Lisbon School of Economics & Management
Universidade de Lisboa

MESTRADO EM
MÉTODOS QUANTITATIVOS PARA A DECISÃO ECONÓMICA E EMPRESARIAL

TRABALHO FINAL DE MESTRADO
DISSERTAÇÃO

META-HEURÍSTICAS PARA O PROBLEMA DO CARTEIRO RURAL ORIENTADO COM LUCROS E INCOMPATIBILIDADES

MARIANA FERREIRA RIBEIRO

ORIENTAÇÃO:
PROFª. DOUTORA RAQUEL MONTEIRO DE NOBRE COSTA BERNARDINO

DOCUMENTO ESPECIALMENTE ELABORADO PARA OBTENÇÃO DO GRAU DE MESTRE

JANEIRO - 2024

Agradecimentos

Em primeiro lugar quero expressar a minha profunda gratidão aos meus pais por terem sido o meu alicerce e apoio ao longo desta jornada tão desafiante, e por me terem encorajado a seguir o meu caminho. Estendo, ainda, o agradecimento à restante família por celebrarem comigo as minhas conquistas pessoais e académicas e serem o meu "porto seguro".

Quero deixar um agradecimento especial à minha orientadora, a Professora Raquel Bernardino, pelo acompanhamento crucial, a disponibilidade e ajuda ao longo de todo o processo de escrita da dissertação e por me ter encorajado quando eu desanimava com os resultados.

Agradeço, também, à Professora Cândida Mourão por me ter orientado no início desta fase, quando eu não sabia que direção seguir.

Por fim, mas não menos importante, gostaria de agradecer aos amigos que foram uma presença constante nesta etapa da minha vida, e que não me deixavam esquecer que eu tinha uma dissertação para acabar. Em particular, quero deixar um agradecimento especial à Ana por ter partilhado comigo esta jornada do mestrado no ISEG.

Índice

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Resumo

O presente trabalho aborda o Problema do Carteiro Rural Orientado com Lucros e Incompatibilidades (em ingles, *Directed Profitable Rural Postman Problema with Incompatibility Constraints*, ou seja, DPRPP-IC). Este generaliza o problema do carteiro rural, adicionando arcos lucrativos e incompatibilidades entre pares de nodos. O DPRPP-IC procura uma rota com início e fim no depósito, que maximize a diferença entre as receitas e os custos totais, que são compostos pelos custos de deslocação e penalizações pagas para eliminar incompatibilidades fracas, satisfazendo as restrições de incompatibilidade. O problema apresentado tem aplicação prática para empresas que partilham serviços de distribuição, transporte ou logística. Uma vez que o DPRPP-IC é um caso particular de um problema NP-difícil, é proposto um algoritmo *Tabu Search* para encontrar soluções de boa qualidade de forma eficiente. *Tabu Search* é uma meta-heurística de pesquisa local que procura escapar a ótimos locais usando uma lista tabu, que não permite movimentos para soluções recentes. Algumas instâncias de referência da literatura são utilizadas para avaliar a qualidade da meta-heurística proposta. Após uma análise aos parâmetros da meta-heurística e a fixação dos mesmos, os resultados analisados mostram que a meta-heurística proposta obtém soluções de fraca qualidade, mas em tempos computacionais bastante inferiores aos registados na literatura para obtenção da solução ótima por métodos exatos.

Palavras-chave: Problema do Carteiro Rural Orientado, incompatibilidades, arcos lucrativos, *Tabu Search*, pesquisa local.

Abstract

This work addresses the Directed Profitable Rural Postman Problem with Incompatibility Constraints (DPRPP-IC). This problem generalizes the rural postman problem by adding profitable arcs and incompatibilities between nodes. The DPRPP-IC searches for a route starting and ending at the depot, that maximizes the difference between the collected profit and the total cost, which is comprised of the travelling costs and the penalties paid to remove weak incompatibilities, while satisfying the incompatibility constraints. The DPRPP-IC has practical application in companies that share shipping, transportation, or logistic services. Since the DPRPP-IC is an NP-hard problem, it is proposed a Tabu Search algorithm to find good quality solutions efficiently. The Tabu Search is a local search based metaheuristic that escapes from local optima using a tabu list, which does not allow movements to recent solutions. Some benchmark instances from the literature are used to evaluate the quality of the proposed metaheuristic. After tuning the parameters of the metaheuristic, the results show that the proposed metaheuristic yields solutions of low quality. However, the computational times required to obtain such solutions are significantly lower than those reported in the literature for obtaining the optimal solutions through exact methods.

Keywords: Directed Rural Postman Problem, Incompatibilities, Profitable arcs, Tabu Search, Local Search.

Capítulo 1

Introdução

No presente Trabalho Final de Mestrado (TFM) o foco é o Problema do Carteiro Rural Orientado com Lucros e Incompatibilidades (DPRPP-IC, do inglês *Directed Profit Rural Postman Problem with Incompatibility Constraints*), e para o qual será proposta uma meta-heurística de modo a obter uma solução admissível de boa qualidade de forma eficiente.

O DPRPP-IC é um problema de roteamento nos arcos, onde cada arco pertencente à solução tem associado um custo de travessia e, caso seja um arco lucrativo, acrescenta, também, uma receita que é recolhida apenas da primeira vez que o arco é atravessado. Este problema tem, ainda, restrições de incompatibilidade associadas a nodos origem de arcos lucrativos. A incompatibilidade entre dois nodos pode ser forte, ou seja, os arcos lucrativos com início nesses nodos não podem pertencer ambos à solução, ou pode ser fraca e, nesse caso, essa incompatibilidade pode ser removida se uma penalização for paga. O objetivo, neste problema, é encontrar uma rota com início e fim no depósito, que maximize a diferença entre a receita e o custo total, que inclui tanto os custos de travessia dos arcos como as penalizações pagas para remover incompatibilidades fracas, satisfazendo as restrições relacionadas com as incompatibilidades impostas.

Os problemas de roteamento nos arcos têm diversas aplicações em várias áreas, tais como, logística, transportes, redes de distribuição, ou outras atividades relacionadas com movimentação de mercadorias.

O DPRPP-IC tem como caso particular o problema do carteiro rural orientado, que é um problema NP-difícil (Colombi and Mansini (2014)), ou seja, é um problema complexo cuja resolução por métodos exatos levaria a tempos computacionais elevados, e portanto, faz sentido desenvolver heurísticas para a sua resolução. Assim, neste TFM é proposta uma meta-heurística para construir uma solução de boa qualidade para o DPRPP-IC, de forma eficiente. A meta-heurística proposta é um algoritmo de *Tabu Search*, que apresenta bons resultados em variados problemas da literatura, tanto em problemas de roteamento nos nodos como problemas de roteamento nos arcos, como por exemplo nos artigos Yu et al. (2019) e Gmira et al.

(2021).

A estratégia da *Tabu Search* controla o processo de pesquisa local, onde em cada iteração é realizado o movimento da solução corrente para uma solução na sua vizinhança, mesmo que este não represente uma melhoria no valor da solução. Assim, este tipo de pesquisa, ao permitir movimentos desfavoráveis, evita a estagnação em ótimos locais. No algoritmo de *Tabu Search* é utilizada, para cada vizinhança explorada, uma lista tabu, que, servindo como uma memória de curto prazo, guarda os movimentos recentes feitos nas pesquisas anteriores. Esta estratégia impede que soluções visitadas recentemente sejam novamente visitadas. A meta-heurística devolve a melhor solução encontrada, entre todas as visitadas.

Durante a pesquisa local começam por ser exploradas três vizinhanças: a primeira consiste em inserir um nodo entre dois nodos consecutivos da solução corrente, a segunda consiste em remover um nodo da solução corrente, e a terceira consiste em trocar a posição de dois nodos da solução corrente. Após a pesquisa exaustiva destas três vizinhanças, é efetuado o movimento para a melhor solução encontrada entre as três. Caso não sejam encontradas soluções admissíveis nas vizinhanças anteriores, são exploradas duas novas vizinhanças: uma que passa por inserir dois nodos em posições consecutivas na solução corrente, e outra que consiste em remover dois nodos consecutivos da solução corrente.

A meta-heurística proposta foi implementada em Visual Basic for Applications (VBA), e, para o teste da sua qualidade, foram utilizadas instâncias da literatura, propostas por Colombi et al. (2017).

A dissertação encontra-se dividida em cinco capítulos. O Capítulo 2 apresenta o problema do DPRPP-IC, as suas aplicações práticas e problemas relacionados. O Capítulo 3 apresenta a metodologia adotada para obter soluções para o problema em estudo. Neste, são descritas as subrotinas utilizadas pela meta-heurística, tanto na sua fase construtiva como na fase melhorativa. O Capítulo 4 apresenta uma análise aos parâmetros a definir e uma avaliação ao desempenho da *Tabu Search*, bem como os resultados gerais e detalhados obtidos e, por fim, no Capítulo 5 são apontadas as principais conclusões do trabalho e que alterações podem ser introduzidas, no futuro, para melhorar a *performance* da *Tabu Search*.

Capítulo 2

Problema do Carteiro Rural Orientado com Lucros e Incompatibilidades

2.1 Revisão de Literatura

Dado o crescente reconhecimento da importância de uma gestão eficiente por parte de organizações e empresas, os problemas de roteamento nos arcos têm sido cada vez mais um foco de estudo.

Os problemas de roteamento nos arcos (ARP, do inglês *Arc Routing Problem*) podem ser definidos numa rede que consiste num conjunto de nodos e ligações entre os nodos. Essas ligações são designadas por arcos quando são direcionadas, ou seja, quando na travessia de um arco de um nodo para outro se diferencia entre o nodo de início e o nodo final do arco. O serviço neste tipo de problemas, contrariamente aos problemas de roteamento nos nodos, está associado aos arcos. O ARP tem diversas aplicações práticas como, por exemplo a distribuição de correspondência, a remoção de neve e recolha de lixo porta-a-porta. Para informações mais aprofundadas sobre ARP, deve ser consultado o livro Corberán and Laporte (2015).

O problema do carteiro rural, identificado em Orloff (1974), é a base do problema a ser estudado neste trabalho, sendo um problema de roteamento nos arcos que consiste em determinar uma rota de custo mínimo que contenha um subconjunto de arcos que requerem serviço do grafo.

O problema do carteiro rural com lucros, baseado no anterior, é um problema onde nem todos os arcos que requerem serviço têm de ser visitados. Existe, no entanto, uma penalização se estes não forem servidos. O artigo de Colombi and Mansini (2014), apresenta a criação de uma mata-heurística para a resolução do problema do carteiro rural orientado com lucros, onde o objetivo é encontrar uma rota que visite determinados arcos maximizando a diferença entre receitas e custos tanto de deslocação como as penalizações. O método de resolução proposto é dividido em dois procedimentos. O primeiro é uma mata-heurística onde, em cada iteração, um

novo subconjunto de arcos que requerem serviço é selecionado, resolvendo uma relaxação do problema e o problema composto pelo subconjunto de arcos é resolvido até à otimalidade. O segundo procedimento é uma pesquisa baseada no *Branch-and-cut* para tentar melhorar a solução encontrada pela mata-heurística. Esta abordagem resultou em soluções de boa qualidade, num conjunto de instâncias, obtidas em tempos computacionais razoáveis.

O DPRPP-IC surge como uma generalização do problema anterior. Até então, a existência de incompatibilidades era aplicada apenas a problemas de roteamento nos nodos, nomeadamente no problema do caixeiro-viajante com famílias e restrições de incompatibilidade, estudado no artigo de Bernardino and Paias (2022). Neste, as cidades a visitar são agrupadas em subgrupos, designados por famílias, e entre alguns pares de famílias existem incompatibilidades, não podendo ser parte da mesma rota. O objetivo, neste problema, é encontrar um conjunto rotas que visite um determinado número de cidades em cada família, de menor custo, evitando as incompatibilidades. Este é apenas um exemplo de problemas de roteamento nos nodos com incompatibilidades, existindo também este tipo de restrições no problema do comprador viajante (Gendreau et al., 2016) ou no *Pickup and Delivery Problem* (Factorovich et al., 2020). O primeiro é um problema de roteamento de veículos onde existem mercados que vendem vários itens e o objetivo é visitar mercados de forma a adquirir todos os itens de uma lista. Pode existir, no entanto, a presença de incompatibilidades entre produtos a transportar, impondo a impossibilidade de carregar dois produtos incompatíveis no mesmo veículo. Para a resolução deste problema foi utilizada uma abordagem *branch-and-price*, e os resultados evidenciam um bom desempenho do algoritmo, otimizando eficientemente instâncias com uma grande frota de veículos e um elevado número de produtos incompatíveis. Já no *Pickup and Delivery Problem with Incompatibility Constrainsts*, o objetivo é satisfazer a procura em cada ponto de entrega (*Delivery point*) e recolher a oferta de cada ponto de recolha (*Pickup point*), podendo também existir produtos incompatíveis que não podem ser transportados ao mesmo tempo no veículo. Para a resolução deste problema, foi, também, proposto um algoritmo *branch-and-cut* que obtém soluções ótimas para o problema de forma eficiente.

No entanto, as restrições de incompatibilidade podem, também, ser aplicadas a problemas de roteamento nos arcos, o que origina novos tópicos de investigação. O DPRPP-IC é caracterizado pela existência de determinados arcos que originam um lucro ao serem atravessados e pela existência de incompatibilidades entre alguns nodos que, consoante o tipo de incompatibilidade, podem ser removidas através do pagamento de uma penalização. O primeiro e único estudo de que se tem conhecimento deste problema aparece num artigo de Colombi et al. (2017) onde é criada uma mata-heurística para encontrar uma solução que maximize a diferença entre a receita e o custo, quer de deslocação quer de penalizações. No referido artigo, é proposta uma mata-heurística que combina duas técnicas de otimização, *Tabu Search* e GRASP (do inglês, *Greedy Randomized Adaptive Search Procedure*), para resolução do problema. A

mata-heurística apresenta bons resultados em comparação com os obtidos por um algoritmo de *branch-and-cut* com um tempo limite de uma hora.

A heurística a ser desenvolvida neste trabalho, para o problema em estudo, é a *Tabu Search*, cuja definição se encontra no livro Glover et al. (2008). *Tabu Search* é uma meta-heurística de pesquisa local que pretende fugir a ótimos locais através de uma memória de curto prazo (a chamada lista tabu), que não permite a realização de movimentos que originem soluções visitadas recentemente. Permite, no entanto, que se façam movimentos para soluções de pior valor para evitar a estagnação num ótimo local. Esta meta-heurística é usada em vários problemas e apresenta bons resultados, como por exemplo no problemas de roteamento nos arcos estudado no artigo de Yu et al. (2019) e no problema de roteamento de veículos estudado no artigo de Gmira et al. (2021). Nestes é aplicado um algoritmo de *Tabu Search* para encontrar uma solução para o *split-delivery mixed capacitated arc-routing problem* e para o *time-dependent vehicle routing problem with time windows*. Assim, pelos resultados demonstrados faz sentido utilizá-la como método de resolução para o DPRPP-IC.

2.2 Definição do Problema

O DPRPP-IC é um problema de roteamento nos arcos onde a seleção de um arco para a solução tem associado um custo e, caso seja um arco lucrativo, tem, também, associado um lucro. O DPRPP-IC é um problema NP-difícil (Colombi and Mansini (2014)), ou seja, é um problema complexo cuja resolução por métodos exatos levaria a tempos computacionais elevados.

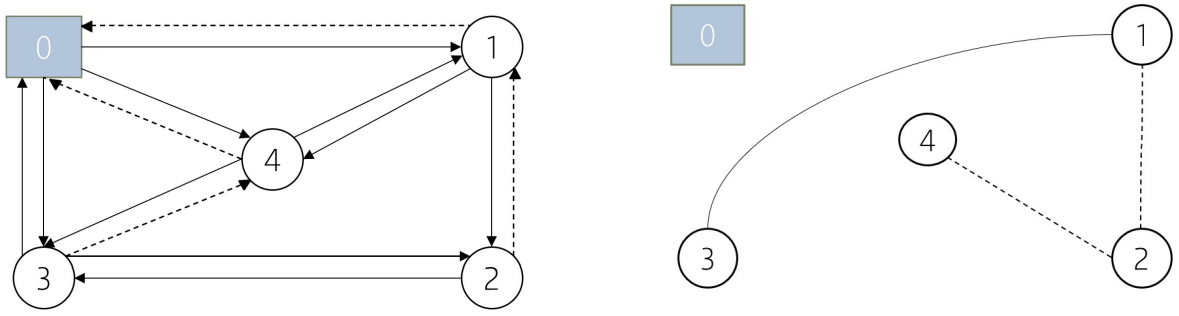
Tal como já foi referido, este problema surge como uma variante do problema do carteiro rural orientado, que pretende encontrar uma rota que minimize os custos satisfazendo as restrições relacionadas com os arcos que requerem serviço. A introdução de restrições de incompatibilidade entre arcos impossibilita ou penaliza a pertença de certos pares de arcos lucrativos à solução, tornando o DPRPP-IC mais complexo.

Existem dois tipos de incompatibilidade entre nodos no DPRPP-IC. A incompatibilidade forte não permite que os arcos lucrativos dos nodos em questão pertençam ambos à solução, isto é, se os nodos i e j forem fortemente incompatíveis, os arcos lucrativos com início no nodo i e os arcos lucrativos com início no nodo j não podem pertencer à solução em conjunto. Na incompatibilidade fraca, os arcos lucrativos podem pertencer à solução mediante o pagamento de uma penalização, ou seja, se os nodos i e j forem fracamente incompatíveis, os arcos lucrativos com início em i e arcos lucrativos com início em j podem ambos pertencer à solução se for paga uma penalização.

Assim, o objetivo do DPRPP-IC é encontrar uma rota com início e fim no depósito, que maximize a diferença entre a receita total total e o custo total (que inclui os custos de deslocação e as penalizações), satisfazendo as restrições de incompatibilidade forte impostas.

O DPRPP-IC pode ser definido da seguinte forma. Seja $G = (V, A)$ um grafo orientado sendo $V = \{0, \ldots, n\}$ o conjunto de nodos a visitar, onde o nodo 0 representa o depósito, e A o conjunto de arcos orientados presentes no grafo. Cada arco $(i, j) \in A$ tem associado um custo de travessia c_{ij}. Existe, ainda, um conjunto de arcos lucrativos $R \subseteq A$, sendo que a travessia de um arco lucrativo $(i, j) \in R$ tem uma receita p_{ij} associado que apenas é recolhido da primeira vez que o arco é atravessado. Assim, define-se lucro como $D_{ij} = p_{ij} - c_{ij}$, ou seja, a diferença entre a receita e o custo de travessia do arco (i, j). É de notar que, p_{ij} = 0 para $(i, j) \in A \setminus R$, isto é, assume-se que a receita dos arcos não lucrativos é zero. Defina-se $V_1 \subseteq V$ como o conjunto de nodos i $(i \in V)$ que são origem de pelo menos um arco lucrativo $(i, j) \in R$, isto é, $V_1 = \{i \in V : (i, j) \in R\}$. Desta forma, V_1 contém todos os nodos que poderão ter restrições de incompatibilidade entre si. Assim, temos o grafo de incompatibilidades $\bar{G} = (V_2, E_1 \cup E_2)$, onde V_2 é o conjunto de nodos $i \in V_1$ que são incompatíveis com pelo menos um outro nodo $j \in V_1 \setminus \{i\}$ e $E_1 \cup E_2$ é o conjunto de pares de nodos incompatíveis, ou seja, $E_1 \cup E_2 = \{\{i, j\} : i \in V_2 \wedge j \in V_2\}$. Considere-se dois nodos $i, j \in V_2$. Se $\{i, j\} \in E_1$, estes são fortemente incompatíveis, enquanto se $\{i, j\} \in E_2$, os nodos i e j são fracamente incompatíveis, sendo possível remover essa incompatibilidade com o pagamento de uma penalização γ_{ij}.

Para ilustrar melhor o problema em estudo, a Figura 2.1 (a) apresenta o grafo G para uma instância do DPRPP-IC, com um depósito, quatro nodos e as ligações possíveis entre nodos, e a Figura 2.1 (b) apresenta o grafo de incompatibilidades $\bar{G}$ com as incompatibilidades existentes entre nodos.



(a) Grafo G. (b) Grafo de incompatibilidades $\bar{G}$.

Figura 2.1: Exemplo de uma instância do DPRPP-IC.

No grafo $G = (V, A)$ da Figura 2.1 (a), temos o conjunto de nodos $V = \{0, 1, 2, 3, 4\}$, onde o nodo 0 representa o depósito, e o conjunto de arcos A = $\{(0, 1); (0, 3); (0, 4); (1, 0); (1, 2); (1, 4); (2, 1); (2, 3); (3, 0); (3, 2); (3, 4); (4, 1); (4, 3); (4, 0)\}$ dos quais R = $\{(1, 0);(2, 1); (3, 4); (4, 0)\}$ são arcos lucrativos assinalados a tracejado. Dos nodos existentes, podem existir incompatibilidades entre os nodos pertencentes a $V_1 = \{1, 2, 3, 4\}$, pois são nodos origem de arcos lucrativos.

O grafo $\bar{G} = (V_2, E_1 \cup E_2)$ presente na Figura 2.1 (b) mostra as incompatibilidades existentes, onde a tracejado são as incompatibilidades fracas, e a linha contínua são as incompatibilidades fortes.

Pela notação anterior temos $V_2 = \{1, 2, 3, 4\}$ como o conjunto de nodos com incompatibilidades, $E_1 = \{\{1, 3\}\}$ com os pares de nodos com incompatibilidade forte, e $E_2 = \{\{1, 2\}; \{2, 4\}\}$ com os pares de nodos que têm incompatibilidade fraca entre si.

Assim, se os nodos 1 e 2 são fracamente incompatíveis, o arco lucrativo com início no nodo 1 e o arco lucrativo com início no nodo 2 apenas podem pertencer ambos à solução mediante o pagamento da penalização γ_{12}. O mesmo acontece com os arcos lucrativos a começar nos nodos 2 e 4.

Já os arcos lucrativos com início no nodo 1 e os arcos lucrativos com início no nodo 3 não podem fazer, de todo, parte da solução em conjunto por serem fortemente incompatíveis.

Desta forma, temos, por exemplo, a solução admissível $(0, 4, 3, 2, 1, 0)$ que envolve o pagamento de uma penalização pois os nodos 1 e 2 são fracamente incompatíveis. Outra solução admissível seria, por exemplo, $(0, 4, 1, 0)$ pois neste problema de roteamento nos arcos, uma solução não precisa de ter os nodos todos e, neste caso, não haveria o pagamento de penalizações. Uma solução não admissível seria, por exemplo $(0, 3, 4, 3, 2, 1, 0)$ pois contém arcos lucrativos com início em nodos fortemente incompatíveis, nomeadamente, os nodos 1 e 3.

O DPRPP-IC tem diversas aplicações, e é particularmente relevante em empresas de transportes que praticam colaboração horizontal, isto é, que partilham serviços de expedição, transporte, logística ou outras atividades auxiliares relacionadas com a movimentação de mercadorias. As incompatibilidades estão relacionadas com as restrições impostas por cada uma das empresas pertencentes a esta associação (conjunto de empresas que colaboram horizontalmente). As restrições de incompatibilidade podem estar relacionadas com o tamanho da encomenda a transportar, por exemplo, mercadorias de pequena dimensão não devem ser transportadas juntamente com mercadorias de média dimensão e não podem ser transportadas com mercadorias de grande dimensão. Podem, ainda, estar relacionadas com o tipo de produtos a transportar. Colombi et al. (2017), por exemplo, identificam produtos alimentares e produtos químicos como sendo fortemente incompatíveis e produtos têxteis e detergentes são fracamente incompatíveis, podendo ser transportados em conjunto caso se pague uma penalização, que pode ser, por exemplo, o custo de um seguro contra possíveis contaminações. Por fim, pode haver incompatibilidades impostas pelo consumidor que pretende um serviço exclusivo, pois este tipo de colaboração envolve empresas ao mesmo nível na cadeia de abastecimento e, muitas vezes, concorrentes. Aqui, pode ser incompatibilidade forte ou fraca caso o consumidor aceite um incentivo monetário, que é a penalização. As empresas de transporte que participam nestas colaborações pretendem satisfazer os requisitos dos seus clientes partilhando os seus serviços de forma a reduzir os custos operacionais, ou aumentar os lucros.

Outra aplicação do DPRPP-IC é o problema de remoção de neve, cujo objetivo é otimizar as rotas dos veículos de remoção de neve tentando cobrir todas as ruas que necessitam de limpeza, minimizando custos de deslocação. Neste caso, os arcos lucrativos seriam as ruas que teriam neve para remover. Quanto a incompatibilidades fortes, poderiam existir incompatibilidades deste tipo entre ruas com configurações muito diferentes, ou seja, se o veículo for apropriado a ruas com determinadas caraterísticas, pode não ser apropriado para ruas com características diferentes pois, por exemplo, pode não ter um tamanho compatível com a largura da rua e assim, ruas fortemente incompatíveis não podem ser intervencionadas na mesma rota. No entanto, se houver a possibilidade de trocar de veículo para se adaptar às diferentes caraterísticas das ruas, a incompatibilidade seria do tipo fraco sendo paga uma penalização associada a essa mudança.

Capítulo 3

Metodologia

Para obter soluções admissíveis para o problema foi desenvolvido um algoritmo de *Tabu Search* que é composto por duas fases: uma fase construtiva onde é aplicada uma heurística construtiva do tipo *greedy* para criação uma solução admissível inicial, e uma fase melhorativa, onde se processa a pesquisa local de modo a tentar melhorar a solução resultante da primeira fase.

3.1 *Tabu Search*

Uma heurística melhorativa é um processo de pesquisa no conjunto de soluções vizinhas de uma dada solução admissível, onde, caso uma solução melhor seja encontrada, atualiza-se a solução e o processo repete-se. As heurísticas são utilizadas para resolver problemas complexos, pois procuram encontrar soluções admissíveis de boa qualidade, não necessariamente ótimas, de forma eficiente. No entanto, as heurísticas melhorativas procuram uma solução melhor dentro de uma vizinhança específica, que pode não ser a melhor solução globalmente. Portanto, as heurísticas melhorativas podem terminar quando encontram um ótimo local, pois não têm mecanismos para sair do ótimo deste tipo.

Por sua vez, uma meta-heurística é um processo iterativo que coordena uma heurística de forma a produzir soluções de elevada qualidade de forma eficiente, e tem como propósito colmatar as falhas das heurísticas melhorativas. Esta combina diferentes métodos para explorar, de forma diversificada, o espaço de soluções e escapar a ótimos locais.

Tanto as heurísticas como as meta-heurísticas não garantem a solução ótima, no entanto, as meta-heurísticas, como já foi referido, usam técnicas para evitar estagnar em ótimos locais.

Tabu Search, apresentada por Glover et al. (2008), é uma meta-heurística que usa uma estratégia de controlo da pesquisa local. A cada iteração, realiza o movimento da solução corrente para uma outra solução da sua vizinhança, permitindo que esse movimento seja não melhorativo. Esta abordagem, ao permitir que se realizem movimentos para soluções com valor da função objetivo pior que o da solução corrente, evita a estagnação em ótimos locais.

No algoritmo de *Tabu Search* é utilizada uma lista tabu, que serve como uma memória de curto prazo, onde são guardados os movimentos recentes das pesquisas anteriores, de forma a impedir que esses movimentos sejam realizados durante as próximas iterações do algoritmo, para evitar voltar a soluções já pesquisadas. Ou seja, sempre que um movimento é realizado de uma solução corrente para uma na sua vizinhança, o movimento inverso é guardado na lista tabu. Por exemplo, se um nodo é adicionado à solução, nas próximas iterações, esse mesmo nodo não pode ser removido. Assim o algoritmo é impedido de voltar a soluções recentes pois, pegando no exemplo anterior, se determinado nodo foi inserido na solução era porque esse movimento originava a melhor solução entre as soluções das vizinhanças. Logo, se o movimento inverso for realizado (nodo for removido da solução), na próxima iteração, é muito provável que esse mesmo nodo seja novamente escolhido para ser inserido e o algoritmo estaria a visitar sempre as mesmas soluções.

Assim, deve ser definido um tamanho fixo para a lista tabu. A dimensão da lista não deve ser muito pequena nem muito grande. Por um lado, se esta for demasiado pequena, podemos estar a percorrer sempre as mesmas soluções. Por outro, se esta for demasiado grande, pode não ser possível obter mais soluções admissíveis a determinada altura da pesquisa, pois existem muitos movimentos proibidos na lista tabu.

Como critério de paragem do algoritmo de *Tabu Search* podem ser utilizados o número máximo de iterações, o número máximo de iterações consecutivas sem melhorar a melhor solução encontrada até ao momento ou, ainda, um tempo computacional máximo.

Algoritmo 3.1 Tabu Search.

1: Determinar a solução admissível inicial x_0
2: Fazer $x = x_0$, $x^* = x_0$ e $T = \emptyset$
3: **while** Critério de paragem não for satisfeito **do**
4: Pesquisar na vizinhança $V'(x)$ a melhor solução $y' \in V'(x)$
5: Fazer $x = y'$
6: Atualizar T com o movimento que transformou x em y'
7: **if** x melhor que x^* **then**
8: $x^* = x$
9: **end if**
10: **end while**
11: **return** x^*

No Algoritmo 3.1 encontra-se o pseudocódigo do algoritmo de *Tabu Search* onde, x é a solução corrente, V'(x) é a vizinhança de x após a remoção das soluções obtidas por movimentos tabu, x^* é a melhor solução encontrada até ao momento e T representa a lista tabu. Para a execução do algoritmo de *Tabu Search*, é necessário definir uma estrutura de vizinhança.

O primeiro passo do algoritmo consiste em executar uma heurística construtiva de forma a obter uma solução admissível inicial, x_0. De seguida, o pseudocódigo iguala tanto a solução corrente como a melhor solução encontrada até ao momento, à solução admissível inicial, e inicializa a lista tabu como um conjunto vazio. A seguir, o algoritmo de *Tabu Search* iniciará um ciclo onde, em cada iteração, é realizada uma pesquisa exaustiva pela vizinhança restrita da solução corrente, ou seja, a vizinhança que é obtida excluindo as soluções que resultam de movimentos tabu. É então realizado o movimento para a melhor solução encontrada na vizinhança (mesmo que seja uma solução de pior valor que o da solução corrente). O movimento efetuado é inserido na lista tabu. Caso a nova solução seja melhor que a melhor solução encontrada até ao momento, esta é atualizada. O ciclo de pesquisa repete-se até que o critério de paragem seja satisfeito.

3.2 Heurística Construtiva

Uma heurística construtiva permite obter uma solução admissível para um problema, partindo de uma solução vazia. Uma heurística construtiva pode, assim, resultar numa solução vazia caso, dadas as características do problema, esta não consiga construir uma solução admissível. Também as meta-heurísticas podem resultar em soluções vazias.

De modo a obter uma solução admissível inicial para o DPRPP-IC, foi desenvolvida uma heurística construtiva do tipo *greedy*, baseada na conhecida heurística do vizinho mais próximo para o problema do caixeiro viajante. A heurística do vizinho mais próximo (Rosenkrantz et al. (1974)), utilizada para problemas de roteamento nos nodos, cria uma rota com início num nodo i, e vai adicionando à rota o nodo mais próximo do último nodo adicionado até que todos os nodos sejam visitados e que, no fim, regresse ao nodo inicial i (ver, por exemplo, Kizilateş and Nuriyeva (2013)). Por nodo mais próximo entende-se o de menor custo, distância ou tempo a partir de determinado nodo, dependendo do objetivo do problema a ser resolvido.

Para o DPRPP-IC, a heurística construtiva usada é a apresentada no Algoritmo 3.2, onde S representa a solução com início no depósito, S' representa a solução inversa, ou seja, a solução com fim no depósito e D_{ij} é a diferença entre a receita p_{ij} e o custo de travessia c_{ij} do arco (i, j), isto é, D_{ij} = p_{ij} - c_{ij}. Além disso, $C(u, v)$ representa o caminho mais curto (calculado apenas com custos de travessia) com início no nodo u e final no nodo v.

Algoritmo 3.2 Heurística Construtiva.

Require: Instância do DPRPP-IC e o caminho mais curto entre cada par de nodos

1: Selecionar nodo i^* que maximiza D_{0i^*}
2: $S = \{0, i^*\}$
3: max_lucro = $+\infty$
4: **while** max_lucro > 0 **do**

max_lucro = 0
nodo_final = -1
Seja j o último nodo adicionado a S
for $i \in V_1$ **do**
if $D_{ji} >$ max_lucro e o arco (j, i) ainda não foi atravessado **then**
if Não existir Incompatibilidade Forte entre i e os nodos de S **then**
max_lucro = D_{ji}
nodo_final = i
end if
end if
end for
if nodo_final $\neq$ -1 **then**
Adicionar $nodo\_final$ a S a seguir ao nodo j
else if nodo_final = 0 **then**
Adicionar $nodo\_final$ a S a seguir ao nodo j
exit While
end if
end while
if S não termina no depósito **then**
$S' = \{0\}$
Selecionar nodo i^* que maximiza D_{i^*0}
$S' = \{i^*, 0\}$
max_lucro = $+\infty$
while max_lucro > 0 **do**
max_lucro = 0
nodo_final_inverso = -1
Seja j o último nodo adicionado a S'
for $i \in V_1$ **do**
if $D_{ij} >$ max_lucro e o arco (i, j) ainda não foi atravessado **then**
if Não existir Incompatibilidade Forte entre i e os nodos de $S \cup S'$ **then**
max_lucro = D_{ij}
nodo_final_inverso = i
end if
end if
end for
if nodo_final_inverso = 0 **then**
$S = S'$

42: **exit** While
43: **else if** nodo_final_inverso $\neq$ -1 **then**
44: Adicionar $nodo\_final\_inverso$ a S' antes de j
45: **end if**
46: **end while**
47: **if** S não termina no depósito **then**
48: **for** $i =$ tamanho de S até 1 **do**
49: Executar o algoritmo Construção de Caminho($S(i)$, $S'(1)$) e obter $C(S(i), S'(1))$
50: **if** Não existir incompatibilidade forte entre nodos de $C(S(i), S'(1))$ e nodos de $S \cup S'$ **then**
51: Adicionar $C(S(i), S'(1))$ a S
52: Juntar S' a S
53: Exit For
54: **end if**
55: **end for**
56: **end if**
57: **end if**
58: **return** S

A heurística construtiva, para o seu correto funcionamento, necessita de uma instância do DPRPP-IC, bem como do caminho mais curto entre cada par de nodos de V.

O algoritmo começa, então, por selecionar o arco $(0, i^*)$, com início no depósito, que maximiza o valor da solução. Isto é, caso existam arcos lucrativos com início no depósito, é selecionado o arco que representa maior diferença entre a sua receita e custo. Caso contrário, é selecionado o arco de menor custo. A solução é então inicializada com o arco $(0, i^*)$.

Após inicializada a solução com o primeiro arco, a heurística entra num ciclo que, em cada iteração, irá adicionar um arco lucrativo com receita positiva ($D_{ij} > 0$) à solução até que não seja possível adicionar mais arcos deste tipo. Assim, o ciclo inicia atribuindo o valor nulo ao maior lucro, e -1 ao nodo final, para que, caso não seja encontrado nenhum nodo final admissível durante o ciclo, e tendo o nodo final ainda este valor, o ciclo seja terminado.

Em cada iteração, são percorridos todos os arcos lucrativos com início no último nodo j adicionado à solução e selecionado o nodo i cujo arco (j, i) tem maior D_{ji}, ou seja, o de maior diferença entre receita e custo. Para isto, é verificado se aquele arco já foi atravessado, pois nesse caso, a sua travessia já não será lucrativa, e é, ainda, verificada a existência de incompatibilidades fortes entre o nodo final desse arco e os nodos já presentes na solução.

Para a verificação da existência de incompatibilidades fortes, foi criada uma função que deteta a presença deste tipo de incompatibilidade que devolve o valor *True* caso esta exista. Esta função está incluída no Algoritmo A.1: Incompatibilidade Forte presente em anexo. Este

algoritmo percorre todos os nodos presentes na solução e verifica se existe algum nodo que seja fortemente incompatível com o nodo final do novo arco selecionado para ser adicionado à solução. Se tal se verificar, o nodo selecionado é descartado. Caso o arco ainda não tenha sido atravessado e não existam incompatibilidades fortes, o arco é adicionado à solução. Este procedimento repete-se até que não seja possível adicionar mais arcos lucrativos a partir do último nodo da solução.

Se a solução criada pelo ciclo anteriormente descrito não terminar no depósito, é iniciado um ciclo idêntico. No entanto, este opera de maneira inversa, ou seja, começa por adicionar o arco com fim no depósito, com maior diferença entre receita e custo e, de seguida, adiciona os arcos com maior D_{ij} com fim no último nodo adicionado j até que não seja possível adicionar mais arcos com D_{ij} positivo. Tal como o ciclo anterior, antes de adicionar qualquer arco, verifica-se sempre se este já foi atravessado e se pode ser adicionado à solução, isto é, se não viola as restrições de incompatibilidade forte nem com os nodos de S' nem com os nodos de S.

No caso de o ciclo inverso encontrar uma solução admissível com início e fim no depósito, essa será considerada a solução admissível obtida pela heurística construtiva e a solução S do ciclo realizado anteriormente é descartada.

No caso contrário, tendo duas soluções, uma com início no depósito e outra com fim no mesmo, a sua junção é feita adicionando o caminho mais curto do último nodo de S para o primeiro nodo de S', denotado como $C(S(i), S'(1))$, e identificado no algoritmo Construção de Caminho apresentado no Algoritmo 3.5, no Capítulo 3.4. Este algoritmo adiciona à solução o caminho mais curto, apenas considerando os custos de travessia, entre o nodo final da solução com início no depósito, S, e o nodo inicial da solução com destino no depósito, S'. Caso não seja possível juntar este caminho à solução por presença de incompatibilidades fortes entre os nodos deste e os nodos de S e S', o algoritmo irá então descartar o último nodo da primeira solução e verificar se é possível adicionar o caminho mais curto entre o penúltimo nodo da solução com início no depósito, S, e o nodo inicial da solução com fim no depósito, S', e assim, sucessivamente. Findo o algoritmo, este devolve uma solução admissível para o problema. Contudo, esta pode ser uma solução vazia, caso o algoritmo não consiga juntar arcos lucrativos ou não consiga determinar um caminho mais curto "admissível", isto é, caso não exista caminho entre os nodos ou caso o caminho mais curto entre os nodos nas extremidades das soluções viole as restrições de incompatibilidade.

É de notar que, para a seleção de arcos lucrativos e para simplificação do código, não são considerados os custos relacionados com a remoção de incompatibilidades fracas, as penalizações.

Nesta fase construtiva da *Tabu Search*, foi implementada uma simplificação da heurística construtiva proposta, onde foi experimentado construir a solução adicionando arcos lucrativos

com origem no último nodo adicionado. Quando não é possível adicionar mais, junta-se o caminho mais curto entre o último nodo e o depósito. No entanto, pretende-se que a heurística construtiva resulte em soluções robustas com a presença de um maior número de arcos na solução, o que permitirá uma pesquisa local mais alargada.

3.3 Pesquisa Local

A fase melhorativa tem como objetivo obter soluções melhores partindo de uma solução admissível inicial. No entanto, esta fase implica que uma vizinhança da solução admissível inicial seja explorada, e portanto, que a mesma esteja definida. Uma vizinhança é um conjunto de soluções "próximas"de uma solução inicial, obtidas através de pequenas alterações à mesma. Desse modo, as soluções vizinhas são semelhantes à solução inicial.

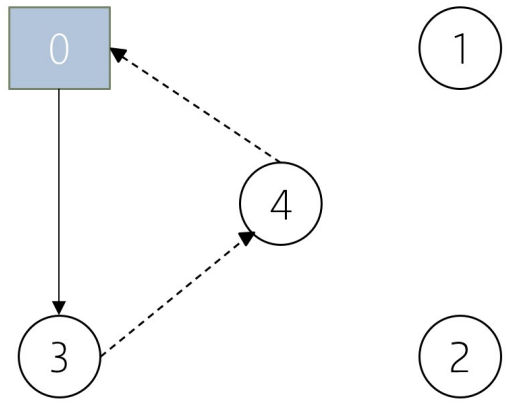
Assim, para o procedimento de pesquisa local serão usadas cinco vizinhanças distintas, que serão apresentadas de seguida.

- Vizinhança $N_i(S)$: Conjunto de soluções admissíveis que são obtidas inserindo um nodo na solução corrente S;
- Vizinhança $N_r(S)$: Conjunto de soluções admissíveis que são obtidas removendo um nodo da solução corrente S;
- Vizinhança $N_t(S)$: Conjunto de soluções admissíveis que são obtidas trocando a posição de dois nodos da solução corrente S.

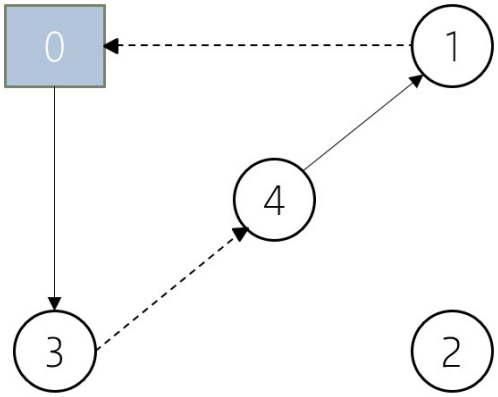
As vizinhanças N_{2i} e N_{2r} apenas serão pesquisadas se não for encontrada nenhuma solução admissível no conjunto das vizinhanças N_i, N_r e N_t.

- Vizinhança $N_{2i}(S)$: Conjunto de soluções admissíveis que são obtidas inserindo um arco na solução corrente S;
- Vizinhança $N_{2r}(S)$: Conjunto de soluções admissíveis que são obtidas removendo um arco da solução corrente S.

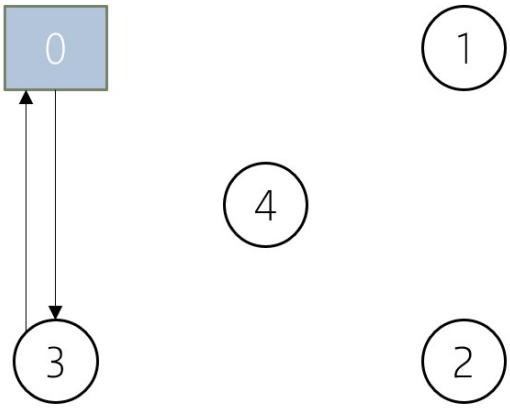
Para exemplificar as três primeiras vizinhanças a pesquisar, a Figura 3.1 apresenta uma solução definida como corrente e um exemplo de uma solução vizinha pertencente a cada vizinhança, utilizando a instância do DPRPP-IC da Figura 2.1.



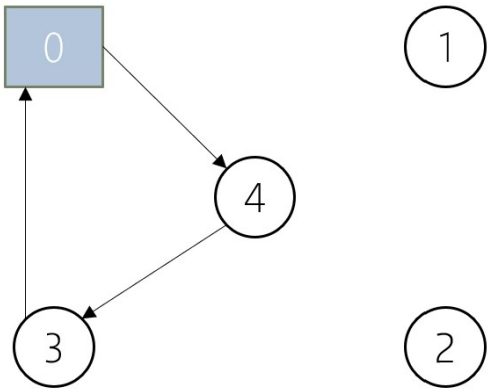
(a) Solução Corrente S.



(b) Solução na vizinhança $N_i(S)$.



(c) Solução na vizinhança $N_r(S)$.

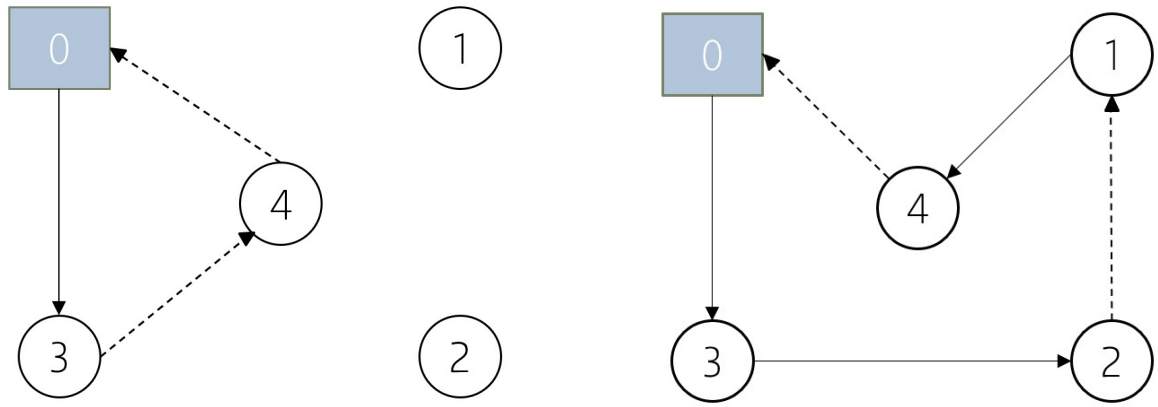


(d) Solução na vizinhança $N_t(S)$.

Figura 3.1: Exemplos das vizinhanças $N_i(S)$, $N_r(S)$ e $N_t(S)$.

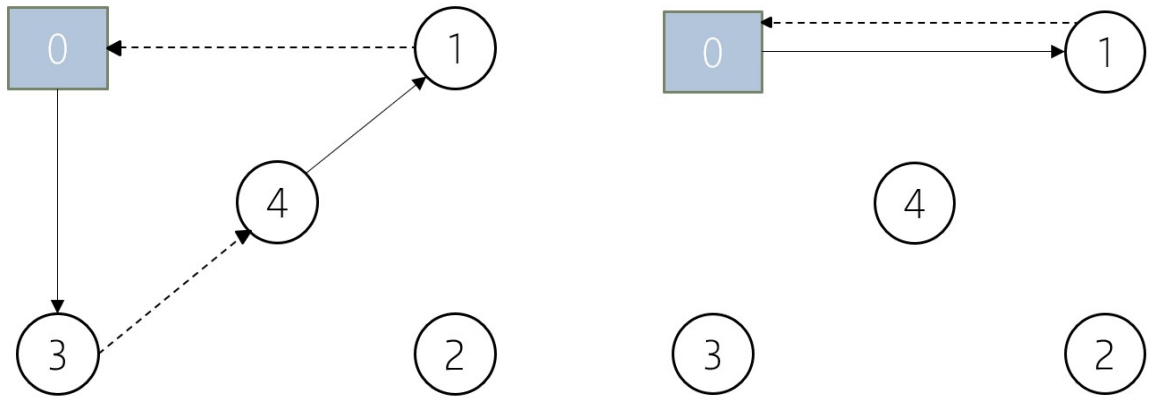
Tomando a solução da Figura 3.1 (a) como a solução corrente S, observa-se que a solução na Figura 3.1 (b) pertence à vizinhança $N_i(S)$ resultando da inserção do nodo 1 na solução S. A solução observada na Figura 3.1 (c) resulta da remoção do nodo 4 da solução corrente S, pertencendo à vizinhança $N_r(S)$. E, por fim, na Figura 3.1 (d) observa-se uma solução da vizinhança $N_t(S)$, que resulta da troca de posição dos nodos 3 e 4.

A pesquisa local passa pela pesquisa exaustiva das três vizinhanças, isto é, é pesquisada a vizinhança $N_i(S) \cup N_r(S) \cup N_t(S)$, e a solução vizinha a selecionar será a melhor encontrada.



(a) Solução Corrente S_1. (b) Solução na vizinhança $N_{2i}(S_1)$.

Figura 3.2: Exemplo da vizinhança $N_{2i}(S_1)$.



(a) Solução Corrente S_2. (b) Solução na vizinhança $N_{2r}(S_2)$.

Figura 3.3: Exemplo da vizinhança $N_{2r}(S_2)$.

A Figura 3.2 (b) apresenta uma solução encontrada na vizinhança N_{2i}, que resulta da inserção do nodo 1 e 2 na solução corrente da Figura 3.2 (a), e a solução da Figura 3.3 (b), tomando como solução corrente a apresentada na Figura 3.3 (a), pertence à vizinhança N_{2r}, resultando da remoção dos nodos 3 e 4.

O critério de escolha da solução vizinha para a qual se vai efetuar o movimento, passa pelo valor incremental. É selecionada a solução vizinha que origina o maior valor incremental dado pela diferença entre as receitas e o custos dos arcos a inserir e remover. Considera-se tanto custos de travessia como as penalizações associadas à remoção de incompatibilidades fracas, que são identificadas pelo Algoritmo A.2 presente em anexo.

No Algoritmo 3.3 apresenta-se o procedimento utilizado para a pesquisa local, que utiliza aos Algoritmos A.3, A.4, A.5, A.6 e A.7 apresentados em anexo.

Algoritmo 3.3 Pesquisa Local.

Require: Instância do DPRPP-IC, Solução admissível inicial S

valor_vizinha = -∞
S' = S
Executar Algoritmo A.3: Pesquisa Vizinhança $N_i(S)$ e atualizar valor_vizinha
Executar Algoritmo A.4: Pesquisa Vizinhança $N_r(S)$ e atualizar valor_vizinha
Executar Algoritmo A.5: Pesquisa Vizinhança $N_t(S)$ e atualizar valor_vizinha
if valor_vizinha = -∞ **then**
    Executar Algoritmo A.6: Pesquisa Vizinhança $N_{2i}(S)$ e atualizar valor_vizinha
    Executar Algoritmo A.7: Pesquisa Vizinhança $N_{2r}(S)$ e atualizar valor_vizinha
    if valor_vizinha $\neq$ -∞ **then**
        Realizar o movimento que gerou o melhor valor_vizinha e obter S' vizinha de S
        Atualizar a lista tabu com o movimento que originou S'
    end if
else
    Realizar o movimento que gerou o melhor valor_vizinha e obter S' vizinha de S
    Atualizar a lista tabu com o movimento que originou S'
end if
return Solução Vizinha S'

Inicialmente, atribui-se o valor de -∞ ao valor_vizinha, que será depois atualizado com o valor incremental das soluções vizinhas admissíveis encontradas. Este valor inicial permite que, durante a pesquisa local, possam ser feitos movimentos para soluções com valor incremental negativo, isto é, soluções piores que a solução corrente. Caso, terminada a pesquisa das vizinhanças, o valor desta variável se mantenha -∞, significa que não existem soluções admissíveis nas vizinhanças. A solução vizinha é igualada a S para que, findo o algoritmo, e não tendo sido encontradas soluções vizinhas admissíveis, S' não seja uma solução vazia.

O algoritmo acima descrito começa por, com recurso ao Algoritmo A.3 presente em anexo, procurar inserir um nodo entre cada par de nodos consecutivos da solução corrente e guarda a melhor solução encontrada na pesquisa dessa vizinhança. De seguida, recorrendo ao Algoritmo A.4, procura remover um nodo da solução corrente e, caso encontre uma solução de melhor valor que a encontrada na vizinhança anterior, essa mesma é guardada e a anterior é descartada. Por fim, através do Algoritmo A.5, percorre a vizinhança N_t, procurando trocar a posição de dois nodos da solução corrente, consecutivos ou não (exceto o depósito). Também neste caso, se for encontrada uma melhor solução que a atual, essa é guardada e a anterior é descartada.

Caso não sejam encontradas soluções admissíveis nas vizinhanças N_i, N_r e N_t, o algoritmo começa então uma pesquisa pelas duas vizinhanças N_{2i} e N_{2r}. Primeiro, pelo Algoritmo

A.6, procura inserir dois nodos em posições consecutivas na solução corrente, guardando novamente a melhor solução encontrada. E, de seguida, procura as soluções vizinhas resultantes da remoção de dois nodos consecutivos da solução corrente, com recurso ao Algoritmo A.7. É de referir que, dado que estas vizinhanças apenas consideram nodos consecutivos, são vizinhanças mais pequenas que as anteriormente referidas. A seleção do movimento a realizar segue a mesma lógica anterior, onde, se uma solução melhor for encontrada em N_{2r} comparativamente a N_{2i}, essa mesma é selecionada. No fim da pesquisa, o movimento guardado é realizado e a lista tabu é atualizada.

Como são pesquisadas várias vizinhanças, é necessário que, para a pesquisa da *Tabu Search*, sejam criadas várias listas tabu. Neste caso, para a vizinhança N_i, N_{2i}, N_r e N_{2r} é criada uma lista conjunta que guarda os movimentos realizados em qualquer uma das vizinhanças para que determinados nodos não sejam nem inseridos nem removidos nas próximas iterações. Já para a vizinhança N_t, a sua lista tabu guarda o par de nodos que foram trocados para que não voltem a ser trocados nas próximas iterações.

Assim, finalizado o algoritmo, este devolve a solução de melhor valor encontrada de entre todas as vizinhanças restritas pesquisadas que, relembramos, pode ser pior que a solução original.

Na procura das soluções vizinhas, é sempre verificado se não existem incompatibilidades fortes entre os nodos da solução e o nodo a inserir ou remover, pois as vizinhanças apenas contêm soluções admissíveis. O mesmo para a troca de nodos, onde, caso se verifique incompatibilidade forte ao trocar a posição de dois nodos na solução, essa solução vizinha não pode ser considerada, pois não é solução admissível.

3.4 Algoritmo de Floyd-Warshall

O algoritmo de Floyd-Warshall calcula o caminho mais curto entre todos os pares de nodos de um grafo (Hougardy (2010)).

Algoritmo 3.4 Algoritmo Floyd-Warshall.

Require: Grafo $G(V, A)$, matriz de custos $c_{i,j}$

```
distancias ← matriz de dimensão V × V inicializada com c_ij
precedencias ← matriz de dimensão V × V inicializada a -1
for i ∈ V do
    distancias[i],[i] = 0
    precedencias[i],[i] = i
end for
for z ∈ V do
    for i ∈ V do
        for j ∈ V do
            distancias[i],[j] = min(distancias[i],[j], distancias[i],[z] + distancias[z],[j])
            if distancias[i],[z] + distancias[z],[j] < distancias[i],[j] then
                precedencias[i],[j] = precedencias[z],[j]
            end if
        end for
    end for
end for
```

Do Algoritmo 3.4 resultam duas matrizes, a matriz de distâncias mínimas (distancias) com a distância de caminho mais curto entre dois nodos, sendo que, para simplificação do algoritmo, apenas foram considerados, neste caso, os custos de travessia, e a matriz de precedências (precedencias), que serve para construir o caminho mais curto entre dois nodos através do Algoritmo 3.5.

Algoritmo 3.5 Construção de Caminho.

Require: Nodo inicial u, nodo final v

1: Caminho(1) $= v$
2: **if** precedencias[u, v] $=$ -1 **then**
3: Caminho = $\emptyset$
4: **else**
5: **while** $u \neq v$ **do**
6: **if** Não existir incompatibilidade forte entre precedente[u,v] e os nodos do Caminho **then**
7: $v =$ precedente[u, v]
8: Adicionar v ao Caminho
9: **else**
10: Caminho = $\emptyset$
11: Exit While
12: **end if**
13: **end while**
14: **end if**
15: **return** Caminho

O Algoritmo 3.5 recebe como argumento o nodo inicial u e o nodo final v do caminho que vai calcular. Se, na matriz de precedências, estiver o valor -1 na linha u, coluna v, então não existe caminho entre esses dois nodos. Caso exista caminho, o algoritmo constrói o caminho mais curto usando essa mesma matriz, partindo do nodo v e retrocedendo através dos precedentes até chegar ao nodo de partida u. Ao construir o caminho, antes de juntar qualquer nodo ao mesmo, é verificado se nenhuma restrição de incompatibilidade é violada entre nodos já no caminho. Na eventualidade de existir incompatibilidades fortes, o algoritmo considera que não existe nenhum caminho entre u e v. Se nenhuma restrição for violada, é devolvido o caminho mais curto entre os nodos u e v.

Capítulo 4

Experiência Computacional

A meta-heurística proposta foi implementada em *Microsoft Visual Basic for Applications* (VBA), versão 7.1, e para a análise da qualidade da solução por ela obtida apenas foram testadas as instâncias da literatura propostas por Colombi et al. (2017), com diferentes características, para as quais existe uma solução ótima conhecida.

Abaixo encontram-se as características do computador, no qual a meta-heurística foi executada, bem como as do software.

- **Edição:** Windows 11 Home
- **Versão:** 22H2
- **Processador:** AMD Ryzen 5 3500U with Radeon Vega Mobile Gfx 2.10 GHz
- **RAM:** 8,00 GB
- **Software:** Microsoft Excel for Microsoft 365 MSO (Version 2308 Build 16.0.16731.20052) 64-bit.

Os resultados obtidos irão ser avaliados em termos de tempo de execução da meta-heurística, em segundos, e da proximidade da melhor solução obtida pela meta-heurística à solução ótima.

Deve referir-se, no entanto, que a comparação feita entre os tempos computacionais da meta-heurística e os tempos computacionais para obtenção do valor ótimo não é justa, visto que os resultados foram obtidos em computadores de características diferentes.

4.1 Análise de Parâmetros

Como mencionado no Capítulo 3.1, é necessário fixar alguns parâmetros para o funcionamento da meta-heurística. Como critério de paragem da *Tabu Search*, será usado o número máximo

de iterações, que deve ser escolhido de modo a garantir uma pesquisa exaustiva do conjunto de soluções. No entanto, este número não deve ser exagerado, pois o objetivo da meta-heurística é obter soluções de boa qualidade de forma eficiente, e um elevado número de iterações contribui para tempos computacionais elevados. Assim, deve ser avaliado a partir de que iteração a solução obtida pela meta-heurística não apresenta melhorias significativas, pois nesse caso as iterações efetuadas apenas estarão a contribuir para tempos computacionais mais demorados. Há que definir, também, o tempo de permanência de um movimento na lista tabu, que, para a meta-heurística proposta, será o número de iterações durante as quais um movimento realizado na solução é considerado tabu. Este parâmetro deve ser definido tendo em conta que um número pequeno de iterações origina a repetição de soluções muito rapidamente e que um número elevado de iterações pode impedir o algoritmo de encontrar mais soluções admissíveis, por ter um grande número de movimentos em lista tabu, e nesse caso, o algoritmo termina.

Para atribuir valores a estes dois parâmetros, foram testados alguns valores em cinco instâncias da literatura com ótimo conhecido, selecionadas aleatoriamente, e anotados os erros relativos das soluções obtidas, Gap, e tempos de execução, em segundos, Tm, apresentados na Tabela 4.1. Os erros relativos são calculados dividindo o valor absoluto da diferença entre o valor ótimo, $Vopt$, e o valor da solução obtida pelo algoritmo *Tabu Search*, Vm, pelo valor do ótimo, multiplicando por 100 para obter o resultado em percentagem. Assim, a formula utilizada é $\frac{Vopt-Vm}{Vopt} \times 100$.

As colunas contêm o número de iterações, I_{tabu}, em que um movimento é mantido tabu, e as linhas o número máximo de iterações, I_{max}, da *Tabu Search*, para cada uma das instâncias. Na segunda coluna, a negrito, encontra-se a identificação da instância K a ser testada.

I_{max}	K	I_{tabu}							
		5		**10**		**15**		**20**	
		$Gap(\%)$	$Tm(s)$	$Gap((\%)$	$Tm(s)$	$Gap(\%)$	$Tm(s)$	$Gap(\%)$	$Tm(s)$
100	**1**	93,63	14,88	93,57	15,56	93,31	17,53	94,1	18,61
	2	97,33	39,08	97,53	31,13	98,36	3,38	98,36	3,47
	3	91,55	27,42	88,87	33,31	88,91	27,76	88,25	37,48
	4	92,22	18,55	92,25	15,98	92,25	15,57	92,25	20,98
	5	97,51	24,78	97,94	20,14	98,28	2,41	98,28	2,43
200	**1**	93,63	30,21	93,57	33,84	93,29	35,71	94,1	40,6
	2	97,33	64,56	97,53	81,54	98,36	3,53	98,36	3,42
	3	91,55	55,89	88,85	79,64	88,79	92,92	88,25	96,56
	4	92,22	53,63	92,25	34,46	92,25	34,05	92,25	47,08
	5	97,51	58,44	97,94	60,41	98,28	2,35	98,28	2,44
400	**1**	93,63	60,68	93,57	78,19	93,12	84,41	94,1	94,12
	2	97,33	152,89	97,53	175,94	98,36	3,38	98,36	3,32
	3	91,55	98,27	88,85	196,23	88,79	239,09	88,25	253,45
	4	92,22	160,67	92,25	83,91	92,25	86,03	92,25	85,93
	5	97,51	143,62	97,94	187,84	98,28	2,34	98,28	2,48
600	**1**	93,63	125,33	93,57	150,14	93,12	190,9	94,1	224,91
	2	97,33	236,46	97,53	248,11	98,36	3,36	98,36	3,69
	3	91,55	144,91	88,85	336,07	88,79	416,37	88,25	455,5
	4	92,22	322,93	92,25	145,72	92,25	140,63	92,25	134,33
	5	97,51	214,99	97,94	387,21	98,28	2,44	98,28	2,39

Tabela 4.1: Análise aos parâmetros.

Para facilitar a interpretação dos resultados para os diferentes valores dos parâmetros, as Tabelas 4.2 e 4.3 apresentam, na segunda coluna, os valores médios dos erros relativos, $\overline{Gap}$, e, na terceira coluna, os tempos computacionais médios, $\overline{Tm}$, das instâncias para cada valor de parâmetros a ser testado.

I_{max}	$\overline{Gap}$	$\overline{Tm}$
100	94,24%	21,80
200	94,23%	45,57
400	94,22%	109,64
600	94,22%	194,32

Tabela 4.2: Análise aos parâmetros: médias para os valores de I_{max}.

I_{tabu}	$\overline{Gap}$	$\overline{Tm}$
5	94,45%	103,66
10	94,03%	119,77
15	94,18%	70,21
20	94,25%	77,7

Tabela 4.3: Análise aos parâmetros: médias para os valores de I_{tabu}.

Por observação da Tabela 4.2, conclui-se que não é necessário um elevado número de iterações para o algoritmo de *Tabu Search* visto que o aumento do mesmo não melhora a solução ou apresenta uma melhoria pouco significativa. Contudo, aumentam os tempos de execução. Por exemplo, por observação da Tabela 4.1, verifica-se que na instância 1, qualquer que seja o I_{tabu}, o valor da melhor solução encontrada após 400 iterações, é o mesmo que o encontrado após 600 iterações, no entanto, o tempo computacional é muito superior neste último. Mais precisamente, o tempo médio após 600 iterações é superior em 90 segundos ao tempo em 400 iterações, isto é, aumenta quase o dobro. Assim sendo, e visto que o valor para I_{max} que apresenta menor erro relativo médio entre os testados é de 400, esse será o escolhido para obter os resultados finais.

Para a duração de um movimento em tabu, foram testadas quatro hipóteses, nomeadamente 5, 10, 15 e 20 iterações. Analisando os resultados obtidos, podemos observar que nas instâncias

2, 4 e 5, qualquer que seja o I_{max}, as soluções de maior valor foram encontradas quando a meta-heurística é testada com um número de iterações em tabu de 5, enquanto que nas instâncias 1 e 3, as soluções de melhor valor são as correspondentes a um I_{tabu} de 15 e 20, respetivamente.

No entanto, pela Tabela 4.3, constata-se que, em termos médios, os valor de I_{tabu} que apresentam resultados ligeiramente melhores são de 10 e 15 iterações, sendo que 10 iterações em tabu apresenta o menor erro relativo médio, contudo, com tempos computacionais mais elevados. Desta forma, foi considerado um número de iterações em tabu de 15, que será suficiente para evitar a repetição de soluções rapidamente.

Assim, perante os resultados obtidos nas cinco instâncias, definiu-se a permanência de um movimento realizado na pesquisa local na lista tabu de 15 iterações e realizar no máximo 400 iterações no algoritmo de *Tabu Search*.

4.2 Análise da *Performance* da *Tabu Search*

De modo a avaliar o impacto da fase de pesquisa local na *Tabu Search*, foram analisados os resultados para as mesmas cinco instâncias selecionadas anteriormente. A Tabela 4.4, regista os valores, Vc, da função objetivo para a solução admissível obtida pela heurística construtiva na segunda coluna, e o valor da função objetivo da solução final, Vm, na terceira coluna. A quarta coluna apresenta a diferença percentual, Gap, entre o valor Vm e o valor Vc, e é calculada através da seguinte fórmula: $\frac{Vm-Vc}{Vm} \times 100$. Foi, ainda, registada, na quinta coluna, a iteração α em que a melhor solução foi encontrada, para se perceber se a pesquisa local efetua muitas iterações sem que nada seja alterado na solução. Na última coluna, a iteração β onde a pesquisa local termina, ou seja, onde nenhuma solução vizinha é encontrada a partir da solução corrente. A primeira coluna identifica a instância K do DPRPP-IC a ser analisada.

K	Vc	Vm	Gap	α	β
1	4798	6972	31,18%	203	400
2	459	824	44,30%	10	13
3	1732	3648	52,52%	114	400
4	3808	5878	35,22%	15	400
5	895	1458	38,61%	11	13

Tabela 4.4: Análise do desempenho da *Tabu Search*.

Por observação da Tabela 4.4, conclui-se que a pesquisa local está efetivamente a encontrar soluções melhores, ou seja, apresenta um bom desempenho para as cinco instâncias analisadas com melhorias significativas no valor das soluções, apresentando valores de Gap entre os 31,18% e 52,52%. No entanto, quanto à iteração em que a melhor solução observa-se que,

para algumas instâncias, são realizadas muitas iterações sem que uma melhor solução seja encontrada. Nas instâncias 1, 3 e 4 a pesquisa local realiza as 400 iterações, mas após a solução de melhor valor ser encontrada, a meta-heurística realiza um elevado número de iterações sem conseguir melhorias, especialmente para a instância 4, onde a melhor solução foi obtida na iteração 15. Já para o caso das instâncias 2 e 5, a pesquisa local termina efetuando muito poucas iterações, ou seja, chegando à iteração 13 a pesquisa local não encontra nenhuma solução vizinha, sendo, portanto, expectável que a meta-heurística resulte em soluções de fraca qualidade para estas instâncias.

Para esta primeira fase da *Tabu Search*, foram testados diversos métodos para a heurística construtiva, por exemplo, numa simplificação da heurística construtiva proposta, foi experimentado construir a solução adicionando arcos lucrativos com origem no último nodo adicionado, e, não conseguindo adicionar mais, juntar o caminho mais curto entre o último nodo e o depósito. No entanto, com este método a meta-heurística, na sua fase melhorativa, parava porque a vizinhança ficava vazia, e portanto, a heurística construtiva proposta, ao possibilitar a presença de mais arcos na solução, permite uma vizinhança maior e consequentemente, um maior número de soluções vizinhas admissíveis. Foram, ainda, testados procedimentos como selecionar aleatoriamente o primeiro arco da solução, desde que com origem no depósito, ou após a pesquisa local não encontrar novas soluções vizinhas, iniciar novamente a heurística construtiva mas dando prioridade a arcos que não estiveram na solução da primeira heurística construtiva realizada. No entanto, após análise de algumas instâncias, estas abordagens não melhoravam a solução e apenas resultavam em tempos computacionais maiores, pelo que a heurística construtiva utilizada para o presente problema foi a descrita anteriormente.

4.3 Resultados Gerais

A implementação da meta-heurística, como mencionado anteriormente, fez-se com recurso ao Visual Basic for Applications (VBA), ferramenta do Excel. Todas as instâncias de referência utilizadas têm 500 nodos e o nodo 0 é considerado o depósito. O número de arcos, pares de nodos com incompatibilidades fraca e pares de nodos com incompatibilidade forte diferem de instância para instância.

Da execução do código da meta-heurística, resulta uma tabela, em Excel, com cabeçalho a cor azul dividida em seis colunas: na primeira encontra-se a identificação da instância, na segunda o valor da solução obtida pela meta-heurística proposta, na terceira o valor ótimo da instância, na quarta e quinta coluna registam-se os tempos computacionais, em segundos, necessários à resolução do problema através da meta-heurística e dos métodos exatos, respetivamente, e a última coluna apresenta o erro relativo, entre o valor da solução obtida e o valor ótimo. Tanto os valores ótimos como os seus tempos computacionais foram retirados do artigo

de Colombi et al. (2017).

O VBA recebe os ficheiros que contêm os dados das instâncias e executa a meta-heurística para cada uma, guardando os resultados obtidos na tabela criada, anteriormente descrita, antes de avançar para a instância seguinte.

Instância	Valor Obtido	Valor Ótimo	Tempo Solução	Tempo Ótimo	Erro Relativo
Instância 1	421	5258	2,3	76,55	91,99%
Instância 2	418	4412	2,29	573,34	90,53%
Instância 3	5246	64772	76,7	10,58	91,90%
Instância 4	191	4610	0,33	559,89	95,86%
Instância 5	21	9458	0,11	1989,47	99,78%
Instância 6	676	11488	2,4	134,8	94,12%
Instância 7	654	9071	2,37	1050,57	92,79%
Instância 8	611	10493	2,3	63,31	94,18%
Instância 9	590	9093	2,29	1334,35	93,51%
Instância 10	52	16866	0,11	2016,73	99,69%
Instância 11	418	3395	2,3	929,2	87,69%
Instância 12	461	3046	2,31	3530,97	84,87%
Instância 13	900	8527	4,55	73,04	89,45%
Instância 14	694	7129	2,31	2406,85	90,27%
Instância 15	1235	13557	4,21	2817,16	90,89%
Instância 16	373	995	2,26	845,54	62,51%
Instância 17	414	4730	2,3	713,97	91,25%
Instância 18	1344	26265	4,63	11,61	94,88%
Instância 19	747	14200	4,61	25,79	94,74%
Instância 20	827	10960	4,57	58,86	92,45%
Instância 21	501	16298	73,2	152,02	96,93%
Instância 22	235	25611	2,89	403,82	99,08%
Instância 23	3580	50382	69,1	2490,12	92,89%
Instância 24	1340	24628	5,04	8,95	94,56%
Instância 25	613	48287	2,46	10,47	98,73%
Instância 26	1237	20380	4,57	42,54	93,93%
Instância 27	1162	21600	4,94	16,21	94,62%
Instância 28	128	40809	0,17	123,07	99,69%
Instância 29	4045	63899	74,12	2,62	93,67%
Instância 30	734	38562	2,01	0,99	98,10%
Instância 31	187	21915	0,31	20,82	99,15%
Instância 32	169	19634	0,32	244,75	99,14%
Instância 33	207	16531	0,32	2219,94	98,75%
Instância 34	297	36687	0,32	7,94	99,19%
Instância 35	272	33114	0,31	6,43	99,18%

Figura 4.1: Output do programa: Instâncias 1 a 35.

Instância	Valor Obtido	Valor Ótimo	Tempo Solução	Tempo Ótimo	Erro Relativo
Instância 36	284	28026	0,32	379,32	98,99%
Instância 37	265	26605	0,35	1191,99	99%
Instância 38	51	36051	0,19	14,32	99,86%
Instância 39	87	32099	0,19	244,75	99,73%
Instância 40	77	26623	0,18	124,29	99,71%
Instância 41	57	23839	0,19	238,54	99,76%
Instância 42	138	56737	0,19	8,86	99,76%
Instância 43	153	51789	0,19	8,92	99,70%
Instância 44	118	43380	0,21	301,02	99,73%
Instância 45	-9	11881	0,11	771,27	100,08%
Instância 46	-4	11018	0,12	640,82	100,04%
Instância 47	57	22017	0,12	126,05	99,74%
Instância 48	42	19856	0,12	148,53	99,79%
Instância 49	440	32886	2,45	13,65	98,66%
Instância 50	219	25344	1,57	3068,65	99,14%
Instância 51	588	53404	2,5	5,52	98,90%
Instância 52	623	41460	2,48	68,22	98,50%
Instância 53	888	54754	2,37	0,58	98,38%
Instância 54	635	42556	1,35	1,11	98,51%
Instância 55	864	40203	2,14	28,55	97,85%
Instância 56	631	77270	0,93	0,75	99,18%
Instância 57	1347	68131	2,21	0,97	98,02%
Instância 58	1111	64555	1,35	4,21	98,28%
Instância 59	744	12796	4,96	6,82	94,19%
Instância 60	830	11844	5,1	73,88	92,99%
Instância 61	3512	44026	226,64	0,95	92,02%
Instância 62	3485	39844	166,9	109,53	91,25%
Instância 63	4992	70368	243,24	0,56	92,91%
Instância 64	5963	64854	279,32	2,71	90,81%
Instância 65	5200	55716	112,39	0,75	90,67%
Instância 66	4603	52512	245,59	194,77	91,23%
Instância 67	6192	58502	81,12	0,78	89,42%
Instância 68	8732	99964	94,77	0,74	91,26%
Instância 69	5106	91675	78,25	0,62	94,43%
Instância 70	5031	79205	72,41	90,4	93,65%
Instância 71	1033	22726	71,69	1,12	95,45%
Instância 72	441	20697	1,92	1,67	97,87%
Instância 73	506	17247	3,04	9,75	97,07%
Instância 74	691	35643	3,04	0,7	98,06%
Instância 75	715	29735	3,01	4,29	97,60%
Instância 76	1614	29015	124,82	6,27	94,44%
Instância 77	2042	57802	15,34	66,47	96,47%
Instância 78	2078	51464	173,5	42,78	95,96%
Instância 79	1901	43117	163,78	245,73	95,59%
Instância 80	2972	88496	135,46	0,78	96,64%
Instância 81	2918	80455	137,7	0,81	96,37%
Instância 82	1482	68675	0,49	4,32	97,84%
Instância 83	4296	86149	8,42	0,78	95,01%
Instância 84	3413	78551	0,94	1,05	95,66%

Figura 4.2: Output do programa: Instâncias 36 a 84.

Instância	Valor Obtido	Valor Ótimo	Tempo Solução	Tempo Ótimo	Erro Relativo
Instância 85	3029	68025	0,93	6,1	95,55%
Instância 86	4898	129501	0,97	0,84	96,22%
Instância 87	4496	118992	0,91	0,84	96,22%
Instância 88	5460	105532	9,14	1,22	94,83%
Instância 89	6972	101368	85	6,51	93,12%
Instância 90	2671	45770	57,66	92,9	94,16%
Instância 91	1458	84959	2,4	0,64	98,28%
Instância 92	2956	70965	90,94	1,87	95,83%
Instância 93	5878	75824	80,23	432,45	92,25%
Instância 94	3648	32530	241,83	3070,85	88,79%
Instância 95	824	50218	2,23	0,72	98,36%

Figura 4.3: Output do programa: Instâncias 854 a 95.

Observa-se que, por um tempo computacional substancialmente menor, é possível encontrar uma solução admissível para o problema, resgistando-se um tempo computacional médio de 35,92 segundos. No entanto, as soluções obtidas são de fraca qualidade. Os valores obtidos pela meta-heurística representam erros relativos muito elevados, tendo um erro relativo médio de 95,38%. Estes encontram-se, assim, muito "longe" do valor ótimo obtido por métodos exatos, pelo que, a meta-heurística proposta não tem uma boa *performance* para o problema em questão. Deve notar-se ainda que, em certas instâncias, como por exemplo nas instâncias 63, 64 e 65, a meta-heurística não se revela útil pois para obtenção de uma solução admissível que se revela muito "afastada" do ótimo, apresenta um tempo computacional muito superior ao tempo computacional gasto para encontrar a solução ótima.

4.4 Impacto do Número de Arcos Lucrativos

Para uma análise mais detalhada dos resultados, na Tabela 4.5 agruparam-se as instâncias pelo rácio, r, entre o número de arcos lucrativos e o número de arcos não lucrativos, isto é $\frac{\#\ arcos\ lucrativos}{\#\ arcos\ não\ lucrativos}$, de modo a tentar perceber o seu impacto nos resultados da meta-heurística. A primeira coluna indica o rácio r, ordenado de forma crescente e a segunda coluna contém o número de instâncias, n, com determinado rácio. A terceira coluna regista os erros relativos médios, $\overline{Gap}$, das instâncias daquele grupo, e a quarta e quinta colunas registam o tempo de execução médio, em segundos, tanto para a meta-heurística, $\overline{Tm}$, como para obtenção do valor ótimo, $\overline{Topt}$, respetivamente.

r	n	$\overline{Gap}$	$\overline{Tm}$	$\overline{Topt}$
23%	12	89,13%	2,15	1018,28
28%	6	99,85%	0,12	948,81
30%	10	93,27%	4,72	313,49
33%	8	96,94%	35,34	22,1
61%	7	99,06%	0,32	581,6
80%	6	98,84%	2,39	595,06
95%	7	91,1%	216,56	482,87
105%	8	96,11%	96,86	56,96
121%	8	99,74%	0,19	132,97
212%	8	98,36%	1,87	4,69
234%	7	92,51%	78,57	431,1
281%	8	94,81%	22,88	3,49

Tabela 4.5: Resultados agrupados por r.

Constata-se que o grupo de instâncias com um erro relativo ligeiramente melhor, é o correspondente ao menor rácio r, isto é, r = 23%. Também nos tempos computacionais não é possível identificar um padrão, pois podemos observar que, por exemplo, os resultados obtidos por métodos exatos, para rácio r = 23%, são obtidos por tempos computacionais mais elevados que qualquer um dos outros rácios. Assim, conclui-se que não existe qualquer relação entre os resultados da meta-heurística e a percentagem de arcos lucrativos da instância a ser testada, tanto a nível de erro relativo como de tempo computacional.

4.5 Impacto das Incompatibilidades

Para analisar o impacto das incompatibilidades presentes nas instâncias, estas foram, então, divididas por instâncias sem incompatibilidades fortes entre nodos, instâncias sem incompatibilidades fracas entre nodos e instâncias com ambos os tipos de incompatibilidades. Comecemos por analisar os resultados para cada tipologia de incompatibilidades presentes nas instâncias e, no fim, será feita a comparação entre as três.

Observemos primeiro as instâncias com apenas incompatibilidades fortes, sem nenhuma incompatibilidade do tipo fraco. Na literatura existe somente uma instância deste tipo para a qual o ótimo é conhecido, o que pode indicar que é difícil de resolver o problema com estas características por métodos exatos.

A Tabela 4.6 apresenta, então, os resultados obtidos nessa instância, onde na primeira coluna está o número de pares de nodos com incompatibilidade forte entre si, IC_{forte}, a segunda coluna apresenta o erro relativo, Gap da solução obtida pela meta-heurística, e na terceira e

quarta colunas apresentam-se os tempos computacionais para a meta-heurística, Tm e para a solução ótima, $Topt$

IC_{forte}	Gap	Tm	$Topt$
1724	62,51%	2,26	845,54

Tabela 4.6: Resultados para instância com apenas incompatibilidades fortes.

Constata-se que, por um tempo computacional muito inferior, é possível encontrar uma solução admissível, no entanto, o seu valor encontra-se afastado do valor ótimo. Contudo, deve realçar-se o facto de esta instância apresentar um erro relativo bastante inferior ao registado nas outras instâncias.

De seguida, avaliam-se os resultados obtidos nas instâncias sem incompatibilidades fortes entre nodos e com incompatibilidades fracas. Aqui, são analisadas 89 instâncias agrupadas por intervalos do número de pares de nodos fracamente incompatíveis, apresentados na primeira coluna da Tabela 4.7. A apresentação da Tabela 4.7 é semelhante à da Tabela 4.5.

$Intervalo$	n	$\overline{Gap}$	$\overline{Tm}$	$\overline{Topt}$
$[1; 500]$	6	93,49%	1,99	480,48
$[501; 1000]$	23	97,59%	5,06	378,83
$[1001; 1500]$	15	96,6%	69,58	256,51
$[1501; 2000]$	21	95,96%	52,43	264,62
$[2001; 3000]$	11	94,34%	24,40	236,88
$[3001; 10000]$	11	96,62%	62,94	238,36
$[10000; 10743]$	2	92,51%	80,85	8,55

Tabela 4.7: Resultados agrupados pelo intervalo de pares de nodos com incompatibilidade fraca.

Não parece existir qualquer tipo de relação entre os resultados obtidos pela meta-heurística e o número de nodos com incompatibilidade fraca na instância. O último grupo foi o que obteve um erro relativo ligeiramente menor, no entanto, é o grupo que apresenta o maior número de pares de nodos com incompatibilidade fraca, indicando que não existe relação entre este fator e os resultados obtidos. Quanto aos tempos computacionais da meta-heurística, parece haver uma relação de proporcionalidade direta entre o número de nodos fracamente incompatíveis e os tempos de execução, sendo que o aumento do primeiro fator parece provocar o aumento do outro. No caso dos tempos computacionais para a obtenção do ótimo, verifica-se o inverso, pois um maior número de pares de nodos fracamente incompatíveis origina tempos computacionais menos demorados.

Mais uma vez, deve realçar-se que a meta-heurística não se revela útil para as instâncias do último grupo, pois é de esperar que uma meta-heurística resolva um problema de otimização em menor tempo que os métodos exatos, mesmo que não encontre o ótimo, e tal não se verifica.

Por último, apresentam-se os resultados obtidos para as instâncias tanto com incompatibilidades fracas entre nodos como incompatibilidades fortes, para o qual apenas existem cinco instâncias com ótimo conhecido. Para cada instância é apresentado na Tabela 4.8, o número de pares de nodos fortemente incompatíveis, IC_{forte}, na segunda coluna, e o número de pares de nodos fracamente incompatíveis, IC_{fraca}, na terceira coluna. A restante tabela apresenta-se organizada de forma semelhante à Tabela 4.5.

K	IC_{forte}	IC_{fraca}	$\overline{Gap}$	$\overline{Tm}$	$\overline{Topt}$
11	155	173	87,69%	2,3	929,2
14	155	186	90,27%	2,31	2406,85
12	173	185	84,87%	2,31	3530,97
13	292	280	89,45%	4,55	73,04
15	307	282	90,89%	4,21	2817,16

Tabela 4.8: Resultados para instâncias com incompatibilidades fortes e fracas.

Neste caso, a meta-heurística apresenta valores da função objetivo ligeiramente melhores, embora sejam instâncias que têm nodos com incompatibilidade fortes, o que proíbe certos movimentos na solução.

As instâncias 13 e 15 apresentam mais nodos com incompatibilidades e têm ainda mais incompatibilidades fortes que fracas, o que impede mais movimentos durante tanto a fase construtiva como melhorativa. Tal poderia justificar erros relativos ligeiramente superiores comparativamente às outras instâncias desta tabela pois as vizinhanças têm poucas soluções e a *Tabu Search* terminaria muito cedo. Contudo, esta diferença não se verifica.

Em termos de tempos computacionais, a meta-heurística regista tempos de execução significativamente melhores nestas instâncias.

Comparando as tabelas apresentadas anteriormente, conclui-se que a meta-heurística parece resultar em soluções ligeiramente melhores para instâncias com incompatibilidades fortes entre nodos. De facto, as instâncias com erros relativos ligeiramente inferiores são as que apenas apresentam incompatibilidade forte entre nodos ou que apresentam ambos os tipos de incompatibilidade. Tal não era previsível pois este tipo de incompatibilidade impede a seleção de certos arcos lucrativos para a solução. No entanto, esse impedimento resulta em valores ótimos inferiores e, portanto, a meta-heurística origina uma solução admissível de valor mais "próximo" do ótimo. Contudo, esta diferença não é significativa ao ponto de se estabelecer uma relação entre os resultados e a presença de nodos com incompatibilidades fortes.

Capítulo 5

Conclusão

Nesta dissertação foi proposta uma meta-heurística *Tabu Search* para a resolução do problema do carteiro rural orientado com lucros e incompatibilidades (DPRPP-IC). A meta-heurística é composta por duas fases. Uma fase construtiva, onde é aplicada uma heurística do tipo *greedy*, com objetivo de criar uma solução admissível inicial, e uma segunda fase, onde é realizada a pesquisa local de modo a melhorar a solução inicial obtida. A meta-heurística proposta foi implementada através da ferramenta *Visual Basic for Applications* (VBA) do Excel.

A meta-heurística inicia com uma heurística construtiva, baseada no método do vizinho mais próximo, inserindo na solução os arcos de maior D_{ij}, que é a diferença entre a receita e o custo de travessia do arco (i, j). Após a construção da solução admissível inicial, é iniciada a pesquisa local da *Tabu Search*, que explora cinco vizinhanças restritas, N_i, N_r, N_t, N_{2i} e N_{2r}, sendo que as últimas duas apenas são pesquisadas caso nenhuma solução admissível seja encontrada nas primeiras três. Na fase melhorativa, a pesquisa local da meta-heurística proposta, realiza 400 iterações, efetuando, em cada, o movimento para a melhor solução encontrada nas vizinhanças da solução corrente. Cada movimento realizado é considerado tabu durante as 15 iterações seguintes.

Pela análise dos resultados de cinco instâncias aleatoriamente selecionadas, esta segunda fase mostrou que, efetivamente, na pesquisa local é encontrada uma solução com melhor valor da função objetivo, com melhorias significativas em todas as instâncias analisadas. Podemos assim concluir que usar a *Tabu Search* é vantajoso comparativamente a usar apenas a heurística construtiva.

Com o intuito de avaliar o desempenho da *Tabu Search* proposta para o problema em estudo, foram testadas as 95 instâncias da literatura, para as quais o valor ótimo é conhecido, com características diferentes.

A meta-heurística resultou em soluções de fraca qualidade para todas as instâncias na qual foi testada, com erros relativos significativamente elevados, ou seja, os valores da função objetivo obtidos encontram-se muito distantes do valor ótimo do problema. Tendo, ainda, sido feita

uma avaliação quanto aos fatores que possam influenciar a *performance* da meta-heurística, conclui-se que não parece existir relação entre os resultados obtidos e o número de arcos lucrativos, nem entre os resultados obtidos e as incompatibilidades estabelecidas.

Constata-se, também, que, para algumas instâncias, a pesquisa local realiza muitas iterações no vazio, ou seja, encontra a melhor solução logo nas primeiras iterações e prossegue com a pesquisa nas vizinhanças, realizando sempre movimentos sem que nenhuma solução de melhor valor seja encontrada, o que contribui para tempos computacionais maiores.

Por fim, conclui-se, também, que a pesquisa local, para algumas instâncias, termina ao fim de poucas iterações. Isto é, o procedimento não encontra nenhuma solução vizinha admissível a partir da solução corrente, o que pode ser justificado pelo facto de existirem muitos movimentos na lista tabu, ou ainda, pelo facto de a solução corrente conter poucos nodos, e não existir nenhuma solução vizinha.

Em suma, a meta-heurística demonstrou uma fraca desempenho para o problema do DPRPP-IC, com uma diferença significativa entre os resultados por ela obtidos e os ótimos obtidos por métodos exatos. Contudo, deve realçar-se que, com valores diferentes para os parâmetros da meta-heurística, os resultados obtidos poderiam ser, também, diferentes.

Como trabalho futuro, o método de pesquisa local poderá ser melhorado. Poderá ser introduzida uma estratégia para que esta não bloqueie, ou seja, para que a meta-heurística não termine quando a pesquisa local não encontra mais soluções admissíveis. Por exemplo, quando não se encontra nenhuma solução nas vizinhanças, poderá ser novamente construída uma solução admissível inicial para recomeçar o processo ou mesmo permitir que um movimento tabu seja realizado de forma a que a pesquisa local possa continuar. Também a pesquisa poderia ser alterada, nomeadamente, a forma como a mesma seleciona os movimentos a realizar. Por exemplo, na meta-heurística proposta, a pesquisa local percorre todas as vizinhanças e realiza o movimento para a solução de melhor valor encontrada. No entanto, resultados diferentes poderiam ser obtidos se, em cada iteração, a pesquisa local terminasse assim que fosse visitada uma solução melhor, realizando uma pesquisa incompleta. Assim, este método já não faria uma pesquisa exaustiva pelas vizinhanças o que contribuiria para uma maior eficiência computacional, e, potencialmente, levaria a diferentes soluções visitadas durante a pesquisa. Poderia, ainda, ser pesquisado o ótimo local da cada vizinhança, ou seja, seria realizado o movimento para o ótimo local da vizinhança a ser explorada e essa seria a solução corrente da vizinhança seguinte.

Outra forma de tentar melhorar os resultados obtidos seria através de uma heurística construtiva que resultasse em soluções iniciais com mais arcos para que a pesquisa local visitasse vizinhanças mais alargadas, ou mesmo mudar a estrutura de vizinhanças a explorar na mesma.

Poderia, também, ser vantajoso observar as soluções ótimas das instâncias para as poder comparar estruturalmente com as soluções obtidas pela meta-heurística e perceber quais as

diferenças entre as mesmas. Assim, poderiam ser desenvolvidas vizinhanças mais precisas que cobrissem essas diferenças.

Bibliografia

Bernardino, R. and Paias, A. (2022). The family traveling salesman problem with incompatibility constraints. *Networks*, 79(1):47–82.

Colombi, M., Corberán, Á., Mansini, R., Plana, I., and Sanchis, J. M. (2017). The directed profitable rural postman problem with incompatibility constraints. *European Journal of Operational Research*, 261(2):549–562.

Colombi, M. and Mansini, R. (2014). New results for the directed profitable rural postman problem. *European Journal of Operational Research*, 238(3):760–773.

Corberán, Á. and Laporte, G. (2015). *Arc Routing: Problems, Methods, and Applications*. SIAM.

Factorovich, P., Méndez-Díaz, I., and Zabala, P. (2020). Pickup and delivery problem with incompatibility constraints. *Computers & Operations Research*, 113:104805.

Gendreau, M., Manerba, D., and Mansini, R. (2016). The multi-vehicle traveling purchaser problem with pairwise incompatibility constraints and unitary demands: A branch-and-price approach. *European Journal of Operational Research*, 248(1):59–71.

Glover, F., Laguna, M., and Marti, R. (2008). *Tabu Search*, volume 16.

Gmira, M., Gendreau, M., Lodi, A., and Potvin, J.-Y. (2021). Tabu search for the time-dependent vehicle routing problem with time windows on a road network. *European Journal of Operational Research*, 288(1):129–140.

Hougardy, S. (2010). The floyd–warshall algorithm on graphs with negative cycles. *Information Processing Letters*, 110(8-9):279–281.

Kizilateş, G. and Nuriyeva, F. (2013). On the nearest neighbor algorithms for the traveling salesman problem. In *Advances in Computational Science, Engineering and Information Technology: Proceedings of the Third International Conference on Computational Science, Engineering and Information Technology (CCSEIT-2013), KTO Karatay University, June 7-9, 2013, Konya, Turkey-Volume 1*, pages 111–118. Springer.

Orloff, C. S. (1974). A fundamental problem in vehicle routing. *Networks*, 4(1):35–64.

Rosenkrantz, D. J., Stearns, R. E., and Lewis, P. M. (1974). Approximate algorithms for the traveling salesperson problem. In *15th Annual Symposium on Switching and Automata Theory (swat 1974)*, pages 33–42.

Yu, M., Jin, X., Zhang, Z., Qin, H., and Lai, Q. (2019). The split-delivery mixed capacitated arc-routing problem: Applications and a forest-based tabu search approach. *Transportation Research Part E: Logistics and Transportation Review*, 132:141–162.

Apêndice A

Algoritmos Auxiliares à Meta-heurística

O Algoritmo A.1 é utilizado para verificar a existência de incompatibilidade forte. Este recebe um nodo e percorre toda a solução verificando se existem nodos fortemente incompatíveis com este. Caso exista, o algoritmo devolve o valor "*True*", Caso contrário devolve "*False*".

Algoritmo A.1 Incompatibilidade Forte.

Require: Solução S, nodo $i \in V_1$

```
1: Incompatibilidade Forte = False
2: for n de 1 até tamanho da solução - 1 do
3:     if {S(n), i} ∈ E_1 then
4:         Incompatibilidade Forte = True
5:     end if
6: end for
7: return Incompatibilidade Forte
```

O Algoritmo A.2 é utilizado para calcular a componente das penalizações no valor incremental de uma solução vizinha. Este recebe um arco lucrativo e percorre toda a solução verificando se existem nodos fracamente incompatíveis com o nodo origem do arco e se os arcos com início nesses nodos são lucrativos. Caso se verifique, o algoritmo devolve o somatório das penalizações a pagar para remover todas as incompatibilidades fracas que existam com esse nodo. Se não existir incompatibilidades deste tipo, o algoritmo devolve o valor 0.

Algoritmo A.2 Incompatibilidade Fraca.

Require: Solução S, arco $(i, j) \in R$

```
Penalização = 0
for n de 1 até tamanho da solução - 1 do
    if {S(n), i} ∈ E2 then
        if (S(n), S(n+1)) ∈ R then
            Penalização = Penalização + γ_{S(n),i}
        end if
    end if
end for
return Penalização
```

O Algoritmo A.3 procura a solução vizinha de melhor valor na vizinhança N_i da solução admissível que lhe é fornecida, tentando inserir um nodo entre cada par de nodos consecutivos presentes na solução.

Algoritmo A.3 Pesquisa Vizinhança N_i.

Require: Instância do DPRPP-IC, Solução admissível inicial S, lista tabu T

```
for i in V do
    for n de 1 até tamanho da solução - 1 do
        Calcular valor_incremental de inserir i entre S(n) e S(n+1)
        if valor_incremental > valor_vizinha, não existir incompatibilidades fortes e i ∉ T
        then
            Guardar o movimento
            valor_vizinha = valor_incremental
        end if
    end for
end for
return Movimento, valor_vizinha
```

O Algoritmo A.4 procura a solução vizinha de melhor valor na vizinhança N_r da solução admissível que lhe é fornecida, tentando remover um nodo da solução.

Algoritmo A.4 Pesquisa Vizinhança N_r.

Require: Instância do DPRPP-IC, Solução admissível inicial S, lista tabu T

1: **for** $n \in S \setminus \{0\}$ **do**
2: Calcular valor_incremental de remover n da solução S
3: **if** valor_incremental $>$ valor_vizinha, não existir incompatibilidades fortes e $n \notin T$ **then**
4: Guardar o movimento
5: valor_vizinha = valor_incremental
6: **end if**
7: **end for**
8: **return** Movimento, valor_vizinha

O Algoritmo A.5 procura a solução vizinha de melhor valor na vizinhança N_t da solução admissível que lhe é fornecida, tentando trocar a posição de dois nodos da solução.

Algoritmo A.5 Pesquisa Vizinhança N_t.

Require: Instância do DPRPP-IC, Solução admissível inicial S, lista tabu T

1: **for** $n \in S \setminus \{0\}$ **do**
2: **for** $z \in S \setminus \{0\}$ **do**
3: Calcular valor_incremental de trocar a posição do nodo n e z
4: **if** valor_incremental $>$ valor_vizinha, não existir incompatibilidades fortes e $\{n, z\} \notin T$ **then**
5: Guardar o movimento
6: valor_vizinha = valor_incremental
7: **end if**
8: **end for**
9: **end for**
10: **return** Movimento, valor_vizinha

O Algoritmo A.6 procura a solução vizinha de melhor valor na vizinhança N_{2i} da solução admissível que lhe é fornecida, tentando inserir dois nodos em posições consecutivas entre cada par de nodos presentes na solução.

Algoritmo A.6 Pesquisa Vizinhança N_{2i}.

Require: Instância do DPRPP-IC, Solução admissível inicial S, lista tabu T

1: **for** i **in** V **do**
2: **for** j **in** V **do**
3: **for** n de 1 até tamanho da solução - 1 **do**
4: Calcular valor_incremental de inserir (i, j) entre $S(n)$ e $S(n + 1)$
5: **if** valor_incremental $>$ valor_vizinha, não existir incompatibilidades fortes e $\{i, j\} \notin T$ **then**
6: Guardar o movimento
7: valor_vizinha = valor_incremental
8: **end if**
9: **end for**
10: **end for**
11: **end for**
12: **return** Movimento, valor_vizinha

O Algoritmo A.7 procura a solução vizinha de melhor valor na vizinhança N_{2r} da solução admissível que lhe é fornecida, tentando remover dois nodos consecutivos da solução.

Algoritmo A.7 Pesquisa Vizinhança N_{2r}.

Require: Instância do DPRPP-IC, Solução admissível inicial S, lista tabu T

1: **for** n de 1 até tamanho da solução - 2 **do**
2: Calcular valor incremental de remover $S(n)$ e $S(n + 1)$ da solução S
3: **if** valor_incremental $>$ valor_vizinha, não existir incompatibilidades fortes e $\{S(n), S(n + 1)\} \notin T$ **then**
4: Guardar o movimento
5: valor_vizinha = valor_incremental
6: **end if**
7: **end for**
8: **return** Movimento, valor_vizinha