UNIVERSIDADE DE LISBOA

FACULDADE DE CIÊNCIAS

DEPARTAMENTO DE ESTATÍSTICA E INVESTIGAÇÃO OPERACIONAL

Meta-heurísticas para o problema do caixeiro viajante com seleção de hotéis

Mestrado em Estatística e Investigação Operacional
Especialização em Investigação Operacional

Raquel Maria Pinheiro Teixeira

Dissertação orientada por:
Prof.ª Doutora Ana Maria Duarte Silva Alves Paias
Prof.ª Doutora Marta Guerreiro Duarte Mesquita de Oliveira

2016

Finalizada esta etapa não poderia deixar de agradecer a todos os que, direta ou indiretamente, contribuíram para a concretização desta dissertação.

Em primeiro lugar, quero agradecer às minhas orientadoras, Professora Doutora Ana Maria Duarte Silva Alves Paias e Professora Doutora Marta Guerreiro Duarte Mesquita de Oliveira, pela colaboração e por todos os conhecimentos partilhados. Um especial agradecimento à Professora Ana Paias por toda a disponibilidade e apoio prestados. Foi uma honra.

Em segundo, agradeço a todos os professores da licenciatura em Matemática e do mestrado em Estatística e Investigação Operacional pelos conhecimentos transmitidos.

Por fim, mas não menos importante, agradeço à minha família e amigos por sempre me apoiarem e acreditarem em mim. Agradeço aos meus colegas de faculdade que me acompanharam nesta jornada, em especial à Catarina Mateus e ao João Silva pela companhia e por me incentivarem e não me deixarem desistir nos momentos mais difíceis. Obrigada a todos por fazerem parte da minha vida.

Raquel

Resumo

O problema do caixeiro viajante (TSP) é um problema com diversas aplicações na medida em que muitas das situações da vida real podem ser modeladas como um problema do caixeiro viajante. Deste modo, ao longo dos anos, o TSP tem sido dos mais estudados em Investigação Operacional. Nesta dissertação é estudada uma variante do TSP, nomeadamente o problema do caixeiro viajante com seleção de hotéis (TSPHS), o qual tem igualmente diversas aplicações práticas.

Dados um conjunto de clientes a visitar e um conjunto de potenciais hotéis para pernoitar, sendo conhecidas as distâncias entre clientes e entre clientes e hotéis, o TSPHS consiste em determinar uma rota que visite todos os clientes começando e acabando num mesmo hotel pré-definido. A rota deverá ser tal que possa ser dividida em percursos com duração não superior a um dado limite de tempo inferior a um dia. Por outro lado, cada percurso deverá ter início e fim em algum dos hotéis disponíveis, não tendo necessariamente que ser o mesmo. O objetivo é minimizar o número de percursos e simultaneamente a distância total percorrida.

Sendo o TSPHS uma extensão do TSP, ele é também um problema NP-difícil, pelo que a existência de métodos heurísticos como auxílio à resolução destes problemas é fundamental. Nesta dissertação são apresentadas duas meta-heurísticas baseadas em algoritmos genéticos combinados com um procedimento de pesquisa local, que considera oito operadores de vizinhança, e um algoritmo de perturbação de soluções.

Foram testados vários parâmetros por forma a obter os melhores resultados para as instâncias de referência utilizadas. Os resultados obtidos foram comparados com os resultados de outros autores. Para uns casos são obtidas melhores soluções para outros não.

Palavras-chave: Caixeiro viajante com seleção de hotéis, Meta-heurísticas, Algoritmos genéticos, Pesquisa local

Abstract

The traveling salesman problem (TSP) has many applications since there are many real-life situations which can be modeled as a traveling salesman problem. Thus, over the years, TSP has been one of the most studied problems in Operational Research. This thesis addresses a variant of TSP, namely the traveling salesman problem with hotel selection (TSPHS). The TSPHS has also many practical applications.

Given a set of customers to be visited and a set of potential hotels to spend the night, the TSPHS consists in determining one route starting and ending in one given hotel that visits all customers. The route is divided into trips lasting no more than a given limit of time. Moreover, each trip must begin and end at any of the available hotels, not necessarily the same. The goal is minimize the number of trips and simultaneously the total traveled distance.

Being the TSPHS an extension of the TSP, it is also an NP-hard problem, so the development of heuristic methods as an aiding tool to solve these problems is crucial. This thesis proposes two meta-heuristics based on genetic algorithms combined with a local search procedure which uses eight neighborhood operators and a solution perturbation algorithm.

Various parameters were tested in order to obtain the best results for the used benchmark instances. The results obtained were compared with results of other authors. For some cases are obtained better solutions for others not.

Keywords: Travelling salesperson problem with hotel selection, Meta-heuristics, Genetic algorithms, Local search

Índice

Lista de Figuras **vii**

Lista de Tabelas **x**

Lista de Algoritmos **xi**

1 Introdução **1**

2 Revisão bibliográfica **4**
2.1 O TSPHS na literatura 4
2.2 Problemas de otimização relacionados com o TSPHS 6

3 Descrição do problema **9**
3.1 Introdução 9
3.2 Formulação 10
3.3 Exemplo de aplicação 13

4 Metodologias de resolução **14**
4.1 Algoritmos genéticos 14
4.2 Pesquisa local 30
4.2.1 Operadores que otimizam a solução ao nível do percurso 32
4.2.2 Operadores que otimizam a solução a nível da rota 40
4.2.3 Outros operadores 44
4.3 Procedimentos de perturbação 54

5 Resultados computacionais **60**
5.1 Instâncias de teste 60
5.2 Afinação das meta-heurísticas 63
5.3 Comparação com os resultados de outros autores 65
5.3.1 Resultados obtidos para o SET1 e para o SET2 66
5.3.2 Resultados obtidos para o SET3 e para o SET4 75
5.3.3 Síntese dos resultados obtidos 82

6 Conclusões **85**
6.1 Principais conclusões 85
6.2 Desenvolvimentos futuros 86

Referências **89**

A Otimização dos parâmetros **91**
A.1 Algoritmo genético 1 93
A.2 Algoritmo genético 2 95

B Resultados finais **99**

Lista de Figuras

4.1 Mecanismo de seleção do tipo roleta usado no AG1 24
4.2 Mecanismo de seleção do tipo torneio usado no AG2 25
4.3 Operador cruzamento OX 28
4.4 Operador mutação baseado na ordenação 29
4.5 Exemplo de aplicação do operador *2-Opt* 33
4.6 Exemplo de aplicação do operador *Swap-within* 35
4.7 Exemplo de aplicação do operador *Move1-within* 37
4.8 Exemplo de aplicação do operador *Move2-within* 38
4.9 Exemplo de aplicação do operador *Swap-between* 42
4.10 Exemplo de aplicação do operador *Move-between* 44
4.11 Exemplo de aplicação do operador *Inversion* 47
4.12 Exemplo de aplicação da fase 1 do operador *Destroy-Insert-Hotels* 50
4.13 Exemplo de aplicação da fase 2 do operador *Destroy-Insert-Hotels* 52
4.14 Exemplos de aplicação da perturbação *Move-chain* 56
4.15 Exemplos de aplicação da perturbação *Swap-chain* 56

5.1 Estrutura de um ficheiro de dados de cada uma das instâncias do TSPHS 62

Lista de Tabelas

5.1 Teste ao efeito do procedimento de pesquisa local 64
5.2 Resultados obtidos para o SET1 69
5.3 Resultados obtidos para o SET2 (grupo com 10 clientes) 70
5.4 Resultados obtidos para o SET2 (grupo com 15 clientes) 71
5.5 Resultados obtidos para o SET2 (grupo com 30 clientes) 72
5.6 Resultados obtidos para o SET2 (grupo com 40 clientes) 73
5.7 Resultados obtidos para o SET1 considerando apenas o procedimento de pesquisa local e o algoritmo de perturbação 74
5.8 Valores atribuídos às penalizações e ao *nger* para cada instância dos SET3 e SET4 75
5.9 Resultados obtidos para o SET3 (grupo com 3 hotéis extra) 78
5.10 Resultados obtidos para o SET3 (grupo com 5 hotéis extra) 79
5.11 Resultados obtidos para o SET3 (grupo com 10 hotéis extra) 80
5.12 Resultados obtidos para o SET4 81
5.13 Resultados médios obtidos por conjunto 82

A.1 Comparação mecanismo de seleção roleta *vs* torneio 91
A.2 Comparação cruzamento uniforme *vs* cruzamento OX na amostra do SET1 92
A.3 Comparação cruzamento uniforme *vs* cruzamento OX na amostra do SET2 92
A.4 Comparação dos mecanismos de substituição incremental na amostra do SET1 92
A.5 Comparação dos mecanismos de substituição incremental na amostra do SET2 93
A.6 Teste ao número de descendentes ($ndesc$=10 e $ndesc$=14) e à probabilidade de mutação ($pmut$=0,2 e $pmut$=0,3) no AG1 94
A.7 Teste ao número de gerações no AG1: comparação dos resultados obtidos com 2000, 4000 e 6000 gerações 94
A.8 Teste ao número de gerações no AG1: comparação dos resultados obtidos com 4000 e 5000 gerações 95
A.9 Teste ao número de descendentes e à probabilidade de mutação considerando o cruzamento CX no AG2 96
A.10 Teste ao número de descendentes e à probabilidade de mutação considerando o cruzamento CX no AG2 96

A.11 Teste ao número de descendentes e à probabilidade de mutação considerando o cruzamento OX no AG2 97
A.12 Teste ao número de gerações no AG2: comparação dos resultados obtidos com 2000 e 4000 gerações 97

B.1 Resultados finais obtidos para o SET1 com MH1 99
B.2 Resultados finais obtidos para o SET1 com MH2 99
B.3 Resultados finais obtidos para o SET2 (10 clientes) com MH1 100
B.4 Resultados finais obtidos para o SET2 (10 clientes) com MH2 100
B.5 Resultados finais obtidos para o SET2 (15 clientes) com MH1 100
B.6 Resultados finais obtidos para o SET2 (15 clientes) com MH2 100
B.7 Resultados finais obtidos para o SET2 (30 clientes) com MH1 101
B.8 Resultados finais obtidos para o SET2 (30 clientes) com MH2 101
B.9 Resultados finais obtidos para o SET2 (40 clientes) com MH1 101
B.10 Resultados finais obtidos para o SET2 (40 clientes) com MH2 101
B.11 Resultados finais obtidos para o SET3 (3 hotéis) com MH1 102
B.12 Resultados finais obtidos para o SET3 (3 hotéis) com MH2 102
B.13 Resultados finais obtidos para o SET3 (5 hotéis) com MH1 102
B.14 Resultados finais obtidos para o SET3 (5 hotéis) com MH2 103
B.15 Resultados finais obtidos para o SET3 (10 hotéis) com MH1 103
B.16 Resultados finais obtidos para o SET4 com MH1 103
B.17 Resultados finais obtidos para o SET4 com MH2 104

Lista de Algoritmos

4.1 Descodificador 20
4.2 Retira-clientes-da-rota 21
4.3 Estrutura básica dos algoritmos genéticos propostos para o TSPHS 31
4.4 Pseudo-código para pesquisar na vizinhança induzida pelo *2-Opt* 34
4.5 Pseudo-código para pesquisar na vizinhança induzida pelo *Swap-within* . . 36
4.6 Pseudo-código para pesquisar na vizinhança induzida pelo *Move1-within* . 39
4.7 Pseudo-código para pesquisar na vizinhança induzida pelo *Move2-within* . 41
4.8 Pseudo-código para pesquisar na vizinhança induzida pelo *Swap-between* . 43
4.9 Pseudo-código para pesquisar na vizinhança induzida pelo *Move-between* . 45
4.10 Pseudo-código para pesquisar na vizinhança induzida pelo *Inversion* . . . 49
4.11 Pseudo-código para pesquisar na vizinhança induzida pela fase 1 do *Destroy-Insert-Hotels* 51
4.12 Pseudo-código para pesquisar na vizinhança induzida pela fase 2 do *Destroy-Insert-Hotels* 53
4.13 Procedimento de pesquisa local 55
4.14 Procedimento da perturbação *Move-chain* 56
4.15 Procedimento da perturbação *Swap-chain* 57
4.16 Procedimento da perturbação *Shake-chain* 57
4.17 Procedimento da perturbação *Inversion-chain* 58
4.18 Procedimento de perturbação de uma solução 59

Capítulo 1

Introdução

Com o aumento da competitividade, principalmente devido à globalização, as empresas vêem-se obrigadas a responder com rapidez e flexibilidade, tornando-se crucial a otimização dos recursos disponíveis reduzindo os custos e cumprindo os serviços a que se propõem.

Problemas relacionados com o desenho de rotas têm sido dos mais estudados na área da investigação operacional devido às suas inúmeras aplicações reais. Considere-se o problema do caixeiro viajante (TSP), um dos problemas de otimização mais estudados na área de Investigação Operacional, o qual consiste em determinar uma rota que minimize a distância total percorrida, visitando cada local uma única vez, e que inicie e termine no mesmo local.

Existem, no entanto, casos em que os locais/clientes que têm que ser visitados não se encontram próximos, tornando-se impossível que o caixeiro viajante visite todos os clientes num mesmo dia. Assim, no final do dia, ele terá de optar por algum local para passar a noite e retomar o trabalho no dia seguinte a partir desse mesmo local. Esta observação motivou a definição de uma nova variante do TSP, nomeadamente o problema do caixeiro viajante com seleção de hotéis (TSPHS).

Considere-se um conjunto de locais/clientes a visitar e um conjunto de hotéis disponíveis, sendo conhecidas as distâncias entre clientes e entre clientes e hotéis. O TSPHS consiste em determinar uma rota com início e fim num dado hotel pré-fixado (depósito) de modo que todos os clientes sejam visitados uma única vez. A rota é dividida em percursos com duração limitada pelo número máximo de horas de trabalho diário. Cada percurso está associado a um dia e contém o hotel de partida, a sequência de clientes a visitar e o hotel de chegada. Pretende-se minimizar o número de percursos necessários para visitar todos os clientes, bem como a distância total percorrida.

De referir que:

– o hotel de partida (depósito) da rota é também o hotel de chegada, deste modo o hotel inicial do primeiro percurso assim como o hotel final do último percurso deverá ser o depósito;

– o caixeiro viajante retoma o trabalho a partir do hotel em que passou a noite, logo o hotel final de um dado percurso corresponderá ao hotel inicial do percurso seguinte;

– cada hotel pode ser utilizado mais do que uma vez ao longo da rota.

De notar ainda que na contextualização destes problemas o custo de um dia extra de trabalho é bastante superior ao custo de percorrer mais alguns quilómetros, pelo que uma rota com poucos percursos é preferível a uma rota com mais percursos mesmo que esta última possua uma menor distância total percorrida.

Existem diversas aplicações reais do TSPHS. Considerem-se de seguida alguns exemplos: um turista que queira visitar um conjunto de atrações mas que devido à distância entre estas tenha que planear a visita por vários dias; um camião de distribuição de gasolina ou gasóleo que devido à sua capacidade não consiga visitar todos os clientes sem reabastecer e tenha de escolher um local para o fazer; as rotas dos veículos elétricos que têm de ser divididas em percursos cuja duração máxima é dada pela autonomia da bateria sendo necessário escolher os locais onde recarregar a bateria. Um outro exemplo, o qual tem particular interesse por parte das empresas e transportadores de mercadorias, diz respeito aos camionistas que têm que proceder às entregas das encomendas em diversos locais da Península Ibérica ou da Europa cujo número de horas de trabalho por dia está limitado por lei (Regulamento (CE) nº 561/2006 de 15 de Março de 2006, [21]). De forma a melhorar as condições de trabalho e a segurança rodoviária, os motoristas de veículos pesados de mercadorias e de passageiros têm regulamentações específicas quanto aos tempos de condução e períodos de repouso que têm de ser compridas.

Esta dissertação tem como objetivo a apresentação de uma nova metodologia de resolução para o TSPHS, o qual é um problema NP-difícil. Deste modo, a existência de métodos heurísticos que permitam obter boas soluções num tempo computacional aceitável é essencial. Foram então desenvolvidas duas meta-heurísticas para o problema.

A dissertação encontra-se organizada em seis capítulos. No capítulo seguinte é feita uma breve revisão à literatura existente, apresentando-se alguns problemas com características semelhantes às do TSPHS. No capítulo 3 procede-se a uma descrição mais pormenorizada do problema e apresenta-se uma formulação matemática para o mesmo. No capítulo 4 é descrita a metodologia de resolução sugerida para o problema, nomeadamente duas variantes de um algoritmo genético combinadas com um procedimento de pesquisa local e um algoritmo de perturbação de soluções. No capítulo 5 procede-se à calibração dos parâmetros dos algoritmos propostos bem como a apresentação e análise dos resultados computacionais obtidos. Por fim, no capítulo 6 são referidas as principais conclusões e sugestões para trabalhos futuros.

Capítulo 2

Revisão bibliográfica

O presente capítulo subdivide-se em duas secções. Na primeira secção procede-se a uma breve apresentação dos métodos desenvolvidos até ao momento para resolver problemas do caixeiro viajante com seleção de hotéis. Posteriormente, numa outra secção, são descritos alguns problemas relacionados com o TSPHS, apresentando-se as principais características comuns e/ou principais diferenças entre os problemas.

2.1 O TSPHS na literatura

O TSPHS é um problema relativamente recente. Foi introduzido pela primeira vez em 2012 por Vansteenwegen *et al.* [19] e desde então diversos trabalhos têm sido desenvolvidos sobre este problema. Vansteenwegen *et al.* apresentaram um modelo matemático de programação linear inteira mista (PLIM) para resolver o TSPHS considerando um número fixo de percursos. Um limite inferior para o número de percursos é obtido tomando em consideração apenas o tempo de serviço dos clientes e o limite para a duração de cada percurso. Posteriormente resolve-se o modelo matemático considerando esse número de percursos. Caso o problema não tenha solução, o número de percursos vai sendo sucessivamente aumentado até que uma solução admissível seja encontrada. Os autores desenvolveram também duas heurísticas que diferem no método de construção da solução inicial mas que incluem a mesma fase de melhoramento da solução. Numa das heurísticas a solução inicial é gerada com base no princípio do vizinho mais próximo e na outra é gerada de forma análoga ao mecanismo *split* proposto por Prins [14]. A fase de melhoramento é composta por trocas locais existentes na literatura bem como algumas desenhadas especialmente para este problema. Os autores compararam os resultados obtidos pelo modelo matemático com os obtidos através da melhor das duas heurísticas, utilizando para tal instâncias de teste por eles introduzidas, e concluíram que a heurística inspirada no mecanismo *split* permitiu a obtenção de boas soluções com tempos computacionais relativamente pequenos.

Em 2013, Castro *et al.* [3] propuseram uma nova formulação matemática para o TSPHS cuja função objetivo minimiza simultaneamente o número de percursos e a distância total percorrida, atribuindo-se uma penalização ao número de percursos da

solução de modo a penalizar soluções com maior número de percursos, devido ao facto do custo associado a um dia extra de trabalho ser geralmente bastante superior ao custo de percorrer mais alguns quilómetros. Como tal, soluções com menor número de percursos são sempre preferíveis a soluções com mais percursos mesmo que correspondam a uma menor distância total percorrida. Os autores sugeriram também um algoritmo memético combinado com *tabu search*. Um algoritmo memético utiliza as operações básicas de um algoritmo genético (cruzamento, seleção e mutação), sendo os indivíduos produzidos melhorados com procedimentos de pesquisa local. Os resultados obtidos com esta meta-heurística verificaram-se mais eficientes, no que respeita à qualidade das soluções, relativamente aos obtidos em [19].

Em 2014, Castro *et al.* [4] apresentaram uma nova formulação matemática para o TSPHS baseada na partição de conjuntos e propuseram uma nova meta-heurística. Uma solução inicial é obtida utilizando um método *order-first split-second*, segundo o qual se obtém primeiramente uma solução para o TSP. Tendo em conta as restrições de tempo, procede-se à partição dessa solução em percursos tornando-a admissível para o TSPHS. Essa partição é feita de um modo inspirado no mecanismo *split* proposto por Prins [14]. Posteriormente procede-se a uma estratégia de melhoramento da solução constituída por uma fase de perturbação, utilizada para introduzir diversidade, seguida de um procedimento de pesquisa local considerando diferentes vizinhanças. Obtiveram-se resultados competitivos face aos obtidos pelo algoritmo memético e com tempos computacionais inferiores a este. No entanto, até ao momento, continua a ser com o algoritmo memético que são obtidas as melhores soluções para as instâncias de teste criadas por Vansteenwegen *et al.*.

Em 2015, Baltz *et al.* [1] introduziram uma variante do TSPHS, nomeadamente o problema do caixeiro viajante com seleção de hotéis e janelas temporais. O objetivo nesta variante é determinar uma rota que visite vários clientes, atendendo às correspondentes janelas temporais, minimizando os custos da rota: salário do caixeiro viajante, as despesas com hotéis, despesas de viagem e as taxas de penalização referentes aos clientes que não são visitados. Os autores desenvolveram um modelo PLIM e uma heurística que combina técnicas como a inserção de menor custo e *2-Opt* com procedimentos aleatórios. Baltz *et al.* mostraram que, para as instâncias de menor dimensão, a heurística proposta produz as mesmas soluções encontradas pelo modelo PLIM num curto tempo computacional. Para instâncias de maior dimensão, e para as quais se conheciam as soluções ótimas, a heurística permitiu a obtenção de soluções de boa qualidade.

Também em 2015, Sousa *et al.* [17] deram seguimento ao estudo do TSPHS. Os autores desenvolveram uma heurística denominada *Client Perturbation Variable Neighborhood Search* inspirada num procedimento de pesquisa local que considera várias vizinhanças. A construção da solução inicial é considerada como pertencente ao grupo de técnicas para geração de soluções *order-first-split-second*. Inicialmente é aplicada a heurística de Lin-Kernighan [11], seguida do algoritmo de Dijkstra. Posteriormente são exploradas soluções vizinhas através de quatro estruturas de vizinhança. Para complementar a eficácia da exploração de soluções vizinhas é introduzido um procedimento que efetua uma perturbação na posição dos clientes na rota. Os resultados obtidos

verificaram-se competitivos com os melhores resultados existentes na literatura e com tempos computacionais inferiores. De um modo geral, os resultados obtidos foram melhores que os obtidos através da meta-heurística apresentada por Castro *et al.* em [4].

2.2 Problemas de otimização relacionados com o TSPHS

O TSPHS possui algumas características comuns com diversos problemas de otimização existentes na literatura, dos quais é possível destacar o problema do caixeiro viajante múltiplo (mTSP), o problema de roteamento de veículos (VRP) e o problema de localização-distribuição (LRP).

O mTSP [2] corresponde a uma extensão do TSP e consiste em determinar m rotas, cada uma para um caixeiro viajante, por forma a que todos os clientes sejam visitados e tendo como objetivo a minimização da distância total percorrida. Tal pode ser abordado como uma única rota que pode ser dividida em diversos percursos (tantos quanto o número de caixeiros viajantes). Cada percurso corresponde à rota de um caixeiro viajante e tem início e fim no depósito. Esta é uma diferença relativamente ao TSPHS, na medida em que no TSPHS cada percurso começa num dado hotel e não tem necessariamente que terminar no mesmo. Uma outra diferença diz respeito ao facto de no mTSP não existir uma restrição à distância total percorrida por cada caixeiro viajante.

O VRP [18] consiste em determinar diferentes rotas (uma para cada um dos veículos disponíveis) por forma a visitar todos os clientes tendo como principal objetivo a minimização da distância total percorrida, sendo que cada veículo possui uma dada capacidade. De modo análogo ao que foi dito relativamente ao mTSP, também aqui as soluções podem ser vistas como uma única rota composta por diferentes percursos, correspondendo cada um deles à rota de um veículo, com início e fim num mesmo hotel (depósito). Também este problema difere do TSPHS na medida em que se considera que cada percurso deve iniciar e terminar no mesmo hotel, porém dada a capacidade dos veículos ser limitada tal pode ser visto como uma restrição à distância máxima percorrida por cada veículo em cada percurso ou como uma restrição à quantidade da procura dos clientes que pode ser afeta a cada veículo (conhecidas as procuras dos clientes), existindo, deste modo, limitações à duração dos percursos.

Dados um conjunto de possíveis localizações de depósitos e um conjunto de clientes a serem visitados, no LRP [12] determinam-se que depósitos serão instalados para servir todos os clientes por forma a minimizar o custo total correspondente aos custos de instalação dos depósitos e aos custos de distribuição, isto é, das rotas de cada veículo. Admite-se que cada veículo deve regressar ao depósito onde inicia a rota. A principal diferença entre o LRP e o TSPHS é que no LRP cada rota de um veículo começa e termina no mesmo hotel (depósito) enquanto no TSPHS os percursos podem iniciar e terminar em hotéis diferentes. No LRP não existe garantia que os depósitos estejam conectados enquanto no TSPHS há que garantir a conectividade entre os percursos. De notar que, no caso das localizações dos depósitos estarem pré-fixadas, o LRP reduz-se ao MDVRP.

Um outro problema relacionado com o TSPHS, é o problema de orientação com seleção de hotéis (OPHS), introduzido em 2013 por Divsalar *et al.* [8]. No OPHS, dados um conjunto de locais cada um com uma dada pontuação associada e um conjunto de possíveis hotéis, pretende-se determinar uma rota definindo a ordem dos locais a visitar por forma a maximizar a soma total das pontuações e garantindo que em cada dia um dado tempo limite de viagem não é excedido. A rota tem início e fim num dado hotel pré-fixado e é dividida em diferentes percursos conectados, cada um com duração não superior ao tempo limite de viagem diário e com início e fim num dos hotéis disponíveis. De notar que no TSPHS temos associado a cada local o tempo de visita, que poderá ser visto como o simétrico da pontuação no OPHS. São óbvias as principais diferenças entre o OPHS e o TSPHS, nomeadamente no TSPHS todos os clientes têm de ser visitados, o que não é obrigatório no OPHS. Além disso, no TSPHS o objetivo é minimizar a distância total percorrida e o número de percursos, enquanto no OPHS o número de percursos está pré-definido. Em 2014, Divsalar *et al.* [9] desenvolveram um algoritmo memético para o OPHS.

De referir ainda um outro problema que possui grande similaridade com o TSPHS, nomeadamente o problema do caixeiro viajante agrupado (CTSP). O CTSP [5] é uma variante do TSP no qual o conjunto dos locais a visitar é particionado em *clusters* de acordo com as localizações geográficas dos clientes. O CTSP tem como objetivo principal encontrar um ciclo hamiltoniano de custo mínimo que visite todos os clientes e que seja tal que os clientes de cada *cluster* sejam visitados em sequência. Os percursos no TSPHS podem ser vistos como os *clusters* no CTSP, onde a escolha da ligação entre dois *clusters* é entendida como a escolha do melhor hotel entre dois percursos no TSPHS.

Capítulo 3

Descrição do problema

No presente capítulo será apresentada uma descrição mais formal do problema em estudo bem como uma formulação matemática para o mesmo. Posteriormente, para melhor perceção do problema, será ilustrado um exemplo de aplicação, o qual será igualmente utilizado nos próximos capítulos como auxílio à explicação de alguns conceitos.

3.1 Introdução

Dados um conjunto $C = \{1, ..., n\}$ composto pelos n clientes que necessitam de ser visitados e um conjunto $H = \{n + 1, ..., n + h\}$ com os h hotéis disponíveis, o TSPHS pode ser definido num grafo completo $G = (V, E)$ onde o conjunto dos vértices $V = H \cup C$ é o conjunto de todos os locais e o conjunto dos arcos $E = \{(i, j)|i, j \in V, i \neq j\}$ é o conjunto de todas as ligações, sendo que (i, j) representa a ligação entre o cliente ou hotel i e o cliente ou hotel j. Cada cliente $i \in C$ requer um tempo de visita τ_i (com $\tau_i = 0, \forall i \in H$). É também conhecido o tempo de viagem b_{ij} proporcional à distância percorrida entre cada par de localidades i e j. O objetivo do TSPHS é determinar uma rota que minimize simultaneamente o número de percursos necessários para visitar todos os clientes e a distância total percorrida.

A rota deve ter início e fim num mesmo hotel pré-fixado (hotel **n+1**). Outras restrições têm que ser tidas em conta, nomeadamente:

- Cada percurso deverá iniciar e terminar num dos hotéis disponíveis, podendo ou não iniciar e terminar no mesmo hotel;
- O hotel inicial de um dado percurso terá de ser igual ao hotel final do percurso imediatamente anterior por forma a garantir a conetividade dos percursos;
- A duração de cada percurso não deve exceder um dado limite de tempo L, sendo que a duração de um percurso é definida como sendo a soma do tempo de viagem com os tempos de servir os clientes visitados.

De referir que, uma vez que um hotel pode ser visitado várias vezes na mesma rota, a solução do TSPHS não é necessariamente um ciclo elementar como acontece no TSP.

Denote-se ainda que as localizações dos hotéis podem coincidir com as localizações dos clientes.

3.2 Formulação

Seja D um majorante para o número de percursos na solução ótima. Por forma a modelar o TSPHS são considerados dois conjuntos de variáveis. Consideram-se variáveis binárias x_{ij}^d que tomam valor 1 se no percurso d após visitar o cliente ou hotel i se visita o cliente ou hotel j e 0 caso contrário, $\forall\,(i,j) \in E, d = 1, ..., D$. Também se consideram as variáveis binárias y^d que tomam valor 1 se algum cliente ou hotel é visitado no percurso d e 0 caso contrário, $d = 1, ..., D$.

Para descrever o TSPHS Castro *et al.* [3] propuseram a seguinte formulação matemática:

$$minimizar\ M_1 \sum_{d=1}^{D} y^d + \sum_{d=1}^{D} \left(\sum_{(i,j)\in E} b_{ij} x_{ij}^d \right) \tag{3.1}$$

sujeito a:

$$\sum_{d=1}^{D} \sum_{i\in V} x_{ij}^d = 1, \qquad \forall j \in C \tag{3.2}$$

$$\sum_{i\in V} x_{ij}^d = \sum_{i\in V} x_{ji}^d, \qquad \forall j \in C, \qquad d = 1, ..., D \tag{3.3}$$

$$\sum_{h\in H} \sum_{j\in V\setminus\{h\}} x_{hj}^d = y^d, \qquad d = 1, ..., D \tag{3.4}$$

$$\sum_{h\in H} \sum_{i\in V\setminus\{h\}} x_{ih}^d = y^d, \qquad d = 1, ..., D \tag{3.5}$$

$$\sum_{(i,j)\in E} (b_{ij} + \tau_j)\ x_{ij}^d \leq L, \qquad d = 1, ..., D \tag{3.6}$$

$$\sum_{j\in V\setminus\{n+1\}} x_{n+1j}^1 = 1 \tag{3.7}$$

$$\sum_{i\in V\setminus\{n+1\}} x_{in+1}^d \geq y^d - y^{d+1}, \qquad d = 1, ..., D-1 \tag{3.8}$$

$$\sum_{i\in V} x_{ih}^d + y^d \geq \sum_{i\in V} x_{hi}^{d+1} + y^{d+1}, \qquad \forall h \in H, \qquad d = 1, ..., D-1 \tag{3.9}$$

$$\sum_{i\in V} x_{ih}^d - \sum_{i\in V} x_{hi}^{d+1} \leq 1 - y^{d+1}, \qquad \forall h \in H, \qquad d = 1, ..., D-1 \tag{3.10}$$

$$x_{ij}^d \leq y^d, \qquad \forall (i,j) \in E, \qquad d = 1, ..., D \tag{3.11}$$

$$y^d \geq y^{d+1}, \quad d = 1, ..., D \tag{3.12}$$

$$\sum_{i \in \kappa} \sum_{j \in \kappa \setminus \{i\}} x_{ij}^d \leq |\kappa| - 1, \quad \kappa \subset C, \quad 2 \leq |\kappa| \leq |C| - 1, \quad d = 1, ..., D \tag{3.13}$$

$$x_{ij}^d \in \{0, 1\}, \quad \forall (i, j) \in E, \quad d = 1, ..., D \tag{3.14}$$

$$y^d \in \{0, 1\}, \quad d = 1, ..., D \tag{3.15}$$

A função objetivo 3.1 corresponde à soma do número de percursos, multiplicado por um peso M_1, com a distância total percorrida. Deste modo, com M_1 suficientemente grande, dá-se mais importância à minimização de percursos do que à minimização da distância total percorrida e, consequentemente, uma solução com poucos percursos terá sempre menor valor do que uma solução que envolva um maior número de percursos.

O conjunto das restrições 3.2 garantem que cada cliente é visitado uma única vez e as restrições 3.3 asseguram a conetividade em cada percurso contido na rota. Isto é, garante-se que o número de arcos que chegam a um dado cliente num dado percurso é igual ao número de arcos que saem desse mesmo cliente.

As restrições 3.4 e 3.5 garantem que cada percurso deve começar e terminar num dos hotéis disponíveis (não tendo necessariamente que começar e terminar no mesmo hotel).

As restrições 3.6 dizem respeito à duração máxima permitida para cada percurso, tendo-se para cada um que a soma do tempo de percorrer o percurso com os tempos necessários para servir os clientes nele visitados não deverá exceder L.

As restrições 3.7 e 3.8 garantem que o hotel n+1 (depósito) corresponde ao hotel inicial e final da rota. Em 3.7 garante-se que o hotel inicial do primeiro percurso é o hotel n+1 e em 3.8 que o hotel final do último percurso, que ocorre quando $y^d = 1$ e $y^{d+1} = 0$, é também o hotel n+1.

As restrições 3.9 e 3.10 garantem que o hotel inicial de um dado percurso, à exceção do primeiro percurso da rota, é igual ao hotel final do percurso anterior.

As restrições 3.11 garantem que se num dado dia d pelo menos um hotel ou cliente é visitado então y^d deverá tomar o valor 1, por outro lado se num dado dia d não se realizou nenhum percurso ($y^d = 0$) então nenhum hotel ou cliente é visitado nesse dia. As restrições 3.12 garantem que se num dado dia se realizou algum percurso ($y^{d+1} = 1$), isto é, se se visitou pelo menos um cliente ou hotel, então no dia anterior também tal aconteceu ($y^d = 1$) e, por outro lado, que se num dado dia não se visitou ninguém ($y^d = 0$) também tal acontecerá no dia seguinte ($y^{d+1} = 0$). Deste modo, em 3.12 garante-se que os percursos são realizados em dias consecutivos, iniciando-se no primeiro dia.

As restrições 3.13 correspondem às restrições de eliminação de subcircuitos. Note-se que, uma vez que o TSPHS admite a existência de soluções com ciclos que comecem e terminem num mesmo hotel, apenas é necessário evitar a existência de subcircuitos dentro de cada percurso.

Por fim, as restrições 3.14 e 3.15 dizem respeito ao domínio das variáveis, garantindo que x_{ij}^d e y^d só tomam valores 0 ou 1.

Denote-se que o TSP é um caso particular que se obtém considerando que não exis-

tem hotéis disponíveis, que L=∞ e que $\tau_i = 0, \forall i \in C$ na formulação acima apresentada para o TSPHS. Assim sendo, o TSPHS é pelo menos tão difícil quanto o TSP, ou seja, é tal como o TSP um problema NP-difícil.

3.3 Exemplo de aplicação

Por forma a compreender melhor o TSPHS considere-se o seguinte exemplo em que se consideram o conjunto de 5 clientes $C = \{1, 2, 3, 4, 5\}$ e o conjunto de 2 hotéis $H = \{6, 7\}$, sendo que o hotel 6 representa o depósito. Considere-se ainda que a duração máxima permitida para cada percurso é de $L = 150$ e que a matriz B com os tempos de viagem entre cada par de localizações e o vetor τ com os tempos de visita em cada cliente são dadas por:

$$B = \begin{bmatrix} 0 & 40 & 30 & 20 & 10 & 15 & 35 \\ 40 & 0 & 15 & 10 & 40 & 15 & 40 \\ 30 & 15 & 0 & 15 & 5 & 10 & 10 \\ 20 & 10 & 15 & 0 & 15 & 25 & 10 \\ 10 & 40 & 5 & 15 & 0 & 10 & 5 \\ 15 & 15 & 10 & 25 & 10 & 0 & 40 \\ 35 & 40 & 10 & 10 & 5 & 40 & 0 \end{bmatrix}$$

$$\tau = \begin{bmatrix} 15 \\ 10 \\ 25 \\ 15 \\ 10 \end{bmatrix}$$

Podemos observar que a rota 6-1-2-3-5-6 é obviamente uma solução não admissível uma vez que o cliente 4 não é visitado. Por outro lado, a rota 6-1-2-3-4-5-6 corresponde a uma solução não admissível pois, apesar de visitar todos os clientes, a duração da rota, a qual é composta por um único percurso, excede o limite L uma vez que $b_{61} + \tau_1 + b_{12} + \tau_2 + b_{23} + \tau_3 + b_{34} + \tau_4 + b_{45} + \tau_5 + b_{56} = 185 > 150$. Já a rota 6-3-2-4-1-5-7 apesar de satisfazer esse limite representa uma solução não admissível na medida em que a rota deve iniciar e terminar no depósito (hotel 6).

A rota 6-5-4-1-2-6-3-6 representa uma solução admissível tal como a rota 6-5-4-1-7-3-2-6, tendo ambas dois percursos (percursos 6-5-4-1-2-6 e 6-3-6 e percursos 6-5-4-1-7 e 7-3-2-6, respetivamente). Cada percurso começa e termina num dos hotéis disponíveis e a sua duração não ultrapassa o limite L. O hotel inicial e final de cada rota corresponde ao depósito. Também as rotas 6-3-2-4-1-5-7-6 e 6-7-3-2-4-1-5-6 são soluções admissíveis apesar de apresentarem dois hotéis seguidos, isto é, de existir um percurso em que não se visita nenhum cliente (percurso 7-6 e 6-7, respetivamente).

Capítulo 4

Metodologias de resolução

No presente capítulo serão apresentadas as meta-heurísticas propostas, nomeadamente dois algoritmos genéticos combinados com pesquisa local e métodos de perturbação. Inicialmente introduz-se o conceito geral de algoritmo genético e é feita uma breve descrição de cada operador e mecanismo usado no algoritmo sugerido. Posteriormente é descrito o mecanismo de pesquisa local, utilizado para melhorar as soluções obtidas pelo algoritmo genético, bem como o processo de perturbação de soluções utilizado para se tentar "escapar" de possíveis ótimos locais.

4.1 Algoritmos genéticos

Sendo o TSPHS uma extensão do TSP ele é também um problema NP-difícil. Deste modo, a existência de técnicas heurísticas com o objetivo de determinar uma solução admissível, preferencialmente com valor próximo do valor ótimo do problema, é fundamental. Nesta dissertação foram desenvolvidas duas meta-heurísticas para o problema em estudo. Uma meta-heurística é um processo iterativo destinado a encontrar uma boa solução para um dado problema, eventualmente a ótima, combinando de forma inteligente diferentes conceitos para explorar o espaço de pesquisa com o objetivo de evitar ótimos locais.

As meta-heurísticas desenvolvidas para o problema em estudo consistem em dois algoritmos genéticos combinados com pesquisa local e métodos de perturbação. Os algoritmos genéticos foram introduzidos pela primeira vez por John Holland na década de 70 e consistem em algoritmos de pesquisa baseados essencialmente na teoria de Darwin acerca da seleção natural e na genética.

O conjunto de características de um indivíduo, determinadas pelo seu material genético, determina a sua capacidade de sobrevivência. Na natureza a competição por recursos escassos em ambientes hostis ou a própria alteração do meio ambiente faz com que os indivíduos mais aptos, isto é, com características favoráveis, consigam adaptar-se mais facilmente às novas condições e, como tal, são os que têm mais hipótese de sobreviver e de se reproduzirem. Conforme o que é fundamentado na genética, os genes dos descendentes

são herdados dos progenitores, ocorrendo uma combinação dos genes de ambos os progenitores. Nessa combinação ocorre a transmissão dos genes mais favoráveis à geração seguinte, podendo ocorrer, embora raramente, mutação nos mesmos, isto é, alteração em uma ou mais posições dos cromossomos. Com o passar do tempo as características mais favoráveis são cada vez mais comuns e os indivíduos são cada vez mais aptos. Este processo é denominado por seleção natural.

Por analogia, num algoritmo genético, o meio ambiente corresponderá ao problema que se pretende resolver e ter-se-á um conjunto de indivíduos (conjunto de soluções) que compõem a população, cada um representado pelo seu cromossoma (codificação da solução). Estes indivíduos evoluem a cada geração, procurando-se preservar as características das melhores soluções sem comprometer a diversidade dentro da população. Os indivíduos são avaliados segundo uma função de aptidão. Os indivíduos mais aptos têm maior probabilidade de serem selecionados para constituírem os progenitores cujos cromossomas serão alvo de cruzamento de modo a originar os cromossomas dos seus descendentes, que poderão ou não ser alvo de mutação. Ao fim de cada geração, espera-se que os indivíduos mais aptos da população atual sobrevivam e como tal integrem a população da geração seguinte e que os menos aptos não sobrevivam e sejam "substituídos" pelos descendentes. Estas manipulações genéticas (cruzamentos e mutações) correspondem às perturbações do algoritmo e promovem a existência de diversidade no mesmo, a fim de evitar ótimos locais. Repete-se este processo por várias gerações até que o critério de paragem estipulado seja verificado.

Os algoritmos genéticos diferem de outros métodos de pesquisa em quatro aspetos principais, os quais fundamentam a robustez do algoritmo:

1. Utilizam uma codificação do problema, ou seja, não consideram o domínio das soluções mas sim o domínio das codificações e o algoritmo é, deste modo, independente do problema;
2. Pesquisam a partir de uma população de soluções e não apenas de uma única solução;
3. Pesquisam usando apenas a função aptidão;
4. Usam operadores estocásticos e não determinísticos, o que permite efetuar uma amostragem mais eficiente do espaço de pesquisa.

Tendo em conta a robustez característica dos algoritmos genéticos e o facto de o TSPHS ser NP-difícil, a utilização de um algoritmo genético para a resolução do problema em estudo pareceu adequada no sentido de nos poder conduzir à obtenção de soluções relativamente boas como ponto de partida. Os passos necessários para a construção de um algoritmo genético são os seguintes:

- Escolher a forma de representar as soluções (cromossomas);
- Gerar a população inicial;
- Avaliar os indivíduos da população (função aptidão);

- Escolher o mecanismo de seleção dos progenitores;
- Definir o operador de cruzamento dos progenitores a aplicar;
- Definir o operador mutação que pode ocorrer nos descendentes;
- Escolher o mecanismo de substituição, isto é, qual o método utilizado para substituir os descendentes por alguns indivíduos na população da geração seguinte.

De seguida serão apresentados os mecanismos e operadores bem como uma descrição da codificação e função aptidão utilizados para a resolução do problema em estudo.

Cromossoma

Cada indivíduo é codificado por um único cromossoma, ou seja, a cada solução corresponderá um dado cromossoma. Deste modo, o espaço das soluções é analisado indiretamente sendo o nosso espaço de pesquisa o das soluções codificadas.

Um cromossoma é representado por uma lista de elementos, designados por alelos, que pode ser uma lista binária, uma lista de números reais, uma permutação, entre outros. Tendo em conta que no problema em estudo todos os clientes têm de ser visitados e a ordem pela qual o são é relevante, considerou-se que cada cromossoma corresponde a uma permutação dos clientes, pelo que em cada alelo existirá um valor entre 1 e n. Aliás, a utilização de permutações para codificar as soluções é algo bastante usual em problemas do TSP.

Dado um cromossoma, para determinar a solução admissível do problema que lhe está associada recorre-se a um descodificador. Recorde-se que, no contexto do problema em estudo, uma solução diz-se admissível se corresponde a uma rota com início e fim no depósito (hotel n+1) composta por percursos com duração não superior ao limite L. Deste modo, para descodificar um cromossoma considera-se que a rota se inicia no depósito e segue-se deste para o cliente existente no primeiro alelo, isto é, no primeiro elemento da permutação. Posteriormente, segue-se desse cliente para o cliente que existe no segundo alelo e assim sucessivamente enquanto a duração máxima permitida por percurso, L, não seja atingida. Quando tal acontecer retira-se o último cliente introduzido no percurso, seja o cliente que se encontra no alelo i, e introduz-se na rota o hotel mais próximo entre o cliente do alelo i-1 e o cliente do alelo i, desde que tal nos conduza a um percurso com duração não superior a L. Caso tal não aconteça, tenta-se introduzir o hotel mais próximo do cliente do alelo i-1. Se mesmo assim ainda não se obtém um percurso com duração não superior a L, repete-se este raciocínio até que se consiga obter um percurso com duração não superior a L. A partir do último hotel inserido na rota segue-se para o cliente no alelo seguinte ao último inserido. O processo repete-se até todos os clientes integrarem a solução. Quando o cliente existente no último alelo for introduzido na rota segue-se deste para o depósito (hotel n+1).

Há que ressalvar as seguintes situações que podem ocorrer:

- Se a duração do percurso corrente coincidir com o limite imposto L, ao introduzir um cliente, averigua-se se a localização desse cliente coincide com a do hotel mais

próximo entre esse cliente e a do cliente seguinte. No caso de ser o último cliente do cromossoma verifica-se se a localização do cliente coincide com o depósito.

- Se a duração do último percurso exceder o limite L, tenta-se inserir um hotel adicional na rota entre o último cliente e o depósito, desde que com a sua inserção a duração do percurso não ultrapasse L. Se a inserção do hotel não conduzir a uma solução admissível, o último cliente é retirado e aplica-se o processo atrás descrito até se obter o percurso com duração não superior a L.

- No caso de todos os clientes no percurso corrente serem retirados, até ao último hotel inserido, tenta-se introduzir um hotel entre esse hotel e o cliente seguinte a visitar. Este processo pode ser repetido se a duração do percurso corrente ainda exceder L. Soluções com três ou mais hotéis consecutivos são consideradas de má qualidade e serão penalizadas, tal como é feito com as soluções não admissíveis.

Por forma a tornar o algoritmo mais eficiente, os hotéis a introduzir são escolhidos no sentido de tentar minimizar a duração total da rota. Começa-se por escolher o hotel que minimiza a distância entre o último elemento (cliente ou hotel) introduzido no percurso corrente e o próximo elemento a ser visitado. Posteriormente, no caso da inserção deste hotel nos conduzir a uma solução não admissível, tenta-se introduzir o hotel mais próximo desse último elemento do percurso corrente.

Note-se que com este descodificador a cada cromossoma está associada uma única solução admissível. No entanto, a mesma permutação pode codificar outras soluções, algumas das quais poderão ser melhores que a obtida com o descodificador.

Repare-se ainda o seguinte: poderíamos ter admitido no descodificador que um cliente que se encontre no alelo i apenas seria introduzido no percurso corrente no caso da sua duração somada com o tempo de ir do cliente do alelo i-1 para o cliente do alelo i e o tempo de servir este último adicionado com o tempo de ir do cliente do alelo i para o hotel mais próximo entre os clientes dos alelos i e i+1 fosse não superior a L. No entanto, tal poder-nos-ia conduzir à obtenção de uma solução com mais percursos do que os que são realmente necessários ao permitir terminar um percurso mais cedo do que era preciso.

Para se compreender melhor o funcionamento do descodificador utilizado será de seguida descrito um exemplo de aplicação do mesmo. Para tal considere-se os dados apresentados no subcapítulo 3.3 e o seguinte cromossoma:

5	4	1	3	2

Vamos construindo a solução passo a passo.

A rota deve iniciar no depósito como tal o primeiro elemento da solução é o hotel 6 (hotel n+1) deste modo tem-se a seguinte rota parcial: "6". De seguida partimos do depósito para o cliente que se encontra no primeiro alelo do cromossoma, cliente 5, deste modo ficamos com "$6-5$". Simultaneamente deve-se ir guardando a duração do percurso corrente por forma a podermos averiguar se o limite L para a duração

máxima de cada percurso é ou não ultrapassado. Ora, temos até ao momento duração $= b_{65} + \tau_5 = 10 + 10 = 20$ que é inferior a L=150 pelo que podemos tentar introduzir o cliente seguinte (alelo 2).

No alelo 2 temos o cliente 4. Introduzindo este cliente no percurso ficamos com "$6-5-4$" cuja duração é igual à duração anterior somada com o tempo de viagem entre o último elemento introduzido na rota (cliente 5) e o cliente 4 e ainda com o tempo de servir o cliente 4. Ou seja, o percurso corrente tem duração $= 20+b_{54}+\tau_4 = 20+15+15 = 50$ que é inferior a L=150.

De seguida introduzimos o cliente existente no alelo 3 (cliente 1) no percurso. Seguindo o mesmo raciocínio anterior, ficamos com "$6-5-4-1$" cuja duração do percurso corrente é $50 + b_{41} + \tau_1 = 50 + 20 + 15 = 85 <$ L=150. Segue-se então para o alelo seguinte. O cliente 3 é introduzido e fica-se com "$6-5-4-1-3$" e com duração do percurso corrente igual a $85 + b_{13} + \tau_3 = 85 + 30 + 25 = 140 <$ L=150.

O alelo seguinte possui o cliente 2 pelo que é este cliente que se vai tentar introduzir. Passamos a ter "$6-5-4-1-3-2$" e ficamos com duração do percurso corrente igual a $140 + b_{32} + \tau_2 = 140 + 15 + 10 = 165$. Esta duração excede o limite imposto L=150, pelo que o cliente 2 deve ser retirado do percurso. Ou seja, voltamos a considerar "$6-5-4-1-3$" com duração igual a 140.

Uma vez que a duração do percurso corrente com a introdução do cliente 2 excede a duração permitida ele só poderá ser visitado no percurso seguinte, isto é, no dia seguinte. Deste modo há que permanecer a noite num dos hotéis. O hotel escolhido para integrar o percurso é o hotel que minimiza a distância entre o último cliente visitado (cliente 3) e o próximo cliente a visitar (cliente 2), desde que tal não viole o limite imposto para a duração do percurso. Vejamos qual é o hotel mais próximo entre os clientes 2 e 3. Temos que $b_{36} + b_{62} = 10 + 15 = 25$ e $b_{37} + b_{72} = 10 + 40 = 50$, pelo que é o hotel 6 que se encontra mais próximo. Considerando a introdução do hotel 6 no percurso, fica-se com "$6-5-4-1-3-6$" que tem duração $= 140 + b_{36} = 140 + 10 = 150 =$ L.

O percurso seguinte inicia-se no hotel 6. De igual modo ao que foi feito antes vamos introduzindo os clientes. Tínhamos terminado no cliente 3 pelo que o cliente seguinte a ser introduzido é o cliente 2. Passamos a ter "$6-5-4-1-3-6-2$" e a duração do percurso corrente igual a $b_{62} = 15$. Sendo a duração inferior a L prosseguíamos introduzindo o cliente seguinte, porém como já todos os clientes foram visitados introduzimos o depósito na rota. Com a introdução do depósito (hotel n+1) ficamos com "$6-5-4-1-3-6-2-6$" e com duração do percurso corrente igual a $15 + b_{26} = 15 + 15 = 30$. Sendo a duração do percurso corrente não superior a L, "$6-5-4-1-3-6-2-6$" é a solução que se obtém com o descodificador a partir do cromossoma dado.

O descodificador utilizado pode ser observado no algoritmo 4.1, considerando-se para tal a seguinte notação:

- *dimcromos*: tamanho do cromossoma de um indivíduo
- *htrip*: indica que o hotel $n + htrip$ é o hotel inicial do percurso corrente
- *kopt*: posição, na permutação, do primeiro cliente a visitar quando se inicia um

percurso

- *time*: duração do percurso corrente
- $s_{i,j}$: indica o hotel mais próximo entre o cliente i e o cliente j
- $u_{i,j}$: indica o hotel mais próximo entre o hotel $n+i$ e o cliente j
- v_i: indica o hotel mais próximo do hotel $n+i$
- w_i: indica o hotel mais próximo do cliente i

População

Cada população é composta por um conjunto de indivíduos, ou seja, por um conjunto de cromossomas, que evoluem ao longo de várias gerações. Na natureza o tamanho da população pode variar de geração em geração, no entanto, para facilitar a implementação, no algoritmo genético considera-se que é constante. O número de indivíduos que vão compor a população bem como o número de gerações que vão existir são parâmetros do algoritmo genético e, como tal, deverão ser alvo de experimentação. À dimensão da população (número de indivíduos por geração) não deve ser atribuído nem um valor pequeno pois corre-se o risco de não ser suficiente para cobrir o espaço das soluções nem demasiado grande pois tal compromete o tempo computacional do algoritmo e consequentemente a sua eficiência.

Em ambas as meta-heurísticas propostas considerou-se que a população inicial é povoada por um conjunto de permutações geradas aleatoriamente.

Função aptidão

A aptidão de um cromossoma mede a qualidade da solução associada a esse cromossoma. Uma vez que se pretende encontrar boas soluções para o problema em estudo, deve existir uma ligação entre a função aptidão e a função objetivo.

Nos algoritmos genéticos propostos para o TSPHS, considerou-se o valor da aptidão de um cromossoma como sendo o valor da função objetivo da solução obtida aplicando o descodificador ao cromossoma. Deste modo, a aptidão de um dado indivíduo é dada pela soma da duração total com o número de percursos que a solução possui multiplicado por uma penalização M_1. A penalização deverá ser suficientemente elevada por forma a considerar mais aptos indivíduos que codifiquem soluções com um menor número de percursos. No caso de um cromossoma codificar uma solução não admissível ou uma solução admissível com 3 ou mais hotéis consecutivos atribui-se uma penalização M_2 ($M_1 << M_2$). A penalização M_2 atribuída deverá ser tal que qualquer outro indivíduo que codifique uma solução que não esteja nestas condições seja mais apto que estes indivíduos. Note-se que se optou por penalizar as soluções não admissíveis caso surjam, no entanto, estas soluções também poderiam ser reparadas ou poderíamos ter optado por gerar novas soluções até que se obtivessem soluções admissíveis, mas tal aumentaria o tempo computacional despendido.

Algoritmo 4.1 Descodificador

Require: *perm* (permutação de clientes)

1: $htrip = 1$
2: $kopt = 1$
3: S $\leftarrow \{n + htrip\}$
4: **while** $kopt \leq dimcromos$ **do**
5: time = 0
6: **for** $j = kopt$ **to** $dimcromos$ **do**
7: S $\leftarrow$ S $+\{perm_j\}$
8: **if** $j = kopt$ **then**
9: $time = time + b_{n+htrip,perm_j} + \tau_{perm_j}$
10: **else**
11: $time = time + b_{perm_{j-1},perm_j} + \tau_{perm_j}$
12: **end if**
13: **if** $time >$ L **then**
14: Aplicar o algoritmo Retira-clientes-da-rota
15: **else if** $time$=L e $((j \neq dimcromos$ e $b_{perm_j,s_{perm_j,perm_{j+1}}} \neq 0)$ ou $(j = dimcromos$ e $b_{perm_j,n+1} \neq 0))$ **then**
16: Aplicar o algoritmo Retira-clientes-da-rota
17: **else if** $time$=L e $((j \neq dimcromos$ e $b_{perm_j,s_{perm_j,perm_{j+1}}} = 0)$ ou $(j = dimcromos$ e $b_{perm_j,n+1} = 0))$ **then**
18: **if** $j = dimcromos$ **then**
19: S $\leftarrow$ S + $\{n + 1\}$
20: **else**
21: S $\leftarrow$ S + $\{s_{perm_j,perm_{j+1}}\}$
22: $htrip = s_{perm_j,perm_{j+1}} - n$
23: **end if**
24: $kopt = j + 1$
25: **else if** $time <$ L e $j = dimcromos$ **then**
26: **if** $time + b_{perm_j,n+1} \leq$ L **then**
27: S $\leftarrow$ S $+\{n + 1\}$
28: $kopt = j + 1$
29: **else**
30: **if** $time + b_{perm_j,u_{perm_j}} \leq$ L e $b_{u_{perm_j},n+1} \leq$ L **then**
31: S $\leftarrow$ S $+\{u_{perm_j}\} + \{n + 1\}$
32: $kopt = j + 1$
33: **else**
34: **if** $time + b_{perm_j,w_{perm_j}} \leq$ L e $b_{w_{perm_j},n+1} \leq$ L **then**
35: S $\leftarrow$ S $+\{w_{perm_j}\} + \{n + 1\}$
36: $kopt = j + 1$
37: **else**
38: Aplicar o algoritmo Retira-clientes-da-rota
39: **end if**
40: **end if**
41: **end if**
42: **end if**
43: **end for**
44: **end while**

Ensure: Rota S

Algoritmo 4.2 Retira-clientes-da-rota

1: $K = kopt$
2: $x \leftarrow j$
3: **while** $x \geq K$ **do**
4: **if** $x = K$ **then**
5: $time = time - b_{n+htrip,perm_x} - \tau_{permx}$
6: S $\leftarrow$ S $-\{perm_x\}$
7: **if** x é o último cliente de $perm$ **then**
8: $aux = n + 1$
9: **else**
10: $aux = perm_{x+1}$
11: **end if**
12: **if** $time + b_{n+htrip,u_{htrip,perm_x}} \leq$ L e $b_{u_{htrip,perm_x}} + \tau_{perm_x} + b_{perm_x,aux} + \tau_{aux} \leq$ L **then**
13: $kopt = x$
14: S $\leftarrow S + \{u_{htrip,perm_x}\}$
15: $htrip = u_{htrip,perm_x} - n$
16: $x = 0$
17: **else**
18: **if** $time + b_{n+htrip,v_{htrip}} \leq$ L e $b_{v_{htrip,perm_x}} + \tau_{perm_x} + b_{perm_x,aux} + \tau_{aux} \leq$ L **then**
19: $kopt = x$
20: S $\leftarrow S + \{v_{htrip}\}$
21: $htrip = v_{htrip} - n$
22: $x = 0$
23: **else**
24: $kopt = dimcromos + 1$
25: $x = 0$
26: **end if**
27: **end if**
28: **else**
29: $time = time - b_{perm_{x-1},perm_x} - \tau_{permx}$
30: S $\leftarrow$ S $-\{perm_x\}$
31: **if** $time + b_{perm_{x-1},s_{perm_{x-1},perm_x}} \leq$ L **then**
32: S $\leftarrow$ S $+\{s_{perm_{x-1},perm_x}\}$
33: $htrip = s_{perm_{x-1},perm_x} - n$
34: $kopt = x$
35: $x = 0$
36: **else**
37: **if** $time + b_{perm_{x-1},w_{perm_{x-1}}} \leq$ L **then**
38: S $\leftarrow$ S $+\{w_{perm_{x-1}}\}$
39: $htrip = w_{perm_{x-1}} - n$
40: $kopt = x$
41: $x = 0$
42: **else**
43: $x = x - 1$
44: **end if**
45: **end if**
46: **end if**
47: **end while**

Repare-se que, deste modo, uma solução não admissível ou com mais do que três hotéis consecutivos terá tendência a ser eliminada da população ao longo das gerações seguintes, na medida em que o indivíduo correspondente será o menos apto. Por outro lado, soluções com menor número de percursos terão tendência a permanecer na população ao longo das gerações em detrimento de soluções admissíveis com maior número de percursos, devido à penalização M_1 atribuída ao número de percursos.

Mecanismo de seleção

De geração em geração há que proceder à seleção dos indivíduos que irão constituir os progenitores e que serão alvo de cruzamento por forma a dar origem aos descendentes, de modo semelhante ao que ocorre na natureza. Existem diversos mecanismos de seleção que podem ser utilizados.

A seleção dos progenitores e consequente reprodução é de certa forma uma versão artificial da seleção natural. De forma a garantir que de uma geração para outra os melhores genes se tornam cada vez mais comuns há que garantir que os indivíduos mais aptos tenham mais hipótese de se reproduzirem e consequentemente de contribuir com mais descendentes que irão integrar a geração seguinte. Tal é assegurado no mecanismo de seleção e tem como fim manter as boas características existentes nas melhores soluções.

As duas meta-heurísticas estudadas para o problema diferem no mecanismo de seleção. Numa considera-se que o processo subjacente à seleção dos progenitores segue a lógica de uma roleta e na outra a de um torneio. A meta-heurística 1 (MH1) utiliza a roleta como mecanismo de seleção no algoritmo genético (AG1) e a meta-heurística 2 (MH2) utiliza o torneio como mecanismo de seleção no algoritmo genético (AG2).

Na roleta cada indivíduo tem associada uma probabilidade de ser selecionado para progenitor, a qual está intimamente relacionada com a sua aptidão. Usualmente essa probabilidade corresponde à fração entre a aptidão do indivíduo e a aptidão total da população (soma das aptidões de todos os indivíduos existentes na população), porém deste modo uma solução com baixo valor terá associada uma probabilidade mais pequena e logo estaremos a dar menos hipóteses de selecionar essa solução. No entanto, e uma vez que estamos num problema de minimização, será às soluções com baixo valor que deverá ser atribuída uma maior probabilidade de seleção pois correspondem a soluções melhores. Como tal, a probabilidade de seleção de um indivíduo será obtida dividindo a diferença entre a aptidão total da população e a aptidão do indivíduo pela aptidão total da população. Deste modo, os indivíduos mais aptos têm mais hipóteses de serem selecionados, o que permite mais facilmente que os melhores genes se possam manter na população. A roleta é portanto enviesada no sentido dos indivíduos mais aptos, os quais correspondem a soluções com menor valor, possuírem maior probabilidade de serem selecionados. Note-se que a seleção dos progenitores com a roleta admite que existe reposição, ou seja, um mesmo indivíduo pode ser selecionado para progenitor várias vezes.

Por forma a reproduzir uma roleta enviesada no AG1 procedeu-se da seguinte forma: para cada indivíduo considera-se a respetiva aptidão complementar dada pela diferença

entre a aptidão total e a aptidão desse indivíduo; começando no primeiro indivíduo e até ao último determina-se a aptidão complementar acumulada; gera-se um número aleatório entre 1 e a soma total obtida; se esse número se encontra abaixo da aptidão complementar acumulada do primeiro indivíduo então é o primeiro indivíduo que é selecionado para progenitor, se esse número se encontra entre os valores da aptidão complementar acumulada dos indivíduos i-1 e i então é o indivíduo i que é selecionado para progenitor, este método é repetido até se obter o número de progenitores pretendido. O primeiro progenitor obtido é cruzado com o segundo, o terceiro com o quarto, e assim sucessivamente. Na figura 4.1 pode-se observar um exemplo de como o mecanismo de seleção utilizado no AG1 funciona, no qual se consideram 8 indivíduos com aptidão A_i $(i = 1, ..., 8)$ e se pretendem obter 4 progenitores P_i que darão origem a 4 descendentes D_j $(j = 1, ..., 4)$.

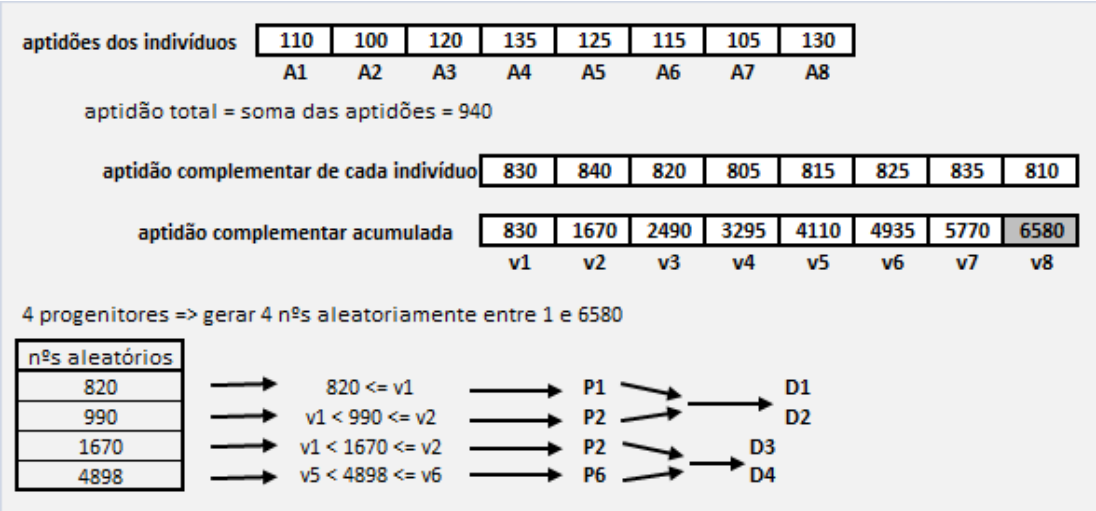

Figura 4.1: Mecanismo de seleção do tipo roleta usado no AG1

Na seleção tipo torneio uma sequência dos indivíduos é obtida permutando aleatoriamente os seus índices. Sucessivos grupos de T indivíduos são comparados entre si, sendo escolhido para progenitor o melhor deles, isto é, o de menor aptidão. Se a lista for toda analisada, outra permutação aleatória dos indivíduos da população é gerada. Este procedimento é repetido até que tenham sido obtidos tantos progenitores como o número de indivíduos existentes na população. Cada um dos progenitores vai ser emparelhado com um progenitor escolhido aleatoriamente. Facilmente se percebe que, deste modo, o melhor indivíduo da população é sempre escolhido e o pior indivíduo nunca o é.

Por forma a reproduzir um torneio no AG2 procedeu-se da seguinte forma: gerou-se aleatoriamente uma permutação dos índices dos indivíduos existentes na população (lista); procedeu-se a sucessivos torneios de T indivíduos, isto é, comparou-se os indivíduos cujo índice correspondem aos primeiros T elementos da lista, tendo sido escolhido para progenitor o melhor deles (que neste caso corresponde ao indivíduo com menor valor da função aptidão), posteriormente comparou-se os indivíduos cujo índice

correspondem aos T elementos seguintes da lista, e assim sucessivamente até perfazer o número de progenitores pretendido; o primeiro progenitor que se obteve é cruzado com o segundo, o terceiro com o quarto, e assim sucessivamente. Na figura 4.2 pode-se observar um exemplo de como o mecanismo de seleção utilizado no AG2 funciona, no qual se consideram 8 indivíduos com aptidão A_i $(i = 1, ..., 8)$ e T=2 com o objetivo de se obter 4 progenitores P_i que darão origem a 4 descendentes D_j $(j = 1, ..., 4)$.

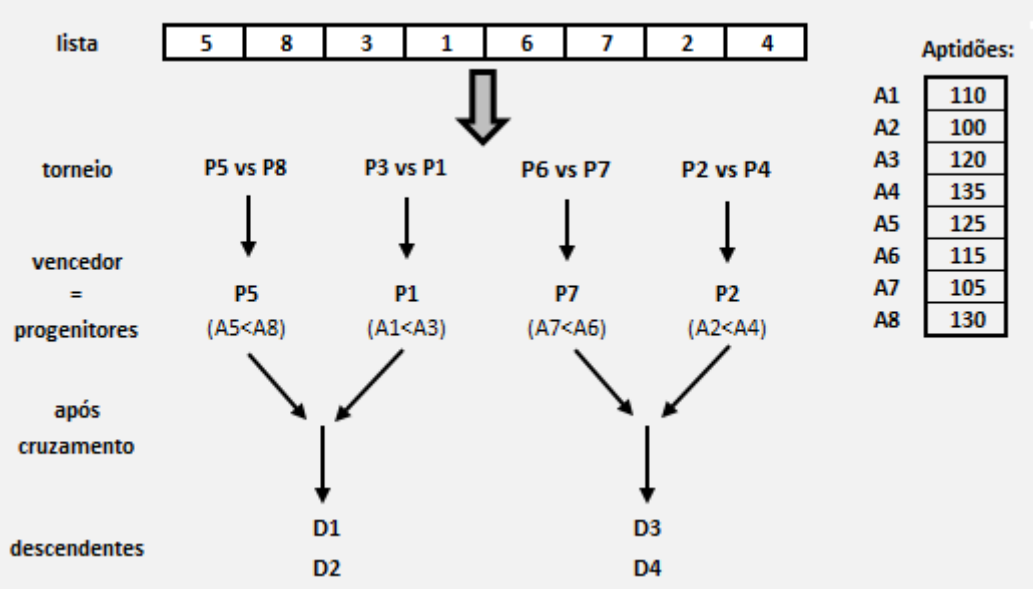

Figura 4.2: Mecanismo de seleção do tipo torneio usado no AG2

O número de descendentes foi considerado como sendo igual ao número de progenitores. O número de progenitores é um parâmetro tanto do AG1 como do AG2 e como tal necessita de ser alvo de alguma experimentação. De notar que o número de progenitores que vai produzir descendentes não deverá ser demasiado pequeno, podendo tal conduzir à perda de diversidade e possível convergência prematura. Por outro lado, a dimensão dos grupos de indivíduos que são comparados no torneio, T, é também um parâmetro do AG2 e foi igualmente alvo de experimentação. Este parâmetro está intimamente relacionado com o número de progenitores, sendo no AG2 o número de progenitores dado pela fração entre o número de indivíduos da população e T. Considera-se que o número de progenitores é um número inteiro par, deste modo, T deve ser escolhido de maneira a que a fração produza um valor inteiro e sempre que esse valor seja ímpar considera-se o número de progenitores igual ao número par imediatamente abaixo desse valor.

Defina-se pressão na seleção como sendo o "grau" com que se favorece os indivíduos mais aptos. A pressão na seleção encontra-se intimamente relacionada com a diversidade na população, sendo esta última essencial para a pesquisa feita num algoritmo genético. De facto, a pressão na seleção é inversamente proporcional à diversidade existente na

população: em geral aumentar a pressão conduz-nos a uma diminuição da diversidade e diminuir a pressão conduz-nos a um aumento da diversidade. Com a pressão de seleção baixa o algoritmo pode demorar mais tempo. Uma pressão de seleção alta poderá conduzir a uma diminuição tal da diversidade que leve à convergência para uma solução de fraca qualidade. Deste modo é fundamental ajustar a pressão na seleção de modo a que se encontre um equilíbrio. Uma vantagem do torneio relativamente à roleta é conseguir-se facilmente ajustar a pressão na seleção fazendo variar o valor de T enquanto na roleta para o fazer tem de se recorrer a possíveis calibrações na função aptidão. Quanto maior/menor o valor atribuído a T, maior/menor pressão à seleção é atribuída, respetivamente.

Operador cruzamento

Após a escolha dos cromossomas que serão progenitores há que proceder ao cruzamento dos mesmos. Na natureza, nem sempre o cruzamento de dois progenitores conduz à origem de descendentes. Por forma a traduzir esta situação, num algoritmo genético considera-se que cada cruzamento entre dois progenitores produz descendentes segundo uma dada probabilidade, a qual é denominada por probabilidade de cruzamento. Deste modo, para cada cruzamento gera-se um número aleatório entre zero e um, se esse número é não superior ao valor atribuído à probabilidade de cruzamento então o operador cruzamento é aplicado aos dois progenitores que se pretendem cruzar, caso contrário não ocorre cruzamento entre esses progenitores sendo os descendentes cópias dos progenitores.

O operador cruzamento tem como principal objetivo a origem de descendentes com material genético herdado de ambos os progenitores. A probabilidade de cruzamento consiste num dos parâmetros de um algoritmo genético e deverá ser alvo de experimentação. A probabilidade de cruzamento deverá ser grande, pois quanto maior for essa probabilidade mais facilmente se podem explorar outras soluções e permitir que o algoritmo evolua no sentido de encontrar uma boa solução.

Muitos são os operadores de cruzamento existentes. Os operadores mais comuns em problemas cuja codificação é uma permutação são os cruzamentos CX, OX e PMX, podendo também ser utilizado o cruzamento uniforme baseado na ordenação. Cada um destes operadores considera que cada cruzamento de dois progenitores, P_1 e P_2, originará dois descendentes, D_1 e D_2.

Estudos feitos no âmbito desta dissertação sugerem que o cruzamento OX é o mais adequado tanto para o AG1 como para o AG2. Segundo este operador, começa-se por gerar aleatoriamente dois pontos de cruzamento. Para tal, nos algoritmos genéticos propostos, geraram-se aleatoriamente dois números distintos entre 2 e n, sendo n o número total de clientes. Posteriormente os elementos existentes na secção, definida por esses dois pontos, do cromossoma do progenitor P_1 são transmitidos ao descendente D_2 para integrarem os mesmos alelos onde estavam em P_1 e, de igual modo, os elementos existentes nessa secção do cromossoma do progenitor P_2 são transmitidos ao descendente D_1. Os restantes alelos do D_1 serão provenientes do progenitor P_1, na verdade eles serão integrados no descendente D_1 a partir do alelo que sucede o último já preenchido

por elementos do progenitor P_2 pela ordem que aparecem no progenitor P_1. Quando o último alelo do cromossoma do descendente D_1 é preenchido e ainda existem alelos do D_1 por preencher, continuamos a integrar os elementos de P_1 mas agora a partir do início do cromossoma de D_1. De notar que, uma vez que os cromossomas dos descendentes deverão igualmente definir permutações, sempre que se pretenda inserir um elemento de P_1 que já existe em D_1, esse elemento é ignorado e tenta-se inserir o elemento do alelo seguinte de P_1. Por um processo análogo os restantes alelos do descendente D_2 serão herdados do progenitor P_2.

Para uma melhor compreensão de como o operador cruzamento OX funciona pode-se observar o exemplo apresentado na figura 4.3, no qual a secção, que se encontra limitada por duas barras, ||, é definida pelos pontos de cruzamento 2 e 4.

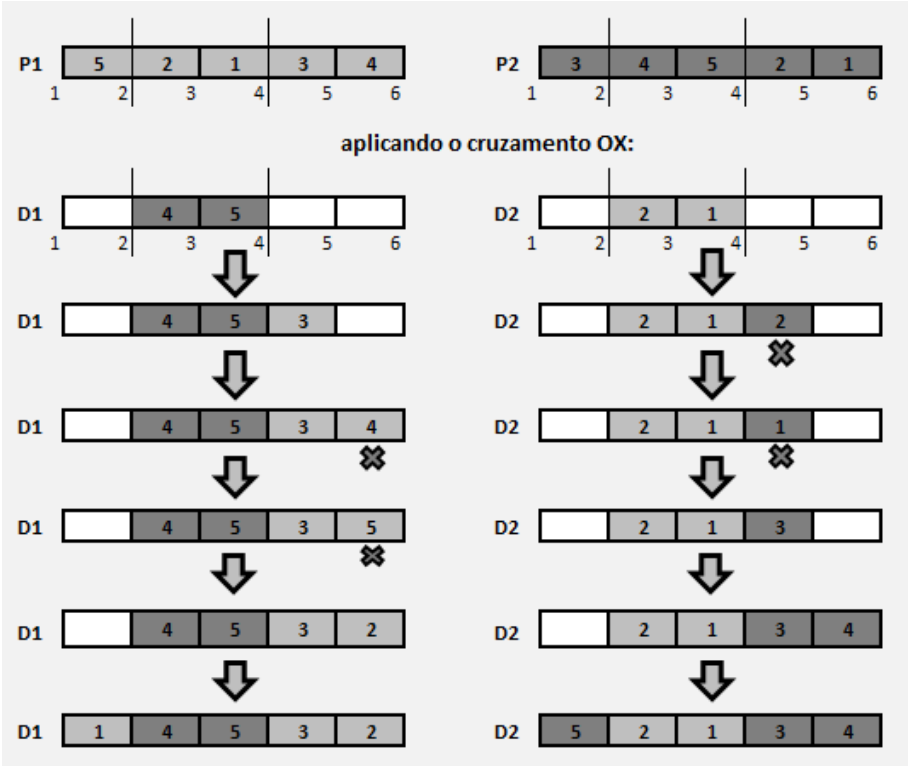

Figura 4.3: Operador cruzamento OX

Operador mutação

É sabido que na natureza nem todos os descendentes sofrem mutação, aliás tal é algo que ocorre raramente. Por forma a traduzir esta situação, num algoritmo genético considera-se que cada descendente pode sofrer mutação segundo uma dada probabilidade, a qual será denominada por probabilidade de mutação. Deste modo, para cada descendente gera-se um número aleatório entre zero e um; se esse número é não superior

ao valor atribuído à probabilidade de mutação então o operador mutação é aplicado ao descendente, caso contrário não se procede a qualquer alteração no cromossoma do descendente.

O operador mutação introduz alterações nos cromossomas, as quais promovem a existência de diversidade no algoritmo, importante para evitar ótimos locais. Para o TSPHS em particular, as soluções dos algoritmos evolutivos tendem a ter falta de diversidade [3]. Deste modo, a probabilidade de mutação não deverá ser muito baixa, mas por outro lado também não poderá ser muito elevada pois pode comprometer a convergência do algoritmo. A probabilidade de mutação consiste num dos parâmetros de um algoritmo genético e deverá ser alvo de experimentação.

Utilizou-se o operador mutação baseado na ordenação. Segundo este, dada uma secção de um cromossoma, a mutação consiste em permutar os elementos (clientes) existentes nessa secção. Uma secção deve integrar pelo menos dois elementos para que possa ocorrer permutação dos mesmos. Deste modo, para definir qual a secção do cromossoma em que vai ocorrer a mutação, geraram-se aleatoriamente dois números entre 1 e $n+1$ de modo que distassem pelo menos duas unidades entre si.

Para uma melhor compreensão de como o operador mutação utilizado funciona observe-se a figura 4.4, na qual são apresentadas três das possíveis mutações que podem ocorrer no cromossoma do descendente D_1 dada a secção, que se encontra limitada por duas barras, ||, definida pelos pontos de mutação 2 e 5.

Figura 4.4: Operador mutação baseado na ordenação

Mecanismo de substituição

Os descendentes obtidos após aplicação do operador cruzamento e possível mutação deverão integrar a população da geração seguinte, deste modo os descendentes irão substituir alguns dos indivíduos existentes na população corrente. A substituição pode ser feita, por exemplo, substituindo os indivíduos menos aptos, escolhidos de forma aleatória ou não, pelos descendentes. Repare-se que deste modo o melhor indivíduo mantém-se

sempre na população (elitismo).

O mecanismo de substituição utilizado foi a substituição incremental. Segundo esta, os indivíduos menos aptos são substituídos pelos descendentes. Deste modo, para se proceder ao mecanismo de substituição nos algoritmos genéticos construídos, procedeu-se à ordenação dos índices dos indivíduos por ordem crescente da sua aptidão numa lista e os indivíduos da população cujo índice está nas últimas x posições dessa lista são substituídos pelos x descendentes originados.

Considerações adicionais

De referir um outro aspeto a ter em atenção num algoritmo genético: as réplicas. As réplicas correspondem à repetição de soluções na população. A existência de muitas réplicas pode conduzir o algoritmo à convergência precoce para soluções de fraca qualidade. Se as réplicas forem muito frequentes podem ser eliminadas da população. De notar, no entanto, que nos algoritmos genéticos construídos tal não foi feito devido ao esforço computacional que isso exigiria.

No algoritmo 4.3 encontra-se apresentado o pseudo-código da estrutura dos algoritmos genéticos propostos para o TSPHS, no qual é considerada a seguinte notação:

- *dimcromos*: tamanho do cromossoma de um indivíduo
- *dimpop*: número de indivíduos existentes na população
- *ger*: indica a geração atual
- *nger*: número máximo de gerações
- *nprog*: número de progenitores
- *ndesc*: número de descendentes
- *pcruz*: probabilidade de cruzamento
- *pmut*: probabilidade de mutação
- A_i: valor da função aptidão do indivíduo i
- P_i: i-ésimo progenitor
- D_i: i-ésimo descendente gerado
- *pop*: população

4.2 Pesquisa local

Os procedimentos clássicos de pesquisa local baseiam-se na exploração iterativa de soluções vizinhas de uma solução. Começam com uma solução admissível (obtida, por exemplo, através de uma heurística construtiva ou de um algoritmo genético), procedendo-se posteriormente a uma pesquisa num conjunto de soluções vizinhas da solução corrente. Os mecanismos de geração das soluções vizinhas baseiam-se em trocas

Algoritmo 4.3 Estrutura básica dos algoritmos genéticos propostos para o TSPHS

Require: n, *dimcromos*, M_1, M_2, T, *nger*, *dimpop*, *pmut*, *pcruz*, τ, B, *nprog*, *ndesc*

1: Inicialização da população: Gerar população *pop* com *dimpop* indivíduos cujo cromossoma é uma permutação de 1 a n
2: Avaliação da população: calcular o valor da função aptidão, A_i, de cada indivíduo i existente em *pop*
3: $ger \leftarrow 1$
4: **while** $ger \leq nger$ **do**
5: Seleção dos *nprog* progenitores: aplicar mecanismo de seleção do tipo roleta no AG1 e aplicar mecanismo de seleção do tipo torneio no AG2
6: $i \leftarrow 1$
7: **while** $i < ndesc$ **do**
8: Gerar aleatoriamente $a \in [0, 1]$
9: **if** $a \leq pcruz$ **then**
10: Aplicação do cruzamento OX a P_i e P_{i+1}, gerando os descendentes D_i e D_{i+1}
11: **else**
12: $D_i \leftarrow P_i$
13: $D_{i+1} \leftarrow P_{i+1}$
14: **end if**
15: $i \leftarrow i + 2$
16: **end while**
17: **for all** descendente D_i **do**
18: Gerar aleatoriamente $a \in [0, 1]$
19: **if** $a \leq pmut$ **then**
20: Aplicar mutação a D_i
21: **end if**
22: **end for**
23: Obtenção da população da geração seguinte: substituir em bloco os *ndesc* indivíduos menos aptos de *pop* pelos *ndesc* descendentes gerados
24: Avaliação dos novos indivíduos da população: calcular o valor da função aptidão de cada descendente D_i
25: $ger \leftarrow ger + 1$
26: **end while**

Ensure: Indivíduo i mais apto da população final (menor A_i)

e/ou outras alterações com o objetivo de tentar encontrar uma solução melhor. Usualmente opta-se por uma das seguintes estratégias de aceitação de uma solução: aceitar a primeira solução vizinha que seja melhor que a solução corrente ou examinar todas as soluções vizinhas e aceitar a melhor delas caso seja melhor que a solução corrente. Repare-se ainda que a solução obtida depende muito da solução de partida e do mecanismo de geração de vizinhanças usado.

Com vista a melhorar as soluções obtidas após a aplicação do algoritmo genético desenvolveu-se então um procedimento de pesquisa local baseado em diferentes operadores de destruição e posterior reconstrução da rota. Para tal utilizaram-se os operadores *2-opt*, *Swap-within*, *Move1-within*, *Move2-within*, *Swap-between*, *Move-between*, *Inversion* e *Destroy-Insert-Hotels*. A aplicação de cada um destes operadores induz uma vizinhança na qual é feita a pesquisa de soluções. Estes operadores atuam na otimização das soluções não só a nível dos percursos como a nível da rota, tendo-se como principal objetivo a minimização da distância total percorrida e do número de percursos das soluções. De referir que se optou por examinar todas as soluções vizinhas e aceitar a melhor desde que melhore a solução corrente. Repete-se este processo até não existirem soluções vizinhas melhores que a solução corrente.

4.2.1 Operadores que otimizam a solução ao nível do percurso

Os operadores *2-opt*, *Swap-within*, *Move1-within* e *Move2-within* otimizam a solução a nível dos percursos, isto é, as trocas são realizadas entre clientes de um mesmo percurso. De seguida serão descritos cada um destes operadores e apresentados alguns exemplos de aplicação para melhor compreensão dos mesmos.

2-Opt

Dado um par de clientes a e b num mesmo percurso da rota, o operador *2-Opt* desconecta o percurso entre a e b e reconstrói-o posteriormente de modo que a sequência de clientes entre os clientes fixos a e b fique invertida. Deste modo, esta troca só poderá ter algum efeito em percursos em que se visitem pelo menos quatro clientes, de forma a garantir a existência de pelo menos dois clientes entre o par a e b considerado.

Repare-se que a aplicação do operador *2-Opt* conduz à obtenção de soluções com igual número de percursos. Note-se ainda que, nos percursos onde não ocorreu alteração, a duração continuará a ser não superior a L. Já no percurso onde existiu a troca continuam a visitar-se os mesmos clientes pelo que a soma dos tempos de serviço nesse percurso não se altera. A alteração da sequência dos clientes a visitar pode levar a uma alteração do tempo de viagem. Se se diminui o tempo de viagem então a duração do percurso onde ocorreu a troca continuará com duração não superior a L, caso contrário tal pode ou não acontecer. Tendo em conta que apenas serão aceites, para atualizar a solução corrente, soluções com duração não superior à da solução corrente, a solução obtida após a aplicação do operador *2-Opt* será necessariamente admissível.

Na figura 4.5 pode-se observar exemplos de aplicação do operador *2-Opt*: uma solução

para a qual, à partida, é sabido que *2-Opt* não terá qualquer efeito e duas outras soluções que exemplificam possíveis trocas que podem ser obtidas com *2-Opt* desde que tal produza uma melhor solução como foi anteriormente referido.

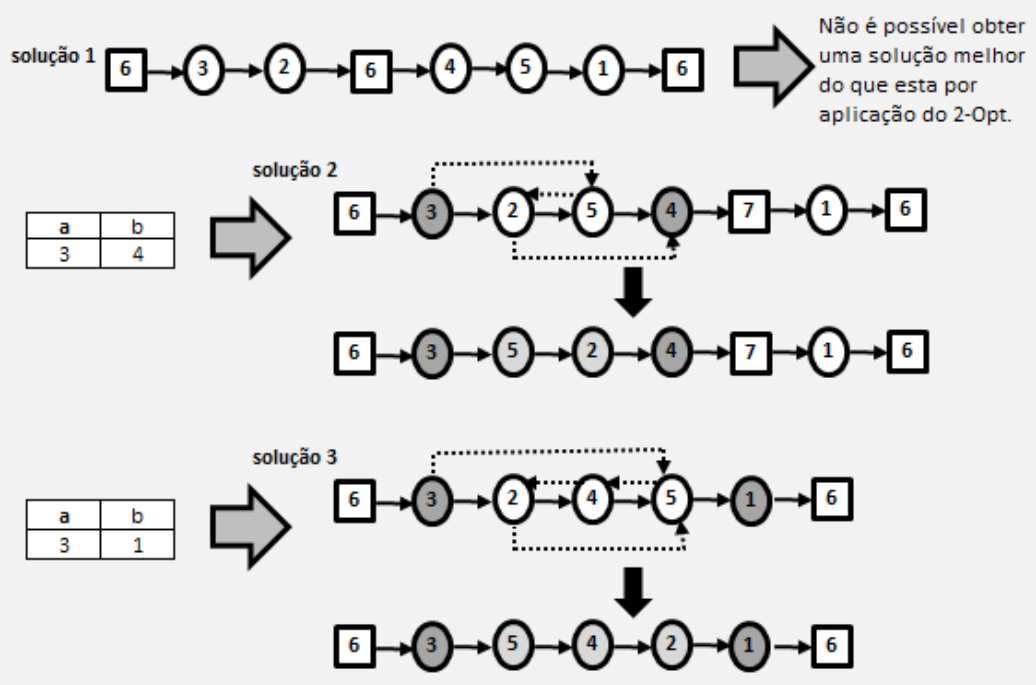

Figura 4.5: Exemplo de aplicação do operador *2-Opt*

No algoritmo 4.4 apresenta-se o pseudo-código para pesquisar soluções na vizinhança induzida pelo operador *2-Opt*. É considerada a seguinte notação:

- N: número de elementos que compõem a solução S
- S_i: elemento na posição i da rota (solução S)
- $V(x)$: valor da solução x
- *besti*: cliente a que permite obter a melhor solução na vizinhança induzida pelo operador *2-Opt*
- *bestj*: cliente b que permite obter a melhor solução na vizinhança induzida pelo operador *2-Opt*

Swap-within

O operador *Swap-within* consiste, em cada percurso, na troca da posição de dois clientes consecutivos, isto é, na troca da ordem da visita desses dois clientes. Para melhor compreensão de como o *Swap-within* funciona considere-se a figura 4.6, na qual se apresentam as possíveis trocas a efetuar com *Swap-within* a partir da solução "6-3-2-7-4-5-1-6" e

Algoritmo 4.4 Pseudo-código para pesquisar na vizinhança induzida pelo *2-Opt*

Require: Solução admissível S'

$S \leftarrow S'$
repeat
  newvalue $\leftarrow V(S)$
  melhorou = 0
  for $i = 2$ **to** N-4 **do**
    if S_i, S_{i+1} e S_{i+2} são clientes **then**
      for $j = i + 3$ **to** N **do**
        if S_j é hotel **then**
          Sair do ciclo for
        else
          $s \leftarrow$ solução obtida procedendo à inversão dos clientes entre o cliente S_i e o cliente S_j
          if $V(s) \leq$ newvalue **then**
            newvalue $\leftarrow V(s)$
            besti $\leftarrow S_i$
            bestj $\leftarrow S_j$
            melhorou = 1
          end if
        end if
      end for
    end if
  end for
  if melhorou=1 **then**
    S $\leftarrow$ solução obtida procedendo à inversão dos clientes entre os clientes *besti* e *bestj*
    $V(S) \leftarrow$ newvalue
  end if
until Solução com valor inferior ou com valor igual a $V(S)$ e que ainda não tenha sido pesquisada seja encontrada

Ensure: Solução admissível com valor não superior ao valor de S'

quais as soluções resultantes. A solução de menor duração substituirá a solução corrente no caso de ter duração não superior.

A aplicação do operador *Swap-within* permite a obtenção de uma solução com o mesmo número de percursos da solução de partida porém com uma duração inferior. De igual modo ao que ocorria com *2-Opt* a solução obtida após esta troca será necessariamente admissível.

Para aplicar *2-Opt* num percurso com quatro clientes necessariamente o cliente a corresponderá ao primeiro cliente visitado nesse percurso e o cliente b ao último e proceder-se-á à inversão dos dois clientes entre a e b. Note-se que, neste caso, esta troca tem o mesmo efeito que aplicar *Swap-within* uma vez que duma solução para a outra o que ocorre é a troca dos clientes consecutivos que se encontram entre a e b. Existe portanto trocas que se podem realizar tanto com *Swap-within* como com *2-Opt*.

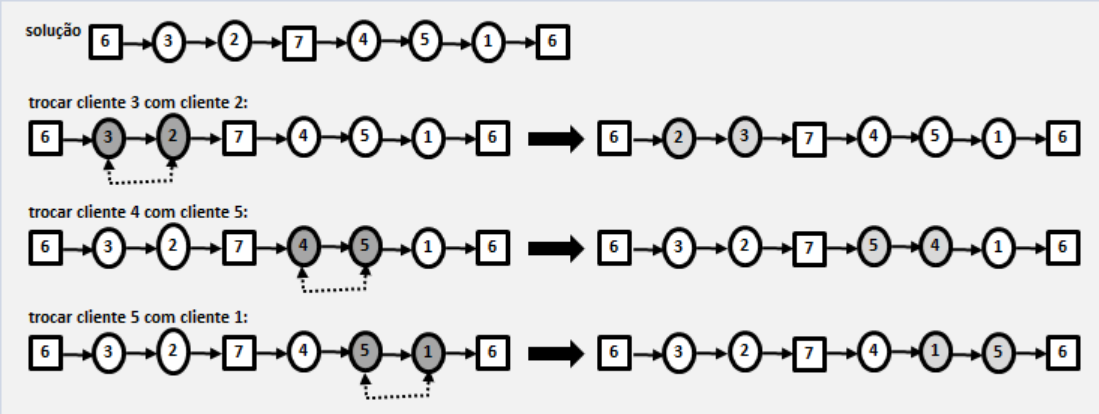

Figura 4.6: Exemplo de aplicação do operador *Swap-within*

No algoritmo 4.5 apresenta-se o pseudo-código para pesquisar soluções na vizinhança induzida pelo operador *Swap-within*. É considerada a seguinte notação:

- N: número de elementos que compõem a solução S
- S_i: elemento na posição i da rota (solução S)
- $V(x)$: valor da solução x
- *besti*: posição do cliente cuja troca com o cliente da posição *bestj* permite obter a melhor solução na vizinhança induzida pelo operador *Swap-within*
- *bestj*: posição do cliente cuja troca com o cliente da posição anterior (*besti*) permite obter a melhor solução na vizinhança induzida pelo operador *Swap-within*

Move1-within

O operador *Move1-within* consiste em, por percurso, retirar um cliente e inseri-lo na melhor posição dentro do mesmo. A melhor posição corresponde à posição do percurso

Algoritmo 4.5 Pseudo-código para pesquisar na vizinhança induzida pelo *Swap-within*

Require: Solução admissível S'

$S \leftarrow S'$
repeat
  newvalue $\leftarrow V(S)$
  melhorou = 0
  for $i = 2$ to N-1 **do**
    if S_i e S_{i+1} são clientes **then**
      $s \leftarrow$ solução obtida trocando a ordem de visita dos clientes nas posições i e i+1 da rota
      if $V(s) \leq$ newvalue **then**
        newvalue $\leftarrow V(s)$
        besti $\leftarrow i$
        bestj $\leftarrow i + 1$
        melhorou = 1
      end if
    end if
  end for
  if melhorou=1 **then**
    S $\leftarrow$ solução obtida trocando o cliente na posição *besti* e o cliente na posição *bestj*
    $V(S) \leftarrow$ newvalue
  end if
until Solução com valor inferior ou com valor igual a $V(S)$ e que ainda não tenha sido pesquisada seja encontrada

Ensure: Solução admissível com valor não superior ao valor de S'

tal que a inserção do cliente nesse local produz a maior redução do valor da solução. Para melhor compreensão de como o operador *Move1-within* funciona veja-se a figura 4.7, na qual são apresentadas todas as trocas analisadas (representadas a tracejado) aquando da aplicação de *Move1-within* à solução "6-3-2-7-4-5-1-6".

A aplicação do operador *Move1-within* permite a obtenção de uma solução com o mesmo número de percursos da solução de partida porém com um valor não superior. De igual modo ao que ocorria com os operadores descritos anteriormente também aqui a solução obtida após esta troca será necessariamente admissível.

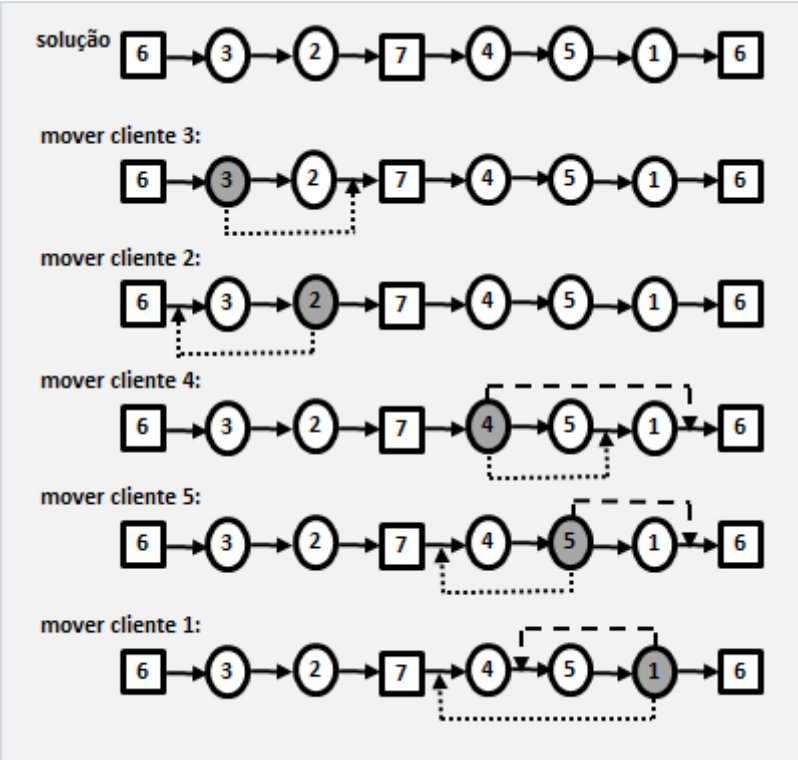

Figura 4.7: Exemplo de aplicação do operador *Move1-within*

No algoritmo 4.6 apresenta-se o pseudo-código para pesquisar soluções na vizinhança induzida pelo operador *Move1-within*. É considerada a seguinte notação:

- N: número de elementos que compõem a solução S
- S_i: elemento na posição i da rota (solução S)
- $V(x)$: valor da solução x
- *besti*: posição do cliente cuja sua remoção permite obter a melhor solução na vizinhança induzida pelo operador *Move1-within*
- *bestj*: melhor posição do cliente ou hotel onde o cliente removido que se encontrava na posição *besti* será inserido, ou seja, o cliente que estava na posição *besti* passa a

ser visitado entre o cliente ou hotel da posição *bestj* e o cliente ou hotel da posição *bestj*+1 na rota

Algoritmo 4.6 Pseudo-código para pesquisar na vizinhança induzida pelo *Move1-within*

Require: Solução admissível S'
1: $S \leftarrow S'$
2: **repeat**
3: newvalue $\leftarrow V(S)$
4: melhorou = 0
5: **for** $i = 2$ **to** N−1 **do**
6: **if** S_i é cliente **then**
7: *inicio* ← posição do hotel que inicia o percurso onde S_i é visitado
8: **for** *inicio* **to** N-1 **do**
9: **if** (S_j é cliente ou (S_j é hotel e $j = inicio$)) e $j \neq i$ e $j \neq i - 1$ **then**
10: $s \leftarrow$ solução obtida inserindo o cliente i entre S_j e S_{j+1}
11: **if** $V(s) \leq$ newvalue **then**
12: newvalue $\leftarrow V(s)$
13: $besti \leftarrow i$
14: $bestj \leftarrow j$
15: melhorou = 1
16: **end if**
17: **else if** S_j é hotel e $j \neq inicio$ **then**
18: Sair do ciclo for
19: **end if**
20: **end for**
21: **end if**
22: **end for**
23: **if** melhorou=1 **then**
24: S ← solução obtida inserindo o cliente que se encontrava na posição *besti* entre o cliente ou hotel na posição *bestj* e o cliente ou hotel na posição *bestj*+1
25: $V(S) \leftarrow$ newvalue
26: **end if**
27: **until** Solução com valor inferior ou com valor igual a $V(S)$ e que ainda não tenha sido pesquisada seja encontrada
Ensure: Solução admissível com valor não superior ao valor de S'

Move2-within

O operador *Move2-within* é análogo ao operador *Move1-within*. Em cada percurso retiram-se dois clientes (consecutivos ou não) e procede-se à inserção de ambos na melhor posição dentro do mesmo.

De notar que enquanto em *Move1-within* só tínhamos um cliente para inserir aqui

temos dois e a ordem pela qual eles são inseridos tem influência no valor da solução. Deste modo, a inserção dos dois clientes (sejam eles a e b) é feita não só tendo em conta qual a melhor posição como também qual a ordem que origina um menor valor (inserir nessa posição a-b ou b-a).

A aplicação do operador *Move2-within* permite a obtenção de uma solução com valor não superior mas com o mesmo número de percursos da solução de partida. Por justificações análogas às apresentadas no operador *2-Opt*, a solução obtida após o *Move2-within* será necessariamente admissível.

Na figura 4.8 são indicadas as trocas analisadas aquando da aplicação de *Move2-within* à solução "6-3-2-7-4-5-1-6", sendo que para cada um dos movimentos apresentados na figura é analisado qual a melhor ordem de os colocar na nova posição, por exemplo, no movimento dos clientes 3 e 2 para à frente do cliente 4 estuda-se as soluções "6-4-**3-2-**5-7-1-6" e "6-4-**2-3-**5-7-1-6".

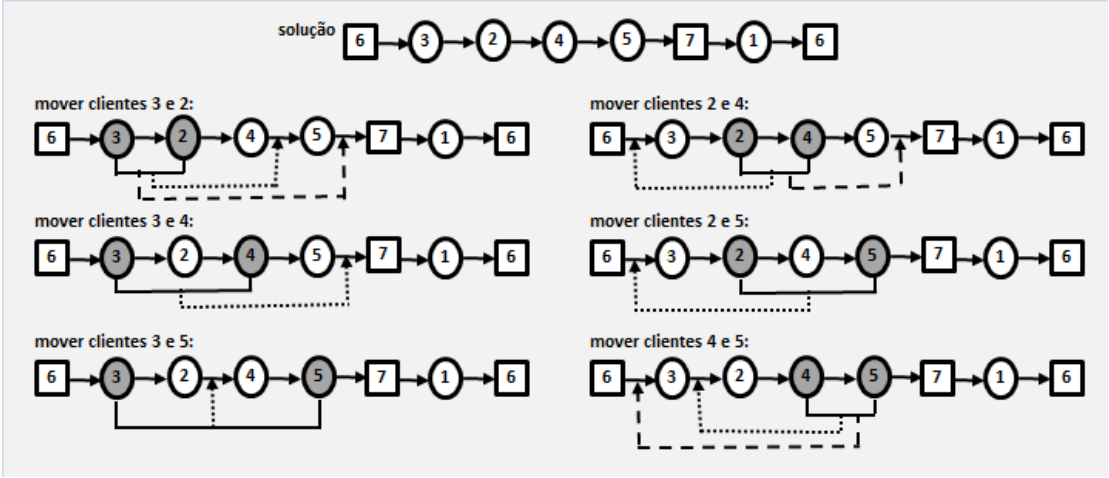

Figura 4.8: Exemplo de aplicação do operador *Move2-within*

No algoritmo 4.7 apresenta-se o pseudo-código para pesquisar soluções na vizinhança induzida pelo operador *Move2-within*. É considerada a seguinte notação:

- N: número de elementos que compõem a solução S
- S_i: elemento na posição i da rota (solução S)
- $V(x)$: valor da solução x
- *besti*: posição de um dos clientes cuja sua remoção permite obter a melhor solução na vizinhança induzida pelo operador *Move2-within*
- *bestj*: posição de um dos clientes cuja sua remoção permite obter a melhor solução na vizinhança induzida pelo operador *Move2-within*
- *bestk*: melhor posição do cliente ou hotel onde os clientes removidos serão inseridos,

sendo que o cliente que se encontrava na posição *besti* será inserido antes do cliente que se encontrava na posição *bestj* da rota

4.2.2 Operadores que otimizam a solução a nível da rota

Os operadores *Swap-between* e *Move-between* otimizam a solução a nível da rota, isto é, as trocas são realizadas entre clientes de diferentes percursos. De seguida serão descritos cada um destes operadores e apresentados alguns exemplos de aplicação para melhor compreensão dos mesmos.

Swap-between

O operador *Swap-between* segue uma lógica semelhante ao *Swap-within*, sendo que aqui a troca é realizada entre dois clientes de percursos distintos.

Repare-se que a aplicação do operador *Swap-between* conduz-nos, tal como os operadores aqui apresentados, à obtenção de uma solução com o mesmo número de percursos da solução de partida. Note-se ainda que, ao contrário do que acontecia em *Swap-within*, neste operador há que averiguar a admissibilidade da solução obtida após a troca, isto porque pode-se obter uma solução com melhor valor porém não admissível. A duração de um ou dos dois percursos onde ocorreu a troca poderá ser superior a L, pois os clientes trocados passam a ser visitados em percursos diferentes e logo tanto a soma dos tempos de serviço como a duração da viagem, devido às novas ligações, nesses percursos podem vir alteradas.

A figura 4.9 exemplifica a aplicação do operador *Swap-between*: a solução 1 possui uma duração inferior à duração da solução de partida e é admissível, sendo a duração de cada percurso inferior a L; a solução 2, apesar de ser uma solução admissível, possui uma duração superior à duração da solução de partida, logo não a irá atualizar e como tal será descartada.

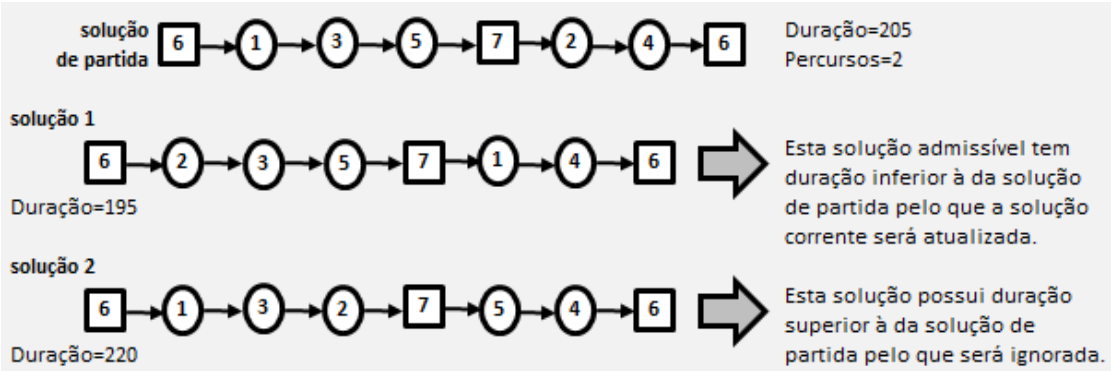

Figura 4.9: Exemplo de aplicação do operador *Swap-between*

Algoritmo 4.7 Pseudo-código para pesquisar na vizinhança induzida pelo *Move2-within*

Require: Solução admissível S'

$S \leftarrow S'$
repeat
  newvalue $\leftarrow V(S)$
  melhorou = 0
  for $i = 2$ **to** N-1 **do**
    if S_i é cliente **then**
      $inicio \leftarrow$ posição do hotel inicial do percurso onde S_i é visitado em S
      $fim \leftarrow$ posição do hotel final do percurso onde S_i é visitado em S
      for $j = i + 1$ **to** N **do**
        if S_j é cliente **then**
          for $k = inicio$ **to** $fim-1$ **do**
            if S_k é cliente ou (S_k é hotel e S_{k+1} é cliente) **then**
              if $k \neq i$ e $k \neq i-1$ e $k \neq j$ e $k \neq j-1$ **then**
                $s_1 \leftarrow$ solução obtida inserindo os clientes i e j (por esta ordem) entre S_k e S_{k+1}
                $s_2 \leftarrow$ solução obtida inserindo os clientes j e i (por esta ordem) entre S_k e S_{k+1}
                if min $\{V(s_1), V(s_2)\} \leq$ newvalue **then**
                  newvalue $\leftarrow \min\{V(s_1), V(s_2)\}$
                  $besti \leftarrow i$ se $\min\{V(s_1), V(s_2)\} = V(s_1)$ ou j se $\min\{V(s_1), V(s_2)\} = V(s_2)$
                  $bestj \leftarrow j$ se $\min\{V(s_1), V(s_2)\} = V(s_1)$ ou i se $\min\{V(s_1), V(s_2)\} = V(s_2)$
                  $bestk \leftarrow k$
                  melhorou = 1
                end if
              end if
            end if
          end for
        else
          Sair do ciclo for
        end if
      end for
    end if
  end for
  if melhorou=1 **then**
    S $\leftarrow$ solução obtida inserindo os clientes que se encontram nas posições *besti* e *bestj*, por esta ordem, entre o cliente ou hotel na posição *bestk* e o cliente ou hotel na posição *bestk*+1 da rota
    $V(S) \leftarrow$ newvalue
  end if
until Solução com valor inferior ou com valor igual a $V(S)$ e que ainda não tenha sido pesquisada seja encontrada

Ensure: Solução admissível com valor não superior ao valor de S'

No algoritmo 4.8 apresenta-se o pseudo-código para pesquisar soluções na vizinhança induzida pelo operador *Swap-between*. É considerada a seguinte notação:

- N: número de elementos que compõem a solução S
- S_i: elemento na posição i da rota (solução S)
- $V(x)$: valor da solução x
- *besti*: posição do cliente cuja troca com o cliente da posição *bestj* permite obter a melhor solução na vizinhança induzida pelo operador *Swap-between*
- *bestj*: posição do cliente cuja troca com o cliente da posição *besti* permite obter a melhor solução na vizinhança induzida pelo operador *Swap-between*

Move-between

O operador *Move-between* segue uma lógica semelhante ao operador *Move1-within*, sendo que nesta o cliente retirado de um dado percurso é colocado num percurso distinto.

A aplicação do operador *Move-between* conduz-nos também à obtenção de uma solução com o mesmo número de percursos da solução de partida. No entanto, ao contrário do que acontecia em *Move1-within*, em *Move-between* há que averiguar a admissibilidade da solução obtida após a troca, isto porque pode-se obter uma solução com melhor valor porém não admissível na medida em que esta troca envolve percursos distintos. Tanto a duração do percurso de onde foi retirado o cliente como a duração do percurso onde ele foi inserido podem aumentar e ser superiores a L. O primeiro caso pode ocorrer mesmo com a remoção do cliente devido às novas ligações, já o segundo caso pode ocorrer pois temos mais um cliente a ser visitado e, como tal, na duração do percurso tem de ser tido em conta o tempo de serviço do cliente e o tempo de viagem.

Na figura 4.10 pode ser observada a aplicação do operador *Move-between*: a solução 1, apesar de ser uma solução admissível, possui uma duração superior à duração da solução de partida; a solução 2 tem uma duração inferior à duração da solução de partida e é admissível.

No algoritmo 4.9 apresenta-se o pseudo-código para pesquisar soluções na vizinhança induzida pelo operador *Move-between*. É considerada a seguinte notação:

- N: número de elementos que compõem a solução S
- S_i: elemento na posição i da rota (solução S)
- $V(x)$: valor da solução x
- *besti*: posição do cliente cuja sua remoção permite obter a melhor solução na vizinhança induzida pelo operador *Move-between*
- *bestj*: melhor posição do cliente ou hotel onde o cliente removido que se encontrava na posição *besti* será inserido

Algoritmo 4.8 Pseudo-código para pesquisar na vizinhança induzida pelo *Swap-between*

Require: Solução admissível S'

$S \leftarrow S'$
repeat
  newvalue $\leftarrow V(S)$
  melhorou $= 0$
  for $i = 2$ **to** N-1 **do**
    if S_i é cliente **then**
      for $j = 2$ **to** N-1 **do**
        if S_j é cliente e S_j não é visitado no mesmo percurso onde S_i é visitado **then**
          value $\leftarrow V(S) - b_{S_{i-1},S_i} - b_{S_i,S_{i+1}} + b_{S_{i-1},S_j} + b_{S_j,S_{i+1}} - b_{S_{j-1},S_j} - b_{S_j,S_{j+1}} + b_{S_{j-1},S_i} + b_{S_i,S_{j+1}}$
          if value $\leq$ newvalue **then**
            $comp_1 \leftarrow$ tempo total de viagem do percurso onde S_i é visitado $-b_{S_{i-1},S_i} - b_{S_i,S_{i+1}} + b_{S_{i-1},S_j} + b_{S_j,S_{i+1}} - \tau_{S_i} + \tau_{S_j}$
            $comp_2 \leftarrow$ tempo total de viagem do percurso onde S_j é visitado $-b_{S_{j-1},S_j} - b_{S_j,S_{j+1}} + b_{S_{j-1},S_i} + b_{S_i,S_{j+1}} - \tau_{S_j} + \tau_{S_i}$
            if $comp_1 \leq$ L e $comp_2 \leq$ L **then**
              newvalue $\leftarrow$ value
              besti $\leftarrow i$
              bestj $\leftarrow j$
              melhorou $= 1$
            end if
          end if
        end if
      end for
    end if
  end for
  if melhorou=1 **then**
    S $\leftarrow$ solução obtida trocando o cliente na posição *besti* e o cliente na posição *bestj*
    $V(S) \leftarrow$ newvalue
  end if
until Solução com valor inferior ou com valor igual a $V(S)$ e que ainda não tenha sido pesquisada seja encontrada

Ensure: Solução admissível com valor não superior ao valor de S'

Algoritmo 4.9 Pseudo-código para pesquisar na vizinhança induzida pelo *Move-between*

Require: Solução admissível S'

$S \leftarrow S'$
repeat
  newvalue $\leftarrow V(S)$
  melhorou $= 0$
  for $i = 2$ **to** N-1 **do**
    if S_i é cliente **then**
      for $j = 2$ **to** N-1 **do**
        if (S_j é cliente e não é visitado no mesmo percurso onde S_i é visitado) ou (S_j é hotel e S_{j+1} é cliente e não é visitado no mesmo percurso onde S_i é visitado) **then**
          value $\leftarrow V(S) - b_{S_{i-1},S_i} - b_{S_i,S_{i+1}} + b_{S_{i-1},S_{i+1}} + b_{S_j,S_i} + b_{S_i,S_{j+1}} - b_{S_j,S_{j+1}}$
          if value $\leq$ newvalue **then**
            $comp_1 \leftarrow$ tempo total de viagem do percurso onde se encontra o cliente da posição i $-b_{S_{i-1},S_i} - b_{S_i,S_{i+1}} + b_{S_{i-1},S_{i+1}} - \tau_{S_i}$
            $comp_2 \leftarrow$ tempo total de viagem do percurso onde se encontra o cliente da posição j $+b_{S_j,S_i} + b_{S_i,S_{j+1}} - b_{S_j,S_{j+1}} + \tau_{S_i}$
            if $comp_1 \leq$ L e $comp_2 \leq$ L **then**
              newvalue $\leftarrow$ value
              $besti \leftarrow i$
              $bestj \leftarrow j$
              melhorou $= 1$
            end if
          end if
        end if
      end for
    end if
  end for
  if melhorou=1 **then**
    S $\leftarrow$ solução obtida inserindo o cliente na posição *besti* entre o cliente ou hotel na posição *bestj* e o cliente ou hotel na posição *bestj*$+1$ da rota
    $V(S) \leftarrow$ newvalue
  end if
until Solução com valor inferior ou com valor igual a $V(S)$ e que ainda não tenha sido pesquisada seja encontrada

Ensure: Solução admissível com valor não superior ao valor de S'

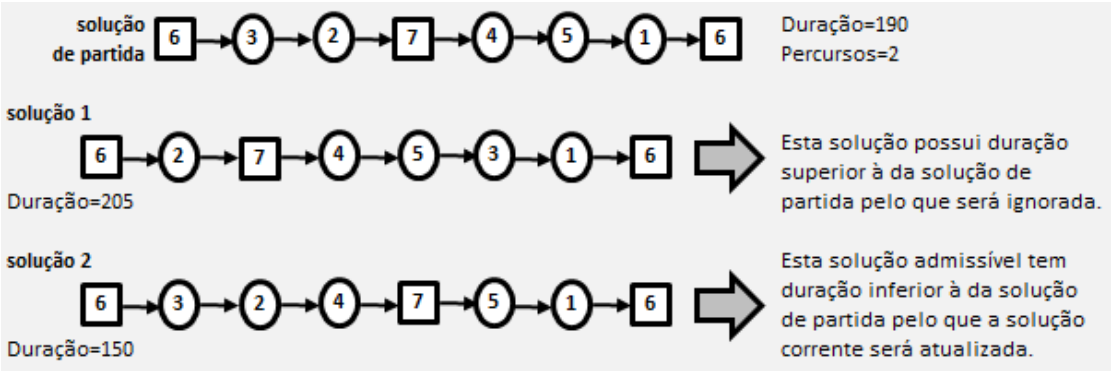

Figura 4.10: Exemplo de aplicação do operador *Move-between*

4.2.3 Outros operadores

Os operadores *Inversion* e *Destroy-Insert-Hotels* tentam otimizar a rota ao nível do percurso e ao nível da rota. O primeiro otimiza a solução dentro e entre percursos e o segundo otimiza a rota quer no número de percursos quer na sua duração. De seguida serão descritos cada um destes operadores e apresentados alguns exemplos de aplicação para melhor compreensão dos mesmos.

Inversion

O operador *Inversion* atua ao nível do percurso incluindo os hotéis inicial e final e consiste em inverter a ordem de visita dos clientes dentro de um dado percurso, possibilitando a substituição dos hotéis inicial e final desse percurso por outros hotéis. Por percurso, procede-se primeiramente à inversão da ordem de visita dos clientes e posteriormente à "atualização dos hotéis", no caso de tal permitir manter a admissibilidade da rota. Esta "atualização" consiste na substituição de cada um dos hotéis inicial e final do percurso por um hotel escolhido com o intuito de minimizar a duração total da rota, tendo em atenção que só se admite esta substituição para hotéis intermédios.

A aplicação do operador *Inversion* conduz-nos à obtenção de uma solução com o mesmo número de percursos da solução de partida, no entanto, com uma duração total não superior. Denote-se que, após a inversão da ordem de visita dos clientes e "atualização dos hotéis" não só poderemos obter uma duração superior a L no percurso onde ocorreu a inversão como nos percursos a ele adjacentes. Deste modo, para além de se analisar a duração total da solução resultante, há que averiguar a sua admissibilidade. No caso de a inversão ter ocorrido no primeiro percurso é apenas necessário verificar se as durações do primeiro e do segundo percurso são respetivamente não superiores a L, pois nos restantes percursos não ocorreu nenhuma alteração. No caso de a inversão ter ocorrido no último percurso basta verificar as durações do penúltimo e do último percurso. Já no caso de ocorrer num percurso intermédio há que verificar se a sua duração

e a duração do percurso antecedente e a do subsequente são não superiores a L.

Inicialmente o hotel escolhido para substituir um dado hotel corresponde ao que minimiza a distância entre o hotel ou cliente que o precede e o hotel ou cliente que o sucede, seja ele o hotel K. No caso de tal nos conduzir a uma solução com duração total superior à da solução corrente necessariamente a escolha de um outro hotel iria nos levar à mesma situação. No entanto, no caso de nos conduzir a uma solução com duração total não superior à da solução corrente, a solução obtida pode ser não admissível, pelo que a escolha de um outro hotel para o substituir é plausível. Assim, neste último caso, tenta-se substituir o hotel pelo hotel diferente de K que produza uma solução admissível e com menor duração total da rota.

De referir ainda que quando se procede à inversão dos clientes dentro de um dado percurso continuam-se a visitar os mesmos clientes, sendo que a única diferença na duração do percurso pode advir dos tempos de viagem da primeira e da última ligação. Considere-se como exemplo o percurso "6-5-4-1-7". Ao proceder à inversão da ordem de visita dos clientes ficamos com "6-1-4-5-7". Continuamos a ligar 1 a 4 e 4 a 5, a diferença está na primeira e na última viagem: enquanto antes tínhamos as ligações 6-5 e 1-7 passamos a ter 6-1 e 5-7, pelo que se consideram outros tempos de viagem e como tal a duração total do percurso pode alterar. Note-se, no entanto, que no caso do hotel inicial e final do percurso serem o mesmo, após a inversão, serão obtidas as mesmas ligações e consequentemente a duração do percurso onde ocorreu a inversão manter-se-á igual. Ou seja, neste caso a inversão conduz-nos à obtenção de uma solução alternativa com o mesmo valor. Assim, considerou-se a aplicação do operador *Inversion* apenas a soluções com mais do que um percurso e em percursos tais que o hotel inicial e final sejam diferentes. Note-se que também só será aplicado a percursos com pelo menos dois clientes, pois caso contrário a inversão não terá qualquer efeito.

Na figura 4.11 pode ser observada a aplicação do operador *Inversion*. Note-se que a solução 1 tem duração não superior à da solução de partida pelo que se averigua a sua admissibilidade. Como é admissível atualizamos a solução corrente, no entanto, caso tal não se verificasse seria necessário experimentar a substituição por um outro hotel como acima descrito.

No algoritmo 4.10 apresenta-se o pseudo-código para pesquisar soluções na vizinhança induzida pelo operador *Inversion*. É considerada a seguinte notação:

- HI_j: hotel inicial de um dado percurso que se encontra na posição j da rota
- HF_k: hotel final de um dado percurso que se encontra na posição k da rota, $k > j$
- $V(x)$: valor da rota x

Algoritmo 4.10 Pseudo-código para pesquisar na vizinhança induzida pelo *Inversion*

Require: Solução admissível S'

$S \leftarrow S'$
repeat
  $newsol \leftarrow$ S
  newvalue $\leftarrow V(S)$
  melhorou = 0
  for all Percurso i da rota S definido pelo HI_j e pelo HF_k **do**
    if $\exists$ pelo menos 2 clientes no percurso i e $HI_j \neq HF_k$ **then**
      Inversão da ordem de visita dos clientes que se encontram entre os hotéis HI_j e HF_k na solução S
      if É o primeiro percurso da rota **then**
        $sol \leftarrow$ solução que se obtém substituindo HF_k pelo hotel mais próximo entre o hotel/cliente que o precede e o hotel/cliente que o sucede
      else if É o último percurso da rota **then**
        $sol \leftarrow$ solução que se obtém substituindo HI_j pelo hotel mais próximo entre o hotel/cliente que o precede e o hotel/cliente que o sucede
      else
        $sol \leftarrow$ solução que se obtém substituindo cada um dos hotéis HI_j e HF_k pelo hotel mais próximo entre o hotel/cliente que o precede e o hotel/cliente que o sucede, respetivamente
      end if
      if $V(sol) \leq$ newvalue **then**
        if *sol* é admissível **then**
          newsol $\leftarrow$ sol
          newvalue $\leftarrow V(sol)$
          melhorou = 1
        else
          if $\exists$ hotéis h_1 e h_2 tal que a substituição de HI_j e HF_k por h_1 e h_2, respetivamente, torna a rota admissível e são os hotéis nessas condições que minimizam a duração total da nova rota **then**
            $newsol \leftarrow$ solução obtida substituindo os hotéis HI_j e HF_k por h_1 e h_2, respetivamente
            $newvalue \leftarrow V(newsol)$
            melhorou = 1
          end if
        end if
      end if
    end if
  end for
  if melhorou=1 **then**
    S $\leftarrow$ newsol
    $V(S) \leftarrow$ newvalue
  end if
until Solução com valor inferior ou com valor igual a $V(S)$ e que ainda não tenha sido pesquisada seja encontrada

Ensure: Solução admissível com valor não superior ao valor de S'

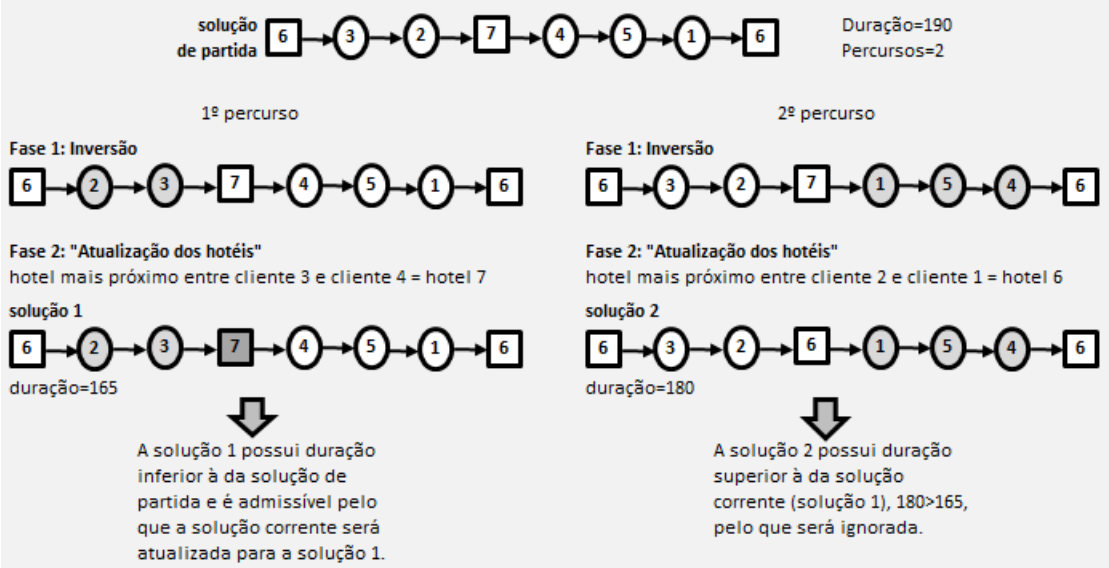

Figura 4.11: Exemplo de aplicação do operador *Inversion*

Destroy-Insert-Hotels

O operador *Destroy-Insert-Hotels* permite obter uma solução com o mesmo número de percursos mas com uma duração total não superior à duração da solução de partida ou uma solução que, apesar de poder ter duração total superior ou igual à duração da solução de partida, seja composta por um menor número de percursos. É um operador constituído por duas fases.

A primeira fase de *Destroy-Insert-Hotels* tem como intuito reduzir o número de percursos da solução. Ao proceder às diferentes trocas anteriormente apresentadas podem existir hotéis consecutivos correspondentes a percursos onde não se visitam clientes. Deste modo, na primeira fase procura-se eliminar percursos procedendo-se da seguinte forma: cada par de hotéis consecutivos é substituído por um único hotel escolhido de modo a minimizar a duração total da rota e a manter a sua admissibilidade. No caso de se ter um percurso sem clientes cujo hotel inicial e final seja o mesmo simplesmente elimina-se um desses hotéis da rota e necessariamente ter-se-á uma solução admissível. No caso do hotel inicial e final do percurso sem clientes diferirem, ambos serão substituídos pelo hotel que minimiza a duração total da rota ou, no caso de estarmos no primeiro ou no último percurso da rota, pelo depósito, desde que os percursos obtidos tenham duração não superior a L.

Na figura 4.12 encontram-se apresentadas seis soluções diferentes, para as quais se indica qual a solução resultante após aplicação da fase 1. Nas soluções 1, 2 e 3 temos no máximo dois hotéis consecutivos, ou seja, um único percurso sem clientes: a solução 1 é composta por dois hotéis consecutivos distintos e uma vez que tal ocorre no último percurso eles são substituídos pelo depósito, o mesmo ocorreria se tal acontecesse no primeiro percurso; a solução 2 tem dois hotéis consecutivos iguais então basta eliminar

um deles, procedendo-se de igual modo se tal ocorresse num percurso intermédio ou no último; a solução 3 possui dois hotéis distintos num percurso intermédio então, neste caso, esses hotéis serão substituídos pelo hotel mais próximo entre o cliente 2 (último cliente que antecede o percurso sem clientes) e o cliente 4 (primeiro cliente que sucede o percurso sem clientes) que corresponde ao hotel 6 pois $b_{26} + b_{64} = 40 < 50 = b_{27} + b_{74}$, considerando os dados apresentados no subcapítulo 3.3. Na solução 4 já se presencia a existência de mais do que dois hotéis consecutivos. Nestas situações procede-se de modo análogo ao que foi acima descrito.

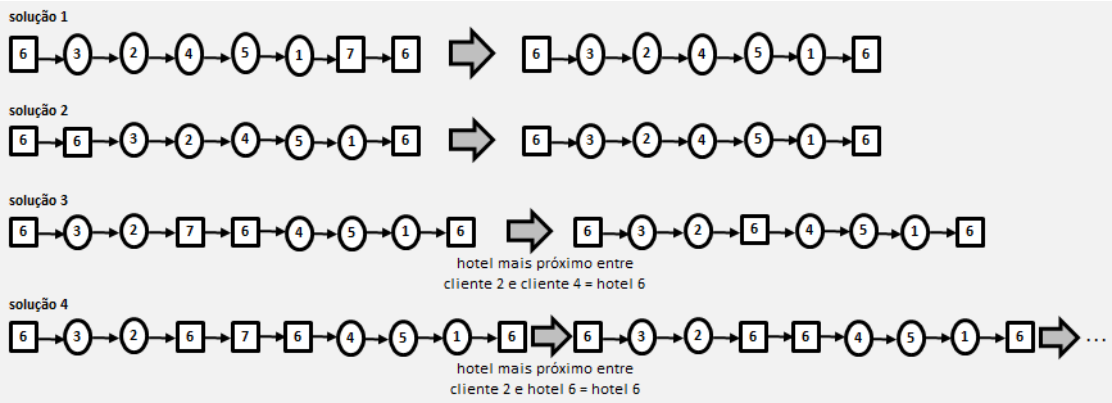

Figura 4.12: Exemplo de aplicação da fase 1 do operador *Destroy-Insert-Hotels*

No algoritmo 4.11 apresenta-se o pseudo-código para pesquisar soluções na vizinhança induzida pela fase 1 do operador *Destroy-Insert-Hotels*. É considerada a seguinte notação:

- N: número de elementos que compõem a solução S
- S_i: elemento na posição i da rota (solução S)
- $s_{i,j}$: hotel mais próximo entre os clientes i e j
- $u_{i,j}$: hotel mais próximo entre o hotel i e o cliente j
- $V(x)$: valor da solução x
- *time(x)*: tempo total de viagem de x

A segunda fase de *Destroy-Insert-Hotels* é aplicada à solução obtida no final da fase um. Removem-se todos os hotéis e posteriormente procede-se à sua inserção. A inserção dos hotéis é feita seguindo a lógica utilizada no descodificador, ou seja, após a remoção de todos os hotéis obtém-se uma permutação com a ordem de visita dos clientes à qual é aplicada o descodificador, garantindo-se desta forma a admissibilidade da solução resultante. A solução obtida aplicando o descodificador à permutação pode diferir da solução obtida aplicando o descodificador à permutação lida no sentido inverso. Assim,

Algoritmo 4.11 Pseudo-código para pesquisar na vizinhança induzida pela fase 1 do *Destroy-Insert-Hotels*

Require: Solução admissível S'

$S \leftarrow S'$
melhorou = 1
while melhorou = 1 **do**
  melhorou = 0
  for $i = 1$ **to** N−1 **do**
    if S_i e S_{i+1} são hotéis, isto é, se existem dois hotéis consecutivos na rota **then**
      if $S_i = S_{i+1}$ **then**
        S ← solução obtida eliminando um desses hotéis da rota
        $V(S) \leftarrow V(S) - M_1$
        melhorou = 1
        Sair do ciclo for
      else if $S_i \neq S_{i+1}$ e definem o primeiro percurso da rota S **then**
        if *time*(segundo percurso) $- b_{S_{i+1},S_{i+2}} + b_{S_i,S_{i+2}} \leq$ L **then**
          S ← solução obtida eliminando o segundo hotel da rota S
          $V(S) \leftarrow V(S) - b_{S_i,S_{i+1}} - b_{S_{i+1},S_{i+2}} + b_{S_i,S_{i+2}} - M_1$
          melhorou = 1
          Sair do ciclo for
        end if
      else if $S_i \neq S_{i+1}$ e definem o último percurso da rota S **then**
        if *time*(penúltimo percurso) $- b_{S_{i-1},S_i} + b_{S_{i-1},S_{i+1}} \leq$ L **then**
          S ← solução obtida eliminando o penúltimo hotel da rota S
          $V(S) \leftarrow V(S) - b_{S_{i-1},S_i} - b_{S_i,S_{i+1}} + b_{S_{i-1},S_{i+1}} - M_1$
          melhorou = 1
          Sair do ciclo for
        end if
      else if $S_i \neq S_{i+1}$ e definem um percurso intermédio **then**
        $h \leftarrow s_{S_{i-1},S_{i+2}}$ se S_{i+2} é cliente ou $u_{S_{i+2}-n,S_{i-1}}$ se S_{i+2} é hotel
        if a substituição de S_i e S_{i+1} na rota por h torna a rota admissível **then**
          S ← solução obtida substituindo os dois hotéis consecutivos por h
          $V(S) \leftarrow V(S) - b_{S_{i-1},S_i} - b_{S_i,S_{i+1}} - b_{S_{i+1},S_{i+2}} + b_{S_{i-1},h} + b_{h,S_{i+2}} - M_1$
          melhorou = 1
          Sair do ciclo for
        else
          if ∃ hotel k tal que a substituição de S_i e S_{i+1} por k torna a rota admissível e k é o hotel que minimiza a duração total da nova rota **then**
            S ← solução obtida substituindo os dois hotéis consecutivos por k
            $V(S) \leftarrow V(S) - b_{S_{i-1},S_i} - b_{S_i,S_{i+1}} - b_{S_{i+1},S_{i+2}} + b_{S_{i-1},k} + b_{k,S_{i+2}} - M_1$
            melhorou = 1
            Sair do ciclo for
          end if
        end if
      end if
    end if
  end for
end while

Ensure: Solução admissível com valor não superior ao valor de S'

à permutação que resulta por remoção dos hotéis é aplicado o descodificador "lendo" a permutação nos dois sentidos, sendo escolhida a que corresponde à melhor solução. No caso de as soluções diferirem no número de percursos opta-se pela solução com menor número de percursos mesmo que possua uma duração total superior à da outra solução e no caso de as soluções possuírem o mesmo número de percursos opta-se pela solução com menor duração.

Na figura 4.13 encontra-se um exemplo de aplicação da fase 2 na solução "6-5-4-1-6-3-2-6". É possível verificar que as soluções 1 e 2 correspondem à mesma codificação, isto é, ao mesmo cromossoma, sendo no entanto soluções distintas. Observa-se ainda que com o descodificador se vão construindo os percursos tentando colocar o maior número possível de clientes e que tal produz uma solução pior nesta situação. As soluções 2 e 3 mostram que efetivamente "ler" a permutação com o descodificador num sentido e no sentido inverso pode conduzir à obtenção de soluções diferentes. Neste caso, verifica-se que nenhuma das soluções 2 e 3 é melhor que a solução 1, pelo que no fim da fase 2 obter-se-á a mesma solução de partida, solução 1. De facto tem-se: valor(solução 1) $= 175 + M_1 \times 2 < 190 + M_1 \times 2 =$ valor(solução 2) e valor(solução 1) $= 175 + M_1 \times 2 < 200 + M_1 \times 2 =$ valor(solução3), recordando que M_1 corresponde à penalização atribuída ao número de percursos existentes na solução.

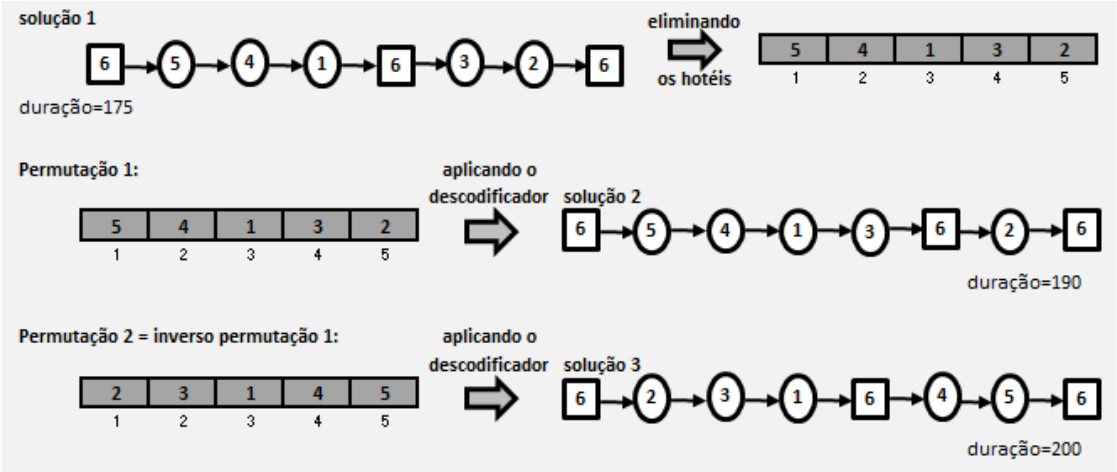

Figura 4.13: Exemplo de aplicação da fase 2 do operador *Destroy-Insert-Hotels*

No algoritmo 4.12 apresenta-se o pseudo-código para pesquisar soluções na vizinhança induzida pela fase 2 do operador *Destroy-Insert-Hotels*, no qual se considera que o valor de uma solução x é denotado por $V(x)$.

Algoritmo 4.12 Pseudo-código para pesquisar na vizinhança induzida pela fase 2 do *Destroy-Insert-Hotels*

Require: Solução admissível S'
$S \leftarrow S'$
$perm_1 \leftarrow$ permutação dos clientes obtida eliminando os hotéis da rota S
$sol_1 \leftarrow$ rota obtida aplicando o *Descodificador* a $perm_1$
$perm_2 \leftarrow$ permutação dos clientes obtida invertendo a ordem dos clientes em $perm_1$
$sol_2 \leftarrow$ rota obtida aplicando o *Descodificador* a $perm_2$
if $V(sol_1) \leq V(sol_2)$ **then**
    $s \leftarrow sol_1$
else
    $s \leftarrow sol_2$
end if
if $V(s) < V(S)$ ou $(V(s) = V(S)$ e $s \neq S)$ **then**
    $V(S) \leftarrow V(s)$
    $S \leftarrow s$
end if
Ensure: Solução admissível com valor não superior ao valor de S'

O mecanismo de pesquisa local utilizado em ambas as meta-heurísticas propostas para o TSPHS encontra-se apresentado no algoritmo 4.13 e é executado enquanto se obtiver uma solução com valor não superior ao valor da melhor solução corrente. Optou-se por se aceitar soluções com valor igual ao valor da solução corrente e não apenas soluções com valor inferior pois tal pode permitir testar outras soluções (outra zona no espaço de pesquisa) e eventualmente possibilitar que se possa "sair" de possíveis ótimos locais. Por outro lado, deste modo poderíamos voltar a pesquisar soluções já analisadas. Tal situação poderia também ocorrer em cada um dos operadores de vizinhança, razão pela qual se decidiu que cada uma seria repetida enquanto fosse encontrada uma solução com valor inferior ou com valor igual mas que ainda não tivesse sido pesquisada.

Foi dicidido primeiro atuar na duração de cada percurso com vista à sua redução, posteriormente atuar na redução do número de percursos e, por fim, na redução da duração total da rota. A ordem pela qual se aplicam os operadores propostos é *Swap-within*, *2-Opt*, *Move1-within*, *Move2-within*, *Destroy-Insert-Hotels*, *Move-between*, *Swap-between* e *Inversion*. Experimentou-se a escolha aleatória dos operadores a aplicar, verificando-se que tal não trazia vantagens em relação à ordem apresentada.

4.3 Procedimentos de perturbação

A aplicação da pesquisa local, mesmo considerando diferentes vizinhanças, pode conduzir-nos a soluções ótimas locais de fraca qualidade. Deste modo, para tentar "sair" desses ótimos locais procedeu-se à perturbação dessas soluções. Esta perturbação consiste na

Algoritmo 4.13 Procedimento de pesquisa local

Require: Solução admissível S''

1: $S \leftarrow S''$ // S é solução corrente
2: $aux = 0$ // número de atualizações de S para soluções com um mesmo valor
3: $y = 1$
4: **while** $y \neq 0$ **do**
5: **if** nº de percursos de $S = 1$ **then**
6: custo ← valor de S
7: Aplicar os operadores *Swap-within*, *2-Opt*, *Move1-within*, *Move2-within* e *Destroy-Insert-Hotels* (fase 2)
8: **if** Se S não foi atualizada no passo 7 **then**
9: $y = 0$
10: **else**
11: **if** valor de S = custo **then**
12: $aux = aux + 1$
13: **if** $aux = 2$ **then**
14: $y = 0$
15: **else**
16: $y = 1$
17: **end if**
18: **else**
19: $aux = 0$
20: $y = 1$
21: **end if**
22: **end if**
23: **else**
24: custo ← valor de S
25: Aplicar os operadores *Swap-within*, *2-Opt*, *Move1-within*, *Move2-within*, *Destroy-Insert-Hotels* (fases 1 e 2), *Move-between*, *Swap-between* e *Inversion*
26: **if** Se S não foi atualizada no passo 25 **then**
27: $y = 0$
28: **else**
29: **if** valor de S = custo **then**
30: $aux = aux + 1$
31: **if** $aux = 2$ **then**
32: $y = 0$
33: **else**
34: $y = 1$
35: **end if**
36: **else**
37: $aux = 0$
38: $y = 1$
39: **end if**
40: **end if**
41: **end if**
42: **end while**

Ensure: Solução admissível com valor não superior ao valor de S''

alteração da posição dos clientes da rota e tem como principal objetivo pesquisar outras regiões do espaço de pesquisa.

Tal como nos algoritmos genéticos, trabalhamos no espaço das codificações pelo que as perturbações foram aplicadas às permutações. Foram utilizados quatro tipos de perturbações, nomeadamente *Move-chain*, *Swap-chain*, *Shake-chain* e *Inversion-chain*, as quais serão apresentadas de seguida.

Move-chain

A perturbação *Move-chain* consiste em mover uma cadeia de c clientes consecutivos, que se encontram desde a posição *indice1* da permutação à posição *indice1*+c-1, para a frente do cliente na posição *indice2*. Foi considerado que a escolha da dimensão da cadeia é feita aleatoriamente por entre as seguintes possibilidades: $c = \{1, 2, 3\}$.

Exemplos de movimentos possíveis com a perturbação *Move-chain* podem ser observados na figura 4.14. No algoritmo 4.14 pode-se observar o pseudo-código do procedimento da perturbação *Move-chain* utilizado.

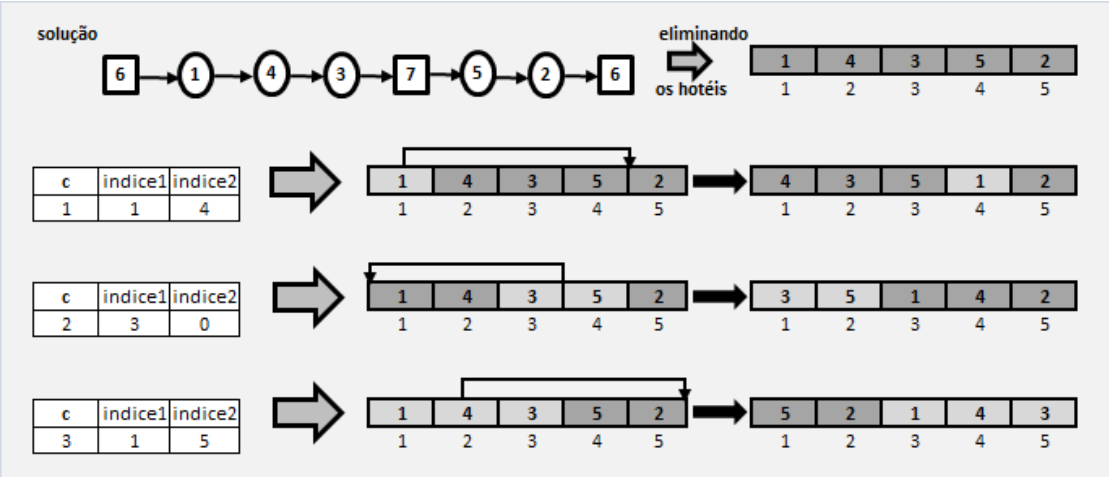

Figura 4.14: Exemplos de aplicação da perturbação *Move-chain*

Algoritmo 4.14 Procedimento da perturbação *Move-chain*

Require: n, *perm*

1: $c \leftarrow$ gerar nº inteiro entre 1 e 3
2: *indice1* $\leftarrow$ gerar nº inteiro entre 1 e n-c+1
3: *indice2* $\leftarrow$ gerar nº inteiro entre 0 e n tal que *indice*2 < *indice*1−1 ou *indice*1+c−1 < *indice*2
4: *newperm* $\leftarrow$ Mover a cadeia de c clientes com início na posição *indice1* de *perm* para a frente do cliente que se encontra na posição *indice2*

Ensure: Permutação *newperm*

Swap-chain

A perturbação *Swap-chain* consiste em trocar uma cadeia de c clientes consecutivos que se encontra a partir da posição *indice1* na permutação por uma cadeia de c clientes consecutivos que se encontra a partir da posição *indice2* na permutação. Foi considerado que a escolha da dimensão da cadeia é feita aleatoriamente por entre as seguintes possibilidades: $c = \{1, 2, 3\}$.

Exemplos de movimentos possíveis com a perturbação *Swap-chain* podem ser observados na figura 4.15. No algoritmo 4.15 pode-se observar o pseudo-código do procedimento da perturbação *Swap-chain* utilizado.

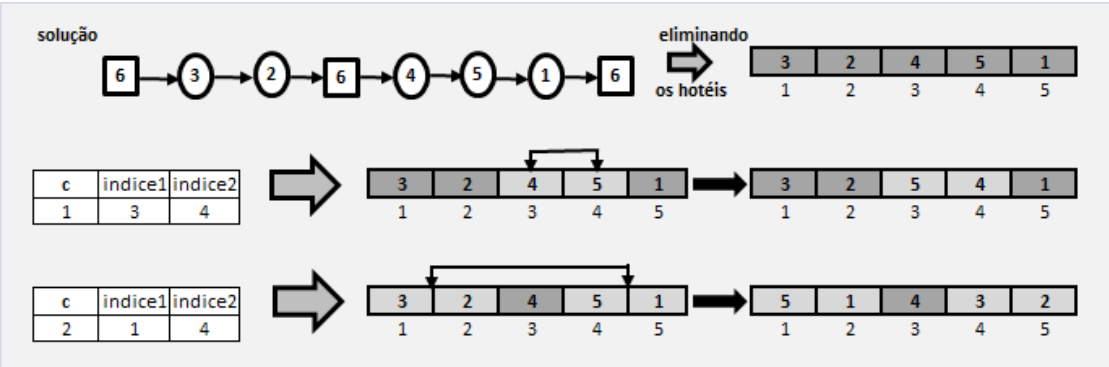

Figura 4.15: Exemplos de aplicação da perturbação *Swap-chain*

Algoritmo 4.15 Procedimento da perturbação *Swap-chain*

Require: n, *perm*

1: $c \leftarrow$ gerar nº inteiro entre 1 e 3
2: *indice1* $\leftarrow$ gerar nº inteiro entre 1 e n-c+1
3: *indice2* $\leftarrow$ gerar nº inteiro entre 1 e n-c+1 tal que $indice2 < indice1 - c + 1$ ou $indice1 + c - 1 < indice2$
4: *newperm* $\leftarrow$ Trocar a cadeia de c clientes com início na posição *indice1* de *perm* com a cadeia de c clientes com início na posição *indice2*

Ensure: Permutação *newperm*

Shake-chain

A perturbação *Shake-chain* considera θ_1 clientes consecutivos que se encontram entre as posições *indice1* e *indice2* e reordena-os aleatoriamente. Esta perturbação tem um efeito semelhante ao da mutação utilizada nos algoritmos genéticos. O parâmetro θ_1, que foi considerado como sendo dependente do número de clientes n, foi alvo de experimentação. De referir que θ_1 deverá tomar valor superior a um, pois caso contrário obtém-se a mesma permutação. No algoritmo 4.16 pode-se observar o pseudo-código do procedimento da

perturbação *Shake-chain* utilizado.

Considere-se como exemplo a aplicação da perturbação *Shake-chain* à permutação de clientes 14352, supondo-se para tal que *indice1* $= 3$, *indice2* $= 5$ e $\theta_1 = 3$. A perturbação *Shake-chain* é então aplicada aos 3 clientes que se encontram desde a posição 3 à posição 5 na permutação, ou seja, aos clientes 352. Uma permutação possível destes clientes é por exemplo 235, ficando-se no final da aplicação desta perturbação com a permutação 14**235**.

Algoritmo 4.16 Procedimento da perturbação *Shake-chain*

Require: n, *perm*, θ_1
1: **repeat**
2: *indice1* $\leftarrow$ gerar nº inteiro entre 1 e n
3: *indice2* $\leftarrow$ gerar nº inteiro entre 1 e n
4: **until** $|indice1 - indice2| + 1 = \theta_1$
5: *newperm* $\leftarrow$ Permutar a cadeia de θ_1 clientes que se encontram desde a posição min(*indice1*,*indice2*) à posição max(*indice1*,*indice2*) de *perm*
Ensure: Permutação *newperm*

Inversion-chain

A perturbação *Inversion-chain* consiste em inverter a ordem de visita de θ_2 clientes consecutivos que se enontram entre as posições *indice1* e *indice2*. O parâmetro θ_2, que foi considerado como sendo dependente do número de clientes n, foi alvo de experimentação. De referir que θ_2 deverá tomar valor superior a um para que possa ocorrer inversão de elementos, pois caso contrário obtém-se a mesma permutação. No algoritmo 4.17 pode-se observar o pseudo-código do procedimento da perturbação *Inversion-chain* utilizado.

Considere-se como exemplo a aplicação da perturbação *Inversion-chain* à permutação de clientes 32451, supondo-se para tal que *indice1* $= 3$, *indice2* $= 1$ e $\theta_2 = 3$. A perturbação *Inversion* é então aplicada aos 3 clientes que se encontram desde a posição 1 à posição 3 na permutação, ou seja, aos clientes 324. Invertendo a ordem de 324 ficamos com 423 e logo no final da aplicação desta perturbação com a permutação **423**51.

Algoritmo 4.17 Procedimento da perturbação *Inversion-chain*

Require: n, *perm*, θ_2
1: **repeat**
2: *indice1* $\leftarrow$ gerar nº inteiro entre 1 e n
3: *indice2* $\leftarrow$ gerar nº inteiro entre 1 e n
4: **until** $|indice1 - indice2| + 1 = \theta_2$
5: *newperm* $\leftarrow$ Inverter a cadeia de θ_2 clientes que se encontram desde a posição min(*indice1*,*indice2*) à posição max(*indice1*,*indice2*) de *perm*
Ensure: Permutação *newperm*

Após aplicação dos procedimentos de perturbação aplica-se o descodificador à permutação resultante para obter a solução correspondente. A esta solução é aplicado o mecanismo de pesquisa local anteriormente descrito. Este método (perturbações seguidas de trocas locais) é repetido *nrep* vezes. Em cada uma das repetições são efetuadas *nperturb* vezes perturbações escolhidas de forma aleatória entre as quatro possibilidades enunciadas. O valor *nperturb* não deverá ser muito elevado para não estragar demasiado a solução nem deverá ser demasiado pequeno pois tal pode não ser suficiente para dar o "salto" necessário para evitar o ótimo local de partida. Os valores a utilizar para *nrep* e *nperturb* foram alvo de alguma experimentação. No algoritmo 4.18 pode-se observar de forma mais sintetizada o procedimento de perturbação de uma solução utilizado.

Algoritmo 4.18 Procedimento de perturbação de uma solução

Require: Solução admissível S; *nrep*; *nperturb*; n; θ_1; θ_2

Retirar os hotéis da melhor solução, obtendo-se a permutação *perm*
contagem = 0
while contagem < *nrep* **do**
  contador = 0
  while contador < *nperturb* **do**
    Gerar um número, *perturb*, aleatório entre 1 e 4
    if *perturb* = 1 **then**
      Aplica-se a perturbação *Move-chain*
    else if *perturb* = 2 **then**
      Aplica-se a perturbação *Swap-chain*
    else if *perturb* = 3 **then**
      Aplica-se a perturbação *Shake-chain*
    else if *perturb* = 4 **then**
      Aplica-se a perturbação *Inversion-chain*
    end if
    perm ← *newperm*
    contador = contador + 1
  end while
  Aplicar o *Descodificador* à lista *perm* dos clientes obtida, obtendo-se uma solução *s*
  Aplicar o procedimento de pesquisa local
  if valor de s < valor de S **then**
    $S \leftarrow s$
  end if
  contagem = contagem + 1
end while

Ensure: Solução admissível

Capítulo 5

Resultados computacionais

No presente capítulo serão apresentados e discutidos os resultados obtidos com a metodologia proposta para as instâncias de teste existentes na literatura para o TSPHS. As meta-heurísticas propostas foram implementadas utilizando o *software* MATLAB versão R2015a. Os resultados apresentados neste capítulo foram obtidos com recurso a um computador com processador *Intel(R) Core(TM) i3-2100 CPU @ 3.10GHz 4GB.*

Este capítulo subdivide-se em três secções. Na primeira secção será explicado como as instâncias de teste utilizadas foram geradas. Na segunda secção serão expostos, de forma sintetizada, todos os testes realizados aos parâmetros necessários para as meta-heurísticas. Na terceira secção apresentar-se-ão as tabelas com os resultados finais obtidos.

5.1 Instâncias de teste

Para testar as meta-heurísticas desenvolvidas para o TSPHS foram utilizadas as instâncias de teste existentes na literatura para o problema, as quais foram geradas por Vansteenwegen *et al.* a partir de instâncias existentes na literatura para o VRP e TSP. As instâncias de teste utilizadas dividem-se em quatro conjuntos e encontram-se disponíveis no *site* [20].

O primeiro conjunto de instâncias (SET1) contém um total de 16 instâncias: 6 instâncias com 100 clientes (c101, c201, r101, r201, rc101 e rc201) criadas a partir das instâncias do VRP com capacidade e janelas temporais (CVRPTW) geradas por Solomon [16] e 10 instâncias (pr01-pr10) cujo número de clientes varia de 48 a 288, criadas a partir das instâncias do VRP com múltiplos depósitos e janelas temporais (MDVRPTW) geradas por Cordeau *et al.* [7]. Nas instâncias do TSPHS geradas a partir das instâncias do CVRPTW, o depósito é o hotel inicial e final da rota. Nas instâncias criadas a partir das instâncias do MDVRPTW, o hotel inicial e final está localizado no centro do conjunto de depósitos disponíveis. Para além do hotel de partida existem 5 hotéis adicionais posicionados nos clientes 1, 11, 21, 31 e 41. As janelas temporais são ignoradas em ambos os casos, mas considera-se que a duração de cada percurso não pode exceder o valor da constante pré-definida L.

O segundo conjunto de instâncias (SET2) é obtido a partir das instâncias do SET1. É constituído por quatro grupos de instâncias. Cada grupo é composto por 13 instâncias das 16 instâncias do SET1, não tendo sido consideradas as instâncias c201, r201 e rc201 por serem facilmente resolvidas considerando rotas com um único percurso. Cada uma das instâncias tem K clientes, K=$\{10, 15, 30, 40\}$, e é obtida a partir da respetiva instância do SET1 considerando os seus K primeiros clientes. Para as instâncias do SET 2, para além do hotel correspondente ao depósito, foi considerado o primeiro hotel dos cinco hotéis adicionais que eram considerados no SET1, nomeadamente o hotel que se encontrava no cliente 1.

O terceiro conjunto de instâncias (SET3) contém instâncias mais complexas que as anteriores. O SET3 é formado por três grupos de instâncias com 3 hotéis, 5 hotéis e 10 hotéis, respetivamente para além do depósito. Cada grupo é composto por 16 instâncias cujo número de clientes varia de 51 a 1002, obtidas a partir de instâncias de referência do TSP. As instâncias foram desenhadas de modo que a solução ótima do TSPHS correspondesse à solução ótima do TSP. Para mais detalhes de como é feito este processo e, consequentemente, da escolha das posições dos hotéis extra, remete-se a explicação no artigo [19]. Neste conjunto considera-se o tempo de visita de cada cliente igual a zero. O tempo limite L para todos os percursos corresponde ao comprimento do maior percurso da solução ótima do TSP. Denote-se que a complexidade destas instâncias provém do facto de apenas um conjunto dos hotéis selecionados gerar soluções de boa qualidade e um pequeno desvio da rota ótima do TSP poderá facilmente conduzir à obtenção de percursos extra.

O quarto conjunto de instâncias (SET4) é, segundo os autores de [19], o mais difícil de resolver, não sendo conhecido o valor ótimo para nenhuma das suas instâncias. É composto por 16 instâncias, as quais foram desenvolvidas a partir das mesmas instâncias do TSP usadas no SET 3. Também neste conjunto o tempo de visita de cada cliente é igual a zero. Em cada uma das instâncias estão disponíveis 10 hotéis localizados nos clientes 1, 6, 11, 16, 21, 25, 31, 35, 41 e 45. O tempo limite L corresponde ao valor obtido dividindo o comprimento da solução ótima do TSP por 5.

O ficheiro de dados contém o número total de hotéis (hotel que deve iniciar e terminar a rota e os hotéis extra), o número de clientes, o limite de tempo L, as localizações de cada um dos hotéis e clientes e os tempos de visita de cada cliente. As localizações são dadas pelas coordenadas no plano, as quais podem ser negativas, dependendo do quadrante em que se localiza o cliente ou hotel, e podem ou não tomar valores inteiros. O ficheiro de dados para cada uma das instâncias é constituído por três colunas seguindo a estrutura apresentada na figura 5.1.

A partir das coordenadas dadas são calculadas as distâncias euclidianas entre cada par de locais (hotel-hotel, hotel-cliente, cliente-hotel, cliente-cliente). Deste modo, para determinar a distância entre dois locais i e j calcula-se: $\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$. De notar, no entanto, que esse valor é posteriormente truncado às décimas. Ou seja, se o valor da raiz fosse, por exemplo, $1,212$ a distância entre os locais seria de $1,2$, já se fosse $1,6532$ a distância a considerar seria de $1,6$.

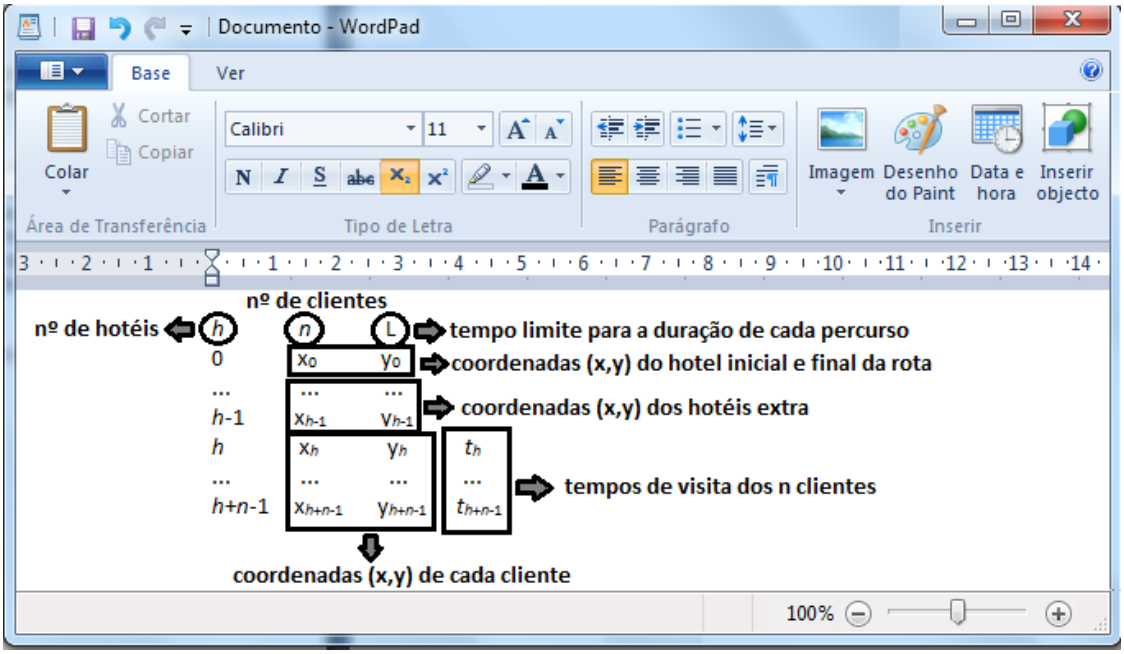

Figura 5.1: Estrutura de um ficheiro de dados de cada uma das instâncias do TSPHS

De referir que para a instância ***lin_105*** do SET4 não serão apresentados resultados pois no artigo [9] os autores verificaram que essa instância possui um cliente que não pode ser visitado em nenhum percurso por forma a satisfazer o tempo limite L. Uma vez que não existe solução admissível para a instância ***lin_105***, ela foi excluída dos testes computacionais realizados e, como tal, o SET4 passa a ser composto por 15 em vez de 16 instâncias.

5.2 Afinação das meta-heurísticas

Como foi referido anteriormente os algoritmos genéticos possuem diversos parâmetros, os quais deverão ser calibrados com vista à obtenção dos melhores resultados. Na construção dos algoritmos genéticos procederam-se a diversas experimentações, nomeadamente procederam-se aos seguintes testes:

- Calibração dos parâmetros: probabilidade de mutação (*pmut*); probabilidade de cruzamento (*pcruz*); número de descendentes (*ndesc*); número de gerações (*nger*); dimensão da população (*dimpop*);
- Escolha do cruzamento a utilizar: cruzamento uniforme *vs* cruzamento OX *vs* cruzamento CX;
- Escolha do mecanismo de substituição mais adequado: mecanismo de substituição incremental onde os x indivíduos a substituir são escolhidos aleatoriamente por

entre os indivíduos com aptidão acima da média *vs* mecanismo de substituição incremental onde os x indivíduos a substituir são os x indivíduos com maior aptidão.

Os testes realizados na calibração dos parâmetros podem ser vistos com mais detalhe em apêndice.

Com base nos testes preliminares efetuados decidiu-se:

1. parâmetros cujo valor é igual para todas as instâncias: probabilidade de mutação (*pmut*), o qual depende se estamos a utilizar o AG1 ou o AG2; probabilidade de cruzamento (*pcruz*); número de descendentes (*ndesc*); dimensão da população (*dimpop*); T, no caso da utilização do AG2.
2. parâmetros cujo valor vai variar consoante o tipo de instância: número de gerações (*nger*); M_1; M_2.

Os valores dos parâmetros utilizados serão apresentados no subcapítulo 5.3.

Após se proceder à afinação dos parâmetros nos algoritmos genéticos foi testado o efeito do procedimento de pesquisa local sugerido nesta dissertação. Para tal aplicou-se o procedimento de pesquisa local proposto às melhores soluções obtidas com AG1 e AG2 para 8 instâncias escolhidas aleatoriamente do SET1 e SET2. Na tabela 5.1 é possível observar os resultados médios obtidos com as melhores soluções no final do algoritmo genético e após aplicar o procedimento de pesquisa local às mesmas.

A partir dos resultados apresentados na tabela 5.1 é possível verificar que antes da aplicação do procedimento de pesquisa local obtiveram-se, em média, melhores resultados com o AG1. No entanto, depois da aplicação do procedimento de pesquisa local, em média, os melhores resultados foram obtidas a partir das soluções do AG2, não existindo no entanto uma diferença significativa relativamente às obtidas a partir das soluções do AG1. De um modo geral, é vantajoso aplicar o procedimento de pesquisa local na medida em que tal nos conduz a uma melhoria das soluções não só a nível do número de percursos como da duração total. Verificando-se que, em média, há uma redução de um percurso em relação ao número de percursos de cada uma das soluções obtidas no final dos algoritmos genéticos.

	antes das trocas		depois das trocas	
	duração	# percursos	duração	# percursos
AG1	4059,66	4,63	3177,99	3,63
AG2	4166,53	4,75	3159,51	3,63

Tabela 5.1: Teste ao efeito do procedimento de pesquisa local

Procedeu-se ao estudo de quais as soluções a que deveriam ser aplicadas o procedimento de pesquisa local. Para tal averiguou-se a aplicação das trocas locais a todas as soluções obtidas no final do genético. Notou-se que existia uma certa tendência para

que as melhores soluções obtidas após as trocas proviessem das soluções existentes na população final do genético com aptidão abaixo da média das soluções distintas. Deste modo, optou-se por se aplicar o procedimento de pesquisa local a três das soluções nessas circunstâncias, garantindo que o indivíduo de menor aptidão fosse necessariamente escolhido. Ou seja, a partir das soluções obtidas na população final do genético aplicou-se o procedimento de pesquisa local à solução com menor aptidão e a outras duas soluções com aptidão abaixo da média das soluções distintas obtidas. Notando que, no caso de existirem menos de quatro soluções com aptidão abaixo dessa média, o procedimento de pesquisa local foi aplicado a todas as soluções nessa situação.

De notar que para as instâncias do SET3 e do SET4, para as quais existe uma maior dificuldade em encontrar soluções admissíveis, o procedimento de perturbação não se verificou muito eficiente na medida em que muitas soluções não admissíveis eram obtidas após aplicação das perturbações. Assim, optou-se por não se aplicar o procedimento de perturbação de soluções a esses conjuntos e, como tal, por se proceder à aplicação do procedimento de pesquisa local a mais soluções. Para os SET3 e SET4 aplicou-se o procedimento de pesquisa local a todas as soluções distintas na população final do algoritmo genético.

De referir ainda que se experimentou combinar o procedimento de pesquisa local com o algoritmo genético do seguinte modo: após um dado número de gerações do algoritmo genético aplicar o procedimento de pesquisa local às soluções distintas existentes na população; prossegue-se com o algoritmo genético substituindo todos os indivíduos da população pelas permutações dos clientes de cada uma das soluções resultantes após aplicação do procedimento de pesquisa local. No entanto, nos testes realizados, tal não trouxe benefícios.

5.3 Comparação com os resultados de outros autores

No presente subcapítulo são apresentadas as melhores soluções obtidas para as instâncias criadas por Vansteenwegen *et al.* considerando cada uma das meta-heurísticas desenvolvidas para o TSPHS no âmbito desta dissertação. Os resultados obtidos serão comparados com os resultados de outros autores, nomeadamente com as soluções obtidas pelo método que produz melhores soluções até ao momento (algoritmo memético, [3]) e com a primeira abordagem apresentada para o TSPHS (heurística I2LS, [19]).

Em cada tabela encontra-se na primeira coluna o nome da instância, seguida, nas tabelas referentes ao SET1, de uma coluna com o número n de clientes. Posteriormente, encontram-se três colunas, cada uma respeitante a um método diferente para o TSPHS, nomeadamente uma coluna com a melhor solução obtida no final de 5 repetições para o MH1 ou MH2, outra com a melhor solução da heurística I2LS e a última com a do algoritmo memético. Será considerada a seguinte notação:

- duração: indica a duração total da rota na melhor solução obtida

- trips: indica o número de percursos na rota da melhor solução obtida
- CPU: indica o tempo de execução em segundos
- *gapLS*: indica o valor do *gap*, em percentagem, dado por $\left(\left(\frac{v_{AG}-vLS}{v_{LS}}\right)\times 100\right)\%$, onde v_{AG} corresponde à duração total da melhor solução obtida ao fim de 5 repetições com a respetiva meta-heurística (MH1 ou MH2) e v_{LS} corresponde à duração total da solução da heurística I2LS para a respetiva instância.
- $gapMA$: indica o valor do *gap*, em percentagem, dado por $\left(\left(\frac{v_{AG}-v_{MA}}{v_{MA}}\right)\times 100\right)\%$, onde v_{AG} corresponde à duração total da melhor solução obtida ao fim de 5 repetições com a respetiva meta-heurística (MH1 ou MH2) e v_{MA} corresponde à duração total da solução do algoritmo memético para a respetiva instância.

5.3.1 Resultados obtidos para o SET1 e para o SET2

Tendo em conta os diversos testes experimentais efetuados foram utilizados os seguintes valores para os parâmetros nos conjuntos de instâncias SET1 e SET2:

1) Parâmetros AG1:

$dimpop = 30$
$nger = 4000$
$ndesc = 14$
$pcruz = 1$
$pmut = 0,2$
$M_1 = 50$ para as instâncias com 10 a 52 clientes
$M_1 = 100$ para as instâncias com 70 a 288 clientes
$M_2 = 100000$

2) Parâmetros AG2:

$dimpop = 30$
$nger = 4000$
$T = 2$
$ndesc = 14$
$pcruz = 1$
$pmut = 0,5$
$M_1 = 50$ para as instâncias com 10 a 52 clientes
$M_1 = 100$ para as instâncias com 70 a 240 clientes
$M_1 = 150$ para as instâncias com 288 a 10002 clientes
$M_2 = 100000$

3) Parâmetros trocas locais + perturbações:

Para averiguar quais os valores a atribuir aos parâmetros necessários para o procedimento de perturbação de uma solução procederam-se a alguns testes para instâncias de diferentes dimensões tendo-se concluído que:

- $nrep = round(0,1 \times n)$
- Para $10 \leq n \leq 96$:
  $nperturb$=$round(0,1 \times n)$, θ_1=$round(0,2 \times n)$ e θ_2=$round(0,1 \times n)$
- Para $100 \leq n \leq 144$:
  $nperturb$=$round(0,1 \times n)$, θ_1=$round(0,2 \times n)$ e θ_2=$round(0,2 \times n)$
- Para $192 \leq n \leq 288$:
  $nperturb$=$round(0,2 \times n)$, θ_1=$round(0,1 \times n)$ e θ_2=$round(0,2 \times n)$

Os resultados obtidos para o SET1 com MH1 e MH2 encontram-se na tabela 5.2.

Denota-se que com a MH1 se obteve soluções com o mesmo número de percursos das soluções obtidas com a heurística I2LS, tendo-se notado que para algumas instâncias se conseguiu soluções com menor duração. Por outro lado, com a MH1 para as instâncias *r101* e *pr05* obtiveram-se rotas com um percurso adicional relativamente às rotas obtidas pelo algoritmo memético.

Com a MH2 obteve-se o mesmo número de percursos das soluções obtidas com a heurística I2LS, à exceção da solução obtida com a instância *r101* para a qual foi obtida uma solução com menos um percurso. Em relação ao número de percursos obtidos com o algoritmo memético, a MH2 apenas nos conduziu à obtenção de uma solução com um número superior para uma instância, nomeadamente para a instância *pr05*. Verifica-se ainda que para algumas instâncias se conseguiu soluções com menor duração em relação às obtidas com a heurística I2LS.

Os resultados obtidos para o SET2 encontram-se nas tabelas 5.3 a 5.6. De referir que para este conjunto as soluções obtidas com o algoritmo memético correspondem na sua maioria às soluções ótimas determinadas em [3] e assinaladas a negrito.

Na tabela 5.3 são apresentados os resultados para o grupo com 10 clientes do SET2, a partir dos quais é possível verificar que se obtiveram as soluções ótimas com as duas meta-heurísticas propostas. Observa-se que se obtiveram melhores resultados comparativamente aos obtidos com a heurística I2LS.

Na tabela 5.4 são apresentados os resultados para o grupo com 15 clientes do SET2. Para estas instâncias também se obtiveram as soluções ótimas para ambas as meta-heurísticas propostas, resultados estes iguais aos obtidos pelo algoritmo memético. Mais uma vez, os resultados obtidos pelas meta-heurísticas propostas são melhores que os obtidos pela heurística I2LS.

Na tabela 5.5 são apresentados os resultados para o grupo com 30 clientes do SET2. Para este grupo já se verificaram algumas diferenças entre os resultados obtidos pelas meta-heurísticas propostas na medida em que para algumas instâncias se obtiveram melhores soluções com MH1 e noutras com MH2. De destacar a instância *rc101* para a qual com MH1 se obteve o número de percursos da solução obtida pelo algoritmo memético ao contrário do que ocorreu com MH2, tendo-se conseguido para as restantes o mesmo número de percursos tanto com MH1 como com MH2. Verifica-se que com MH2 se obteve soluções com o mesmo número de percursos das soluções obtidas com a heurística

I2LS, mas com menor duração, e que com MH1 se obteve o mesmo número de percursos das soluções obtidas com o algoritmo memético.

Na tabela 5.6 são apresentados os resultados para o grupo com 40 clientes do SET2. De modo análogo ao que acontece com o grupo de 30 clientes, também aqui para algumas instâncias se obtiveram melhores soluções com MH1 e para outras com MH2. Neste caso, verifica-se que para a instância *r101* com o MH2 se consegue obter uma solução com o número de percursos da solução obtida pelo algoritmo memético enquanto que com o MH1 se obteve uma solução com mais um percurso.

instância	n	MH1					MH2					I2LS		MA+TS	
		duração	trips	$gapLS(\%)$	$gapMA(\%)$	CPU(s)	duração	trips	$gapLS(\%)$	$gapMA(\%)$	CPU(s)	duração	trips	duração	trips
c101	100	9667,9	9	-0,18	0,75	27,87	9696,7	9	0,11	1,05	83,74	9685,6	9	9595,6	9
r101	100	1766,5	9	-1,93	3,63	109,13	1792,6	8	-0,48	5,16	106,96	1801,3	9	1704,6	8
rc101	100	1774,0	8	2,89	5,97	115,44	1749,1	8	1,45	4,48	117,61	1724,1	8	1674,1	8
c201	100	9586,8	3	-0,14	0,28	91,84	9593,0	3	-0,07	0,35	101,54	9600,0	3	9560,0	3
r201	100	1664,5	2	-0,80	1,28	149,92	1654,3	2	-1,41	0,66	151,92	1678,0	2	1643,4	2
rc201	100	1661,7	2	-0,50	1,16	127,95	1643,8	2	-1,57	0,07	154,19	1670,0	2	1642,7	2
pr01	48	1419,0	2	-1,87	0,48	16,73	1419,0	2	-1,87	0,48	16,19	1446,0	2	1412,2	2
pr02	96	2596,8	3	1,07	2,10	42,83	2614,5	3	1,76	2,80	95,16	2569,3	3	2543,3	3
pr03	144	3589,0	4	0,14	5,09	450,03	3555,6	4	-0,80	4,11	441,35	3584,1	4	3415,1	4
pr04	192	4476,3	5	2,52	6,14	1411,63	4469,7	5	2,37	5,98	1193,05	4366,3	5	4217,4	5
pr05	240	5256,1	6	2,62	6,00	5089,16	5255,1	6	2,60	5,98	3691,37	5122,1	6	4958,7	5
pr06	288	6345,1	7	3,39	6,41	7068,27	6413,7	7	4,50	7,56	16535,22	6137,3	7	5963,1	7
pr07	72	2110,3	3	0,93	1,93	36,69	2107,9	3	0,81	1,82	35,40	2090,9	3	2070,3	3
pr08	144	3501,6	4	-0,09	3,84	296,04	3500,8	4	-0,11	3,82	440,23	3504,7	4	3372,0	4
pr09	216	4711,9	5	2,04	6,60	2126,40	4686,7	5	1,50	6,03	2178,55	4617,6	5	4420,3	5
pr10	288	6342,1	7	4,01	6,76	5046,35	6364,7	7	4,38	7,14	7485,73	6097,5	7	5940,5	7
médias			4,94	0,88	3,65	1387,89		4,88	0,82	3,59	2051,76		4,94		4,81

Tabela 5.2: Resultados obtidos para o SET1

instância	MH1					MH2					I2LS		MA+TS	
	duração	trips	$gapLS(\%)$	$gapMA(\%)$	CPU(s)	duração	trips	$gapLS(\%)$	$gapMA(\%)$	CPU(s)	duração	trips	duração	trips
c101	955,1	1	-0,03	0,00	5,50	955,1	1	-0,03	0,00	4,94	955,4	1	955,1	1
r101	272,8	2	-4,68	0,00	6,67	272,8	2	-4,68	0,00	6,03	286,2	2	272,8	2
rc101	237,5	1	0,00	0,00	5,80	237,5	1	0,00	0,00	5,83	237,5	1	237,5	1
pr01	426,6	1	0,00	0,00	5,73	426,6	1	0,00	0,00	4,94	426,6	1	426,6	1
pr02	661,9	1	0,00	0,00	5,59	661,9	1	0,00	0,00	4,98	661,9	1	661,9	1
pr03	553,3	1	-1,04	0,00	5,57	553,3	1	-1,04	0,00	5,02	559,1	1	553,3	1
pr04	476,4	1	-6,79	0,00	5,52	476,4	1	-6,79	0,00	5,03	511,1	1	476,4	1
pr05	528,9	1	-5,71	0,00	5,48	528,9	1	-5,71	0,00	5,25	560,9	1	528,9	1
pr06	597,4	1	-1,11	0,00	5,57	597,4	1	-1,11	0,00	5,00	604,1	1	597,4	1
pr07	670,2	1	-5,35	0,00	5,66	670,2	1	-5,35	0,00	5,10	708,1	1	670,2	1
pr08	573,4	1	0,00	0,00	5,53	573,4	1	0,00	0,00	5,04	573,4	1	573,4	1
pr09	645,5	1	-0,31	0,00	5,56	645,5	1	-0,31	0,00	5,02	647,5	1	645,5	1
pr10	461,5	1	0,00	0,00	5,68	461,5	1	0,00	0,00	5,01	461,5	1	461,5	1
médias		1,08	-1,92	0,00	5,68		1,08	-1,92	0,00	5,17		1,08		1,08

Tabela 5.3: Resultados obtidos para o SET2 (grupo com 10 clientes)

instância	MH1					MH2					I2LS		MA+TS	
	duração	trips	$gapLS(\%)$	$gapMA(\%)$	CPU(s)	duração	trips	$gapLS(\%)$	$gapMA(\%)$	CPU(s)	duração	trips	duração	trips
c101	1452,2	2	-0,31	0,00	6,54	1452,2	2	-0,31	0,00	5,98	1456,7	2	1452,2	2
r101	379,8	2	-2,99	0,00	7,85	379,8	2	-2,99	0,00	6,32	391,5	2	379,8	2
rc101	303,2	2	-0,95	0,00	7,00	303,2	2	-0,95	0,00	6,22	306,1	2	303,2	2
pr01	590,4	1	0,00	0,00	6,13	590,4	1	0,00	0,00	5,40	590,4	1	590,4	1
pr02	745,6	1	-0,73	0,00	6,27	745,6	1	-0,73	0,00	5,49	751,1	1	745,6	1
pr03	632,9	1	-2,50	0,00	6,16	632,9	1	-2,50	0,00	5,93	649,1	1	632,9	1
pr04	683,4	1	0,00	0,00	6,19	683,4	1	0,00	0,00	5,53	683,4	1	683,4	1
pr05	621,2	1	-5,94	0,00	6,07	621,2	1	-5,94	0,00	5,61	660,4	1	621,2	1
pr06	685,2	1	0,00	0,00	6,13	685,2	1	0,00	0,00	5,44	685,2	1	685,2	1
pr07	795,3	1	-2,12	0,00	6,29	795,3	1	-2,12	0,00	5,88	812,5	1	795,3	1
pr08	707,2	1	0,00	0,00	6,20	707,2	1	0,00	0,00	5,63	707,2	1	707,2	1
pr09	771,7	1	-0,25	0,00	6,61	771,7	1	-0,25	0,00	5,69	773,6	1	771,7	1
pr10	611,9	1	0,00	0,00	6,02	611,9	1	0,00	0,00	6,16	611,9	1	611,9	1
médias		1,23	-1,21	0,00	6,42		1,23	-1,21	0,00	5,79		1,23		1,23

Tabela 5.4: Resultados obtidos para o SET2 (grupo com 15 clientes)

instância	MH1					MH2					I2LS		MA+TS	
	duração	trips	$gapLS(\%)$	$gapMA(\%)$	CPU(s)	duração	trips	$gapLS(\%)$	$gapMA(\%)$	CPU(s)	duração	trips	duração	trips
c101	2863,3	3	-1,53	0,00	9,31	2863,6	3	-1,52	0,01	8,47	2907,8	3	2863,2	3
r101	669,1	3	-1,05	2,12	10,74	655,2	3	-3,11	0,00	9,80	676,2	3	655,2	3
rc101	714,9	3	-4,30	1,33	11,51	683,8	4	-8,46	-3,08	10,84	747,0	4	705,5	3
pr01	964,8	1	0,00	0,00	9,73	964,8	1	0,00	0,00	9,17	964,8	1	964,8	1
pr02	1088,7	2	-4,55	0,96	9,58	1083,9	2	-4,97	0,52	8,77	1140,6	2	1078,3	2
pr03	952,5	1	-0,48	0,00	11,34	952,5	1	-0,48	0,00	10,03	957,1	1	952,5	1
pr04	1091,6	2	-5,02	0,00	9,32	1091,6	2	-5,02	0,00	9,06	1149,3	2	1091,6	2
pr05	924,7	1	-1,24	0,00	10,36	924,7	1	-1,24	0,00	9,14	936,3	1	924,7	1
pr06	1063,2	2	-4,59	0,00	9,65	1063,2	2	-4,59	0,00	8,75	1114,4	2	1063,2	2
pr07	1130,4	2	-2,38	0,00	9,77	1130,4	2	-2,38	0,00	8,55	1158,0	2	1130,4	2
pr08	1006,2	2	-4,72	0,00	9,21	1006,2	2	-4,72	0,00	8,95	1056,1	2	1006,2	2
pr09	1091,4	2	-3,68	0,00	9,61	1102,4	2	-2,71	1,01	8,50	1133,1	2	1091,4	2
pr10	918,9	1	-0,88	0,00	10,17	918,9	1	-0,88	0,00	9,24	927,1	1	918,9	1
médias		1,92	-2,65	0,34	10,02		2,00	-3,08	-0,12	9,17		2,00		1,92

Tabela 5.5: Resultados obtidos para o SET2 (grupo com 30 clientes)

instância	MH1					MH2					I2LS		MA+TS	
	duração	trips	$gapLS(\%)$	$gapMA(\%)$	CPU(s)	duração	trips	$gapLS(\%)$	$gapMA(\%)$	CPU(s)	duração	trips	duração	trips
c101	3870,1	4	-2,02	0,10	12,43	3867,3	4	-2,09	0,03	11,69	3950,0	4	3866,1	4
r101	880,8	5	-1,64	2,09	14,36	880,0	4	-1,73	1,99	14,02	895,5	4	862,8	4
rc101	850,3	4	-0,11	0,00	15,88	850,3	4	-0,11	0,00	13,71	851,2	4	850,3	4
pr01	1175,0	2	-3,97	1,25	12,48	1178,2	2	-3,71	1,53	11,68	1223,6	2	1160,5	2
pr02	1365,8	2	-3,69	2,16	20,97	1374,3	2	-3,10	2,80	11,67	1418,2	2	1336,9	2
pr03	1320,6	2	-4,78	1,32	13,31	1320,6	2	-4,78	1,32	11,52	1386,9	2	1303,4	2
pr04	1259,5	2	-2,58	0,00	13,19	1259,5	2	-2,58	0,00	12,11	1292,9	2	1259,5	2
pr05	1208,3	2	0,62	0,63	12,48	1209,8	2	0,75	0,76	12,92	1200,8	2	1200,7	2
pr06	1257,6	2	-1,69	1,18	10,22	1250,2	2	-2,27	0,59	12,72	1279,2	2	1242,9	2
pr07	1407,0	2	-1,37	0,00	12,21	1407,0	2	-1,37	0,00	12,33	1426,5	2	1407,0	2
pr08	1222,2	2	-6,41	0,00	12,68	1222,2	2	-6,41	0,00	12,14	1305,9	2	1222,2	2
pr09	1290,5	2	0,27	0,49	11,74	1296,0	2	0,70	0,92	11,77	1287,0	2	1284,2	2
pr10	1213,9	2	-1,60	1,12	12,64	1200,6	2	-2,68	0,02	11,96	1233,6	2	1200,4	2
médias		2,54	-2,23	0,80	13,43		2,46	-2,26	0,77	12,33		2,46		2,46

Tabela 5.6: Resultados obtidos para o SET2 (grupo com 40 clientes)

Experimentou-se para o SET1 não aplicar o algoritmo genético, isto é, experimentou-se apenas a aplicação do procedimento de pesquisa local e do algoritmo de perturbação. Para tal geraram-se aleatoriamente 25 permutações, às quais foi aplicado o descodificador e posteriormente à solução resultante foi aplicado o procedimento de pesquisa local e o algoritmo de perturbação. Na tabela 5.7 podem-se observar as melhores soluções obtidas por este processo, denominado PLP.

instância	n	PLP					I2LS		MA+TS	
		duração	trips	$gapLS(\%)$	$gapMA(\%)$	CPU(s)	duração	trips	duração	trips
c101	100	9644,4	9	-0,43	0,51	533,36	9685,6	9	9595,6	9
r101	100	1780,7	8	-1,14	4,46	674,56	1801,3	9	1704,6	8
rc101	100	1783,9	8	3,47	6,56	762,44	1724,1	8	1674,1	8
c201	100	9598,6	3	-0,01	0,40	633,90	9600,0	3	9560,0	3
r201	100	1652,1	2	-1,54	0,53	1102,50	1678,0	2	1643,4	2
rc201	100	1650,5	2	-1,17	0,47	1010,10	1670,0	2	1642,7	2
pr01	48	1413,9	2	-2,22	0,12	38,32	1446,0	2	1412,2	2
pr02	96	2584,7	3	0,60	1,63	762,20	2569,3	3	2543,3	3
pr03	144	3516,9	4	-1,87	2,98	3169,06	3584,1	4	3415,1	4
pr04	192	4463,2	5	2,22	5,83	5891,48	4366,3	5	4217,4	5
pr05	240	5238,3	6	2,27	5,64	29657,38	5122,1	6	4958,7	5
pr06	288	6360,6	7	3,64	6,67	60624,86	6137,3	7	5963,1	7
pr07	72	2104,1	3	0,63	1,63	212,56	2090,9	3	2070,3	3
pr08	144	3487,8	4	-0,48	3,43	3113,50	3504,7	4	3372,0	4
pr09	216	4712,2	5	2,05	6,60	17428,78	4617,6	5	4420,3	5
pr10	288	6379,3	7	4,62	7,39	60211,72	6097,5	7	5940,5	7
médias			4,88	0,66	3,43	11614,17		4,94		4,81

Tabela 5.7: Resultados obtidos para o SET1 considerando apenas o procedimento de pesquisa local e o algoritmo de perturbação

A partir da análise dos resultados obtidos na tabela 5.7 verifica-se que as soluções obtidas com o PLP possuem maior duração comparativamente às soluções obtidas com o algoritmo memético. Por outro lado, as soluções apresentadas nessa tabela são, para algumas instâncias, melhores que as soluções obtidas com a heurística I2LS. Destacando a solução obtida *r101* que possui menos um percurso em relação ao número de percursos da solução obtida pela heurística I2LS.

Observa-se igualmente que, em média, se obtiveram melhores soluções comparativamente aos resultados obtidos pelas meta-heurísticas propostas no que diz respeito à duração, no entanto, a melhoria não era significativa. Observa-se ainda que os tempos de execução obtidos aplicando o PLP às soluções geradas aleatoriamente são bastante superiores aos obtidos através da aplicação das meta-heurísticas, uma vez que se consideram como soluções iniciais soluções piores não só no número de percursos como em duração.

Conclui-se que o procedimento de pesquisa local e o algoritmo de perturbação de soluções funcionam bastante bem sozinhos, tendo-se conseguido para todas as instâncias o respetivo número de percursos da solução obtida no algoritmo memético, porém denota-se que a utilização do algoritmo genético é importante no sentido de permitir que se possa aplicar o PLP a soluções cujo valor não seja tão elevado.

5.3.2 Resultados obtidos para o SET3 e para o SET4

Nos conjuntos SET3 e SET4 existe uma maior dificuldade em encontrar soluções admissíveis, sendo, nestes casos, a aplicação do algoritmo genético importante no sentido de encontrar soluções admissíveis para o problema. De referir que o procedimento de perturbação não se verificou muito eficiente na medida em que muitas soluções não admissíveis eram obtidas após aplicação das perturbações. Assim, optou-se por não se aplicar o procedimento de perturbação de soluções a estes conjuntos. Recorde-se que estes conjuntos possuem instâncias mais complexas e que um pequeno desvio da rota ótima do TSP poderá facilmente conduzir à obtenção de percursos extra.

Os valores dos parâmetros utilizados para as instâncias dos conjuntos SET3 e SET4 foram os mesmos que os utilizados para as instâncias dos conjuntos SET1 e SET2 à exceção dos parâmetros $nger$, M_1 e M_2. Ou seja, utilizaram-se os seguintes valores:

- **AG1:** $dimpop$=30, $ndesc$=14, $pcruz$=1, $pmut$=0,2
- **AG2:** $dimpop$=30, T=2, $ndesc$=14, $pcruz$=1, $pmut$=0,5

Os valores atribuídos aos parâmetros $nger$, M_1 e M_2 encontram-se apresentados na tabela 5.8.

	AG1			AG2		
	M_1	M_2	$nger$	M_1	M_2	$nger$
eil51	50	100000	16000	50	100000	20000
berlin52	50	100000	16000	50	100000	20000
sr70	50	100000	16000	50	100000	20000
eil76	50	100000	16000	50	100000	20000
pr76	50	1000000	16000	50	1000000	20000
kroa100	50	500000	16000	50	500000	20000
kroc100	50	500000	16000	50	500000	20000
krod100	50	500000	16000	50	500000	20000
rd100	50	100000	16000	50	100000	20000
eil101	50	100000	16000	50	100000	20000
lin105	50	500000	16000	50	500000	20000
ch150	100	100000	16000	150	100000	20000
tsp225	100	100000	16000	200	100000	20000
a280	150	100000	16000	200	100000	20000
pcb442	150	1000000	20000	250	1000000	30000
pr1002	150	5000000	60000	250	5000000	60000

Tabela 5.8: Valores atribuídos às penalizações e ao $nger$ para cada instância dos SET3 e SET4

Os resultados obtidos para o SET3 encontram-se nas tabelas 5.9 a 5.11.

Na tabela 5.9 são apresentados os resultados para o grupo com 3 hotéis do SET3. Neste grupo já se nota que as meta-heurísticas propostas funcionam pior do que acontecia nos conjuntos SET1 e SET2. Observa-se que existem diferenças entre os resultados obtidos com as duas meta-heurísticas, existindo casos em que para umas instâncias se

obtiveram melhores soluções com a MH1 e para outras com as melhores soluções foram obtidas com a MH2. De um modo geral, para este grupo a MH1 funciona melhor que a MH2, permitindo a obtenção de melhores soluções não só no número de percursos como na duração da rota. Para todas as instâncias, observa-se que o número de percursos das rotas obtidas pelas meta-heurísticas é superior ao número obtido com o algoritmo memético. No entanto, para algumas instâncias obteve-se um número de percursos igual ao obtido com a heurística I2LS. Existem instâncias para as quais as soluções obtidas pelas meta-heurísticas propostas são melhores do que as obtidas pela heurística I2LS e vice-versa. Note-se que para a instância *eil76* com a MH1 foi obtida uma solução com duração inferior no entanto com mais um percurso em relação à solução obtida pela heurística I2LS, deste modo a solução obtida pela heurística é melhor do que a obtida com MH1 na medida em que é preferível uma rota com menos percursos a uma rota com mais percursos mesmo que esta última possua uma menor duração.

Na tabela 5.10 são apresentados os resultados para o grupo com 5 hotéis do SET3. Mais uma vez, existem instâncias para as quais se obtiveram melhores soluções com a MH1 e noutras com a MH2. Também aqui os resultados obtidos foram para todas as instâncias piores do que os obtidos com o algoritmo memético não só em termos do número de percursos como da duração das rotas. Verifica-se ainda que, enquanto no grupo com 3 hotéis tinham-se obtido soluções melhores que as obtidas com a heurística I2LS, aqui tal não acontece com a MH1, sendo que para a MH2 se obteve uma melhor solução relativamente à heurística apenas para a instância *eil101*. Neste grupo verifica-se que as meta-heurísticas propostas têm, em média, comportamento semelhante.

No grupo com 10 hotéis do SET3 a dificuldade em encontrar soluções admissíveis foi bastante maior do que para os restantes grupos deste conjunto. Para este grupo as meta-heurísticas propostas não funcionaram bem, tendo-se obtido para a maioria das instâncias, uma população final do algoritmo genético povoada apenas por soluções não admissíveis. Observando o ficheiro de dados destas instâncias verificou-se que o tempo limite L era bastante inferior em relação aos tempos limites dos grupos anteriores. Como tal optou-se por se realizar um pré-processamento antes de aplicar a meta-heurística proposta: aplicar 5 repetições do algoritmo genético considerando um valor superior para L, nomeadamente considerou-se $L'=L+(L/2)$, e depois aplicar o procedimento de pesquisa local a todas as soluções diferentes obtidas na população final das 5 repetições. Posteriormente procedeu-se a 5 repetições da meta-heurística considerando o valor original L e a população inicial com 30 permutações dos clientes das soluções obtidas após o procedimento de pesquisa local. Optou-se por se proceder a 30000 gerações no algoritmo genético utilizado no pré-processamento para a instância *pr1002*, 10000 gerações para a instância *pcb442* e 4000 gerações para as restantes instâncias. No algoritmo genético inserido na meta-heurística procederam-se às z gerações necessárias para perfazer o *nger* definido na tabela 5.8 para a respetiva instância. Optou-se por se aplicar este processo apenas considerando a MH1 e analisar os resultados obtidos para este caso. Na tabela 5.11 podem ser observadas as melhores soluções obtidas para o grupo com 10 hotéis do SET3. Verifica-se que, à exceção da instância *berlin52*, se obteve soluções com número de percursos superior aos obtidos pela heurística I2LS e consequentemente em relação

às soluções obtidas pelo algoritmo memético. Relativamente a essa instância verifica-se também que foi o único caso em que se conseguiu uma solução melhor que a obtida pela heurística I2LS.

Os resultados obtidos para o SET4 encontram-se na tabela 5.12. Mais uma vez notam-se diferenças entre as duas meta-heurísticas propostas, obtendo-se para algumas instâncias melhores soluções com a MH1 e noutras com a MH2. Observa-se que, em média, com a MH2 se obtiveram soluções com menor número de percursos relativamente aos obtidos com MH1 e que, por outro lado, com a MH1 obtiveram-se melhores $gap's$ em relação aos obtidos com a MH2. De notar que com a MH2 se obtiveram, para as instâncias *eil51* e *ch150* melhores soluções que as obtidas com a heurística I2LS e que com a MH1 se obteve uma solução para a instância *eil51* com menor duração mas com mais um percurso do que a solução obtida com a heurística. Para ambas as meta-heurísticas as soluções obtidas tiveram qualidade inferior às obtidas com o algoritmo memético.

instância	MH1					MH2					I2LS		MA+TS	
	duração	trips	$gapLS(\%)$	$gapMA(\%)$	CPU(s)	duração	trips	$gapLS(\%)$	$gapMA(\%)$	CPU(s)	duração	trips	duração	trips
eil51	458,7	5	-3,84	7,68	73,09	460,0	5	-3,56	7,98	93,63	477	5	426	4
berlin52	8446,2	5	3,63	11,99	68,93	8065,3	5	-1,04	6,94	85,81	8150	5	7542	4
st70	774,8	5	2,76	14,79	97,71	823,5	6	9,22	22,00	123,09	754	5	675	4
eil76	608,3	6	-3,29	13,07	100,10	577,6	5	-8,17	7,36	123,98	629	5	538	4
pr76	132015,1	5	18,65	22,06	98,17	139508,4	5	25,39	28,98	132,44	111260	4	108159	4
kroa100	26460,5	6	16,02	24,33	161,78	26844,2	6	17,71	26,14	187,88	22806	5	21282	4
kroc100	27035,2	6	14,89	30,30	139,09	28024,1	7	19,09	35,06	206,01	23532	5	20749	4
krod100	26146,6	6	5,13	22,79	163,62	28019,3	6	12,66	31,58	168,21	24870	5	21294	4
rd100	8865,7	5	-0,04	12,08	141,80	9370,4	6	5,65	18,46	186,24	8869	5	7910	4
eil101	735,4	5	0,74	16,92	133,75	719,0	5	-1,51	14,31	157,37	730	5	629	4
lin105	18410,2	6	9,08	28,04	174,61	19282,6	7	14,25	34,10	236,13	16878	5	14379	4
ch150	8366,9	6	12,67	28,17	216,99	8099,7	6	9,07	24,08	308,01	7426	5	6528	4
tsp225	4646,7	6	2,01	18,66	515,80	4756,1	6	4,41	21,45	765,07	4555	5	3916	4
a280	3388,1	6	12,82	30,76	1171,17	3611,2	6	20,25	39,37	1404,8	3003	5	2591	5
pcb442	61363,7	6	9,46	20,85	2985,36	61872,6	6	10,37	21,85	5354,11	56058	5	50778	4
pr1002	382248,1	7	31,29	47,56	59818,06	352309,7	6	21,00	36,00	85825,12	291158	5	259045	4
médias		5,69	8,25	21,88	4128,75		5,81	9,68	23,48	5959,87		4,94		4,06

Tabela 5.9: Resultados obtidos para o SET3 (grupo com 3 hotéis extra)

instância	MH1					MH2					I2LS		MA+TS	
	duração	trips	$gapLS(\%)$	$gapMA(\%)$	CPU(s)	duração	trips	$gapLS(\%)$	$gapMA(\%)$	CPU(s)	duração	trips	duração	trips
eil51	491,3	7	4,75	15,33	81,86	487,0	7	3,84	14,32	101,63	469	7	426	6
berlin52	8942,3	8	9,05	18,57	82,53	8984,5	8	9,57	19,13	87,60	8200	6	7542	6
st70	853,4	8	9,83	26,43	110,64	795,5	8	2,38	17,85	143,46	777	7	675	6
eil76	643,3	8	3,59	19,57	102,92	655,2	7	5,51	21,78	134,52	621	7	538	6
pr76	134262,7	8	8,92	24,13	117,87	124907,9	8	1,33	15,49	160,22	123266	7	108159	6
kroa100	27368,9	9	20,87	28,60	168,00	29756,0	9	31,41	39,82	193,41	22644	7	21282	6
kroc100	25541	8	9,18	23,10	159,74	29320,2	10	25,33	41,31	167,18	23394	8	20749	6
krod100	30012,6	10	23,57	40,94	155,47	30404,5	10	25,19	42,78	182,64	24287	7	21294	6
rd100	10005,3	9	15,83	26,49	155,89	9537,6	8	10,41	20,58	222,47	8638	7	7910	6
eil101	776,3	8	6,78	23,42	172,58	705,8	8	-2,92	12,21	214,53	727	8	629	6
lin105	20155,5	10	25,33	40,17	164,18	19752,3	10	22,82	37,37	178,56	16082	7	14379	6
ch150	8232,2	9	10,59	26,11	316,18	8501,6	9	14,21	30,23	439,06	7444	8	6528	6
tsp225	4821,4	8	0,61	23,12	817,33	4946,4	9	3,22	26,31	911,82	4792	8	3916	6
a280	3513,4	10	12,54	32,78	1392,12	3601,0	10	15,34	36,09	1393,83	3122	8	2646	7
pcb442	64061,9	8	9,52	26,16	5785,06	62490,6	8	6,83	23,07	6466,27	58494	8	50778	6
pr1002	400642,6	11	34,07	54,23	67483,29	382413,3	10	27,97	47,21	53679,32	298827	8	259774	7
médias		8,69	12,81	28,07	4829,10		8,69	12,65	27,85	4042,28		7,38		6,13

Tabela 5.10: Resultados obtidos para o SET3 (grupo com 5 hotéis extra)

instância	MH1					I2LS		MA+TS	
	duração	trips	$gapLS(\%)$	$gapMA(\%)$	CPU(s)	duração	trips	duração	trips
eil51	523,1	14	13,72	22,79	177,51	460	12	426	10
berlin52	7945,6	9	-3,95	1,04	160,15	8272	9	7864	8
st70	983,2	17	45,66	45,66	231,35	675	10	675	10
eil76	635,0	14	4,61	18,03	274,72	607	13	538	11
pr76	137768,8	15	13,05	27,38	252,96	121861	14	108159	11
kroa100	25928,8	15	10,91	21,83	307,49	23379	12	21282	11
kroc100	26727,3	16	14,53	28,81	324,19	23337	13	20749	11
krod100	26808,7	15	13,94	25,90	328,69	23529	13	21294	11
rd100	9891,0	12	8,00	25,04	403,08	9158	11	7910	10
eil101	755,7	14	12,62	20,14	356,33	671	12	629	11
lin105	19537,9	12	20,85	35,88	355,09	16167	11	14379	10
ch150	7690,0	14	6,76	17,80	879,36	7203	12	6528	11
tsp225	4913,0	15	12,25	25,46	2684,65	4377	13	3916	11
a280	3659,3	19	17,78	40,04	4553,32	3107	14	2613	11
pcb442	66723,3	15	17,24	28,87	19185,86	56912	13	51774	11
pr1002	391816,3	19	35,35	51,25	136826,20	289477	13	259045	11
médias		14,69	15,21	27,25	10456,31		12,19		10,56

Tabela 5.11: Resultados obtidos para o SET3 (grupo com 10 hotéis extra)

instância	MH1					MH2					I2LS		MA+TS	
	duração	trips	$gapLS(\%)$	$gapMA(\%)$	CPU(s)	duração	trips	$gapLS(\%)$	$gapMA(\%)$	CPU(s)	duração	trips	duração	trips
eil51	451,3	7	-5,78	5,20	73,00	450,5	6	-5,95	5,01	93,97	479	6	429	6
berlin52	9185,5	7	4,11	6,29	72,51	9181,1	7	4,06	6,24	94,33	8823	7	8642	7
st70	807,7	7	8,42	11,72	104,13	795,7	7	6,81	10,06	136,93	745	7	723	6
eil76	608,5	7	2,27	11,04	103,71	613,5	6	3,11	11,95	136,75	595	6	548	6
pr76	138864,4	8	6,99	17,62	107,24	139549,7	7	7,52	18,20	133,68	129789	7	118061	6
kroa100	25542,9	7	11,89	14,32	166,23	28159,9	7	23,36	26,03	187,23	22828	7	22343	6
kroc100	27948,1	8	17,71	33,51	142,72	25488,8	8	7,35	21,76	167,61	23744	7	20933	6
krod100	27310,3	7	9,66	26,06	171,57	26723,8	8	7,31	23,36	205,95	24904	6	21664	6
rd100	9867,2	7	12,52	19,69	137,24	10569,2	7	20,53	28,20	180,96	8769	6	8244	6
eil101	701,1	7	1,17	10,58	140,92	720,9	7	4,03	13,71	168,30	693	6	634	6
ch150	7914,2	7	3,06	19,06	257,24	7616,6	7	-0,81	14,59	352,59	7679	7	6647	6
tsp225	5081,8	8	5,45	11,17	785,86	5110,1	7	6,04	11,79	740,46	4819	7	4571	6
a280	3478,2	9	11,37	31,45	1072,89	3482,6	8	11,51	31,62	1271,82	3123	7	2646	6
pcb442	66232,9	8	3,78	21,89	3286,00	69743,2	8	9,28	28,35	5488,42	63822	7	54339	6
pr1002	442749,4	11	34,05	51,27	75740,50	440652,6	11	33,42	50,55	87680,55	330282	7	292690	7
médias		7,67	8,45	19,39	5490,78		7,40	9,17	20,09	6469,30		6,67		6,13

Tabela 5.12: Resultados obtidos para o SET4

5.3.3 Síntese dos resultados obtidos

Com o intuito de sintetizar os resultados obtidos e de comparar as meta-heurísticas propostas serão apresentados e analisados os resultados médios obtidos para cada conjunto.

Na tabela 5.13 podem ser observadas as médias obtidas por conjunto considerando as melhores soluções obtidas para cada uma das instâncias e para cada uma das meta-heurísticas.

instância	MH1				MH2			
	trips	$gapLS(\%)$	$gapMA(\%)$	CPU(s)	trips	$gapLS(\%)$	$gapMA(\%)$	CPU(s)
SET1	4,94	0,88	3,65	1387,89	4,88	0,82	3,59	2051,76
SET2 (10 clientes)	1,08	-1,92	0,00	5,68	1,08	-1,92	0,00	5,17
SET2 (15 clientes)	1,23	-1,21	0,00	6,42	1,23	-1,21	0,00	5,79
SET2 (30 clientes)	1,92	-2,65	0,34	10,02	2,00	-3,08	-0,12	9,17
SET2 (40 clientes)	2,54	-2,23	0,80	13,43	2,46	-2,26	0,77	12,33
SET3 (3 hotéis)	5,69	8,25	21,88	4128,75	5,81	9,68	23,48	5959,87
SET3 (5 hotéis)	8,69	12,81	28,07	4829,10	8,69	12,65	27,85	4042,28
SET3 (10 hotéis)	14,69	15,21	27,25	10456,31				
SET4	7,67	8,45	19,39	5490,78	7,40	9,17	20,09	6469,30

Tabela 5.13: Resultados médios obtidos por conjunto

A experiência computacional efetuada, resumida na tabela 5.13, sugere que:

- Relativamente ao SET1 os resultados obtidos por ambas as meta-heurísticas foram, em média, de qualidade inferior aos obtidos pela heurística I2LS e pelo algoritmo memético, no entanto, a metodologia proposta nesta dissertação revelou-se competitiva com os resultados obtidos em [19] e [3]. Notou-se que para o SET1, em média, se obtiveram melhores soluções, quer em termos do número de percursos quer em relação à duração, com a MH2. A meta-heurística MH1 conduziu-nos, em média, à obtenção de tempos computacionais menores.

- Relativamente ao SET2 verifica-se que os resultados obtidos por ambas as meta-heurísticas foram, em média, competitivos em relação aos obtidos pela heurística I2LS e pelo algoritmo memético, tendo superado os resultados obtidos, em termos da qualidade das soluções, pela heurística I2LS. Notou-se que para o SET2, em média, se obtiveram as mesmas soluções para os grupos com 10 e 15 clientes com ambas as meta-heurísticas propostas. Para o grupo com 30 clientes obtiveram-se, em média, melhores soluções com a MH1, já para o grupo com 40 clientes foi com a MH2. De um modo geral, a meta-heurística MH2 conduziu-nos, em média, à obtenção de tempos computacionais inferiores.

- A maior complexidade das instâncias dos conjuntos SET3 e SET4 reflete-se na qualidade das soluções obtidas. Verifica-se que, no SET3, com o aumento do número de hotéis disponíveis a complexidade aumenta e consequentemente a qualidade das soluções obtidas piora. No grupo com 3 hotéis do SET3 obtiveram-se, em média, melhores soluções com MH1, no grupo com 5 hotéis foi com MH2 mas com uma diferença pouco significativa e no SET4, em média, a MH2 conduziu-nos a soluções

com menos percursos e a MH1 com menores durações. De um modo geral, a metaheurística MH1 conduziu-nos, em média, à obtenção de tempos computacionais inferiores.

- Muitos autores consideram o mecanismo de seleção torneio preferível em relação à roleta por ser, em geral, mais robusto no que diz respeito a evitar a falta de diversidade das soluções. No entanto, verificou-se que tal não aconteceu nos testes realizados nesta dissertação, isto é, foi com a roleta que se obteve mais de diversidade nos algoritmos genéticos.

Capítulo 6

Conclusões

No presente capítulo são apresentadas as principais conclusões do estudo efetuado ao TSPHS. Adicionalmente são referidas algumas sugestões de trabalho futuro de forma a complementar o trabalho realizado no âmbito desta dissertação.

6.1 Principais conclusões

O problema do caixeiro viajante com seleção de hotéis (TSPHS) é um problema de otimização NP-difícil, o qual tem inúmeras aplicações reais. Na presente dissertação pretendeu-se estudar uma nova metodologia para resolver o TSPHS. Mais concretamente foram estudadas duas meta-heurísticas, as quais consistem num algoritmo genético combinado com um procedimento de pesquisa local, baseado em oito operadores de destruição e posterior reconstrução da rota, e um algoritmo de perturbação de soluções. As meta-heurísticas diferem no mecanismo de seleção utilizado no algoritmo genético, nomeadamente na MH1 utilizou-se a roleta e na MH2 o torneio.

Os resultados obtidos foram comparados com os resultados de outros autores, nomeadamente compararam-se os resultados com os obtidos pela heurística I2LS [19] e pelo algoritmo memético [3]. Analisando os resultados finais concluiu-se que as meta-heurísticas propostas funcionaram melhor para os conjuntos SET1 e SET2 que para os conjuntos SET3 e SeT4.

No SET1 os resultados obtidos foram competitivos em relação aos da heurística I2LS, obtendo-se para alguns casos melhores soluções. No entanto, o mesmo não se passou em relação às soluções obtidas pelo algoritmo memético, o qual apresentou melhores soluções para todas as instâncias. A média dos $gap's$ das soluções obtidas em relação às do algoritmo memético é inferior a 4%. Os resultados obtidos para o SET2 revelaram-se competitivos em relação ao algoritmo memético tendo sido obtidas para diversas instâncias as mesmas soluções (na maioria dos casos soluções ótimas) e para uma instância uma solução melhor que a solução obtida pelo algoritmo memético com tempos computacionais relativamente baixos. Para este conjunto as soluções obtidas nesta dissertação foram melhores do que as obtidas pela heurística I2LS.

Nos testes realizados constatou-se haver uma maior dificuldade em resolver as instâncias

dos conjuntos SET3 e SET4, concluindo-se que a escolha dos hotéis tem um grande impacto na qualidade das soluções finais e que existe uma certa dificuldade em determinar uma sequência de hotéis de boa qualidade. Para muitas instâncias o número de percursos obtido foi bastante superior ao obtido com o algoritmo memético.

É possível concluir que não existiu uma meta-heurística que superasse a outra. Contudo, para os conjuntos SET1 e SET2 é a MH2 que nos conduz a melhores resultados, essencialmente no que diz respeito ao número de percursos das soluções.

Concluiu-se também que o procedimento de pesquisa local revelou-se fundamental para a qualidade dos resultados obtidos. Não tirando mérito ao algoritmo genético, o qual também se verificou ser bastante importante principalmente no sentido da redução do número de percursos das soluções e na obtenção de soluções admissíveis, em particular nos conjuntos SET3 e SET4.

6.2 Desenvolvimentos futuros

Diversos aspetos podem ser melhorados com o intuito de tornar os algoritmos mais eficientes e eficazes, algo que é importante para a resolução de problemas do caixeiro viajante com seleção de hotéis correspondentes a situações da vida real.

Como foi referido uma dada permutação pode codificar soluções melhores que a obtida aplicando o descodificador, o qual tem um carácter *greedy* tentando visitar o máximo de clientes por percurso. Deste modo, poderá ter interesse investir num estudo de um descodificador exato, no qual se poderia resolver o modelo matemático considerando a ordem de visita dos clientes fixa, isto é, a ordem de precedências das visitas dada pela ordem dos clientes na permutação a que é aplicada o descodificador.

Por outro lado, também foi constatado a existência de falta de diversidade nas populações dos algoritmos genéticos propostos. Desta forma, o desenvolvimento de um procedimento que permitisse a existência de uma população mais variada, sem comprometer a eficiência do algoritmo, poderia ser vantajoso. Uma sugestão seria a possibilidade de variar a probabilidade de mutação ao longo dos algoritmos genéticos de modo inversamente proporcional à diversidade na população, no sentido de prevenir ou de pelo menos atenuar a possível existência de convergência prematura.

Como trabalho futuro sugere-se ainda o estudo da utilização de uma codificação diferente, nomeadamente uma codificação que tenha em consideração os hotéis, em especial para os conjuntos SET3 e SET4. Poderia ter interesse estudar um algoritmo genético que codificasse as soluções através de chaves aleatórias ou até mesmo a utilização de um cromossoma constituído pela permutação dos clientes seguida de uma lista que codificasse a ordem dos hotéis na solução. Neste último caso, seriam considerados os dois níveis de decisão: a seleção dos hotéis e a ordem dos clientes a visitar. De notar que há que ter atenção ao facto de não se saber *a priori* o número de percursos da solução e consequentemente o número de hotéis da mesma.

Por fim, é de referir que no *software* MATLAB é feita a compilação e execução do código simultaneamente, pelo que se poderiam tentar diminuir os tempos de execução

obtidos implementando as meta-heurísticas numa outra linguagem, tornando deste modo os algoritmos mais eficientes.

Referências

[1] Baltz, A., Ouali, M. E., Jäger, G., Sauerland, V., Srivastav, A. (2015). Exact and heuristic algorithms for the Travelling Salesman Problem with Multiple Time Windows and Hotel Selection. *Journal of the Operational Research Society*, vol. 66, pp. 615-626.

[2] Bektas, T. (2006). The multiple traveling salesman problem: An overview of formulations and solution procedures. *Omega*, vol. 34(3), pp. 209-219.

[3] Castro, M., Sörensen, K., Vansteenwegen, P., Goos, P. (2013). A memetic algorithm for the travelling salesperson problem with hotel selection. *Computers & Operations Research*, vol. 40, pp. 1716-1728.

[4] Castro, M., Sörensen, K., Vansteenwegen, P., Goos, P. (2014). A fast metaheuristic for the travelling salesperson problem with hotel selection. *4OR A Quarterly Journal of Operations Research*.

[5] Chisman, J. A. (1975). The clustered traveling salesman problem. *Computers & Operations Research*, vol. 2, nº 2, pp. 115-119.

[6] Cordeau, J. F., Gendreau, M., Laporte, G. (1997). A tabu search heuristic for periodic and multi-depot vehicle routing problems. *Networks*, vol. 30(2), pp. 105-119.

[7] Cordeau J. F., Laporte G., Mercier A. (2001). A unified tabu search heuristic for vehicle routing problems with time windows. *Journal of the Operational Research Society*, vol. 52, pp. 928-936.

[8] Divsalar, A., Vansteenwegen, P., Cattrysse, D. (2013). A variable neighborhood search method for the orienteering problem with hotel selection. *Journal of the Operational Research Society*, vol. 145, pp. 150-160.

[9] Divsalar, A., Vansteenwegen, P., Sörensen, K., Cattrysse, D. (2014). A memetic algorithm for the orienteering problem with hotel selection. *European Journal of Operational Research*, vol. 237, pp. 29-49.

[10] Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization & Machine Learning*. Addison-Wesley: New York.

[11] Lin, S., Kernighan, B. W. (1973). An effective heuristic algorithm for the travelling-salesman problem. *Operations Research*, vol. 21, no. 2, pp. 498-516.

[12] Nagy, G., Salhi, S. (2007). Location-routing: issues, models and methods. *European Journal of Operational Research*, vol. 177(2), pp. 649-672.

[13] Polacek, M., harti, R.F., Doerner, K., Reimann, M. (2004). A variable neighborhood search for the multi depot vehicle routing problem with time windows. *Journal of Heuristics*, vol. 10(6), pp. 613-627.

[14] Prins, C. (2004). A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers & Operations Research*, vol. 31(12), pp. 1985-20021.

[15] Reeves, C. (1993). *Modern Heuristic Techniques for Combinatorial Problems*. Genetic algorithms, pp 151-196. Oxford:Blackwell.

[16] Solomon, M.M. (1987). Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints. *Operations Research*, vol.35, pp.254-265.

[17] Sousa, M.M., Ochi, L.S., Coelho, I.M., Gonçalves, L.B. (2015). *A Variable Neighborhood Search Heuristic for the Traveling Salesman Problem with Hotel Selection*. XLI Latin American Computing Conference.

[18] Toth, P., Vigo, D. (2002). *The vehicle routing problem*. Discrete Mathematics and Applications, vol. 9. Society for Industrial and Applied Mathematics.

[19] Vansteenwegen, P., Souffriau, A., Sörensen, K. (2012). The travelling salesperson problem with hotel selection. *Journal of the Operational Research Society*, vol. 63, pp. 207-217.

[20] http://antor.uantwerpen.be/instances-in-the-paper-a-memetic-algorithm-for-the-travelling-salesperson-problem-with-hotel-selection/

[21] http://www.frota-azul.pt/downloads/Faqs_Descanso.pdf

Apêndice A

Otimização dos parâmetros

Na presente secção serão apresentados os testes realizados aos parâmetros. De referir que se optou por se realizarem estes testes aos conjuntos SET1 e SET2 por se acreditar serem mais representativos de situações da vida real enquanto os conjuntos SET3 e SET4 foram construídos de modo a que pequenas variações na sequência ótima de visitas aos clientes tendem a criar poderá facilmente conduzir à obtenção de percursos extra (soluções de má qualidade).

Nos testes preliminares começou-se por comparar os resultados obtidos com os mecanismos de seleção roleta e torneio. Considerou-se o mecanismo de substituição incremental 1, $dimpop$=30, M_1=50, M_2=100000, $pmut$=0.05, $pcruz$=0.95, $nger$=2000 e $ndesc$=2. Para proceder a este teste selecionou-se um conjunto aleatório de 6 instâncias do SET1 e outras 6 instâncias do SET2. Devido à aleatoriedade associada ao algoritmo genético procederam-se a 5 repetições, ou seja, para cada instância aplicou-se 5 vezes o algoritmo genético. Na tabela A.1 apresentam-se os resultados médios obtidos para esta experimentação considerando as melhores soluções conseguidas após as 5 repetições para cada uma das instâncias testadas.

	SET1		SET2	
	roleta	torneio	roleta	torneio
duração	5772,52	6411,00	2103,62	2352,15
# percursos	10,67	11,83	4,50	4,83
tempos (s)	193,94	277,36	34,79	33,45

Tabela A.1: Comparação mecanismo de seleção roleta *vs* torneio

A partir dos resultados apresentados na tabela A.1 observa-se que o mecanismo de seleção que produz melhores resultados é a roleta. No entanto, muitos autores consideram o mecanismo de seleção torneio preferível em relação à roleta, pelo que se optou por prosseguir os testes com estes dois mecanismos independentemente uma vez que ainda não tínhamos os parâmetros otimizados.

Posteriormente, utilizando os mesmos parâmetros, comparou-se a utilização do cruzamento uniforme *vs* cruzamento OX tanto com o mecanismo de seleção torneio como

com a roleta. Nas tabelas A.2 e A.3 podem-se observar os resultados médios obtidos para esta experimentação considerando as melhores soluções conseguidas após as 5 repetições para cada uma das instâncias testadas.

	roleta (AG1)			torneio (AG2)		
cruzamento	duração	# percursos	tempos (s)	duração	# percursos	tempos (s)
uniforme	5772,52	10,67	193,94	6411,00	11,83	277,36
OX	5712,82	10,33	20,78	6551,15	12,17	20,09

Tabela A.2: Comparação cruzamento uniforme *vs* cruzamento OX na amostra do SET1

	roleta (AG1)			torneio (AG2)		
cruzamento	duração	# percursos	tempos (s)	duração	# percursos	tempos (s)
uniforme	2103,62	4,50	34,79	2352,15	4,83	33,45
OX	2054,20	4,17	17,80	2367,03	5,00	17,10

Tabela A.3: Comparação cruzamento uniforme *vs* cruzamento OX na amostra do SET2

Atendendo aos resultados apresentados nas tabelas A.2 e A.3 concluiu-se o seguinte:
– utilizar o cruzamento OX com a roleta permitiu a obtenção de melhores soluções em média não só em termos do número de percursos como da sua duração total tanto para as instâncias testadas do SET1 como do SET2; por outro lado, também permitiu uma redução bastante significativa nos tempos computacionais comparativamente aos obtidos com o cruzamento uniforme;
– no mecanismo de seleção torneio denotou-se que os resultados obtidos com o cruzamento OX eram piores que os obtidos com o cruzamento uniforme, no entanto, observa-se que essa diferença não é de todo significativa e que utilizar o cruzamento OX permite a obtenção de tempos computacionais médios bastante inferiores aos obtidos pelo cruzamento uniforme.

Deste modo optou-se por se utilizar o cruzamento OX em detrimento do cruzamento uniforme tanto para a roleta como para o torneio.

Posteriormente comparou-se a utilização do mecanismo de substituição incremental 1 *vs* mecanismo de substituição incremental 2. Para tal usou-se os valores para os parâmetros acima atribuídos e o cruzamento OX. Os resultados médios obtidos para esta experimentação considerando as melhores soluções conseguidas após as 5 repetições para cada uma das instâncias do SET1 e SET2 selecionadas anteriormente podem ser observados nas tabelas A.4 e A.5.

	roleta (AG1)			torneio (AG2)		
substituição	duração	# percursos	tempos (s)	duração	# percursos	tempos (s)
incremental 1	5712,82	10,33	20,78	6551,15	12,17	20,09
incremental 2	5668,00	10,17	7,86	6483,62	12,33	7,64

Tabela A.4: Comparação dos mecanismos de substituição incremental na amostra do SET1

	roleta (AG1)			torneio (AG2)		
substituição	duração	# percursos	tempos (s)	duração	# percursos	tempos (s)
incremental 1	2054,20	4,17	17,80	2367,03	5,00	17,10
incremental 2	2043,47	4,33	5,13	2332,28	4,83	4,69

Tabela A.5: Comparação dos mecanismos de substituição incremental na amostra do SET2

Atendendo aos resultados apresentados nas tabelas A.4 e A.5 verificou-se quer em relação aos resultados obtidos para o SET1 como para o SET2 e quer utilizando a roleta como o torneio que são obtidas em média soluções com menores durações com o mecanismo de substituição incremental 2 comparativamente às obtidas com o mecanismo de substituição incremental 1. Verifica-se que nuns casos é com o mecanismo de substituição incremental 1 que se obtém em média soluções com um menor número de percursos e noutros casos é com o mecanismo de substituição incremental 2, no entanto, quer num caso quer no outro as diferenças não são muito significativas. Por outro lado, observa-se que utilizar o mecanismo de substituição incremental 2 permite, em média, a obtenção de tempos computacionais bastante inferiores aos obtidos pelo mecanismo de substituição incremental 1. Tendo em conta estes resultados optou-se por se utilizar o mecanismo de substituição incremental 2 em detrimento do mecanismo de substituição incremental 1 tanto para a roleta como para o torneio.

Outros testes permitiram concluir que utilizar $pcruz$=1 e $pmut$=0,1 em vez de $pcruz$=0,95 e $pmut$=0,05 produz melhores resultados. Tendo-se testado posteriormente $pmut$=0,1 *vs* $pmut$=0,2 e $ndesc$=2 *vs* $ndesc$=6 *vs* $ndesc$=10, tendo-se optado por ficar com $pmut$=0,2 e $ndesc$=10.

Testou-se ainda o parâmetro *dimpop* a utilizar tendo-se optado por $dimpop$=30.

Procederam-se ainda a outros testes. De seguida apresenta-se a otimização dos parâmetros realizada de um modo mais pormenorizado para cada um dos algoritmos genéticos (AG1 cujo mecanismo de seleção é a roleta e AG2 cujo mecanismo de seleção é o torneio).

A.1 Algoritmo genético 1

Considerou-se o mecanismo de seleção roleta, o mecanismo de substituição incremental 2, $dimpop$=30, M_1=50, M_2=100000, $nger$=2000, $pmut$=0,2 e $ndesc$=10.

Teste ao número de descendentes e à probabilidade de mutação:

Testou-se o aumento da probabilidade de mutação e do número de descendentes, utilizando-se para tal um conjunto de 6 instâncias escolhidas do SET1 e do SET2. Para esta experimentação considerou-se a melhor solução conseguida e a média das soluções obtidas após as 5 repetições para cada uma das instâncias selecionadas. Na tabela A.6 apresentam-se a média desses resultados.

	ndesc	10			14		
pmut		duração	# percursos	tempos (s)	duração	# percursos	tempos (s)
0,2	melhores	2371,88	4,17	23,50	2287,85	4,00	32,82
	médias	2426,26	4,30		2381,26	4,23	
0,3	melhores	2324,35	4,00	24,58	2337,25	4,17	34,95
	médias	2427,42	4,27		2396,78	4,27	

Tabela A.6: Teste ao número de descendentes (*ndesc*=10 e *ndesc*=14) e à probabilidade de mutação (*pmut*=0,2 e *pmut*=0,3) no AG1

Como pode ser observado na tabela A.6 o aumento no número de descendentes conduz a um aumento no tempo computacional o que era algo de esperar. Optou-se pela combinação *pmut*=0,2 e *ndesc*=14, a qual, apesar de ter associado tempos computacionais superiores, foi a que nos conduziu a melhores resultados em média.

Teste ao número de gerações:

Até aqui tem sido considerado *nger*=2000. De seguida apresenta-se o estudo realizado ao número de gerações, *nger*, a utilizar. Comparou-se inicialmente *nger*=2000 *vs* *nger*=4000 *vs* *nger*=6000, considerando a combinação optada anteriormente: *pmut*=0.2 e *ndesc*=14. Para estes testes utilizaram-se as 6 instâncias do teste anterior. Considerou-se a melhor solução conseguida e a média das soluções obtidas após as 5 repetições para cada uma das instâncias selecionadas. Na tabela A.7 apresentam-se a média desses resultados.

nger		duração	# percursos	tempos (s)
2000	melhores	2287,85	4,00	32,82
	médias	2381,26	4,23	
4000	melhores	2131,88	3,83	64,70
	médias	2276,00	4,03	
6000	melhores	2219,62	4,00	98,06
	médias	2291,99	4,03	

Tabela A.7: Teste ao número de gerações no AG1: comparação dos resultados obtidos com 2000, 4000 e 6000 gerações

Tal como era de esperar o tempo computacional aumenta com o aumento do número de gerações, *nger*. Observando-se que os resultados melhoram, em média, ao passar de *nger*=2000 para *nger*=4000, porém pioram ao passar de *nger*=4000 para *nger*=6000. Deste modo, experimentou-se comparar *nger*=4000 *vs* *nger*=5000. Os resultados obtidos podem ser observados na tabela A.8. Optou-se por *nger*=4000 na medida em que permite obter em média melhores resultados, apesar da diferença não ser muito significativa, e um tempo computacional inferior.

nger		duração	# percursos	tempos (s)
4000	melhores	2131,88	3,83	64,70
	médias	2276,00	4,03	
5000	melhores	2194,92	3,83	100,92
	médias	2278,46	4,00	

Tabela A.8: Teste ao número de gerações no AG1: comparação dos resultados obtidos com 4000 e 5000 gerações

De referir que os valores optados para os diferentes parâmetros foram experimentados nalgumas instâncias do SET3 e SET4 escolhidas aleatoriamente, tendo-se denotado que era necessário proceder à continuação da afinação desses parâmetros. Testou-se qual o valor a utilizar para a probabilidade de mutação e para o número de gerações. Os resultados obtidos permitiram verificar que $pmut$=0,2 continuava a ser a mais adequada, no entanto, que o número de gerações necessitava de ser aumentado. Outros testes foram feitos, nomeadamente aos valores a atribuir a M_1 e M_2. Os valores escolhidos poderão ser observados no subcapítulo 5.3.

A.2 Algoritmo genético 2

Considerou-se o mecanismo de seleção torneio, o mecanismo de substituição incremental 2, $dimpop$=30, M_1=50, M_2=100000, $nger$=2000, $pmut$=0,2 e $ndesc$=10.

Testou-se inicialmente o aumento da probabilidade de mutação: $pmut$=0,2 *vs* $pmut$=0,3. Através deste teste denotou-se que $pmut$=0,3 produzia melhores resultados que $pmut$=0,2, denotando-se que com o mecanismo de seleção torneio era preciso ainda mais diversidade do que com a roleta. Após se proceder a uma análise das primeiras populações iniciais verificou-se que, como se cruzam permutações muito semelhantes e devido ao próprio funcionamento do cruzamento OX, se geram muitas réplicas. Posto isto, antes de se testar com mais detalhe o efeito do aumento da probabilidade de mutação e do número de descendentes, procedeu-se ao estudo de um outro cruzamento: o cruzamento CX.

Teste ao operador cruzamento: cruzamento OX *vs* cruzamento CX:

Inicialmente procedeu-se ao estudo dos valores mais adequados para os parâmetros *pmut* e *ndesc* utilizando o cruzamento CX, tendo-se estudado $pmut$=0,2 e $pmut$=0,3 com $ndesc$=10 e $ndesc$=14. Considerou-se para tal um conjunto de 6 instâncias escolhidas aleatoriamente do SET1 e do SET2. Considerou-se a melhor solução conseguida e a média das soluções obtidas após as 5 repetições para cada uma das instâncias selecionadas. Na tabela A.9 apresentam-se a média desses resultados.

A partir dos resultados apresentados na tabela A.9 denota-se que, com o aumento do número de descendentes e com o aumento da probabilidade de mutação, o tempo computacional aumenta, no entanto, o aumento é pouco significativo. Optou-se por se escolher a combinação $pmut$=0,3 e $ndesc$=14, a qual conduziu a melhores resultados em média.

ndesc		10			14		
pmut		duração	# percursos	tempos (s)	duração	# percursos	tempos (s)
0,2	melhores	2541,32	4,50	15,91	2510,37	4,67	21,90
	médias	2656,13	4,87		2614,07	4,83	
0,3	melhores	2495,02	4,67	17,16	2444,38	4,33	23,11
	médias	2625,00	4,90		2599,33	4,73	

Tabela A.9: Teste ao número de descendentes e à probabilidade de mutação considerando o cruzamento CX no AG2

Ora, com o cruzamento CX optou-se por se utilizar a combinação *pmut*=0,3 e *ndesc*=14 e até ao momento com o cruzamento OX tínhamos optado pela combinação *pmut*=0,3 e *ndesc*=10. Na tabela A.10 pode-se observar os resultados médios obtidos para cada um desses casos, verificando-se que o cruzamento OX conduz a melhores resultados, em média, do que o cruzamento CX.

Note-se que ao analisar as primeiras populações obtidas com o cruzamento CX verificou-se que continuavam a existir muitas réplicas, mais do que o verificado com o cruzamento OX. De referir ainda que se optou por não se estudar o efeito do cruzamento PMX uma vez que à partida o PMX produziria um efeito semelhante ao obtido com o cruzamento OX. Tal aconteceria na medida em que quando se analisou as primeiras populações obtidas com o cruzamento OX tinha-se verificado que existiam muitas permutações semelhantes graças à secção que é "transmitida" no cruzamento, que também é algo característico do cruzamento PMX.

	cruzamento CX *pmut*=0,3 e *ndesc*=14			cruzamento OX *pmut*=0,3 e *ndesc*=10		
	duração	# percursos	tempos (s)	duração	# percursos	tempos (s)
melhores	2444,38	4,33	23,11	2391,63	4,17	20,34
médias	2599,33	4,73		2531,77	4,60	

Tabela A.10: Teste ao número de descendentes e à probabilidade de mutação considerando o cruzamento CX no AG2

Teste ao número de descendentes e à probabilidade de mutação:

Considerando o cruzamento OX, testou-se o aumento da probabilidade de mutação e do número de descendentes, utilizando-se para tal um conjunto de 8 instâncias escolhidas aleatoriamente do SET1 e do SET2. Considerou-se a melhor solução conseguida e a média das soluções obtidas após as 5 repetições para cada uma das instâncias selecionadas. Na tabela A.11 apresentam-se a média desses resultados.

Como pode ser observado na figura A.11 o aumento no número de descendentes, *ndesc*, conduz a um aumento no tempo computacional, o que era algo de esperar. O aumento da probabilidade de mutação, *pmut*, também conduz a um aumento no tempo computacional. Optou-se por se escolher a combinação *pmut*=0,5 e *ndesc*=14, a qual produziu melhores resultados em média.

	ndesc	10			14		
pmut		duração	# percursos	tempos (s)	duração	# percursos	tempos (s)
0,3	melhores	4581,24	5,13	25,09	4472,14	5,13	34,56
	médias	4723,66	5,50		4628,33	5,30	
0,5	melhores	4496,28	5,13	26,48	4310,79	4,88	36,01
	médias	4666,14	5,45		4495,68	5,18	
0,7	melhores	4408,18	4,88	28,60	4353,74	5,13	39,83
	médias	4598,87	5,28		4516,11	5,23	

Tabela A.11: Teste ao número de descendentes e à probabilidade de mutação considerando o cruzamento OX no AG2

Teste ao número de gerações:

Posteriormente procedeu-se ao estudo do aumento do número de gerações. Até aqui eram consideradas 2000 gerações. Testou-se então $nger$=2000 *vs* $nger$=4000, utilizando-se para tal as 8 instâncias anteriores. Considerou-se a melhor solução conseguida e a média das soluções obtidas após as 5 repetições para cada uma das instâncias selecionadas. Na tabela A.12 apresentam-se a média desses resultados.

nger		duração	# percursos	tempos (s)
2000	melhores	4310,79	4,88	36,01
	médias	4495,68	5,18	
4000	melhores	4166,53	4,75	75,40
	médias	4370,74	5,05	

Tabela A.12: Teste ao número de gerações no AG2: comparação dos resultados obtidos com 2000 e 4000 gerações

A partir dos resultados apresentados na tabela A.12 é possível concluir que com o aumento do número de gerações os tempos computacionais, em média, também aumentam. Optou-se por se escolher $nger$=4000 pois, apesar de conduzir a tempos computacionais superiores aos obtidos por $nger$=2000, permite obter melhores resultados. De notar que, através de uma análise mais pormenorizada aos resultados, foi possível averiguar que para uma das instâncias obteve-se com $nger$=2000 uma solução bastante boa (*outlier*), no entanto, para as restantes instâncias testadas obteve-se melhores soluções com $nger$=4000.

Denote-se que se testou valores superiores para o número de gerações os quais conduziram, em média, à obtenção de melhores resultados, no entanto, a melhoria verificada foi pouco significativa e, por outro lado, os tempos computacionais aumentaram significativamente.

De referir que os valores optados para os diferentes parâmetros foram experimentados nalgumas instâncias do SET3 e SET4 escolhidas aleatoriamente, tendo-se denotado que era necessário proceder à continuação da afinação desses parâmetros. Testou-se qual o

valor a utilizar para a probabilidade de mutação e para o número de gerações. Os resultados obtidos permitiram verificar que $pmut$=0,5 continuava a ser a mais adequada, no entanto, que o número de gerações necessitava de ser aumentado. Outros testes foram feitos, nomeadamente aos valores a atribuir a M_1 e M_2. Os valores escolhidos poderão ser observados no subcapítulo 5.3.

Apêndice B

Resultados finais

Nesta secção são apresentadas, para cada conjunto e para cada uma das meta-heurísticas propostas, a melhor solução e a média das melhores soluções obtidas ao fim de 5 repetições.

		melhor solução					média das melhores				
instância	n	duração	trips	$gapLS(\%)$	$gapMA(\%)$	CPU(s)	duração	trips	$gapLS(\%)$	$gapMA(\%)$	CPU(s)
c101	100	9667,9	9	-0,18	0,75	27,87	9739,5	9,0	0,56	1,50	80,79
r101	100	1766,5	9	-1,93	3,63	109,13	1778,0	9,0	-1,29	4,31	109,04
rc101	100	1774,0	8	2,89	5,97	115,44	1814,5	8,0	5,24	8,39	115,82
c201	100	9586,8	3	-0,14	0,28	91,84	9612,3	3,0	0,13	0,55	90,66
r201	100	1664,5	2	-0,80	1,28	149,92	1668,9	2,0	-0,54	1,55	128,71
rc201	100	1661,7	2	-0,50	1,16	127,95	1667,3	2,0	-0,16	1,50	123,21
pr01	48	1419,0	2	-1,87	0,48	16,73	1434,1	2,0	-0,82	1,55	16,10
pr02	96	2596,8	3	1,07	2,10	42,83	2618,3	3,0	1,91	2,95	82,44
pr03	144	3589,0	4	0,14	5,09	450,03	3624,9	4,0	1,14	6,14	379,68
pr04	192	4476,3	5	2,52	6,14	1411,63	4531,4	5,0	3,78	7,44	1286,52
pr05	240	5256,1	6	2,62	6,00	5089,16	5294,9	6,0	3,37	6,78	4160,06
pr06	288	6345,1	7	3,39	6,41	7068,27	6389,5	7,0	4,11	7,15	9034,69
pr07	72	2110,3	3	0,93	1,93	36,69	2149,0	3,0	2,78	3,80	32,56
pr08	144	3501,6	4	-0,09	3,84	296,04	3530,3	4,0	0,73	4,69	376,63
pr09	216	4711,9	5	2,04	6,60	2126,40	4754,7	5,0	2,97	7,57	2255,80
pr10	288	6342,1	7	4,01	6,76	5046,35	6358,9	7,0	4,29	7,04	6080,63

Tabela B.1: Resultados finais obtidos para o SET1 com MH1

		melhor solução					média das melhores				
instância	n	duração	trips	$gapLS(\%)$	$gapMA(\%)$	CPU(s)	duração	trips	$gapLS(\%)$	$gapMA(\%)$	CPU(s)
c101	100	9696,7	9	0,11	1,05	83,74	9712,8	9,0	0,28	1,22	85,02
r101	100	1792,6	8	-0,48	5,16	106,96	1785,6	8,8	-0,87	4,75	108,37
rc101	100	1749,1	8	1,45	4,48	117,61	1798,5	8,8	4,32	7,43	110,84
c201	100	9593,0	3	-0,07	0,35	101,54	9612,5	3,0	0,13	0,55	98,31
r201	100	1654,3	2	-1,41	0,66	151,92	1667,5	2,0	-0,63	1,46	147,78
rc201	100	1643,8	2	-1,57	0,07	154,19	1665,5	2,0	-0,27	1,39	138,22
pr01	48	1419,0	2	-1,87	0,48	16,19	1428,9	2,0	-1,18	1,18	16,05
pr02	96	2614,5	3	1,76	2,80	95,16	2640,5	3,0	2,77	3,82	92,08
pr03	144	3555,6	4	-0,80	4,11	441,35	3609,3	4,0	0,70	5,69	425,97
pr04	192	4469,7	5	2,37	5,98	1193,05	4523,1	5,0	3,59	7,25	1424,39
pr05	240	5255,1	6	2,60	5,98	3691,37	5288,9	6,0	3,26	6,66	3889,26
pr06	288	6413,7	7	4,50	7,56	16535,22	6426,7	7,0	4,72	7,77	11144,29
pr07	72	2107,9	3	0,81	1,82	35,40	2124,1	3,0	1,59	2,60	33,99
pr08	144	3500,8	4	-0,11	3,82	440,23	3325,4	4,0	-5,12	-1,38	445,61
pr09	216	4686,7	5	1,50	6,03	2178,55	4744,7	5,0	2,75	7,34	2149,95
pr10	288	6364,7	7	4,38	7,14	7485,73	6394,7	7,0	4,87	7,65	6505,86

Tabela B.2: Resultados finais obtidos para o SET1 com MH2

	melhor solução					média das melhores				
instância	duração	trips	$gapLS(\%)$	$gapMA(\%)$	CPU(s)	duração	trips	$gapLS(\%)$	$gapMA(\%)$	CPU(s)
c101	955,1	1	-0,03	0,00	5,50	955,2	1,0	-0,03	0,01	5,43
r101	272,8	2	-4,68	0,00	6,67	274,7	2,0	-4,01	0,70	6,28
rc101	237,5	1	0,00	0,00	5,80	238,1	1,2	0,25	0,25	5,84
pr01	426,6	1	0,00	0,00	5,73	426,6	1,0	0,00	0,00	5,41
pr02	661,9	1	0,00	0,00	5,59	661,9	1,0	0,00	0,00	5,66
pr03	553,3	1	-1,04	0,00	5,57	553,3	1,0	-1,04	0,00	5,46
pr04	476,4	1	-6,79	0,00	5,52	476,4	1,0	-6,79	0,00	5,57
pr05	528,9	1	-5,71	0,00	5,48	528,9	1,0	-5,71	0,00	5,42
pr06	597,4	1	-1,11	0,00	5,57	604,8	1,0	0,12	1,24	5,53
pr07	670,2	1	-5,35	0,00	5,66	684,6	1,0	-3,32	2,15	5,50
pr08	573,4	1	0,00	0,00	5,53	574,9	1,0	0,26	0,26	5,42
pr09	645,5	1	-0,31	0,00	5,56	645,5	1,0	-0,31	0,00	5,51
pr10	461,5	1	0,00	0,00	5,68	469,6	1,0	1,75	1,75	5,46

Tabela B.3: Resultados finais obtidos para o SET2 (10 clientes) com MH1

	melhor solução					média das melhores				
instância	duração	trips	$gapLS(\%)$	$gapMA(\%)$	CPU(s)	duração	trips	$gapLS(\%)$	$gapMA(\%)$	CPU(s)
c101	955,1	1	-0,03	0,00	4,94	955,1	1,0	-0,03	0,00	4,89
r101	272,8	2	-4,68	0,00	6,03	274,7	2,0	-4,01	0,70	5,78
rc101	237,5	1	0,00	0,00	5,83	237,5	1,0	0,00	0,00	5,67
pr01	426,6	1	0,00	0,00	4,94	428,7	1,0	0,49	0,49	4,92
pr02	661,9	1	0,00	0,00	4,98	661,9	1,0	0,00	0,00	4,96
pr03	553,3	1	-1,04	0,00	5,02	553,3	1,0	-1,04	0,00	4,92
pr04	476,4	1	-6,79	0,00	5,03	476,4	1,0	-6,79	0,00	4,89
pr05	528,9	1	-5,71	0,00	5,25	528,9	1,0	-5,71	0,00	4,90
pr06	597,4	1	-1,11	0,00	5,00	601,1	1,0	-0,50	0,62	4,90
pr07	670,2	1	-5,35	0,00	5,10	670,2	1,0	-5,35	0,00	4,93
pr08	573,4	1	0,00	0,00	5,04	573,4	1,0	0,00	0,00	4,91
pr09	645,5	1	-0,31	0,00	5,02	645,9	1,0	-0,25	0,06	4,92
pr10	461,5	1	0,00	0,00	5,01	475,2	1,0	2,96	2,96	4,90

Tabela B.4: Resultados finais obtidos para o SET2 (10 clientes) com MH2

	melhor solução					média das melhores				
instância	duração	trips	$gapLS(\%)$	$gapMA(\%)$	CPU(s)	duração	trips	$gapLS(\%)$	$gapMA(\%)$	CPU(s)
c101	1452,2	2	-0,31	0,00	6,54	1452,2	2,0	-0,31	0,00	6,55
r101	379,8	2	-2,99	0,00	7,85	379,8	2,0	-2,99	0,00	7,13
rc101	303,2	2	-0,95	0,00	7,00	303,2	2,0	-0,95	0,00	7,39
pr01	590,4	1	0,00	0,00	6,13	596,2	1,0	0,98	0,98	6,10
pr02	745,6	1	-0,73	0,00	6,27	746,7	1,0	-0,59	0,15	6,18
pr03	632,9	1	-2,50	0,00	6,16	636,1	1,0	-2,00	0,51	6,07
pr04	683,4	1	0,00	0,00	6,19	683,4	1,0	0,00	0,00	6,03
pr05	621,2	1	-5,94	0,00	6,07	628,4	1,0	-4,84	1,16	5,98
pr06	685,2	1	0,00	0,00	6,13	686,9	1,0	0,24	0,24	6,09
pr07	795,3	1	-2,12	0,00	6,29	796,3	1,0	-2,00	0,12	6,20
pr08	707,2	1	0,00	0,00	6,20	707,2	1,0	0,00	0,00	6,08
pr09	771,7	1	-0,25	0,00	6,61	771,7	1,0	-0,25	0,00	6,13
pr10	611,9	1	0,00	0,00	6,02	621,3	1,0	1,54	1,54	6,08

Tabela B.5: Resultados finais obtidos para o SET2 (15 clientes) com MH1

	melhor solução					média das melhores				
instância	duração	trips	$gapLS(\%)$	$gapMA(\%)$	CPU(s)	duração	trips	$gapLS(\%)$	$gapMA(\%)$	CPU(s)
c101	1452,2	2	-0,31	0,00	5,98	1452,2	2,0	-0,31	0,00	5,94
r101	379,8	2	-2,99	0,00	6,32	385,9	2,0	-1,44	1,60	6,36
rc101	303,2	2	-0,95	0,00	6,22	309,2	2,0	1,01	1,98	6,30
pr01	590,4	1	0,00	0,00	5,40	608,5	1,0	3,06	3,06	5,42
pr02	745,6	1	-0,73	0,00	5,49	746,7	1,0	-0,59	0,15	5,74
pr03	632,9	1	-2,50	0,00	5,93	639,4	1,0	-1,50	1,02	5,45
pr04	683,4	1	0,00	0,00	5,53	683,4	1,0	0,00	0,00	5,44
pr05	621,2	1	-5,94	0,00	5,61	627,8	1,0	-4,94	1,06	5,45
pr06	685,2	1	0,00	0,00	5,44	686,9	1,0	0,24	0,24	5,53
pr07	795,3	1	-2,12	0,00	5,88	798,0	1,0	-1,78	0,34	5,63
pr08	707,2	1	0,00	0,00	5,63	709,4	1,0	0,31	0,31	5,48
pr09	771,7	1	-0,25	0,00	5,69	772,1	1,0	-0,20	0,05	5,63
pr10	611,9	1	0,00	0,00	6,16	613,9	1,0	0,33	0,33	5,43

Tabela B.6: Resultados finais obtidos para o SET2 (15 clientes) com MH2

	melhor solução					média das melhores				
instância	duração	trips	$gapLS(\%)$	$gapMA(\%)$	CPU(s)	duração	trips	$gapLS(\%)$	$gapMA(\%)$	CPU(s)
c101	2863,3	3	-1,53	0,00	9,31	2871,0	3,0	-1,26	0,27	9,47
r101	669,1	3	-1,05	2,12	10,74	677,5	3,6	0,19	3,40	10,68
rc101	714,9	3	-4,30	1,33	11,51	690,7	3,8	-7,54	-2,10	11,79
pr01	964,8	1	0,00	0,00	9,73	990,6	1,8	2,67	2,67	9,40
pr02	1088,7	2	-4,55	0,96	9,58	1098,3	2,0	-3,71	1,86	9,41
pr03	952,5	1	-0,48	0,00	11,34	956,4	1,0	-0,07	0,41	9,91
pr04	1091,6	2	-5,02	0,00	9,32	1105,3	2,0	-3,83	1,26	9,22
pr05	924,7	1	-1,24	0,00	10,36	924,7	1,0	-1,24	0,00	10,18
pr06	1063,2	2	-4,59	0,00	9,65	1068,1	2,0	-4,15	0,46	9,43
pr07	1130,4	2	-2,38	0,00	9,77	1134,1	2,0	-2,07	0,33	9,20
pr08	1006,2	2	-4,72	0,00	9,21	1011,1	2,0	-4,26	0,48	9,36
pr09	1091,4	2	-3,68	0,00	9,61	1126,5	2,0	-0,59	3,21	9,30
pr10	918,9	1	-0,88	0,00	10,17	925,8	1,0	-0,14	0,75	10,47

Tabela B.7: Resultados finais obtidos para o SET2 (30 clientes) com MH1

	melhor solução					média das melhores				
instância	duração	trips	$gapLS(\%)$	$gapMA(\%)$	CPU(s)	duração	trips	$gapLS(\%)$	$gapMA(\%)$	CPU(s)
c101	2863,6	3	-1,52	0,01	8,47	2876,4	3,0	-1,08	0,46	8,54
r101	655,2	3	-3,11	0,00	9,80	669,2	3,8	-1,03	2,14	10,00
rc101	683,8	4	-8,46	-3,08	10,84	685,7	4,2	-8,21	-2,81	10,76
pr01	964,8	1	0,00	0,00	9,17	979,6	1,2	1,53	1,53	9,02
pr02	1083,9	2	-4,97	0,52	8,77	1097,2	2,0	-3,81	1,75	8,61
pr03	952,5	1	-0,48	0,00	10,03	961,4	1,0	0,45	0,93	9,15
pr04	1091,6	2	-5,02	0,00	9,06	1109,1	2,0	-3,50	1,60	8,79
pr05	924,7	1	-1,24	0,00	9,14	926,8	1,0	-1,01	0,23	9,39
pr06	1063,2	2	-4,59	0,00	8,75	1066,4	2,0	-4,31	0,30	8,73
pr07	1130,4	2	-2,38	0,00	8,55	1143,3	2,0	-1,27	1,14	8,57
pr08	1006,2	2	-4,72	0,00	8,95	1024,9	2,0	-2,95	1,86	8,62
pr09	1102,4	2	-2,71	1,01	8,50	1117,3	2,0	-1,39	2,37	8,65
pr10	918,9	1	-0,88	0,00	9,24	924,0	1,0	-0,33	0,56	9,64

Tabela B.8: Resultados finais obtidos para o SET2 (30 clientes) com MH2

	melhor solução					média das melhores				
instância	duração	trips	$gapLS(\%)$	$gapMA(\%)$	CPU(s)	duração	trips	$gapLS(\%)$	$gapMA(\%)$	CPU(s)
c101	3870,1	4	-2,02	0,10	12,43	3895,0	4,0	-1,39	0,75	11,99
r101	880,8	5	-1,64	2,09	14,36	892,0	5,0	-0,39	3,39	14,06
rc101	850,3	4	-0,11	0,00	15,88	868,6	4,4	2,05	2,15	15,63
pr01	1175,0	2	-3,97	1,25	12,48	1185,2	2,0	-3,14	2,12	12,18
pr02	1365,8	2	-3,69	2,16	20,97	1381,1	2,0	-2,62	3,30	12,42
pr03	1320,6	2	-4,78	1,32	13,31	1329,9	2,0	-4,11	2,03	12,32
pr04	1259,5	2	-2,58	0,00	13,19	1281,0	2,0	-0,92	1,71	12,19
pr05	1208,3	2	0,62	0,63	12,48	1214,0	2,0	1,10	1,11	11,91
pr06	1257,6	2	-1,69	1,18	10,22	1263,3	2,0	-1,24	1,64	12,23
pr07	1407,0	2	-1,37	0,00	12,21	1426,2	2,0	-0,02	1,36	12,30
pr08	1222,2	2	-6,41	0,00	12,68	1256,4	2,0	-3,79	2,80	12,92
pr09	1290,5	2	0,27	0,49	11,74	1314,5	2,0	2,14	2,36	11,52
pr10	1213,9	2	-1,60	1,12	12,64	1231,5	2,0	-0,17	2,59	12,08

Tabela B.9: Resultados finais obtidos para o SET2 (40 clientes) com MH1

	melhor solução					média das melhores				
instância	duração	trips	$gapLS(\%)$	$gapMA(\%)$	CPU(s)	duração	trips	$gapLS(\%)$	$gapMA(\%)$	CPU(s)
c101	3867,3	4	-2,09	0,03	11,69	3885,0	4,0	-1,65	0,49	11,35
r101	880,0	4	-1,73	1,99	14,02	886,0	4,6	-1,06	2,69	13,81
rc101	850,3	4	-0,11	0,00	13,71	850,9	4,8	-0,03	0,08	14,76
pr01	1178,2	2	-3,71	1,53	11,68	1181,8	2,0	-3,42	1,84	11,22
pr02	1374,3	2	-3,10	2,80	11,67	1385,0	2,0	-2,34	3,60	11,81
pr03	1320,6	2	-4,78	1,32	11,52	1334,0	2,0	-3,81	2,35	11,80
pr04	1259,5	2	-2,58	0,00	12,11	1276,2	2,0	-1,29	1,33	11,76
pr05	1209,8	2	0,75	0,76	12,92	1223,4	2,0	1,89	1,89	12,51
pr06	1250,2	2	-2,27	0,59	12,72	1262,8	2,0	-1,28	1,60	11,49
pr07	1407,0	2	-1,37	0,00	12,33	1429,4	2,0	0,20	1,59	11,84
pr08	1222,2	2	-6,41	0,00	12,14	1265,3	2,0	-3,11	3,52	11,35
pr09	1296,0	2	0,70	0,92	11,77	1318,2	2,0	2,42	2,65	11,98
pr10	1200,6	2	-2,68	0,02	11,96	1216,9	2,0	-1,35	1,38	11,63

Tabela B.10: Resultados finais obtidos para o SET2 (40 clientes) com MH2

	melhor solução					média das melhores				
instância	duração	trips	$gapLS(\%)$	$gapMA(\%)$	CPU(s)	duração	trips	$gapLS(\%)$	$gapMA(\%)$	CPU(s)
eil51	458,7	5	-3,84	7,68	73,09	481,3	5,4	0,89	12,97	72,99
berlin52	8446,2	5	3,63	11,99	68,93	8980,0	5,8	10,18	19,07	70,36
st70	774,8	5	2,76	14,79	97,71	828,7	5,8	9,91	22,77	94,44
eil76	608,3	6	-3,29	13,07	100,10	624,8	6,0	-0,67	16,13	97,83
pr76	132015,1	5	18,65	22,06	98,17	134432,8	5,6	20,83	24,29	96,01
kroa100	26460,5	6	16,02	24,33	161,78	28073,8	6,4	23,10	31,91	153,02
kroc100	27035,2	6	14,89	30,30	139,09	28728,5	6,8	22,08	38,46	158,04
krod100	26146,6	6	5,13	22,79	163,62	28472,5	6,0	14,49	33,71	153,03
rd100	8865,7	5	-0,04	12,08	141,80	10112,0	6,0	14,02	27,84	142,26
eil101	735,4	5	0,74	16,92	133,75	750,8	5,8	2,85	19,36	126,61
lin105	18410,2	6	9,08	28,04	174,61	20287,8	7,2	20,20	41,09	164,87
ch150	8366,9	6	12,67	28,17	216,99	8655,9	6,2	16,56	32,60	264,32
tsp225	4646,7	6	2,01	18,66	515,80	4831,3	6,0	6,07	23,37	554,66
a280	3388,1	6	12,82	30,76	1171,17	3572,5	6,6	18,97	37,88	943,81
pcb442	61363,7	6	9,46	20,85	2985,36	63214,2	6,0	12,77	24,49	3294,00
pr1002	382248,1	7	31,29	47,56	59818,06	385161,1	7,0	32,29	48,69	41635,52

Tabela B.11: Resultados finais obtidos para o SET3 (3 hotéis) com MH1

	melhor solução					média das melhores				
instância	duração	trips	$gapLS(\%)$	$gapMA(\%)$	CPU(s)	duração	trips	$gapLS(\%)$	$gapMA(\%)$	CPU(s)
eil51	460,0	5	-3,56	7,98	93,63	484,0	5,6	1,47	13,62	89,69
berlin52	8065,3	5	-1,04	6,94	85,81	8776,7	5,6	7,69	16,37	87,08
st70	823,5	6	9,22	22,00	123,09	853,2	6,0	13,16	26,41	120,44
eil76	577,6	5	-8,17	7,36	123,98	624,5	5,8	-0,72	16,08	121,98
pr76	139508,4	5	25,39	28,98	132,44	136569,5	5,8	22,75	26,27	129,51
kroa100	26844,2	6	17,71	26,14	187,88	28347,2	6,0	24,30	33,20	193,17
kroc100	28024,1	7	19,09	35,06	206,01	30729,0	7,2	30,58	48,10	199,36
krod100	28019,3	6	12,66	31,58	168,21	28468,4	6,4	14,47	33,69	186,25
rd100	9370,4	6	5,65	18,46	186,24	9833,1	6,0	10,87	24,31	180,89
eil101	719,0	5	-1,51	14,31	157,37	740,6	5,4	1,45	17,74	162,51
lin105	19282,6	7	14,25	34,10	236,13	20294,2	7,0	20,24	41,14	205,72
ch150	8099,7	6	9,07	24,08	308,01	8669,7	6,4	16,75	32,81	329,59
tsp225	4756,1	6	4,41	21,45	765,07	4865,4	6,0	6,81	24,24	602,23
a280	3611,2	6	20,25	39,37	1404,8	3673,1	6,6	22,32	41,77	1169,62
pcb442	61872,6	6	10,37	21,85	5354,11	63119,7	6,0	12,60	24,31	5346,98
pr1002	352309,7	6	21,00	36,00	85825,12	369241,4	6,6	26,82	42,54	61013,00

Tabela B.12: Resultados finais obtidos para o SET3 (3 hotéis) com MH2

	melhor solução					média das melhores				
instância	duração	trips	$gapLS(\%)$	$gapMA(\%)$	CPU(s)	duração	trips	$gapLS(\%)$	$gapMA(\%)$	CPU(s)
eil51	491,3	7	4,75	15,33	81,86	490,1	7,8	4,51	15,06	80,40
berlin52	8942,3	8	9,05	18,57	82,53	9280,4	8,2	13,18	23,05	81,45
st70	853,4	8	9,83	26,43	110,64	964,7	9,2	24,15	42,92	108,71
eil76	643,3	8	3,59	19,57	102,92	655,5	8,0	5,56	21,84	113,50
pr76	134262,7	8	8,92	24,13	117,87	139209,5	8,6	12,93	28,71	114,82
kroa100	27368,9	9	20,87	28,60	168,00	28857,0	9,6	27,44	35,59	163,82
kroc100	25541	8	9,18	23,10	159,74	31010,6	10,2	32,56	49,46	153,27
krod100	30012,6	10	23,57	40,94	155,47	32821,4	11,0	35,14	54,13	151,36
rd100	10005,3	9	15,83	26,49	155,89	10891,8	9,4	26,09	37,70	166,08
eil101	776,3	8	6,78	23,42	172,58	788,5	8,6	8,46	25,36	166,46
lin105	20155,5	10	25,33	40,17	164,18	21632,4	11,5	34,51	50,44	152,92
ch150	8232,2	9	10,59	26,11	316,18	8853,2	9,6	18,93	35,62	296,80
tsp225	4821,4	8	0,61	23,12	817,33	5066,7	9,0	5,73	29,38	813,54
a280	3513,4	10	12,54	32,78	1392,12	3733,4	10,4	19,58	41,10	1134,47
pcb442	64061,9	8	9,52	26,16	5785,06	65930,5	8,8	12,71	29,84	3595,33
pr1002	400642,6	11	34,07	54,23	67483,29	408343,2	11,0	36,65	57,19	61105,82

Tabela B.13: Resultados finais obtidos para o SET3 (5 hotéis) com MH1

instância	melhor solução duração	trips	$gapLS(\%)$	$gapMA(\%)$	CPU(s)	média das melhores duração	trips	$gapLS(\%)$	$gapMA(\%)$	CPU(s)
eil51	487,0	7	3,84	14,32	101,63	491,8	7,6	4,85	15,44	106,53
berlin52	8984,5	8	9,57	19,13	87,60	9654,6	8,2	17,74	28,01	96,01
st70	795,5	8	2,38	17,85	143,46	894,9	8,6	15,17	32,58	144,41
eil76	655,2	7	5,51	21,78	134,52	646,0	7,8	4,02	20,07	140,39
pr76	124907,9	8	1,33	15,49	160,22	131788,6	8,2	6,91	21,85	150,49
kroa100	29756,0	9	31,41	39,82	193,41	30673,6	10,0	35,46	44,13	189,90
kroc100	29320,2	10	25,33	41,31	167,18	32494,5	11,0	38,90	56,61	185,33
krod100	30404,5	10	25,19	42,78	182,64	30892,2	10,4	27,20	45,07	183,78
rd100	9537,6	8	10,41	20,58	222,47	10344,1	9,0	19,75	30,77	205,89
eil101	705,8	8	-2,92	12,21	214,53	754,7	8,2	3,80	19,98	202,77
lin105	19752,3	10	22,82	37,37	178,56	22610,0	11,4	40,59	57,24	180,91
ch150	8501,6	9	14,21	30,23	439,06	8766,2	9,6	17,76	34,29	402,16
tsp225	4946,4	9	3,22	26,31	911,82	5138,4	9,4	7,23	31,22	778,17
a280	3601,0	10	15,34	36,09	1393,83	3682,6	10,0	17,96	39,18	1371,61
pcb442	62490,6	8	6,83	23,07	6466,27	64845,4	8,8	10,86	27,70	5767,68
pr1002	382413,3	10	27,97	47,21	53679,32	398343,9	10,4	33,30	53,34	66075,62

Tabela B.14: Resultados finais obtidos para o SET3 (5 hotéis) com MH2

instância	melhor solução duração	trips	$gapLS(\%)$	$gapMA(\%)$	CPU(s)	média das melhores duração	trips	$gapLS(\%)$	$gapMA(\%)$	CPU(s)
eil51	523,1	14	13,72	22,79	177,51	530,1	14,8	15,23	24,43	178,93
berlin52	7945,6	9	-3,95	1,04	160,15	8419,9	9,0	1,79	7,07	157,45
st70	983,2	17	45,66	45,66	231,35	1047,9	18,8	55,25	55,25	232,58
eil76	635,0	14	4,61	18,03	274,72	654,2	14,4	7,77	21,59	273,49
pr76	137768,8	15	13,05	27,38	252,96	142766,7	15,6	17,16	32,00	255,33
kroa100	25928,8	15	10,91	21,83	307,49	26671,8	15,6	14,08	25,33	305,86
kroc100	26727,3	16	14,53	28,81	324,19	26952,0	16,0	15,49	29,90	325,62
krod100	26808,7	15	13,94	25,90	328,69	27218,4	16,0	15,68	27,82	327,02
rd100	9891,0	12	8,00	25,04	403,08	9900,4	12,4	8,11	25,16	402,76
eil101	755,7	14	12,62	20,14	356,33	773,9	14,2	15,33	23,03	367,22
lin105	19537,9	12	20,85	35,88	355,09	19245,7	12,6	19,04	33,85	351,15
ch150	7690,0	14	6,76	17,80	879,36	8027,5	14,8	11,45	22,97	869,77
tsp225	4913,0	15	12,25	25,46	2684,65	4947,7	15,0	13,04	26,35	2661,05
a280	3659,3	19	17,78	40,04	4553,32	3756,6	19,0	20,91	43,77	4600,00
pcb442	66723,3	15	17,24	28,87	19185,86	65801,1	15,8	15,62	27,09	17494,02
pr1002	391816,3	19	35,35	51,25	136826,20	407411,3	19,8	40,74	57,27	143627,33

Tabela B.15: Resultados finais obtidos para o SET3 (10 hotéis) com MH1

instância	melhor solução duração	trips	$gapLS(\%)$	$gapMA(\%)$	CPU(s)	média das melhores duração	trips	$gapLS(\%)$	$gapMA(\%)$	CPU(s)
eil51	451,3	7	-5,78	5,20	73,00	478,2	7,0	-0,16	11,48	75,60
berlin52	9185,5	7	4,11	6,29	72,51	9921,8	8,0	12,45	14,81	73,04
st70	807,7	7	8,42	11,72	104,13	841,9	7,8	13,01	16,45	103,20
eil76	608,5	7	2,27	11,04	103,71	614,8	7,0	3,32	12,18	101,02
pr76	138864,4	8	6,99	17,62	107,24	145838,8	8,0	12,37	23,53	107,64
kroa100	25542,9	7	11,89	14,32	166,23	27610,6	8,0	20,95	23,58	150,77
kroc100	27948,1	8	17,71	33,51	142,72	29047,6	8,4	22,34	38,76	135,94
krod100	27310,3	7	9,66	26,06	171,57	28285,1	7,8	13,58	30,56	157,46
rd100	9867,2	7	12,52	19,69	137,24	10316,4	7,8	17,65	25,14	150,53
eil101	701,1	7	1,17	10,58	140,92	735,1	7,0	6,08	15,95	142,52
ch150	7914,2	7	3,06	19,06	257,24	8362,7	7,2	8,90	25,81	249,39
tsp225	5081,8	8	5,45	11,17	785,86	5211,6	8,4	8,15	14,01	663,54
a280	3478,2	9	11,37	31,45	1072,89	3547,9	9,0	13,60	34,08	1219,86
pcb442	66232,9	8	3,78	21,89	3286,00	72726,2	9,6	13,95	33,84	3737,74
pr1002	442749,4	11	34,05	51,27	75740,50	459663,2	11,4	39,17	57,05	75658,83

Tabela B.16: Resultados finais obtidos para o SET4 com MH1

instância	melhor solução					média das melhores				
	duração	trips	$gapLS(\%)$	$gapMA(\%)$	CPU(s)	duração	trips	$gapLS(\%)$	$gapMA(\%)$	CPU(s)
eil51	450,5	6	-5,95	5,01	93,97	477,0	6,8	-0,41	11,20	96,09
berlin52	9181,1	7	4,06	6,24	94,33	10108,1	8,2	14,57	16,96	92,23
st70	795,7	7	6,81	10,06	136,93	890,7	8,0	19,55	23,19	132,96
eil76	613,5	6	3,11	11,95	136,75	621,5	6,6	4,46	13,42	133,72
pr76	139549,7	7	7,52	18,20	133,68	140597,2	8,0	8,33	19,09	136,69
kroa100	28159,9	7	23,36	26,03	187,23	28556,3	8,0	25,09	27,81	176,33
kroc100	25488,8	8	7,35	21,76	167,61	27333,9	8,2	15,12	30,58	171,10
krod100	26723,8	8	7,31	23,36	205,95	28972,0	8,2	16,33	33,73	200,55
rd100	10569,2	7	20,53	28,20	180,96	10417,1	7,8	18,79	26,36	193,11
eil101	720,9	7	4,03	13,71	168,30	744,1	7,0	7,38	17,37	164,51
ch150	7616,6	7	-0,81	14,59	352,59	8185,7	7,4	6,60	23,15	327,79
tsp225	5110,1	7	6,04	11,79	740,46	5368,5	8,2	11,40	17,45	707,67
a280	3482,6	8	11,51	31,62	1271,82	3701,9	9,0	18,54	39,91	1417,21
pcb442	69743,2	8	9,28	28,35	5488,42	70602,2	8,8	10,62	29,93	6117,87
pr1002	440652,6	11	33,42	50,55	87680,55	446369,5	11,2	35,15	52,51	67325,76

Tabela B.17: Resultados finais obtidos para o SET4 com MH2