais próximo do depósito (MOND)	Mais próximo do anterior (MONN)
(692,103) - 23	(692,103) - 23
(1108,72) - 23 32	(1108,72) - 23 32
(1358,60) - 23 24	(1358,60) - 23 24
(1380,49) - 23 32 8	(1380,49) - 23 32 8
(1610,47) - 23 32 13	(1610,47) - 23 32 13
(1806,43) - 23 32 2	(1703,38) - 23 32 8 24
(1836,41) - 23 32 24 31	(1732,35) - 23 32 8 29
(2020,38) - 23 32 24 8	(1791,32) - 23 32 8 31 24
(2063,37) - 23 32 24 27	(2011,31) - 23 32 8 31 29
(2182,36) - 23 32 24 25	(2078,30) - 23 32 8 2
(2244,34) - 23 32 24 2	(2156,28) - 23 32 8 31 2
(2250,32) - 23 32 24 13	(2258,27) - 23 32 8 31 27 29
(2372,26) - 23 32 24 8 29	(2487,21) - 23 32 8 31 27 30 1
(2522,24) - 23 32 24 8 13	(2625,19) - 23 32 8 31 27 30 25
(2718,23) - 23 32 24 8 2	(2783,16) - 23 32 8 31 27 30 22
(2729,22) - 23 32 24 8 29 31	(3221,14) - 23 32 8 31 27 30 17 22
(2803,21) - 23 32 24 8 29 27	(3329,10) - 23 32 8 31 27 30 17 1 25
(2840,20) - 23 32 24 8 29 30	(3615,9) - 23 32 8 31 27 30 17 1 22
(3137,18) - 23 32 24 8 29 2	(3789,8) - 23 32 8 31 27 30 17 1 29 25
(3140,17) - 23 32 24 8 29 22	(3845,6) - 23 32 8 31 27 30 17 1 28 25
(3390,16) - 23 32 24 8 29 25 27	(4191,4) - 23 32 8 31 27 30 17 1 28 22
(3430,15) - 23 32 24 8 29 25 30	(4621,3) - 23 32 8 31 27 30 17 1 29 28 22
(3635,13) - 23 32 24 8 29 25 31 27	(4875,2) - 23 32 8 31 27 30 17 1 28 2 22
(3711,12) - 23 32 24 8 29 25 31 30	(5293,1) - 23 32 8 31 27 30 17 1 29 6 3 22
(3851,10) - 23 32 24 8 29 25 31 27 30	(5436,0) - 23 32 8 31 27 30 17 1 29 26 6 3 22
(4007,9) - 23 32 24 8 29 25 31 30 1	
(4147,8) - 23 32 24 8 29 25 31 27 30 1	
(4325,7) - 23 32 24 8 29 25 31 27 30 7	
(4583,6) - 23 32 24 8 29 25 31 30 1 28	
(4723,5) - 23 32 24 8 29 25 31 27 30 1 28	
(4901,4) - 23 32 24 8 29 25 31 27 30 7 28	
(5163,3) - 23 32 24 8 29 25 31 27 30 1 6 26	
(5495,2) - 23 32 24 8 29 25 31 30 17 1 7 28	
(5856,1) - 23 32 24 8 29 25 31 27 6 26 1 30 22	
(6220,0) - 23 32 24 8 29 25 31 30 1 13 6 26 22	

Tabela 6.2: Comparação entre execuções do algoritmo truncado por reordenações de mercados diferentes

6.2. Escalada do Monte

De forma a poder-se avaliar a qualidade do algoritmo truncado, decidiu-se implementar um algoritmo de Escalada do Monte (*Hill Climbing – HC*). A secção 2.4.1 descreve o seu funcionamento.

A escalada do monte é um algoritmo iterativo que começa com uma solução arbitrária e tenta melhorá-la a cada iteração, através da sua vizinhança. O algoritmo termina quando cumpre um certo critério de paragem.

6.2.1. Geração da fronteira inicial

Antes de iniciar o *Hill Climbing*, é necessária uma fronteira inicial. Essa fronteira é, normalmente, preenchida com soluções aleatórias. No entanto, como já se explicou antes, as soluções aleatórias trazem alguma diversidade, mas causam uma enorme demora na convergência para uma fronteira sub-ótima.

Sendo assim, optou-se por gerar a fronteira inicial com algumas soluções aleatórias e outras escolhidas pela sua proximidade em relação ao depósito.

6.2.1.1. Geração de soluções aleatórias

São geradas 1000 soluções aleatórias com uma forte tendência para a geração de soluções com muitos mercados, pois são mais frequentes.

6.2.1.2. Geração de soluções por proximidade do depósito

No algoritmo exato, escolheu-se a ordenação de mercados pelo mais próximo em relação ao depósito (MOND), pois esta oferece excelentes resultados.

Na geração da fronteira inicial para o *Hill Climbing*, usa-se o MOND da seguinte forma. Suponhamos que temos uma instância com 4 mercados: A, B, C e D. Após aplicar o MOND, obtém-se a ordenação: CABD. As soluções geradas para a fronteira são as seguintes: C, CA, CAB, CABD.

6.2.2. Cálculo da vizinhança

A otimização de uma solução passa pela análise de soluções muito parecidas com a atual. A esse conjunto de soluções chama-se vizinhança de uma solução.

Para modificar ligeiramente uma solução, pensou-se em três maneiras: inserir um mercado não presente na solução atual, remover um dos mercados da solução, ou trocar a posição de dois mercados seguidos.

6.2.2.1. Inserção de um mercado

Este método procede à inserção de todos os mercados ausentes da solução, e em todas as posições da solução. Toma-se como exemplo um conjunto de 5 mercados: A, B, C, D e E. Toma-se como solução atual ACE. Para começar, verifica-se que os mercados ausentes da solução são B e D. Procede-se então à inserção dos mercados B e D em todas as posições da solução, obtendo-se o seguinte conjunto de soluções : BACE, ABCE, ACBE, ACEB, DACE, ADCE, ACDE, ACED.

6.2.2.2. Remoção de um mercado

A remoção de um mercado gera novas soluções, onde foram removidos cada um dos mercados presentes inicialmente. Tomando como exemplo a solução ACE da secção anterior, a remoção de um mercado gera o seguinte conjunto de soluções: CE, AE, AC.

6.2.2.3. Permutação de dois mercados seguidos

É possível obter bons vizinhos apenas permutando mercados seguidos. Este método percorre a solução e troca as posições de todos os mercados vizinhos. Prosseguindo com o exemplo da solução ACE, após o uso deste método, obtém-se o seguinte conjunto de soluções: CAE, AEC.

6.2.3. Funcionamento do algoritmo

Na sua primeira fase, o algoritmo de otimização percorre toda a fronteira, calculando as vizinhanças de todos os seus elementos e inserindo novas soluções sempre que estas são encontradas, pois todas as soluções da vizinhança são candidatas a entrar na fronteira. Deste modo, é muito provável que a dimensão da fronteira cresça consoante a complexidade da instância em análise. Trata-se de efetuar uma primeira otimização de modo a eliminar as piores soluções logo no início.

Numa segunda fase, entra-se num ciclo infinito que faz novamente o cálculo da vizinhança e a sua introdução na fronteira, se surgirem boas soluções.

O critério de paragem destas iterações é um dos seguintes:

- quando a iteração atual não inseriu nenhuma solução na fronteira;

- quando passa o limite de tempo de 60 segundos.

A cada iteração, a fronteira é atualizada com novas soluções que podem levar a soluções ainda melhores.

7. Resultados

Estes algoritmos foram corridos num portátil com o sistema operativo Ubuntu 12.04 LTS de 64 bits, com um processador Intel(R) Core(TM) i5 CPU M 480 @ 2,67 GHz e com memória RAM de 3,7 GB.

As instâncias de teste usadas neste trabalho são as de Singh e van Oudheusden, já brevemente descritas na secção 3.5.

Como o algoritmo exato é muito pesado, optou-se por criar variantes da instância "Singh33_2.33.50.1.5.tpp", onde foram retirados mercados, pois este conjunto só contém instâncias de 33 mercados (incluindo o depósito). É necessário haver um conjunto de instâncias com menos mercados para se poder testar a eficácia do algoritmo exato ao longo da sua implementação.

Segue uma breve legenda das colunas usadas nas tabelas de dados:

- M: número de mercados;
- P: número de produtos;
- Ordenação: ordenação dos mercados, tal como discutida em 5.1.1;
- #F: número de soluções na fronteira não dominada;
- t: tempo de computação em segundos;
- Gerações: número de soluções geradas;
- Expansões: número de soluções expandidas (não expansões podem indicar cortes);
- Avaliações: número de soluções avaliadas.

Estas instâncias pequenas são apenas usadas para testar a qualidade e eficiência do algoritmo exato. As instâncias originais (de 33 mercados) são tratadas pelos algoritmos aproximados.

7.1. Algoritmo exato

De forma a bem ilustrar a evolução do algoritmo exato, indicam-se tabelas de dados e tabelas comparativas de várias fases do desenvolvimento.

7.1.1. Profundidade primeiro com cortes básicos

Antes de se implementar o *branch and bound*, o algoritmo exato era um simples algoritmo de profundidade primeiro que usava os cortes básicos, descritos na secção 5.2.

M	Ordenação	#F	t	Gerações	Expansões	Avaliações
5	1 2 3 4	6	0	42	37	42
6	1 2 3 4 5	13	0	221	203	221
7	1 2 3 4 5 6	14	0.04	1399	1310	1399
8	1 2 3 4 5 6 7	12	0.27	10286	9736	10286
9	1 2 3 4 5 6 7 8	15	2.44	85782	81785	85782
10	1 2 3 4 5 6 7 8 9	19	29.84	798677	765587	798677

Tabela 7.1: Eficiência do algoritmo de profundidade primeiro com cortes básicos

7.1.2. Branch and bound

Ao aplicar-se o *branch and bound* na sua forma mais básica, as melhorias foram enormes apesar deste ainda não estar totalmente terminado.

M	Ordenação	#F	t	Gerações	Expansões	Avaliações
5	1 2 3 4	6	0	40	33	40
6	1 2 3 4 5	13	0	166	102	166
7	1 2 3 4 5 6	14	0.03	753	358	753
8	1 2 3 4 5 6 7	12	0.09	2019	653	2019
9	1 2 3 4 5 6 7 8	15	0.5	10651	2995	10651
10	1 2 3 4 5 6 7 8 9	19	6.55	127168	37344	127168

Tabela 7.2: Eficiência da primeira versão do algoritmo branch and bound

7.1.3. Ordenação dos mercados

A ordenação prévia dos mercados trouxe ainda mais melhorias ao *branch and bound*, pois permitia efetuar imensos cortes mais cedo.

M	Ordenação	#F	t	Gerações	Expansões	Avaliações
5	4 3 2 1	6	0	34	21	34
6	4 5 3 2 1	13	0.01	165	91	165
7	6 3 5 2 4 1	14	0.03	708	304	708
8	7 2 3 1 6 4 5	12	0.1	2171	693	2171
9	7 8 5 6 2 3 4 1	15	0.51	10475	2901	10475
10	9 8 7 1 6 3 2 5 4	19	4.91	94933	26725	94933

Tabela 7.3: Eficiência do algoritmo branch and bound com ordenação de mercados

A tabela comparativa abaixo mostra significativas melhorias na maior parte dos casos quando se aplicou a ordenação de mercados antes de correr o algoritmo exato.

As percentagens positivas significam melhorias (redução de valores) e as negativas significam piorias (aumento de valores).

M	#F	t	Gerações	Expansões	Avaliações
5	IGUAIS	0.00%	15.00%	36.36%	15.00%
6	IGUAIS	0.00%	0.60%	10.78%	0.60%
7	IGUAIS	0.00%	5.98%	15.08%	5.98%
8	IGUAIS	-11.11%	-7.53%	-6.13%	-7.53%
9	IGUAIS	-2.00%	1.65%	3.14%	1.65%
10	IGUAIS	25.04%	25.35%	28.44%	25.35%

Tabela 7.4: Comparação do branch and bound antes e depois da ordenação dos mercados

7.1.4. Recálculo das distâncias

O recálculo das distâncias foi feito por razões lógicas. Faz muito mais sentido trabalhar com instâncias que respeitam a desigualdade triangular.

M	Ordenação	#F	t	Gerações	Expansões	Avaliações
5	4 3 2 1	7	0	34	21	34
6	4 5 3 2 1	13	0.01	160	81	160
7	6 3 5 2 4 1	14	0.03	778	335	778
8	7 2 6 5 3 4 1	13	0.13	2848	929	2848
9	7 8 3 6 5 4 1 2	18	0.88	18419	5499	18419
10	9 8 7 4 6 1 5 2 3	21	15.02	236149	77347	236149

Tabela 7.5: Eficiência do algoritmo branch and bound com ordenação de mercados e recálculo de distâncias

Nota-se que a eficiência do algoritmo piorou um pouco. Isso deve-se principalmente ao facto de haver mais soluções na fronteira não dominada do que antes.

Existem agora novas distâncias entre mercados, logo as soluções ótimas tendem a variar e podem aparecer novas soluções. O crescimento da fronteira implica um maior tempo de computação.

M	#F	t	Gerações	Expansões	Avaliações
5	1	0.00%	0.00%	0.00%	0.00%
6	0	0.00%	3.03%	10.99%	3.03%
7	0	0.00%	-9.89%	-10.20%	-9.89%
8	1	-30.00%	-31.18%	-34.05%	-31.18%
9	3	-72.55%	-75.84%	-89.56%	-75.84%
10	2	-205.91%	-148.75%	-189.42%	-148.75%

Tabela 7.6: Comparação do algoritmo exato antes e depois do recálculo de distâncias

7.1.5. Descartar mercados que não melhoram o preço da solução

Após utilização do recálculo de distâncias, é possível descartar mercados que não melhoram o preço da solução. Verifica-se se um mercado é útil durante a execução do algoritmo exato.

Apesar desta alteração baixar o número de gerações e expansões, o tempo de computação aumenta. Provavelmente, os métodos que suportam esta alteração são demasiado pesados.

Abaixo segue uma comparação dos resultados anteriores com os resultados obtidos ao descartar "maus" mercados.

M	t	Gerações	Expansões	Avaliações
5	0.00%	0.00%	0.00%	0.00%
6	0.00%	0.00%	0.00%	0.00%
7	0.00%	0.00%	0.00%	0.00%
8	0.00%	0.81%	0.22%	0.81%
9	-1.94%	1.09%	0.11%	1.09%
10	15.44%	3.75%	1.88%	3.75%
11	18.14%	6.31%	1.06%	6.31%
12	18.32%	8.23%	2.50%	8.23%

Tabela 7.7: Comparação entre antes e depois de se descartar mercados que não melhoram o preço

O baixo desempenho do algoritmo nestas condições, levou a que não se descartasse os mercados sem utilidade.

7.1.6. Maiores instâncias resolvidas

Nos resultados anteriores, mostraram-se apenas resoluções de instâncias até 10 mercados, para não se ir para tempos de computação muito altos. No entanto, conseguiu-se resolver instâncias até 14 mercados usando este algoritmo exato.

M	Ordenação	#F	t	Gerações	Expansões	Avaliações
11	10 8 9 5 7 6 3 1 2 4	18	28.41	335529	79961	335529
12	11 8 10 9 6 2 1 7 4 3 5	19	201.8	1887302	425291	1887302
13	12 2 1 11 10 7 9 8 5 3 4 6	25	919.08	8927574	1860538	8927574
14	1 13 2 3 12 11 8 10 4 9 6 5 7	26	1791.82	15306683	2663898	15306683

Tabela 7.8: Maiores instâncias resolvidas até ao momento

7.2. Algoritmos aproximados

As escolhas efetuadas nas afinações destes algoritmos estão ilustradas e justificadas no capítulo 6. De seguida, apresentam-se algumas comparações entre os dois algoritmos aproximados desenvolvidos.

7.2.1. Comparação individual

Antes de se proceder a comparações gerais, é importante mostrar as semelhanças e diferenças dos resultados obtidos em ambos os algoritmos aproximados.

Começa-se com a instância "*Singh33_2.33.50.1.5*" que retorna fronteiras mais curtas pois os preços variam apenas no intervalo [0, 5], e mostra-se a instância mais complexa "*Singh33_2.33.500.5.50000*", cujos preços variam entre 0 e 50000, e que resulta em fronteiras muito maiores.

Para podermos comparar de forma simples, dados tão diferentes e de tamanha dimensão, recorre-se a gráficos de linhas. As linhas vermelhas representam as soluções obtidas no algoritmo exato truncado, baseado em *branch and bound* (B&B). Já as linhas azuis representam as fronteiras obtidas pela escalada do monte.

Na Figura 7.1 e na Figura 7.2 podem observar-se as linhas do HC quase sempre abaixo das linhas do B&B, mostrando que a escalada do monte supera o algoritmo truncado, como seria de esperar.

O algoritmo truncado corre apenas durante 60 segundos, o que é muito pouco para instâncias de 33 mercados.

No entanto, as linhas do B&B estão muito próximas das do HC. Isso indica que o HC procurou na vizinhança de soluções muito boas. A fronteira inicial foi muito bem escolhida quando se inseriu soluções por proximidade do depósito. Esse fator torna o HC muito parecido com o algoritmo truncado.

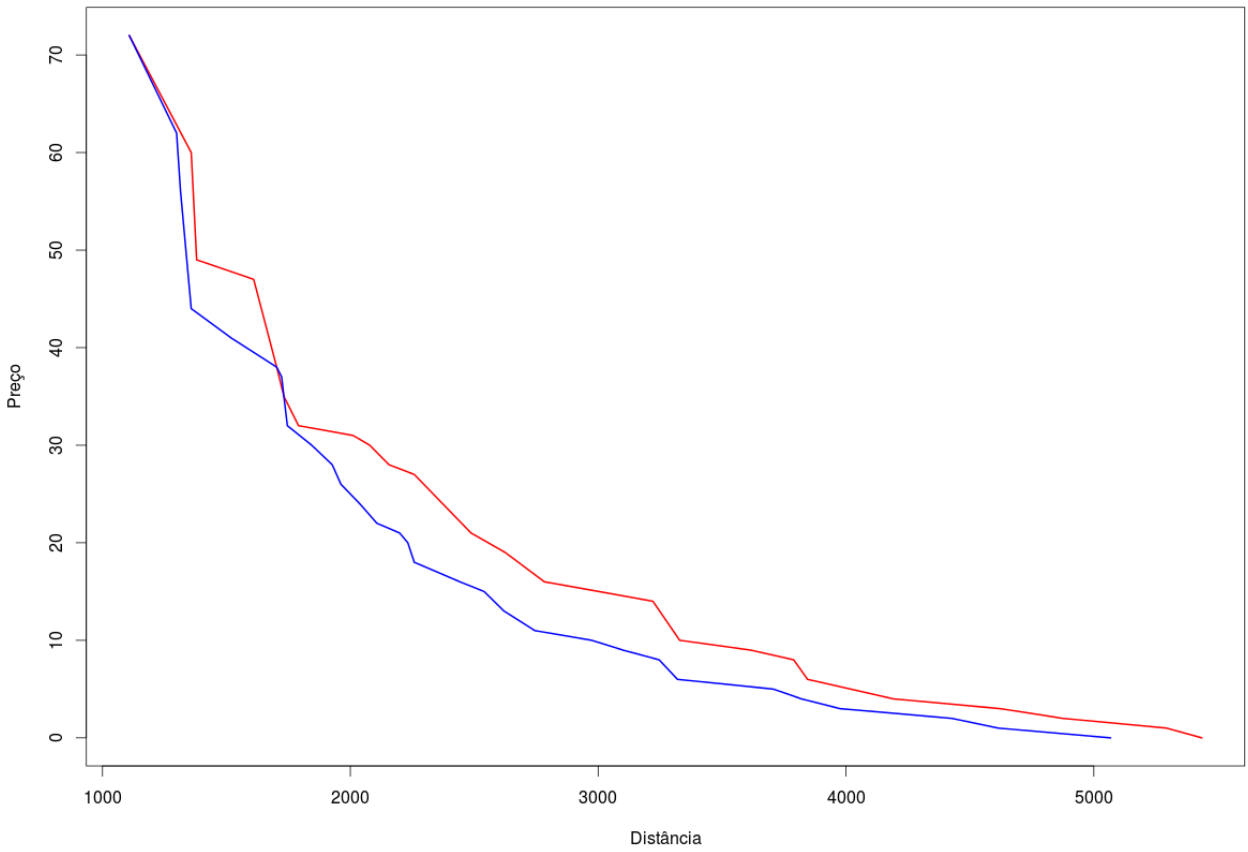

Figura 7.1: Comparação entre os dois algoritmos aproximados para a instância "Singh33_2.33.500.1.5"

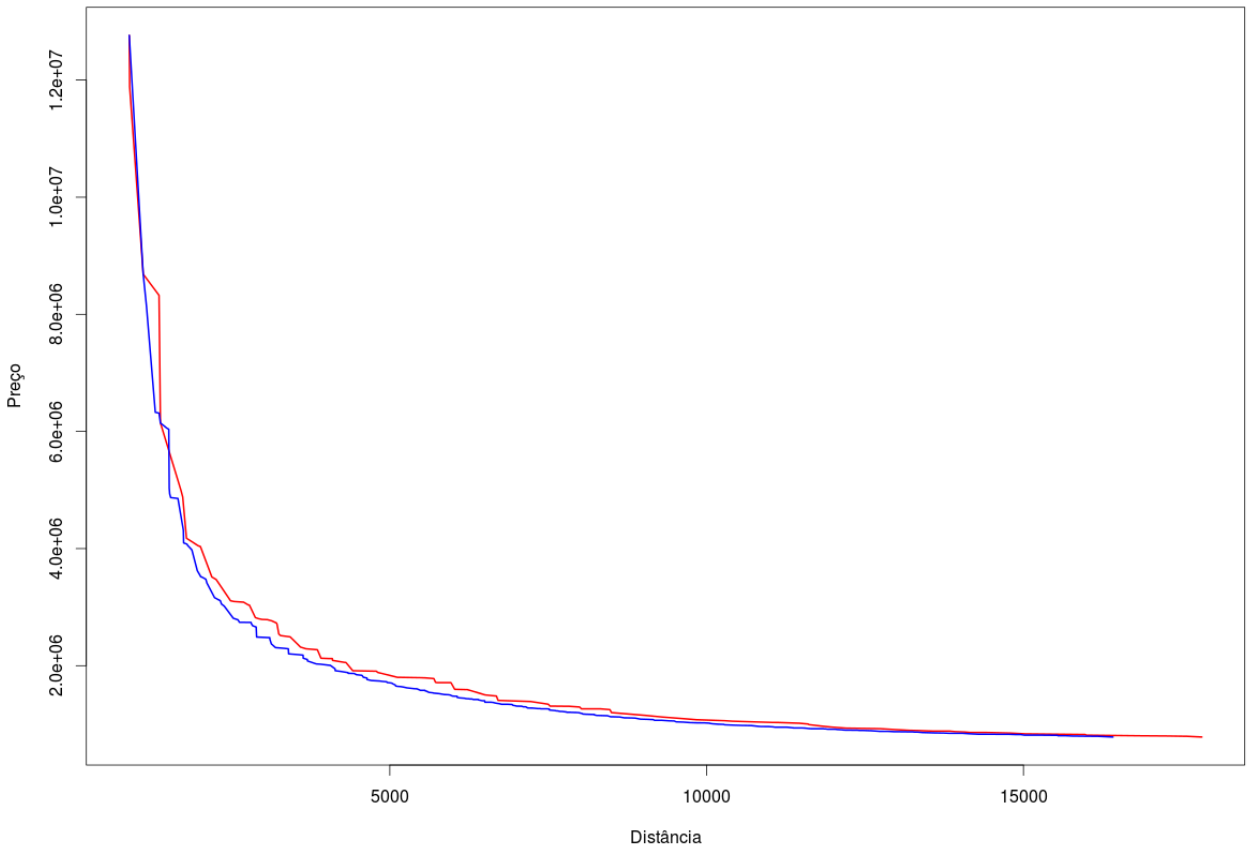

Figura 7.2: Comparação entre os dois algoritmos aproximados para a instância "Singh33_2.33.500.5.50000"

7.2.2. Comparação geral

Foram calculadas as soluções para as 450 instâncias descritas em 3.5 usando ambos os algoritmos aproximados e, tal como já foi referido em 6.2.1, a escalada do monte supera o algoritmo truncado.

A análise efetuada em 7.2.1 não foi repetida para todas as restantes instâncias. No entanto, para as 450 instâncias, foram recolhidos os seguintes dados:

- dimensão da fronteira
- menor distância e maior preço obtidos (primeira solução da fronteira)
- maior distância e menor preço obtidos (última solução da fronteira)

Usando esses dados, foi possível efetuar uma pequena análise estatística onde se constatou o seguinte:

(a) 98,9% das instâncias resultaram em soluções de maior dimensão usando o HC;

(b) as distâncias mínimas, os preços máximos e os preços mínimos mantém-se praticamente iguais entre as corridas dos dois algoritmos;

(c) em 95,1% das instâncias o HC conseguiu obter uma distância máxima menor do que o algoritmo truncado

Tendo em conta os pontos (a) e (c), verifica-se que já se havia constatado o mesmo em 7.2.1 e que esses factos tinham levado à conclusão de que o desempenho do HC é um pouco melhor do que o do algoritmo exato truncado.

7.3. Comparação com outros autores

A comparação de resultados com os de outros autores é de extrema importância num projeto desta envergadura. No entanto, tendo em conta que este trabalho é sobre o TPP bi-objetivo, o baixíssimo número de trabalhos desta natureza dificulta imenso na criação de dados comparativos de qualidade.

O único trabalho, de meu conhecimento, semelhante a este é o de Riera e Salazar (Riera & Salazar, 2005a). Assim, este será o único com o qual se pode tentar fazer uma comparação. No entanto, os resultados apresentados por estes autores têm pouquíssima informação relevante para este trabalho.

Riera e Salazar apresentam alguns resultados com as instâncias de Singh e van Oudheusden (Singh & van Oudheusden, 1997). Usam apenas as instâncias onde os valores dos preços estão no intervalo [1; 500], e onde o número de mercados é 50, 100, 150 e 200. Para cada valor do número de mercados, há 5 instâncias, sendo então o total de instâncias usadas nos resultados de 20. São calculadas as médias dos valores obtidos nas 5 instâncias com mesmo número de mercados. Assim, a tabela de resultados tem apenas 4 linhas.

Os indicadores usados por estes autores parecem ser demasiado referentes ao próprio algoritmo. Isso dificulta a comparação, pois não se consegue identificar indicadores gerais, tais como:

- número de soluções na fronteira obtida

- distâncias mínima e máxima da fronteira
- preços mínimo e máximo da fronteira

Assim, realiza-se uma comparação muito limitada, apenas baseando-nos no tempo de computação.

M	P	t
33	50	1196.2
	100	4462.2
	150	14726.2
	200	25296.0

Tabela 7.9: Resumo dos resultados de Riera & Salazar (2005a)

M	P	t
33	50	58.0
	100	56.8
	150	61.0
	200	61.6

Tabela 7.10: Resumos dos resultados do nosso algoritmo de Escalada do Monte

Esta comparação não nos revela muito além do facto de que Riera & Salazar parecem ter conseguido resolver as instâncias até ao fim, apesar da sua enorme complexidade.

Teria sido muito interessante poder comparar resultados usando os indicadores mencionados acima (número de soluções da fronteira, distâncias e preços mínimos e máximos), mas tal não foi possível.

8. Conclusão

O algoritmo exato, que lida com a questão bi-objetivo, funciona bastante bem, visto que o espaço de resultados aumenta exponencialmente com a adição de mercados. Apesar de não se ter conseguido resolver o leque de 450 instâncias de Singh & van Oudheusden (descritas em 3.5), o objetivo de resolver, até à otimalidade, instâncias de reduzida dimensão, foi cumprido. Conseguiu-se resolver instâncias até 14 mercados em tempo útil, obtendo-se fronteiras de eficiência ótimas. Ainda, o algoritmo exato desenvolvido providenciou excelentes heurísticas, cortes simples e outras ideias que podem ser aperfeiçoadas e usadas em algoritmos híbridos.

Verificou-se que é possível melhorar a eficiência de uma meta-heurística com a utilização de um algoritmo exato. Este trabalho oferece múltiplas ideias de cortes básicos fora do algoritmo principal. Esses cortes podem servir de base para trabalhos futuros de complexidade exponencial, e não apenas relacionados com o TPP.

As fronteiras exatas obtidas (até 14 mercados), usando o algoritmo baseado em *branch and bound*, servem de base de comparação para outros autores interessados neste tema. Além disso, o algoritmo exato considera dois objetivos (distância e preço) separadamente, em vez de somados, como na maioria da literatura. Tendo em conta que não existem resultados detalhados de algoritmos exatos sobre o TPP bi-objetivo, as fronteiras exatas obtidas aqui podem revelar-se muito úteis na realização de futuros trabalhos semelhantes a este.

Como o algoritmo exato não produziu o efeito desejado, desenvolveram-se dois algoritmos aproximados.

O algoritmo truncado é apenas o algoritmo exato com um limite de tempo, portanto, oferece resultados satisfatórios, pois as melhores soluções aparecem logo no início da procura, graças à boa reordenação de mercados. O objetivo de se criar um algoritmo híbrido é, de certa forma, alcançado com o algoritmo exato truncado.

O algoritmo de escalada do monte produz resultados ainda melhores do que o algoritmo truncado, pois procura apenas na vizinhança de soluções inicial que já são muito boas. O grande ponto forte deste HC é o facto deste usar heurísticas idênticas às do algoritmo exato.

O objetivo de se estudarem algumas meta-heurísticas, também não foi totalmente cumprido, pois só se conseguiu implementar um algoritmo de escalada do monte.

O impacto da reordenação de mercados nos resultados, de qualquer um dos algoritmos desenvolvidos, é excelente. Trata-se de uma das grandes descobertas deste trabalho. A escolha de uma boa ordenação inicial é fundamental para que o TPP seja resolvido com mais eficiência.

É muito importante, pensar-se na implementação de um método quantitativo de comparação de algoritmos cujos resultados são fronteiras de Pareto. Neste trabalho, as comparações foram feitas de forma aproximada e usando apenas algumas soluções das fronteiras. No entanto, um trabalho futuro poderá ser o desenvolvimento de um algoritmo de comparação de fronteiras de eficiência. Por exemplo, fica lançada uma direção para se comparar fronteiras de eficiência usando a área do polígono composto pelos pontos de abcissa igual ao valor da distância de uma solução e ordenada igual ao preço. Quanto menor a área, tanto melhor é a fronteira de eficiência.

Finalmente, a base de dados de soluções aproximadas (Rodrigues, 2016) são um bom resultado deste trabalho, que poderá servir de base de comparação para trabalhos futuros.

Bibliografia

Boctor, Fayez F., Gilbert Laporte, and Jacques Renaud. "Heuristics for the traveling purchaser problem." Computers & Operations Research 30.4 (2003): 491-504.

Bontoux B., Dominique Feillet D., Ant colony optimization for the traveling purchaser problem, Computers & Operations Research 35 (2008) 628 – 637

Burstall, Rodney Matineau. "A heuristic method for a job-scheduling problem." OR (1966): 291-304.

Buzacott, John A., and Sujit K. Dutta. "Sequencing many jobs on a multi-purpose facility." Naval Research Logistics Quarterly 18.1 (1971): 75-82.

Ehrgott, Matthias. Multicriteria optimization. Springer Science & Business Media, 2006

El-Dean, R. A. H. Z.. A Tabu Search Approach for solving the Travelling Purchaser Problem, 2008

Glover, Fred. "Future paths for integer programming and links to artificial intelligence." Computers & operations research 13.5 (1986): 533-549.

Glover, Fred. "Tabu search-part I." ORSA Journal on computing 1.3 (1989): 190-206.

Glover, Fred. "Tabu search—part II." ORSA Journal on computing 2.1 (1990): 4-32.

Glover, Fred, Gary A. Kochenberger. Handbook of metaheuristics. Springer Science & Business Media, 2003.

Goldbarg M. C., Bagi L. B., Goldbarg E. F. G., Transgenetic algorithm for the traveling purchaser problem, European Journal of Operational Research, 199(1), pp36-45, 2009

Kirkpatrick, Scott, C. Daniel Gelatt, and Mario P. Vecchi. "Optimization by simulated annealing." science 220.4598 (1983): 671-680.

Laporte G., Riera-Ledesma J. Salazar-González J. J., A Branch-and-Cut Algorithm for the Undirected Traveling Purchaser Problem, Operations Research, Vol. 51, No. 6, pp. 940-951, 2003

Ochi, Luiz S., Lucia Drummond, and Rosa Figueiredo. "Design and implementation of a parallel genetic algorithm for the travelling purchaser problem." Proceedings of the 1997 ACM

symposium on Applied computing. ACM, 1997.

Pearn W. L., Chien R. C., Improved solutions for the traveling purchaser problem, Computers & Operations Research, 25(11), pp879-885, 1998

Ravi, R., and F. Sibel Salman. "Approximation algorithms for the traveling purchaser problem and its variants in network design." Algorithms-ESA'99. Springer Berlin Heidelberg, 1999. 29-40.

Riera-Ledesma J., Salazar-González J. J., A heuristic approach for the Travelling Purchaser Problem, European Journal of Operational Research 162 (2005) 142–152

Riera Ledesma J., The Traveling Purchaser Problem, PhD thesis, DEIOC, Universidad de La Laguna, 2002

Riera-Ledesma J, Salazar-González J. J., The biobjective travelling purchaser problem, European Journal of Operational Research, Volume 160, Issue 3, Pages 599–613, Decision Analysis and Artificial Intelligence, 2005

Riera-Ledesma J., Salazar-González J. J., Solving school bus routing using the multiple vehicle traveling purchaser problem: A branch-and-cut approach, Computers & Operations Research , Volume 39, Issue 2, Pages 391–404, 2012

Rodrigues J, Soluções dos algoritmos aproximados da dissertação "O Problema do Mercador Viajante", https://sites.google.com/site/joaorodriguesinfor/mestrado-mw/dissertacao [19 de dezembro de 2016]

Russell, Stuart, and Peter Norvig. "Artificial intelligence: a modern approach." (1995).

Sean, Luke, "1 Essentials of Metaheuristicsl.", Second Edition , Online Version 2.2 , October, 2015.

Sels, Veronique, et al. "Hybrid tabu search and a truncated branch-and-bound for the unrelated parallel machine scheduling problem." Computers & Operations Research 53 (2015): 107-117.

Singh K. N., van Oudheusden D. L., A branch and bound algorithm for the traveling purchaser problem, European Journal of Operational Research, Volume 97, Pages 571–579, 1997

Sörensen, Kenneth. "Metaheuristics—the metaphor exposed." International Transactions in Operational Research 22.1 (2015): 3-18.

Voß S., Dynamic tabu search strategies for the traveling purchaser problem, Annals of

Operations Research 63.2, pp253-275, 1996

Woeginger, Gerhard J. "Exact algorithms for NP-hard problems: A survey." Combinatorial Optimization—Eureka, You Shrink!. Springer Berlin Heidelberg, 2003. 185-207.