UNIVERSIDADE ESTADUAL DE CAMPINAS

Instituto de Matemática, Estatística e Computação Científica

RAMOM SANTANA REBOUÇAS

Problema do Caixeiro Viajante com Coleta de Prêmios e Janelas de Tempo

Campinas

2016

Ramom Santana Rebouças

Problema do Caixeiro Viajante com Coleta de Prêmios e Janelas de Tempo

Dissertação apresentada ao Instituto de Matemática, Estatística e Computação Científica da Universidade Estadual de Campinas como parte dos requisitos exigidos para a obtenção do título de Mestre em Matemática Aplicada.

Orientador: Francisco de Assis Magalhães Gomes Neto

Este exemplar corresponde à versão final da Dissertação defendida pelo aluno Ramom Santana Rebouças e orientada pelo Prof. Dr. Francisco de Assis Magalhães Gomes Neto.

Campinas

2016

Agência(s) de fomento e nº(s) de processo(s): CNPq

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca do Instituto de Matemática, Estatística e Computação Científica
Maria Fabiana Bezerra Muller - CRB 8/6162

Rebouças, Ramom Santana, 1989-
R242p Problema do caixeiro viajante com coleta de prêmios e janelas de tempo / Ramom Santana Rebouças. – Campinas, SP : [s.n.], 2016.

Orientador: Francisco de Assis Magalhães Gomes Neto.
Dissertação (mestrado) – Universidade Estadual de Campinas, Instituto de Matemática, Estatística e Computação Científica.

1. Problema do caixeiro viajante. 2. Janela de tempo. 3. Otimização combinatória. 4. Meta-heurísticas. 5. Heurística. I. Gomes Neto, Francisco de Assis Magalhães,1964-. II. Universidade Estadual de Campinas. Instituto de Matemática, Estatística e Computação Científica. III. Título.

Informações para Biblioteca Digital

Título em outro idioma: Prize collecting traveling salesman problem
Palavras-chave em inglês:
Traveling-salesman problem
Time window
Combinatorial optimization
Methaheuristics
Heuristic
Área de concentração: Matemática Aplicada
Titulação: Mestre em Matemática Aplicada
Banca examinadora:
Francisco de Assis Magalhães Gomes Neto [Orientador]
Antonio Carlos Moretti
Antônio Augusto Chaves
Data de defesa: 06-07-2016
Programa de Pós-Graduação: Matemática Aplicada

Dissertação de Mestrado defendida em 06 de julho de 2016 e aprovada

Pela Banca Examinadora composta pelos Profs. Drs.

Prof.(a). Dr(a). FRANCISCO DE ASSIS MAGALHÃES GOMES NETO

Prof.(a). Dr(a). ANTONIO CARLOS MORETTI

Prof.(a). Dr(a). ANTÔNIO AUGUSTO CHAVES

A Ata da defesa com as respectivas assinaturas dos membros encontra-se no processo de vida acadêmica do aluno.

Dedico este trabalho aos meus pais, que são minha fortaleza. Também à meu orientador, por não ter desistido de mim.

Agradecimentos

As minhas conquistas, nem só por mim me são direito. Existem pessoas fundamentais, sem as quais eu não teria finalizado a minha jornada para me tornar um Mestre em Matemática.

Devo agradecer primeiramente aos meus pais, José Fernandes e Francisca Dayse, que mesmo com toda a dificultade decorrente de ter um filho longe de casa, ficaram muito felizes, me apoiaram e me incentivaram a todo momento, do início ao fim desse mestrado. Quando me via em meio a solidão e confusões mentais, eles não me deixavam parar. Mesmo quado eu pensei que tudo estava indo por água abaixo, eles me diziam para tentar. Por isso e muito mais, sou eternamente grato à eles.

Também sou grato à meu orientador, carinhosamente conhecido no IMECC como professor Chico. O que dizer de uma pessoa, que mesmo sem cobrar explicações sobre suas dificuldades - sejam financeiras, emocionais ou familiares -, nunca lhe virou as costas quando você o procurou pedindo ajuda? Assim foi o meu orientador, professor Francisco de Assis Magalhães, o qual passei a admirar não só profissionalmente, mas também, como pessoa. Um dia espero ter condições de recebê-lo para passar uns dias com sua família em minha cidade. Sou realmente muito grato por sua paciência e ajuda.

Alguns amigos importantes saberão que têm toda minha gratidão. Agradeço pelas conversas, pelo tempo passado juntos, pelos conselhos e incentivos. Por nos reunirmos para almoçar ou jantar e pelos momentos divertidos. Viver em Campinas teria sido ainda mais complicado sem eles.

Por fim, sou grato ao CNPq, pela colaboração de dois anos com a bolsa de mestrado.

Resumo

O Problema do Caixeiro Viajante tem sido objeto de estudo desde o início do século XIX, mantendo-se como um amplo ramo de pesquisa até hoje, tendo em vista o forte impacto do transporte no comércio mundial. Essa dissertação contém algumas variações desse problema, bem como técnicas para obtenção de sua solução. Como destaque, é apresentado o Problema do Caixeiro Viajante com Coleta de Prêmios e Janelas de Tempo, no qual a função objetivo relaciona custos, penalidades por clientes não atendidos e prêmios por clientes atendidos, os quais devem ser visitados dentro de suas respectivas janelas de tempo. GENIUS é a principal heurística utilizada para obtenção de solução inicial do problema em destaque, e a meta-heurística VNS é responsável pelo refinamento da solução inicial. Para o algoritmo que implementamos, são apresentados vários resultados numéricos.

Palavras-chave: Caixeiro Viajante, Janelas de tempo, Coleta de prêmios, Meta-heurística, VNS.

Abstract

The Traveling Salesman Problem has been studied since the beginning of the XIX century, and it remains a wide research field today, in view of the strong impact of transportation in world trade. This dissertation contains some variations of this problem, as well as techniques to obtain its solution. Most notably, we present the Prize Collecting Traveling Salesman Problem With Time Windows, for which the objective function relates costs, penalties for clients that are not visited, and prizes for visited clients. Costumers who are served must be visited within their respective time windows. GENIUS is the main heuristic used to produce a initial solution for the highlighted problem, and the VNS methaheuristic is responsible for refining the initial solution. For the implemented algorithm, we present several numerical results.

Keywords: Traveling Salesman, Time windows, Prize collecting, Methaheuristic, VNS.

Sumário

Introdução **11**
0.1 Descrição do Problema **11**
0.2 Objetivos e Delimitação do Problema **11**
0.3 Justificativa e Importância **11**
0.4 Método de Pesquisa **12**
0.5 Organização do Texto **13**

1 O PROBLEMA DO CAIXEIRO VIAJANTE COM COLETA DE PRÊMIOS E JANELAS DE TEMPO **14**
1.1 O Problema do Caixeiro Viajante **14**
1.2 Problema do Caixeiro Viajante com Janelas de Tempo **16**
1.3 Problema do Caixeiro Viajante com Coleta de Prêmios **18**
1.4 Problema do Caixeiro Viajante com Coleta de Prêmios e Janelas de Tempo **20**
1.4.1 Reformulação do Problema 22
1.4.1.1 Restrições e função objetivo 23

2 GENIUS **25**
2.1 GENIUS para PCV **25**
2.1.1 GENI 25
2.1.1.1 Inserção tipo 1 26
2.1.1.2 Inserção tipo 2 27
2.1.1.3 Algoritmo 27
2.1.2 US 28
2.1.2.1 Remoção tipo 1 28
2.1.2.2 Remoção tipo 2 29
2.2 GENIUS para PCVJT **31**
2.2.1 Adaptação do método 31
2.2.2 Uma heurística para o PCVJT 32

3 METODOLOGIA **35**
3.1 Heurísticas e Meta-heurísticas **35**
3.1.1 Heurísticas 35
3.1.1.1 Heurísticas de Construção 36
3.1.1.2 Heurísticas de Refinamento 37
3.1.2 Meta-heurísticas 39

3.2 Busca em Vizinhança Variável 39
3.2.1 Estruturas de Vizinhança 40
3.2.2 Busca Local 41
3.2.3 Esquemas básicos de VNS 42
3.2.3.1 Descida em Vizinhança Variável 43
3.2.3.2 VNS Reduzida 43
3.2.3.3 VNS Básico 44
3.2.3.4 VNS Geral 44
3.2.4 Representação de uma solução 45
3.2.5 Alguns movimentos adicionais 46
3.3 O Método Proposto 47

4 EXPERIMENTOS COMPUTACIONAIS 50
4.1 Comparação dos resultados 51
4.2 Resultados para o PCVCPJT 54

5 CONSIDERAÇÕES FINAIS 57

REFERÊNCIAS 59

Introdução

0.1 Descrição do Problema

Pode-se dizer que o Problema do Caixeiro Viajante com Coleta de Prêmios e Janelas de Tempo (PCVCPJT) é uma junção do Problema do Caixeiro Viajante com Janelas de Tempo (PCVJT) [29] com o Problema do Caixeiro Viajante com Coleta de Prêmios (PCVCP)[5]. O primeiro está associado a um caixeiro que precisa partir do depósito e visitar todos os clientes, cada um uma única vez, sendo ainda atrelada a cada cliente i uma janela de tempo $J_i = [a_i, b_i]$, de modo que o vendedor atenda cada consumidor em um momento $t_i \in J_i$, e ao final retorne ao depósito, adotando o percurso de menor custo possível. O segundo problema associa um prêmio para cada cliente visitado e uma penalidade para cliente não visitado, e consiste em estabelecer uma rota, também com início e fim no depósito, que inclua uma quantidade conveniente de clientes para que se colete um prêmio mínimo preestabelecido e, ao mesmo tempo, minimize a soma dos custos de viagem e das penalidades.

0.2 Objetivos e Delimitação do Problema

Este trabalho tem como principal objetivo apresentar técnicas heurísticas para a resolução do PCVCPJT. Esse problema pertence à classe de problemas provenientes do Problema do Caixeiro Viajante (PCV)[2], que tem sido objeto de estudo de vários pesquisadores desde o início do século XIX, e continua um forte ramo de pesquisa até hoje (como em [16] e [27], por exemplo) tendo em vista o forte impacto do transporte rodoviário no comércio mundial.

0.3 Justificativa e Importância

A importância do estudo do PCVCPJT é justificada por sua semelhança com problemas enfrentados no dia-a-dia de muitas empresas, principalmente aquelas que estão ligadas à área de logística e distribuição, como transportadoras de cargas, ou empresas cujo sucesso de seus negócios depende da eficiência dessas áreas, tais como distribuidoras em geral. O aumento da informatização tem tornado os clientes cada vez mais exigentes, o que aumenta a dificuldade de atendimento e acentua a importância dos processos logísticos. Desse modo, a distribuição e os processos logísticos tornaram-se fundamentais nas operações gerenciais, pois para se manter competitivo dentro um mercado cada vez mais concorrido é indispensável a adoção de processos ágeis e adaptáveis ao perfil dos

clientes. Além disso, companhias de engenharia de tráfego impõem uma série de restrições quanto aos veículos, como os horários em que podem circular para entrega ou coleta de produtos, e seus tamanhos, que influenciam sua capacidade de entrega.

Como um exemplo prático do PCVCPJT, podemos citar a entrega de um produto, por um distribuidor a seus revendedores. Cada revendedor informa dias e horários em que pode receber a mercadoria, ou até quando precisa recebê-la, o que impõe ao distribuidor tomar decisões quanto à ordem em que deve atendê-los, para que não se atrase e perca uma entrega ou para que não chegue muito cedo e perca tempo demais esperando o horário de início de atendimento. Ainda mais, pode ocorrer de a demanada do produto por parte dos revendedores exceder a capacidade máxima do veículo da distribuidora. Neste caso, para cada revendedor que não receber o produto, o distribuidor paga uma multa contratual, que pode variar de acordo com o contrato firmado com os revendedores.

Assim como o PCV e suas variações, o problema proposto é um problema NP-completo, isto é, um problema para o qual não se conhece um algoritmo exato que encontre uma solução em tempo polinomial.[1]. Métodos exatos, como *Branch-and-Bound* ou *Branch-and-Cut* resolvem problemas NP-completos em tempo exponencial. Em casos de problemas não simétricos, o número de soluções possíveis pode alcançar $\sum_{i=1}^{n}(i-1)!$, e mesmo com o avanço tecnológico no desenvolvimento de processadores mais eficientes, é inviável enumerar todas as soluções para elevados valores de n . Além disso, em muitos casos, uma solução viável deve ser gerada em um curto espaço de tempo, como no caso de uma transportadora que, frente a uma alta demanda de serviços a realizar, não pode aguardar a certeza de ter encontrado a solução ótima.

0.4 Método de Pesquisa

Diante dessa situação, torna-se necessário o uso de métodos heurísticos - métodos que exploram apenas uma pequena parte do espaço de soluções, fornecendo uma solução de boa qualidade com baixo custo computacional - para a obtenção da solução de problemas dessa complexidade. O presente trabalho propõe a abordagem do PCVCPJT via um método, de forma tradicional, composto de duas fases. A primeira fase consiste na produção de uma solução inicial por meio de heurísticas de construção, sendo a heurística GENIUS a principal delas. Após obter uma solução inicial, partimos para a segunda etapa, na qual é feito um refinamento da solução inicial. Nessa fase o algoritmo é regido pela Meta-heurística de Busca em Vizinhança Variável (VNS, do inglês, *Variable Neighborhood Search*), que consiste em explorar diferentes vizinhanças de soluções em busca de outras melhores.

[1] Para informações sobre teoria de complexidade computacional, veja [6]

0.5 Organização do Texto

Esse trabalho encontra-se organizado da seguinte forma: No Capítulo 2, descrevemos o Problema do Caixeiro Viajante e suas variações, destacando o problema com janelas de tempo e coleta de prêmios. No Capítulo 3, apresentamos com bom detalhamento a principal heurística de construção utilizada neste trabalho, a saber, a GENIUS. No Capítulo 4, detalhamos a metodologia utilizada para a abordagem do problema proposto, descrevendo o método heurístico utilizado. No Capítulo 5, expomos alguns experimentos computacionais. Finalmente, no Capítulo 6, apresentamos nossas conclusões.

1 O Problema do Caixeiro Viajante com Coleta de Prêmios e Janelas de Tempo

No início do século XIX, vários problemas envolvendo grafos, intimamente relacionados ao Problema do Caixeiro Viajante (PCV), foram propostos pelo Matemático Irlandês William Hamilton. Um deles, nomeado por Ball como o *Jogo de Hamilton*,[1] consistia em determinar uma rota ao longo das arestas de um dodecaedro regular, passando por todos os vértices, sendo cada um visitado uma única vez. Apesar dos problemas propostos, a primeira abordagem formal do PCV foi apresentada por Hassler Witney, em um seminário na Universidade de Princenton, em 1934.

Considere que um viajante necessita visitar um certo número de cidades e retornar ao lugar de origem, de tal maneira que cada cidade seja visitada exatamente uma vez e que a distância total percorrida seja a menor possível. Supondo conhecida a distância entre cada par de cidades, que roteiro deve ser escolhido?

Além dessa descrição do PCV, é comum apresentá-lo considerando os custos ou os tempos dos trajetos entre as cidades, ao invés das distâncias entre elas. O PCV se encaixou muito bem como aplicação para diversos problemas do dia-a-dia, principalmente para empresas que trabalham com distribuição. Ele também deu origem a vários outros problemas, como veremos neste capítulo. No que segue, usaremos o vértice 1 para representar a cidade, ou mesmo o depósito, de onde o caixeiro viajante partirá para fazer suas visitas.

1.1 O Problema do Caixeiro Viajante

Mais formalmente, o PCV pode ser definido por um grafo $G = (V, A)$, tal que o conjunto de vértices V representa as cidades a serem incluídas no roteiro, e o conjunto de arestas A representa as possibilidades existentes de se viajar de uma cidade para outra. Para garantir que seja possível a viagem de ida e de volta entre cada par de cidades, supomos que o grafo é completo, de maneira que $|A| = |V|(|V| - 1)$. Em termos de Teoria dos Grafos, encontrar uma solução ótima para o PCV, consiste em determinar o circuito Hamiltoniano de menor custo no grafo G.

Suponha conhecidos o número de cidades e os custos dos trajetos entre elas, de modo que $V = \{1, \ldots, n\}$ e, para cada par de vértces $i, j \in V$, c_{ij} seja o custo associado ao arco $(i, j) \in A$. A variável de decisão do problema indica se um arco de A pertence ou não

[1] Para saber mais sobre o jogo de Hamilton e outros problemas como este, veja [7]

à solução que está sendo produzida:

$$x_{ij} = \begin{cases} 1, \text{ se } j \text{ é visitado logo após } i \\ 0, \text{ caso contrário} \end{cases}$$

A variável x_{ii} não tem nenhum significado, de modo que só há variáveis para $i \neq j$. Como o critério para a tomada de decisões é a obtenção do ciclo Hamiltoniano de menor custo, temos a seguinte função objetivo:

$$z = \sum_{i \in V} \sum_{j \in V \setminus \{i\}} c_{ij} x_{ij}.$$

A formulação matemática do problema, proposta por Dantizig, Fulkerson e Johnson (1954), é dada a seguir.

$$min \quad \sum_{i \in V} \sum_{j \in V \setminus \{i\}} c_{ij} x_{ij} \tag{1.1}$$

$$s.a. \quad \sum_{j \in V \setminus \{i\}} x_{ij} = 1, \qquad i = 1, \ldots, n \tag{1.2}$$

$$\sum_{i \in V \setminus \{j\}} x_{ij} = 1, \qquad j = 1, \ldots, n \tag{1.3}$$

$$\sum_{i \in S} \sum_{j \in S \setminus \{i\}} x_{ij} \leqslant |S| - 1, \quad \forall S \subset V, S \neq \varnothing \tag{1.4}$$

$$x_{ij} \in \{0, 1\}, \qquad i, j = 1, \ldots, n, i \neq j. \tag{1.5}$$

As restrições (1.2) e (1.3) implicam que a cada nó chega e de cada nó sai, respectivamente, apenas uma aresta. Isso significa que todo nó é visitado uma única vez. Para explicarmos a restrição (1.4), considere o caso em que $n = 6$ e todas as variáveis sejam iguais a zero, salvo

$$x_{13} = x_{35} = x_{51} = x_{26} = x_{64} = x_{42} = 1.$$

Essa solução satisfaz as restrições (1.2) e (1.3). No entanto, não forma um circuito hamiltoniano. Embora todos os nós estejam incluídos na solução exatamente uma vez, esta solução forma dois circuitos: $\{1, 3, 5, 1\}$ e $\{2, 6, 4, 2\}$. Eles formam o que chamamos de *subrota*, que são circuitos que não incluem todos os vértices do grafo.

Para que uma solução, nesse caso, não possua subrota no subconjunto $S = \{2, 6, 4\} \subset V$, por exemplo, precisamos garantir que no máximo duas arestas de um circuito formado por estes três vértices estejam presentes. De fato, é impossível que se tenha um circuito com menos arestas do que nós, de modo que a equação

$$x_{26} + x_{64} + x_{42} \leqslant 2,$$

impede que a solução possua uma subrota no conjunto S. Para garantir que jamais se formarão subrotas em qualquer subconjunto S não vazio estritamente contido em V, generalizamos a equação anterior para

$$\sum_{i \in S} \sum_{j \in S \setminus \{i\}} x_{ij} \leqslant |S| - 1, \forall S \subset V.$$

Essa formulação possui $2n$ restrições de designação, sendo n em cada uma das restrições (1.2) e (1.3), além de $2^n - 2$ restrições para eliminação de subrotas em (1.4), uma vez que não estamos considerando os subconjuntos próprio e vazio de V.

1.2 Problema do Caixeiro Viajante com Janelas de Tempo

No problema do caixeiro viajante com janelas de tempo (PCVJT), além de encontrar o caminho de custo mínimo que deve ser percorrido por um veículo que parte do depósito e passa por cada cliente uma única vez, cada consumidor deve começar a ser atendido dentro de um intervalo de tempo preestabelecido.

Observe que, na descrição do problema, nos referimos a clientes ao invés de cidades. Isso se deve ao fato de que o problema vem tomando cada vez mais um foco comercial. Para formalizar matematicamente o problema, supomos conhecido um grafo $G = (V, A)$, e definimos as variáveis

- x_{ij} - variável binária, exatamente como descrita na seção anterior
- t_i - variável de tempo que indica o início do atendimento do cliente i.

Além disso, temos os seguintes dados:

- s_i - tempo de atendimento (serviço) do cliente (nó) i.
- v_{ij} - tempo de travessia do arco (i, j).
- a_i - instante mínimo para dar início ao atendimento do cliente i (início da janela de tempo).
- b_i - prazo limite para inicar o atendimento no nó i (fim da janela de tempo).
- c_{ij} - matriz de custos, assim como na seção anterior.

Normalmente, instâncias do PCVJT fornecem uma matriz com os valores T_{ij}, que representam o tempo mínimo entre o início do atendimento do nó i e o início do atendimento do nó j, quando este é visitado imediatamente após o nó i. Estes valores são

dados por $T_{ij} = s_i + v_{ij}$. Também é usual que essa matriz seja usada como a matriz de custos, de modo que $T_{ij} = c_{ij}$. A separação dos dados s_i, v_{ij}, T_{ij} e c_{ij}, foi feita para ajudar na compreensão do modelo abaixo, em que apenas c_{ij} aparece. A seguinte formulação de programação linear inteira mista, baseada no método M-grande, proposta por Desrochers e Laporte em 1991 [30] tem base na formulação proposta originalmente por Miller, Tucker e Zemlin 1960 [29]:

$$min \quad \sum_{i \in V} \sum_{j \in V \setminus \{i\}} c_{ij} x_{ij} \tag{1.6}$$

$$s.a. \quad \sum_{j \in V \setminus \{i\}} x_{ij} = 1, \qquad i = 1, \ldots, n \tag{1.7}$$

$$\sum_{i \in V \setminus \{j\}} x_{ij} = 1, \qquad j = 1, \ldots, n \tag{1.8}$$

$$t_i + c_{ij} - (1 - x_{ij}) \cdot M \leqslant t_j, \quad \forall (i,j) \in A, j \neq 1 \tag{1.9}$$

$$a_i \leqslant t_i \leqslant b_i, \qquad i = 1, \ldots, n \tag{1.10}$$

$$x_{ij} \in \{0, 1\}, \qquad i, j = 1, \ldots, n, i \neq j. \tag{1.11}$$

A restrição (1.10) garante que o cliente i seja atendido dentro de sua janela de tempo $[a_i, b_i]$. Para explicar a restrição (1.9), partimos da ideia de que devemos estabelecer uma ordem entre as variáveis de tempo, isto é, se i precede j numa solução, então deve-se ter $t_i < t_j$. Primeiramente, veja que, se $x_{ij} = 1$, o instante de início de atendimento do nó j não acontece antes de se visitar o nó i, atendê-lo e percorrer o arco (i, j), de modo que se tem

$$t_i + c_{ij} \leqslant t_j. \tag{1.12}$$

A equação acima é exatamente o que obtemos de (1.9) quando $x_{i,j} = 1$. Quando $x_{ij} = 0$, (1.9) resulta em

$$t_i + c_{ij} - M \leqslant t_j.$$

Nesse caso, M deve ser escolhido de modo que essa desigualdade seja satisfeita para quaisquer valores de t_i e t_j. Para a escolha de M, fazemos $M = max\{M_{ij};\ (i,j) \in A\}$, onde $M_{ij} = max\{b_i + c_{ij} - a_j, 0\}$, $\forall (i,j) \in A$.

Além de estabelecer uma ordem para as variáveis t_i, a restrição (1.9) garante que uma solução não possua subrotas. De fato, se houvesse um ciclo começando em $i \neq 1$, estaríamos iniciando o atendimento neste nó no instante t_i e, ao final, retornaríamos para i, obtendo outro valor t_i, pois (1.9) nos informa que se i antecede j na solução, então $t_i < t_j$, já que $c_{ij} > 0$. Disso resultaria que $t_i < t_i$, o que é um absurdo.

Como a restrição (1.9) tem $O(n^2)$, é mais interessante mantê-la do que a restrição para eliminação de subrotas utilizada na formulação do PCV da seção anteior, que cresce mais rapidamente por ter $O(2^n)$.

Observe que, para $x_{ij} = 1$, segue de (1.10) e (1.12) que

$$t_j \geqslant max\{t_i + c_{ij}, a_j\}.$$

No método proposto neste trabalho, adotamos

$$t_j = \max\{t_i + c_{ij}, a_j\}.$$

1.3 Problema do Caixeiro Viajante com Coleta de Prêmios

O Problema do Caixeiro Viajante com Coleta de Prêmios (PCVCP), referido na língua inglesa como *Prize Collecting Traveling Salesman Problem (PCTSP)*, está associado a um caixeiro viajante que coleta um prêmio não negativo para cada cidade (ou cliente) que ele visita, além de pagar uma penalidade, também não negativa, para cada cidade não visitada. O problema encontra-se em minimizar a soma dos custos da viagem e das penalidades, incluindo na rota um número suficiente de cidades que permita a coleta de um prêmio mínimo preestabelecido.

Apesar de ser uma extensão do PCV, o PCVCP foi primeiramente formulado por Egon Balas [5] como um modelo para programação da operação diária de uma fábrica de produção de lâminas de aço. Por razões que tinham a ver com os desgastes dos rolos, dentre outros fatores, a ordem do processamento era essencial. Esse problema serviu como base para o desenvolvimento de um software implementado por Balas e Martin, e foi resolvido aproximadamente por meio da combinação de várias heurísticas (Balas, 2001).

Para uma definição matemática formal do problema, considere um grafo $G = (V, A)$, exatamente como na Seção 1.1, para o qual são fornecidos os seguintes dados:

- p_i - prêmio coletado ao visitar o cliente $i \in V$.
- π_i - penalidade paga por não visitar o cliente $i \in V$.
- c_{ij} - custo associado ao arco $(i, j) \in A$.
- p_{min} - prêmio mínimo que deve ser coletado na rota percorrida.

Observe que a grande diferença desse problema para os anteriores é que nem todas cidades precisam ser visitadas, de modo que continuamos com as variáveis x_{ij} desempenhando o mesmo papel como nos problemas anteriores, mas adicionamos as variáveis

- $y_i = \begin{cases} 1, & \text{se } i \text{ é visitado,} \\ 0, & \text{caso contrário.} \end{cases}$

- f_{ij}, para $i = 1, \ldots, n$; $j = 1, \ldots, n$.

As variáveis f_{ij} indicam a ordem em que o arco (i, j) é percorrido. Por exemplo, $f_{ij} = 10$ caso (i, j) seja o décimo arco percorrido na solução. Caso (i, j) não esteja na solução, tem-se $f_{ij} = 0$. A variável f_{ij} também é conhecida como o fluxo do arco (i, j) [27, 31].

A seguir, apresentamos a formulação do PCVCP proposta por Balas, onde foram inseridas as restrições envolvendo as variáveis de fluxo [28].

$$min \quad \sum_{i \in V} \sum_{j \in V \setminus \{i\}} c_{ij} x_{ij} + \sum_{i \in V} \pi_i \cdot (1 - y_i) \tag{1.13}$$

$$s.a. \quad \sum_{j \in V \setminus \{i\}} x_{ij} = y_i, \qquad i = 1, \ldots, n \tag{1.14}$$

$$\sum_{i \in V \setminus \{j\}} x_{ij} = y_j, \qquad j = 1, \ldots, n \tag{1.15}$$

$$\sum_{i \in V} p_i y_i \geqslant p_{min} \tag{1.16}$$

$$\sum_{j \in V \setminus \{i\}} f_{ij} - \sum_{j \in V \setminus \{i\}} f_{ji} = y_i, \qquad \forall i \in V \setminus \{1\} \tag{1.17}$$

$$f_{ij} \leqslant (n - 1) \cdot x_{ij}, \qquad \forall (i, j) \in A \tag{1.18}$$

$$x_{ij} \in \{0, 1\}, \qquad i, j = 1, \ldots, n, i \neq j \tag{1.19}$$

$$y_i \in \{0, 1\}, \qquad i = 1, \ldots, n \tag{1.20}$$

$$f_{ij} \geqslant 0, \qquad \forall (i, j) \in A. \tag{1.21}$$

A função objetivo (1.13) é composta basicamente por dois componentes: o custo total de deslocamento, representado pelo primeiro somatório, e o custo das penalidades pagas por não visitação de vértices, representado pelo segundo somatório. As restrições (1.14) e (1.15) garantem que exatamente um arco chega e exatamente um sai de cada nó presente numa rota solução. A restrição (1.16) garante que o prêmio coletado na rota solução não seja inferior ao prêmio mínimo preestabelecido.

As restrições (1.18) e (1.21) garantem que $f_{ij} = 0$ se o arco (i, j) não está inserido na solução. Por fim, as restrições (1.18) , (1.21) e (1.17) garantem a continuidade do fluxo, isto é, a eliminação de subrotas. De fato, suponha que um nó $k \in V \setminus \{1\}$ esteja na solução, e que seu antecessor e sucessor sejam i e j, respectivamente. Por conta de (1.14) e (1.15), a restrição (1.17) nos informa que

$$f_{kj} - f_{ik} = y_k \Rightarrow f_{kj} - f_{ik} = 1 \Rightarrow f_{kj} = 1 + f_{ik}.$$

Isso significa que o fluxo que sai de um nó supera em uma unidade o fluxo que chega ao mesmo nó. Ainda mais, sempre que um arco (i_1, j_1) é percorrido antes de um arco (i_2, j_2), tem-se que $f_{i_1 j_1} < f_{i_2 j_2}$. Portanto, caso houvesse um circuito com início e término no nó k,

por exemplo, teríamos $f_{kj} < f_{ik}$, o que é um absurdo para uma solução. Logo, a restrição (1.17) impede a existência de subrotas.

1.4 Problema do Caixeiro Viajante com Coleta de Prêmios e Janelas de Tempo

Nesta seção, apresentamos o problema central dessa dissertação, a saber, o Problema do Caixeiro Viajante com Coleta de Prêmios e Janelas de Tempo (PCVCPJT). Pode-se dizer que esse problema é a junção do PCVCP com o PCVJT, de modo que, para formulá-lo, foi decidido que devem ser mantidas as principais características de cada um dos dois problemas componentes:

1. Clientes visitados geram prêmios e clientes não visitados geram penalidades.
2. É necessário atender uma quantidade de clientes que proporcione uma coleta de prêmios que não seja inferior ao prêmio mínimo preestabelecido.
3. Os clientes devem ser visitados dentro de suas janelas de tempo, também preestabelecidas.

Obviamente, a terceira caracterísitca é relativa ao PCVJT, enquanto as outras dizem respeito ao PCVCP. Observe que, dessa forma, nem todos os clientes precisam ser atendidos. No entanto, aqueles que o forem devem ser visitados dentro de suas respectivas janelas de tempo. A descrição do problema é a seguinte:

- O problema consiste em minimizar a soma dos custos de viagem e das penalidades, incluindo na sua rota uma quantidade suficiente de clientes para coletar um prêmio maior ou igual ao prêmio mínimo preestabelecido, de modo que os clientes inclusos na rota sejam visitados uma única vez e sejam atendidos dentro de suas respectivas janelas de tempo.

Mais precisamente, consideremos um grafo $G = (V, A)$, exatamente como nas seções anteriores. As variáveis são:

- $x_{ij} = \begin{cases} 1, \text{ se o arco } (i,j) \text{ é percorrido} \\ 0, \text{ caso contrário} \end{cases}$
- $y_i = \begin{cases} 1, \text{ se o nó } i \text{ é atendido} \\ 0, \text{ caso contrário} \end{cases}$
- t_i - início do atendimento ao nó i

- f_{ij} - fluxo ou ordem do arco (i, j)

As variáveis x_{ij} e y_i são de decisão, que determinam a presenção ou não de arcos e nós, respectivamente, na rota solução. As variáveis t_i são de tempo, que indicam o início do serviço no cliente i, enquanto as variáveis de fluxo, f_{ij}, indicam a ordem em que o arco (i, j) é percorrido.

Por sua vez, os dados são os seguintes:

- c_{ij} - Custo associado arco $(i, j) \in A$.

- v_{ij} - tempo de viagem do arco $(i, j) \in A$.

- s_i - tempo de serviço no nó i.

- a_i - início da janela de tempo referente ao nó i.

- b_i - término da janela de tempo referente ao nó i.

- p_i - prêmio coletado por visitar o nó $i \in V$.

- π_i - penalidade paga por não visitar o nó $i \in V$.

- p_{min} - prêmio mínimo que deve ser coletado em uma rota solução.

Novamente, inibimos os dados s_i e v_{ij}, adotando $c_{ij} = T_{ij}$, onde $T_{ij} = s_i + v_{ij}$, de modo que c_{ij} representa o tempo mínimo entre o início do atendimento do cliente i e o início do atendimento do cliente j, quando este é visitado logo após i.

Segue a apresentação da formulação matemática do PCVCPJT:

$$min \quad \sum_{i \in V} \sum_{j \in V \setminus \{i\}} c_{ij} x_{ij} + \sum_{i \in V} \pi_i \cdot (1 - y_i) \tag{1.22}$$

$$s.a. \quad \sum_{j \in V \setminus \{i\}} x_{ij} = y_i, \qquad i = 1, \ldots, n \tag{1.23}$$

$$\sum_{i \in V \setminus \{j\}} x_{ij} = y_j, \qquad j = 1, \ldots, n \tag{1.24}$$

$$\sum_{i \in V} p_i y_i \geqslant p_{min} \tag{1.25}$$

$$\sum_{j \in V \setminus \{i\}} f_{ij} - \sum_{j \in V \setminus \{i\}} f_{ji} = y_i, \qquad \forall i \in V \setminus \{1\} \tag{1.26}$$

$$f_{ij} \leqslant (n-1) \cdot x_{ij}, \qquad \forall (i,j) \in A \tag{1.27}$$

$$t_i + c_{ij} - (1 - x_{ij}) \cdot M \leqslant t_j, \qquad \forall (i,j) \in A, j \neq 1 \tag{1.28}$$

$$a_i \leqslant t_i \leqslant b_i, \qquad i = 1, \ldots, n \tag{1.29}$$

$$x_{ij} \in \{0,1\}, \qquad i,j = 1, \ldots, n, i \neq j \tag{1.30}$$

$$y_i \in \{0,1\}, \qquad i = 1, \ldots, n \tag{1.31}$$

$$f_{ij} \geqslant 0, \qquad \forall (i,j) \in A. \tag{1.32}$$

1.4.1 Reformulação do Problema

Há dois motivos pelos quais um cliente pode vir a não ser atendido:

1. Não é possível atendê-lo dentro de sua janela de tempo.
2. A função objetivo tem melhor valor sem incluí-lo na rota.

Além disso, esse problema pode ser empregado para dois objetivos diferentes:

1. Atender todos os clientes possíveis, deixando de fora da rota apenas aqueles para os quais não é possível realizar o atendimento dentro da janela de tempo.
2. Assim como no PCVCP, atender à quantidade de clientes que resulte num melhor valor de função objetivo.

O objetivo adotado neste trabalho, é o primeiro, isto é, atender o máximo possível de clientes, com exlusão de alguns deles apenas na impossibilidade de atendimento dentro da janela de tempo.

1.4.1.1 Restrições e função objetivo

- Primeiramente, foi observado que, o que se espera para um arco (i, j) pertencente a uma solução do PCVCPJT, é que o início do atendimento no cliente j não comece antes de se passar pelo cliente i e percorrer o arco (i, j), de modo que a inequação (1.12) - obtida a partir da restrição (1.28), quando $x_{ij} = 1$ - deve ser satisfeita. Essa inequação é suficiente para estabelecer uma ordem entre as variáveis de tempo relativas aos nós visitados, e determinar seus valores. Dessa forma, é desnecessário que as variáveis t_i para os quais $y_i = 0$ (i não é visitado), bem como arcos não percorridos, influenciem na determinação das demais variáveis de tempo. Com base nesse pensamento, para $x_{ij} = 0$, podemos notar que, ao substituir M por $b_i + c_{ij}$ na restrição (1.28), obtemos:

  $$t_i + c_{i,j} - (b_i + c_{i,j}) \leqslant t_j,$$

  donde segue que

  $$t_i - b_i \leqslant t_j. \tag{1.33}$$

  Da restrição (1.29), sabemos que $t_i \leqslant b_i$, e assim, (1.33) nos informa que $t_j \geqslant 0$. Além disso, como $t_j \geqslant 0$, (1.33) também nos informa que $t_i \leqslant b_i$. Em outras palavras, a equação acima não tem influência sobre os valores das variáveis de tempo. Por esse motivo, optamos por não calcular a constante M, substituindo a restrição (1.28) por

  $$t_i + c_{i,j} - (b_i + c_{i,j}) \cdot (1 - x_{i,j}) \leqslant t_j$$

  .

- Observando a formulação proposta para o PCVCPJT, as restrições envolvendo as variáveis f_{ij}, são utilizadas para que não sejam admitidas subrotas nas soluções. No entanto, esse papel também é desempenhado pelas restrições de janelas de tempo, como vimos na formulação do PCVJT. Logo, há uma redundância nas restrições, o que é desnecessário na formulação do problema. Como além desse papel, as restrições de janela de tempo também estabelecem que o cliente será atendido dentro do horário combinado, é natural que optemos por elas em detrimento das restrições envolvendo as variáveis de fluxo f_{ij}.

- Seguindo a linha que vem sendo adotada em trabalhos que tratam do PCVCP - como em [28], por exemplo -, escolhemos suprimir a restrição de prêmio mínimo

  $$\sum_{i \in V} p_i y_i \geqslant p_{min}.$$

  Isso significa que a formulação do problema passará a aceitar soluções que antes seriam infactíveis, pois sem essa restrição, poderá haver alguma solução na qual não

se colete o prêmio mínimo. Em contrapartida, a função objetivo ganha um novo termo, sendo agora composta por três parcelas, cada qual relacionada com um dado diferente do problema.

- Os custos de deslocamento de todos os arcos $(i, j) \in V$ pertencentes à rota são incluídos na primeira parcela da função objetivo:

$$\sum_{i \in V} \sum_{j \in V \setminus \{i\}} c_{ij} x_{ij}.$$

- A segunda parcela da função objetivo corresponde ao custo total das penalidades π_i pagas pela não visitação de vértices :

$$\sum_{i \in V} \pi_i \cdot (1 - y_i).$$

- Por fim, a última parcela penaliza a solução na qual se coleta um prêmio inferior ao prêmio mínimo, por meio de um peso $\alpha \in [0, 1]$.

$$\alpha \cdot max\{0, p_{min} - \sum_{i \in V} p_i y_i\}$$

O PCVCPJT ganha então a nova formulação:

$$\min \sum_{i \in V} \sum_{j \in V \setminus \{i\}} c_{ij} x_{ij} + \sum_{i \in V} \pi_i \cdot (1 - y_i) + \alpha \cdot max\{0, p_{min} - \sum_{i \in V} p_i y_i\} \tag{1.34}$$

$$s.a. \qquad \sum_{j \in V \setminus \{i\}} x_{ij} = y_i, \qquad i = 1, \ldots, n \tag{1.35}$$

$$\sum_{i \in V \setminus \{j\}} x_{ij} = y_j, \qquad j = 1, \ldots, n \tag{1.36}$$

$$t_i + c_{ij} - (b_i + c_{i,j}) \cdot (1 - x_{i,j}) \leqslant t_j, \quad \forall (i, j) \in A, j \neq 1 \tag{1.37}$$

$$a_i y_i \leqslant t_i \leqslant b_i y_i, \qquad i = 1, \ldots, n \tag{1.38}$$

$$x_{ij} \in \{0, 1\}, \qquad i, j \in V, i \neq j \tag{1.39}$$

$$y_i \in \{0, 1\}, \qquad i = 1, \ldots, n. \tag{1.40}$$

Essa é a vesão definitiva, mais clara e enxuta, para o modelo do problema que estamos tratando neste trabalho.

2 GENIUS

Neste capítulo, abordamos a principal heurística deste trabalho, para a obtenção de solução inicial. Trata-se da heurística GENIUS, proposta por Gendreau *et al.* [9]. Dentre todas as técnicas que compõem a fase inicial do processo de encontrar soluções para o nosso problema, ela é quem tem o maior potencial de nos fornecer uma primeira solução de boa qualidade. O capítulo começa detalhando como funciona a versão original dessa heurística - criada para resolver o PCV puro - e, posteriormente, expande-a para a versão que utilizamos, que leva em consideraçao as restrições de janela de tempo.

2.1 GENIUS para PCV

Em geral, heurísticas para o PCV podem ser classificadas em dois tipos: (1) de construção, que consiste em gradualmente montar uma solução por adicionar um novo vértice ao circuito, a cada passo, e (2) de melhoramento, onde a partir de uma solução factível, busca-se obter outra com melhor valor de função objetivo, por meio de várias trocas sistemáticas sobre a solução corrente. Os melhores métodos costumam ser de duas fases, combinando procedimentos que possuam essas duas propriedades. A heurística GENIUS possui tais características, sendo composta por uma fase do tipo (1) chamada GENI, e uma do tipo (2), chamada US.

2.1.1 GENI

Começamos tratando da fase de construção da heurística GENIUS. Essa etapa consiste em produzir uma solução factível, por meio de inserções. A principal característica desse *procedimento de inserção generalizada*, ou GENI - do inglês, *generalized* (GEN) *insertion* (I) -, é que a inserção de um vértice v no circuito que está sendo construído toma lugar entre dois vértices que, a princípio, não necessariamente são adjacentes. No entanto, quando realizada a inserção, esses dois vértices tornam-se adjacentes a v.

Na prática, suponha que seja dado um percurso e que nele desejemos inserir um vértice v, entre dois nós v_i e v_j, já presentes nesse percurso. Para mais clareza, deixamos estabelecida a seguinte notação que vale para todo o capítulo, salvo menção explícita em contrário:

- v_k - Vértice presente no caminho de v_j até v_i.
- v_l - Vértice presente no caminho de v_i até v_j.

- v_{h-1}, v_{h+1} - Antecessor e sucessor, respectivamente, do um vértice v_h, considerando a orientação do circuito.

A seguir, apresentamos as duas estratégias que compõem a fase GENI, descrevendo como a inserção de um nó v é realizada no percurso por meio da exclusão de certos arcos e inclusão de outros, inidicando como alguns trechos são afetados por conta disso. Nas figuras 1 e 2, (a) e (b) representam os percursos antes e após a inserção do vértice v, respectivamente. Além disso, para todas as figuras neste capítulo, os arcos tracejados em (a) representam os arcos removidos, enquanto os arcos tracejados em (b), são os incluídos. Embora representem os mesmos trechos, os segmentos pontilhados em (a) possuem orientação contrária a dos segmentos pontilhados em (b).

2.1.1.1 Inserção tipo 1

Nesse primeiro procedimento, é considerado um vértice v_k, tal que $v_k \neq v_i$ e $v_k \neq v_j$. Para inserir v no percurso, é feita a remoção dos arcos (v_i, v_{i+1}), (v_j, v_{j+1}) e (v_k, v_{k+1}), e a inclusão dos arcos (v_i, v), (v, v_j), (v_{i+1}, v_k) e (v_{j+1}, v_{k+1}). Como consequência disso, os caminhos $(v_{i+1}, \ldots, v_j)$ e $(v_{j+1}, \ldots, v_k)$ são invertidos no processo. Observe que pelas restrições dadas a v_k, para que seja possível realizar tal inserção, é preciso que exista ao menos um nó entre v_j e v_i, isto é, $v_{j+1} \neq v_i$.

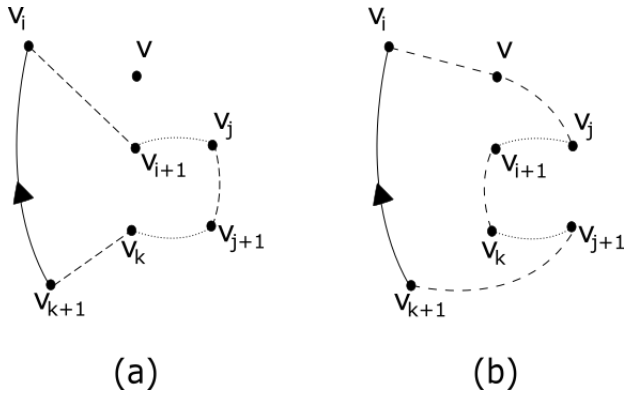

Figura 1 – Inserção tipo 1 do nó v entre os nós v_i e v_j

Algoritmo 1: GENI: inserção tipo 1

Requer: v, v_i, v_j, v_k, P
Fornece: P
if $v_k \neq v_i$ *and* $v_k \neq v_j$ **then**
  remova os arcos (v_i, v_{i+1}), (v_j, v_{j+1}) e (v_k, v_{k+1}) de P
  insira os arcos (v_i, v), (v, v_j), (v_{i+1}, v_k) e (v_{j+1}, v_{k+1}) em P
end

2.1.1.2 Inserção tipo 2

Aqui, devemos ter $v_k \neq v_j$, $v_k \neq v_{j+1}$, $v_l \neq v_i$ e $v_l \neq v_{i+1}$. Por esse procedimento, inserir v no percurso, significa remover os arcos (v_i, v_{i+1}), (v_{l-1}, v_l), (v_j, v_{j+1}) e (v_{k-1}, v_k), e acrescentar os arcos (v_i, v), (v, v_j), (v_l, v_{j+1}), (v_{k-1}, v_{l-1}) e (v_{l+1}, v_k). Note que com isso, os caminhos $(v_{i+1}, \ldots, v_{l-1})$ e $(v_l, \ldots, v_j)$ devem ser invertidos.

Observe que pelas restrições dadas à v_k e v_l, para que seja possível realizar tal inserção, é preciso que existam ao menos dois nós no caminho entre v_i e v_j e no caminho entre v_j e v_i. Desse modo, a possiblidade da inserção tipo 1, é necessária para a inserção tipo 2, e a possibilidade da inserção tipo 2, é suficiente para a do primeiro tipo. Isso pode ser aproveitado em termos de programação.

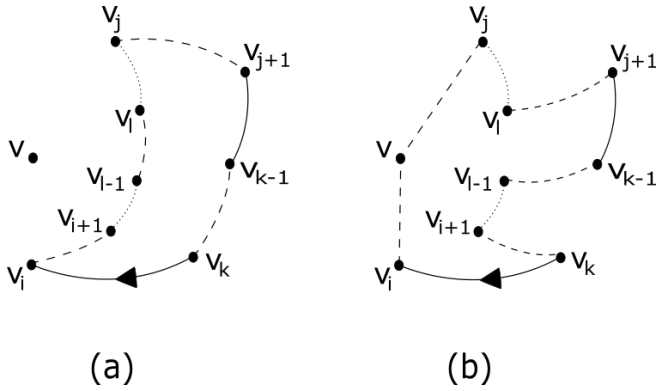

Figura 2 – Inserção tipo 2 do nó v entre os nós v_i e v_j

Algoritmo 2: GENI: inserção tipo 2

Requer: v, v_i, v_j, v_k, v_l, P
Fornece: P
if $v_k \neq v_i$ *and* $v_k \neq v_j$ **then**
  remova os arcos (v_i, v_{i+1}), (v_{l-1}, v_l), (v_j, v_{j+1}) e (v_{k-1}, v_k) de P
  insira os arcos (v_i, v), (v, v_j), (v_l, v_{j+1}), (v_{k-1}, v_{l-1}) e (v_{l+1}, v_k) em P
end

2.1.1.3 Algoritmo

O algoritmo GENI considera as duas possíveis orientações do percurso para realizar cada inserção. Para realizar cada processo de inserção, precisamos estabelecer v_i, v_j, v_k e v_l. Uma vez que a solução cresce a cada iteração, torna-se mais caro considerar todas as possibilidades para tais vértices. Portanto, limita-se a busca para determiná-los. Para cada vértice $v \in V$, define-se a vizinhança $N_p(v)$ como o conjunto de p vértices já sobre o percurso, mais próximos - com respeito aos c_{ij}'s - de v. Caso o percurso possua menos que p vértices, então todos pertencem a $N_p(v)$.

Dessa forma, para um dado parâmetro p, seleciona-se na seguinte ordem:

1. v_i e v_j em $N_p(v)$
2. $v_k \in N_p(v_{i+1})$
3. $v_l \in N_p(v_{j+1})$

Na prática, p é um número relativamente pequeno. No seguinte algoritmo, S representa o conjunto formado pelos nós que ainda não foram inseridos no percurso P.

Algoritmo 3: GENI

Requer: S, p
Fornece: P
Monte a rota inicial P, formada por um subconjunto aleatório de 3 vértices contidos em S.
Inicialize a p-vizinhança de todos os vértices.
while $S \neq \varnothing$ **do**
  Selecione arbritariamente $v \in S$.
  repeat
    Escolha v_i, v_j, v_k e v_l.
    Para cada orientação de P, aplique os algoritmos 1 e 2.
  until *Esgotar as possibilidades de escolhas para a quádrupla* (v_i, v_j, v_k, v_l).;
  Defina P como o percurso obtido pela inserção de menor custo.
  Atualize a p-vizinhança de todos os vértices, pois agora v está em P.
end

2.1.2 US

Em adição ao GENI, foi desenvolvido um algoritmo de pós otimização US - do inglês, *Unstringing* and *Stringing* -, que consiste em remover um vértice de um percurso factível (*Stringing*) e inserí-lo de volta (*Unstringing*). É claro, esse procedimento pode ser aplicado a uma rota produzida por qualquer algoritmo. O procedimento *Stringing* é idêntico ao GENI, enquanto o procedimento *Unstringing*, é justamente o inverso, já que, como veremos nesta seção, os arcos removidos (inseridos) por essa técnica, equivalem aos arcos inseridos (removidos) no método GENI.

No que segue, considere $v_j \in N_p(v_{i+1})$ e $v_k \in N_p(v_{i-1})$. Além disso, nas figuras 3 e 4, (a) e (b) representam os percursos antes e após o processo de remoção, respectivamente. A seguir, descrevemos as duas formas de reconectar a rota após a remoção um vértice v_i por meio de exclusão e inclusão de arcos, o que afeta alguns trechos do percurso.

2.1.2.1 Remoção tipo 1

Para uma dada orientação da rota, considere o vértice v_k no caminho $(v_{i+1}, \ldots, v_{j-1})$. Remover v_i do percurso significa excluir os arcos (v_{i-1}, v_i), (v_i, v_{i+1}), (v_k, v_{k+1}), e incluir os

arcos (v_{i-1}, v_k), (v_{i+1}, v_j), (v_{k+1}, v_{j+1}). Os caminhos $(v_{i+1}, \ldots, v_k)$ e $(v_{k+1}, \ldots, v_j)$ devem ser revertidos na nova rota.

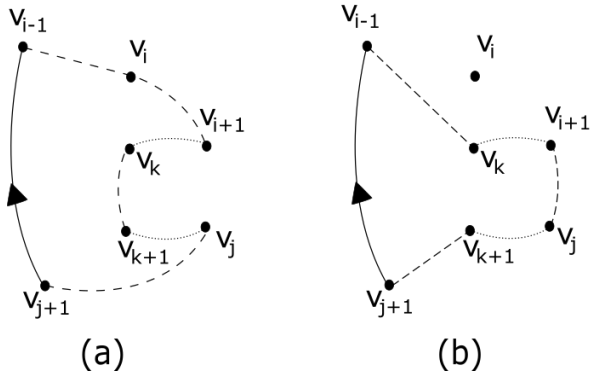

Figura 3 – Remoção tipo 1 do vértice v_i

Algoritmo 4: US: remoção tipo 1

Requer: v_i, v_j, v_k, P
Fornece: P
Remova os arcos (v_{i-1}, v_i), (v_i, v_{i+1}), (v_k, v_{k+1}) de P
Insira os arcos (v_{i-1}, v_k), (v_{i+1}, v_j), (v_{k+1}, v_{j+1}) em P

2.1.2.2 Remoção tipo 2

Para uma dada orientação da rota, considere o vértice v_k no caminho $(v_{j+1}, \ldots, v_{i-2})$; e também $v_l \in N_p(v_{k+1})$ no caminho $(v_j, \ldots, v_{k-1})$. Remover v_i do percurso significa excluir os arcos (v_{i-1}, v_i), (v_i, v_{i+1}), (v_{j-1}, v_j), (v_l, v_{l+1}) e (v_k, v_{k+1}) e incluir os arcos (v_{i-1}, v_k), (v_{l+1}, v_{j-1}), (v_{i+1}, v_j) e (v_l, v_{k+1}). Os caminhos $(v_{i+1}, \ldots, v_{j-1})$ e $(v_{l+1}, \ldots, v_k)$ devem ser revertidos.

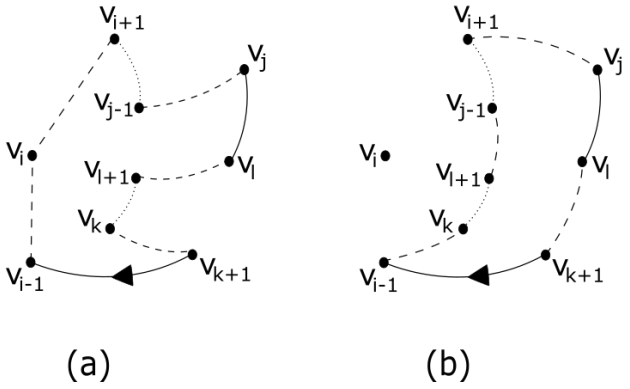

Figura 4 – Remoção tipo 2 do vértice v_i

O funcionamento do método US consiste basicamente em avaliar, para todo vértice v pertencente ao percurso solução P, o custo do caminho resultante após a aplicação

Algoritmo 5: US: remoção tipo 2

Requer: v_i, v_j, v_k, v_l, P
Fornece: P
remova os arcos (v_{i-1}, v_i), (v_i, v_{i+1}), (v_{j-1}, v_j), (v_l, v_{l+1}) e (v_k, v_{k+1}) de P
insira os arcos (v_{i-1}, v_k), (v_{l+1}, v_{j-1}), (v_{i+1}, v_j) e (v_l, v_{k+1}) em P

das técnicas de remoção e inserção, e escolher a combinação desses movimentos que resultar num melhor valor de função objetivo. Desde que o caminho obtido seja mais barato que a solução corrente, a troca é feita. No algoritmo a seguir, z representa o custo do percurso P. O procedimento é realizado da seguinte forma:

Algoritmo 6: US

Requer: p, P, z
Fornece: P^*, z^*
$P^* \leftarrow P$, $z^* \leftarrow z$, $i \leftarrow 2$
while $i < n + 1$ **do**
  $P \leftarrow P^*$
  repeat
    Determine v_j, v_k e v_l.
    Para cada orientação da rota, aplicar sobre P os algoritmos 4 e 5, para remover v_i
    $S \leftarrow \{v_i\}$
    $v \leftarrow v_i$
    Para cada um dos caminhos obtidos na linha acima, aplique o algoritmo 3.
  until *Esgotar as possibilidades de escolha para a tripla* (v_j, v_k, v_l).;
  Dentre os percursos obtidos, defina P' como o percurso com melhor custo z'.
  $P \leftarrow P'$, $z \leftarrow z'$
  if $z' < z^*$ **then**
    $P^* \leftarrow P'$, $z^* \leftarrow z$, $i \leftarrow 2$
  end
  $i \leftarrow i + 1$
end

Oberve que a segunda instrução dentro do comando "repeat"do algoritmo 6 equivale a aplicar os dois métodos de remoção acima apresentados, para o nó v_i, e em seguida, realizando assim a etapa *Unstringing*. Na última linha dentro do mesmo comando, aplicamos o método GENI, executando assim, a etapa *Stringing*. Quando i atinge $n + 1$, o método não consegue realizar mais nenhuma melhoria. É claro que não basta obter um percurso com menor custo apenas por remover o vértice. A solução só é trocada se, ao reinserí-lo, obtivermos uma solução com custo menor que a anterior, o que evita a obtenção de soluções com inserções repetidas.

2.2 GENIUS para PCVJT

Diferentemente do PCV, a solução do PCVJT é um caminho começando em v_1 e terminando v_{n+1}, e não um passeio. Apesar de tanto v_1 quanto v_{n+1} representarem o depósito e o caminho solução ser equivalente a um circuito, o depósito deve estar sempre presente como ponto de partida e de chegada na rota que está sendo criada, para que, a cada passo da construção da solução em questão, seja possível manter a verificação das restrições de tempo.

Mesmo que os tempos de viagem sejam simétricos, a solução é um caminho direcionado, pois mesmo se um subcaminho $(p_i, \ldots, p_j)$ é factível, o seu reverso, $(p_j, \ldots, p_i)$ pode violar as janelas de tempo.

2.2.1 Adaptação do método

Levando em conta a natureza assimétrica do PCVJT, substitui-se $N_p(v)$ por duas vizinhanças orientadas:

- $N_p^+(v)$ - vizinhança à direita de v, contendo os p nós sucessores mais próximos de v já no caminho.
- $N_p^-(v)$ - vizinhança à esquerda de v, contendo os p nós antecessores mais próximos de v já no caminho.

Optou-se por um algoritmo que preserva a factibilidade em todo momento, diferentemente de alguns algoritmos para Roteamento de Veículos que permitem passar por soluções infactíveis [8]. A principal razão para essa escolha é que geralmente há muita dificuldade em recuperar a factibilidade de uma solução que viola as restrições de janelas de tempo, como confirmado por testes preliminares realizados por Gendreau[10]. O número de inserções factíveis é mais limitado que no caso do PCV, especialmente tendo em vista o fato de que a inserção de um vértice no caminho parcialmente construído tem impacto sobre todos os seus sucessores.

Em alguns estágios no curso do algoritmo, pode provar-se impossível a inserção de um novo vértice no caminho, caso em que um procedimento de 'retrocesso' pode ser aplicado. No entanto, ainda assim a factibilidade não pode ser garantida, e o algoritmo pode terminar prematuramente sem que alguns nós sejam inseridos.

Por fim, vizinhanças baseadas em distâncias não funcionam bem para o PCVJT, já que dois nós próximo podem ser incompatíveis ou terem janelas de tempo muito distantes. Assim, tanto as distâncias como as janelas de tempo são incorporadas na definição de vizinhança.

2.2.2 Uma heurística para o PCVJT

Como mencionado, para definir as vizinhanças no PCVJT devemos levar em conta distâncias e janelas de tempo. Dados dois vértices v_i e v_j define-se a proximidade entre suas janelas de tempo como:

$$r_{ij} = min\{b_j, b_i + T_{ij}\} - max\{a_j, a_i + T_{ij}\}. \tag{2.1}$$

Um meio natural de definir uma 'pseudo-distância' d_{ij} entre v_i e v_j, seria usar a combinação convexa de T_{ij} e r_{ij}, isto é,

$$d_{ij} = \alpha T_{ij} + (1 - \alpha) r_{ij}, \text{ onde } \alpha \in [0, 1]. \tag{2.2}$$

No entanto, testes realizados por Gendreau mostraram que definir α é problemático, e essa regra produz resultados instáveis; distâncias tendem erroneamente a serem priorizadas em alguns casos, enquanto o mesmo ocorre para janelas de tempo em outros. Um meio mais satisfatório de definir vizinhanças é incluir em $N_p^+(v)$ os p_1 sucessores mais próximos de v já sobre o percurso, com relação a T_{ij}, e os p_2 sucessores mais próximos de v já sobre o percurso, com respeito a r_{ij}, de modo que $p_1 + p_2 = p$. Analogamente define-se $N_p^-(v)$ em termos dos predecessores de v. Além disso, o depósito é sempre incluído tanto em $N_p^+(v)$ quanto em $N_p^-(v)$.

Quando inserimos um vértice no caminho corrente, devemos tomar cuidado para manter a factibilidade. Para isso, computa-se um tempo crítico de partida z_i para cada vértice $v_i \in V$. Esse coeficiente mede o tempo limite para deixar o vértice e é atualizado no decorrer do algoritmo. Os coeficientes z_i são computados do final para o começo do caminho, começando do vértice v_{n+1}.

$$\begin{cases} z_{n+1} = b_{n+1} \\ z_i = min\{b_i, z_{i+1} - T_{i,i+1}\} \end{cases}$$

Quando um vértice v é inserido no caminho corrente usando GENI, a ordem de vários vértices pode ser afetada. Após realizada uma inserção, seja $(v_1, \ldots, v_{i_\alpha}, \ldots, v_{i_\beta}, \ldots, v_{n+1})$ o caminho obtido, onde $(v_{i_\alpha}, \ldots, v_{i_\beta})$ representa a porção modificada pela inserção. Como sabemos, uma condição necessária para que uma inserção potencial seja factível é que o tempo de chegada em cada vértice v_i satisfaça:

$$t_i \leqslant b_i, \ i = i_\alpha, \ldots, i_\beta.$$

Ainda mais, a inserção deve ser factível para todo vértice após v_{i_β}, isto é,

$$t_{i_{\beta+1}} = t_{i_\beta} + T_{i_\beta i_{\beta+1}} \leqslant z_{i_{\beta+1}}$$

De modo contrário ao que acontece em GENI, os vértices não são inseridos de modo aleatório, pois essa regra tende a tornar problemática a inserção dos últimos vértices. Ao invés disso, os vértices são ordenados de acordo com um critério que mede suas dificuldades de inserção, e então, eles são inseridos em ordem não crescente de dificuldade. Alguns critérios podem ser usados:

1. Ordem não decrescente do comprimentos das janelas de tempo; $b_i - a_i$.
2. Ordem não decrescente de a_i.
3. Ordem não decrescente de b_i.

Testes realizados por Gendreau [10], revelam que a primeira regra é a mais satisfatória.

Para realizar uma inserção, vértices não alocados da lista ordenada são considerados separadamente, em cada etapa de inserção. O primeiro vértice que pode ser factivelmente alocado é inserido em sua melhor porção possível no caminho. Quando nenhum vértice ainda não alocado pode ser inserido factivelmente no caminho, o seguinte procedimento de retrocesso é aplicado.

Seja S a lista dos nós ainda não alocados. Começando do sucessor do depósito no caminho, remova por vez, um vértice v_i e coloque-o em S, no final da lista, e tente inserir no caminho cada um dos vértices de S, exceto aquele que acabou de ser removido do caminho, pela seguinte ordem:

1. Se uma inserção é factível, ela é realizada e o procedimento de inserção padrão é retomado.
2. Se nenhuma inserção é factível, o mesmo processo é repetido para o sucessor de v_i, até que se alcance v_{n+1}.

Alguma inserção será factível desde que, no pior caso, todos os vértices exceto v_1 e v_{n+1} possam ser removidos. Para evitar ciclagem, o algoritmo verifica o número de vezes θ_i que um vértice i é colocado de volta em S, e o processo termina com uma solução infactível sempre que um limite superior aceitável $\overline{\theta}$ para os valores θ_i é atingido para algum i.

Se o algoritmo tem sucesso em determinar uma solução factível, um procedimento de pós otimização baseado em US é aplicado, como descrito na seção anterior. Esse processo é aplicado repetidamente até que se varra completamente o caminho sem produzir nenhuma melhoria. Lembre que um rearranjo do percurso é considerado somente se ele preserva a factibilidade sobre as janelas de tempo para todos os vértices.

O pseudo-código do algoritmo é dado a seguir.

Algoritmo 7: GENIUS para PCVJT

Requer: V, a, b, $\overline{\theta}$
Fornece: P
Insira todos os vértices de $V\backslash\{v_1, v_{n+1}\}$ em um conjunto ordenado S, em ordem não decrescente dos comprimentos de suas janelas de tempo.
Construa o primeiro caminho $P = (v_1, v_{n+1})$.
Para todo $i \in V$ faça $\theta_{v_i} = 0$
$k \leftarrow 1$; $s \leftarrow |S|$
while $S \neq \varnothing$ **do**
  Aplique GENI ao vértice $v = S(k)$, para inserí-lo entre dois de seus p-vizinhos.
  if *não houve inserção realizada em GENI* **then**
    $k \leftarrow k + 1$
    if $k = s + 1$ **then**
      Remova o vértice v_i logo após o depósito e coloque-o no final da lista S.
      $\theta_{v_i} \leftarrow \theta_{v_i} + 1$
      if $\theta_{v_i} = \overline{\theta}$ **then**
        Pare, o algoritmo não pode identificar uma solução factível.
      end
      $k \leftarrow 1$
    end
  end
  Atribua a P o melhor caminho obtido na inserção GENI.
  Atualize S e todas as vizinhanças, pois agora v está em P.
  $k \leftarrow 1$; $s \leftarrow |S|$
end
$k \leftarrow 2$
while $k \leqslant n$ **do**
  Aplique US ao vértice v_k
  if *Há solução melhor que a corrente* **then**
    Atribua a P a melhor solução obtida no processo US.
    $k \leftarrow 2$
  end
  $k \leftarrow k + 1$
end

3 Metodologia

Sendo uma junção do PCVCP com o PCVJT, o PCVCPJT, assim como eles, pode apresentar alto grau de complexidade de resolução, sendo inviável resolver problemas grandes de forma exata, devido a restrições tais como recursos computacionais ou tempo de resposta disponível. Sendo assim, é interessante o estudo e o desenvolvimento de métodos que gerem boas soluções (mesmo que não sejam ótimas) de forma rápida. Heurísticas e meta-heurísticas são técnicas desenvolvidas com a finalidade de resolver satisfatoriamente problemas com essas características.

Neste capítulo apresentamos algumas técnicas heurísticas e a meta-heurísticas utilizadas neste trabalho para resolver o PCVCPJT, e finalizamos com um resumo do método adotado para a resolução do mesmo problema.

3.1 Heurísticas e Meta-heurísticas

Um problema de otimização pode ser escrito como

$$min\{f(P); P \in X, X \subset S\}, \tag{3.1}$$

onde S, X, P e f denotam respectivamente o *espaço solução*, o *conjunto factível*, uma solução factível e a função objetivo, que assume valores reais. Se $S = \mathbb{R}^n$, trata-se de um problema de *otimização contínua*. Já se for S um conjunto finito, trata-se de um problema de *otimização combinatória*, como é o nosso caso.

Heurísticas e Meta-heurísticas são técnicas desenvolvidas para resolver, de forma efciente, problemas de otimização combinatória de porte relativamente grande. Essas técnicas não garantem que a solução ótima do problema em questão seja encontrada, tão pouco informam o quão próximo a solução encontrada está da otimalidade. No entanto elas fornecem boas soluções, com um custo de resolução menor que o dos métodos exatos.

3.1.1 Heurísticas

Heurísticas são algoritmos que exploram apenas uma pequena parte do espaço de soluções de um problema, fornecendo uma solução de boa qualidade a baixo custo computacional. Esses algoritmos são particularmente indicados para problemas com alto grau de dificuldade associado a sua resolução exata, como os problemas combinatórios. As heurísticas podem ser classificas de acordo com seu objetivo, sendo de construção ou de refinamento.

3.1.1.1 Heurísticas de Construção

Normalmente utilizadas no início do processo de resolução de um problema, as heurísticas de construção consistem em construir uma solução factível, elemento a elemento, segundo algum critério de otimização, até que todos os elementos façam parte da solução ou até que algum critério de parada seja satisfeito. A seguir, apresentamos exemplos de heurísticas de construção específicas para o PCV, dentre as quais algumas utilizadas neste trabalho.

É importante ressaltar que, sendo nosso interesse a resolução de um problema que possui janelas de tempo, devemos adicionar a essas heurísticas, a verificação de factibilidade com respeito as restrições de tempo, em cada passo da construção.

- **Heurísticas de ordenação (Sorting heuristics)**

  Consistem em produzir uma rota que representa uma sequência dos nós sob algum critério de ordenação. Dentre as várias possibilidades de ordenação, escolhemos as seguintes:

  1. Checar se a sequência trivial é factível:

  $$(1, 2, 3, \cdots, n)$$

  2. Ordenar os nós de acordo com a ordem crescente dos limites superiores de suas respectivas janelas de tempo e checar se essa sequência é factível.

  3. Ordenar os nós de acordo com a ordem crescente dos limites inferiores de suas respectivas janelas de tempo e checar se essa sequência é factível.

  4. Ordenar os nós de acordo com o ponto médio de suas respectivas janelas de tempo e checar se essa sequência é factível. O ponto médio da janela de tempo de um nó 'i', é dado por:

  $$m_i = \frac{a_i + b_i}{2}$$

- **Heurística do Vizinho mais próximo**

  Consiste em partir de um caminho inicial formado por um arco factível $(1, v) \in A$ e adicionar, a cada iteração, o cliente ainda não alocado que está mais próximo do último inserido na rota corrente, desde que o circuito resultante seja factível. Mais precisamente, partindo de $(1, v) \in A$, após um certo número de passos obtemos um caminho parcial $P' = (1, v_1, \ldots, v_k)$. Escolhemos então, um nó v_l ainda não alocado, tal que v_l está factivelmente mais próximo de v_k, isto é, tal que acrescentar

o arco (v_k, v_l) ao final de P' gera o menor aumento de função objetivo e a rota $(1, v_1, \ldots, v_k, v_l)$ é factível. Note que para cada v escolhido para iniciar a construção do caminho, obtem-se uma rota diferente. Desse modo, com essa heurística podemos obter até $n-1$ soluções, e nos reservar ao direito de escolher a melhor delas.

- **Heurísticas de inserção (Insertion heuristics)**

Nessa classe de heurísticas, partindo do depósito (nó 1), construimos um caminho inicial escolhendo o nó $v_i \in V\backslash\{1\}$, tal que o caminho $(1, v_i, 1)$, seja o mais curto (ou mais barato). Assim, na etapa k de inserção, teremos um caminho parcial da forma $P' = (1, v_1, \ldots, v_k, 1)$ e o conjunto $W = V\backslash\{1, v_1, \ldots, v_k\}$, que contém os nós ainda não inseridos no caminho. Nesse caso, para cada $j \in W$, definimos

$$d_j := min\{c_{v_l j} + c_{j v_{l+1}} - c_{v_l v_{l+1}}; j \in W, v_l \in P' \text{ e } (1, v_1, \ldots, v_l, j, v_{l+1}, \ldots, v_k, 1) \text{ é factível}\}.$$

O caminho para o qual exigimos a factibilidade na definição de d_j é resultado da inserção do nó j em P', entre os nós v_l e v_{l+1}. Adotamos duas estratégias de inserção que resultam em heurísticas distintas, uma priorizando a inserção mais barata, e outra priorizando a urgência de atendimento.

(a) Inserção visando o menor custo de inserção:

Dentre todos os nós ainda não inseridos, escolha o nó $j \in W$ que causa o menor aumento no comprimento do caminho P', inserindo-o na posição que proporciona esse menor acréscimo no caminho.

(b) Inserção visando urgência de atendimento:

Dentre todos os nós ainda não inseridos, escolha o nó $j \in W$ para o qual há o menor número de inserções factíveis possíveis, inserindo-o na posição que gere o menor aumento de comprimento do caminho.

3.1.1.2 Heurísticas de Refinamento

As heurísticas de refinamento, são técnicas de busca local que buscam melhorar uma solução, por meio da exploração de sua vizinhança. Isso se dá por alterações realizadas na solução fornecida, até que não haja melhoria possível ou até que algum critério de parada seja satisfeito. Apresentamos a seguir exemplos de heurísticas de refinamento, dentre as quais algumas utilizadas neste trabalho.

- **k-optimal**

As heurísticas k-optimal (ou simplesmente, k-opt) foram introduzidas por Croes [13] e Lin [12], com o intuito de melhorar soluções do PCV. Para um dado número k, e uma

solução S, essa heurística consiste em remover k arcos de S, e substituí-los por outros k arcos diferentes, obtendo assim uma nova rota S'. Caso S' satisfaça as restrições do problema, tem-se uma nova solução factível. O objetivo deste procedimento é diminuir o custo de deslocamento em S, o que ocorre quando S' tem custo menor que S ou, em outras palavras, quando a soma dos custos das arestas que entram na solução é menor do que a soma dos custos das arestas removidas.

A Figura 5 mostra um exemplo de aplicação da heurística k-opt para $k = 2$. Os arcos removidos são $(2, 4)$ e $(3, 5)$, enquanto que os os novos arcos são $(2, 3)$ e $(4, 5)$. De modo geral, após a remoção de k arestas, essa técnica testa todas as possibilidades de reconexão dos nós e escolhe aquela que resulta na solução de menor custo. Neste trabalho, usamos essa heurística de refinamento com valores $k = 2, 3$. Apesar de ser possível obter melhores mudanças na solução para valores maiores de k, a complexidade do método também aumenta exponencialmente com o valor de k, de modo que não é comum empregar k maior que 3 ou 4.

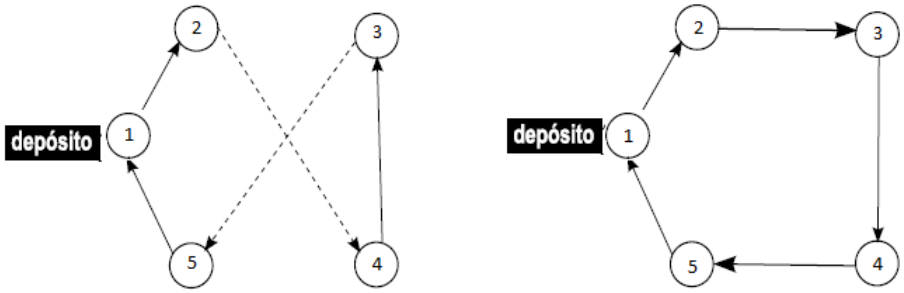

Figura 5 – Aplicação do método 2-opt.

- **Swap heuristic**

  Dada uma solução factível S, essa heurística consiste em percorrer todo o percurso dado, verificando se a troca de dois nós consecutivos v_i e v_{i+1}, $i = 1, \ldots, n$, resulta numa solução factível com melhor valor de função objetivo. Caso isso ocorra, a troca é efetuada.

- **Troca de dois nós (Two-Node-Exchange heuristic)**

  Trata-se de uma generalização da heurística anterior, onde quaisquer dois nós (não somente nós consecutivos) na solução corrente pode ser trocados. Se isso resultar numa rota factível mais barata que a anterior, a troca é feita.

- **Reinserção de um nó (Node-Reinsertion heuristic)**

Dada uma solução S e um nó interno v_j, esta técnica consiste em remover v_j de S, obtendo assim um caminho S'. Em seguida, tenta-se resinserir o nó v_j em S', em todas as posições possíveis, escolhendo-se aquela que gere o menor custo de inserção, e tal que a rota obtida seja factível. Se a solução obtida tem custo menor que o de S, então a reinserção é feita.

- **Reinserção de um arco (Arc-Reinsertion heuristic)**

Dada uma solução S, essa heurística consiste em remover dois nós consecutivos v_i e v_{i+1}, obtendo S'. Em seguida, tenta-se reinserir o par (v_i, v_{i+1}) em todas as posições possíveis de S', guardando o caminho obtido de menor custo com essa inserção, dentre os que são factíveis. Se esse caminho tiver custo menor que S, então ele é aceito.

3.1.2 Meta-heurísticas

De acordo com a definição original, meta-heurísticas são métodos de solução que coordenam procedimentos de busca locais com estratégias de mais alto nível, de modo a criar um processo capaz de escapar de mínimos locais e realizar uma busca robusta no espaço de soluções de um problema. Posteriormente, a definição passou a abranger quaisquer procedimentos que empregassem estratégias para escapar de mínimos locais em espaços complexos de soluções. Em especial foram incorporados procedimentos que utilizam o conceito de vizinhança para estabelecer um meio de fugir dos mínimos locais. Uma meta-heurística, portanto, visa produzir um resultado satisfatório para um problema, porém sem qualquer garantia de otimalidade[11].

3.2 Busca em Vizinhança Variável

Busca em Vizinhança Variável - do inglês, *Variable Neighborhood Search* (VNS) - é uma meta-heurística com o objetivo de resolver problemas de otimização combinatória e global.

Uma solução de 3.1, $P^* \in X$, é ótima se

$$f(P^*) \leqslant f(P), \forall P \in X.$$

Um algoritmo exato para o problema 3.1, se existe, encontra a solução ótima $P*$, ou mostra que não existe solução factível, isto é, que $X = \varnothing$. Entretanto, na prática, o tempo necessário para resolver o problema deve ser finito e não muito longo.

Muitas instâncias práticas de problemas da forma 3.1 são muito grandes para que uma solução exata seja encontrada em tempo razoável. É bem conhecido da Teoria

da Complexidade [6] [14] que milhares de problemas são NP-completos, de modo que não se conhece qualquer algoritmo que seja capaz de resolvê-los em um número de passos polinomial com relação ao tamanho das instâncias. Além disso, se uma solução de um problema NP-completo fosse encontrada em tempo polinomial, então todo problema da classe NP poderia ser resolvido em tempo polinomial. [16] Isso explica a necessidade de recorrer a heurísticas que produzam rapidamente uma solução aproximada, ou mesmo, uma solução ótima, para a qual não se tenha prova de otimalidade.

A meta-heurística VNS inclui uma heurística de busca local para resolver problemas de otimização combinatória e global. Desde sua criação, ela tem passado por muitos desenvolvimentos e tem sido aplicada em numerosas áreas. Sua ideia básica consiste numa troca sistemática de vizinhança, combinada com uma busca local. Essa ideia tem tido muitos predecessores. Isso permite a troca de *estruturas de vizinhanças* dentro dessa busca. Vejamos então como obter tais estruturas e como essa busca funciona.

3.2.1 Estruturas de Vizinhança

Entendamos por estrutura de vizinhança, o termo que indica o método pelo qual obtemos novas soluções a partir de uma solução dada e denotemos por N_k $(k = 1, \ldots, k_{max})$ a k-ésima estrutura de vizinhança do nosso algoritmo. $N_k(P)$ é o conjunto finito das soluções obtidas a partir de P pelo método da estrutura de vizinhança N_k. Nesse caso, dizemos que $N_k(P)$ é a k-ésima vizinhança de P, e que um elemento pertencente a este conjunto é um vizinho de P.

Neste trabalho, as estruturas de vizinhanças são baseadas nas heurísticas de refinamento. As heurísticas apresentadas em 3.1.1.2 são usualmente adotadas para a resolução do PCV e suas variantes.[1] Dada uma rota P, a escolha da estrutura de vizinhança N_k determina o *movimento* que será aplicado à P para se obter uma nova solução.

Para fixar ideias, considere um PCV de tamanho $n = 4$, para o qual o circuito $P = (1, 2, 3, 4)$ representa uma solução. Digamos, por exemplo, que N_4 seja a estrutura de vizinhança baseada na heurística Troca de dois nós [34]. Dessa forma, a vizinhança $N_4(P)$ é o conjunto de todas as soluções obtidas a partir de P, por aplicar um movimento de troca de dois nós. Mais explicitamente, neste exemplo, tem-se:

$$N_4(P) = \{(2,1,3,4), (3,2,1,4), (4,2,3,1)$$
$$(1,3,2,4), (1,4,3,2), (1,2,4,3)\}.$$

Seguindo esse raciocínio, supondo que N_5 se baseia no método 2-opt, o conjunto de todas as possíveis soluções obtidas a partir de P por substituir dois de seus arcos (movimento 2-opt) é dado por:

$$N_5(P) = \{(1,2,4,3), (1,3,2,4)\},$$

[1] Ver por exemplo, [26], [27] e [34]

onde a primeira rota desse conjunto é obtida da troca dos arcos $(2, 3)$ e $(4, 1)$ pelos arcos $(2, 4)$ e $(3, 1)$, respectivamente, enquanto a segunda, da troca dos arcos $(1, 2)$ e $(3, 4)$ pelos arcos $(1, 3)$ e $(2, 4)$, respectivamente. Na Seção 3.3, o leitor encontrará a descrição de cada estrutura de vizinhança adotada no algoritmo proposto.

Uma *solução ótima* é uma solução factível onde o mínimo do problema 3.1 é alcançado. Diz-se que $P' \in X$ é um *mínimo local* do problema 3.1 com respeito a N_k se não existe solução $P \in N_k(P') \subset X$ tal que

$$f(P) < f(P').$$

A VNS se baseia nas seguintes afirmações:

1. Um mínimo local com respeito a uma estrutura de vizinhança, não necessariamente é um mínimo local com respeito a outra estrutura vizinhança.
2. Um mínimo global é um mínimo local com respeito a todas possíveis estruturas de vizinhanças.
3. Os mínimos locais com respeito a diferentes estruturas de vizinhanças estão relativamente próximos uns dos outros.

A VNS é uma meta-heurística que sistematicamente explora a ideia de troca de vizinhanças. Essa troca de vizinhanças pode produzir algoritmos mais eficientes, uma vez que um mínimo local de uma vizinhança pode ter melhor valor que o mínimo de outra. Isso aumenta as chances de se encontrar um mínimo global. Portanto, realizar buscas locais em diferentes vizinhanças para resolver problemas NP-difíceis pode muito bem conduzir a maiores benefícios.

3.2.2 Busca Local

Uma heurística de busca local consiste em escolher uma solução inicial P, encontrar uma direção de descida a partir de P, dentro de uma vizinhança $N(P)$, e mover para o mínimo de $f(P)$ dentro de $N(P)$ na direção encontrada. Se não existe direção de descida, a heurística para. Normalmente, a direção de máxima descida, também referida como direção de *maior melhoria* (do inglês, *best improvement*), é usada. Esse conjunto de regras é resumido no Algoritmo 8, em que assumimos que uma solução inicial P seja dada. O algoritmo fornece um mínimo local, denotado por P'. Observe que a estrutura de vizinhança $N(P)$ é definida para todo $P \in X$. Em problemas de otimização combinatória, como é o nosso caso, essa vizinhança consiste de todos os vetores obtidos a partir de P, por alguma modificação, como por exemplo, a troca de dois nós quaisquer de P. A cada passo do algoritmo, a vizinhança $N(P)$ é explorada completamente.

Algoritmo 8: BestImprovement

Requer: P
Fornece: P'
$z \leftarrow \infty$
while $f(P) < z$ **do**
  $P' \leftarrow P$
  $z \leftarrow f(P')$
  $P \leftarrow arg\ min_{Q \in N(P)} f(Q)$
end

Uma vez que explorar completamente grandes vizinhanças pode consumir muito tempo, uma alternativa mais atraente é usar a heurística de *primeira melhoria* ou *primeira descida* (do inglês, *first improvement*, ou *first descent*). Nesse caso, vetores $P_i \in N(P)$ são enumerados sistematicamente e um movimento é feito assim que uma direção de descida é encontrada. Essa estratégia está resumida no Algoritmo 9. $|N(P)|$ e P_i indicam o número de elementos do conjunto $N(P)$ e o i-ésimo elemento de $N(P)$, respectivamente.

Algoritmo 9: FirstImprovement

Requer: P
Fornece: P'
$z \leftarrow \infty$
while $f(P) < z$ **do**
  $P' \leftarrow P$; $z \leftarrow f(P')$; $i \leftarrow 1$
  while $f(P) \geqslant z\ and\ i \leqslant |N(P)|$ **do**
    $P \leftarrow P_i$
    $i \leftarrow i + 1$
  end
end

3.2.3 Esquemas básicos de VNS

Visando resolver o problema 3.1, podemos usar a meta-heurística VNS de três diferentes modos: determinístico; estocástico; combinado, ou seja, com mudanças de vizinhança determinísticas e estocásticas. Primeiro, apresentamos o Algoritmo 10, que fornece a função de troca de vizinhança, usada nesse trabalho. Essa função, que chamaremos de NeighborhoodChange, consiste em comparar um novo valor $f(P')$, com o valor atual, $f(P)$, obtido na vizinhança k. Se ocorre uma melhoria, k é retornado para seu valor inicial, e a solução P é atualizada. Caso contrário, a próxima vizinhança é considerada.

Algoritmo 10: NeighborhoodChange

Requer: P, P', k
Fornece: P, k
if $f(P') < f(P)$ **then**
  $P \leftarrow P'$
  $k \leftarrow 1$
end
$k \leftarrow k + 1$

3.2.3.1 Descida em Vizinhança Variável

O método de *Descida em Vizinhança Variável* - do inglês, *Variable Neighborhood Descent* (VND) - é definido por uma troca de vizinhança determinística. Seus passos são apresentados no Algoritmo 11. Na descrição dos próximos algoritmos, consideramos que uma solução inicial P é dada. A maioria das heurísticas baseadas em busca local, usa muito poucas vizinhanças, não passando de duas, isto é $k_{max} \leqslant 2$. Note que a solução final pode ser um mínimo local com respeito a todas as vizinhanças. Desse modo, as chances de alcançar um mínimo global são maiores usando a VND, do que usando uma única estrutura de vizinhança. Além disso, dado que as estruturas de vizinhança da VND são percorridas em uma certa ordem, pode-se desenvolver uma estratégia de aninhamento, ou seja, a vizinhança $N_k(P)$ pode estar contida em $N_{k+1}(P)$.

Algoritmo 11: VND

Requer: P, k_{max}
Fornece: P
$z \leftarrow \infty$
while $f(P) < z$ **do**
  $k \leftarrow 1$
  $z \leftarrow f(P)$
  while $k < k_{max}$ **do**
    $P' \leftarrow arg\ min_{Q \in N_k(P)} f(Q)$
    NeighborhoodChange(P, P', k)
  end
end

3.2.3.2 VNS Reduzida

O método *VNS Reduzido* - do inglês, *Reduced VNS* (RVNS) - é obtido se pontos aleatórios são definidos em $N_k(P)$, sem que se exija que a direção seja de descida. Nesse caso, os valores dos novos pontos são comparados com aquele da solução atual, e tomam o lugar desta em caso de melhoria. Os passos da RVNS são apresentados no Algoritmo 12. Veja que, nesse algoritmo, uma função *Shake* é utilizada. Essa função gera um ponto P' aleatório na k-ésima vizinhança de P, isto é, $P' \in N_k(P)$.

A RVNS é muito útil para instâncias muito grandes, para as quais a busca local é cara. Tem sido observado, como dito em [16], que o valor frequentemente adotado para o parâmetro k_{max} é 2. Além disso, podemos considerar como critério de parada, dentre várias possibilidades, o tempo máximo de CPU permitido, t_{max}, ou o número máximo de iterações entre duas melhorias.[2].

Algoritmo 12: RVNS

Requer: P, k_{max}, t_{max}
Fornece: P
$t \leftarrow 0$
while $t \leqslant t_{max}$ **do**
  $e \leftarrow cputime$
  $k \leftarrow 1$
  while $k < k_{max}$ **do**
    $P' \leftarrow Shake(P, k)$
    NeighborhoodChange(P, P', k)
  end
  $t \leftarrow t + cputime - e$
end

3.2.3.3 VNS Básico

O método *VNS Básico* [23] - do inglês, *Basic VNS* (BVNS) - combina mudanças de vizinhanças determinísticas e estocásticas. Muitas vezes, vizinhanças N_k consecutivas são aninhadas. Os passos do método são fornecidos no Algoritmo 13. Observe que o ponto P' é gerado de forma aleatória, com o objetivo de evitar ciclagem, que pode ocorrer se uma regra determinística for aplicada. Em seguida, a heurística de busca local de primeira melhoria (Algoritmo 9) é adotada. No entanto, esta pode ser substituida pela heurística de busca local de máxima descida (Algoritmo 8). Um exemplo ilustrativo de aplicação do VNS Básico pode ser visto em [16].

3.2.3.4 VNS Geral

O método *VNS Geral* - do inglês, *General VNS* (GVNS) - pode ser obtido do BVNS, fazendo-se uma simples mudança na busca local do algoritmo. No GVNS, ao invés do FirstImprovement, usa-se a VND. O uso do VNS Geral tem produzido os maiores sucessos já reportados na literatura.[3] Os passos do VNS Geral são dados no Algoritmo 14.

Apesar do GVNS ser mais completo, para termos um algoritmo mais rápido e podermos explorar uma quantidade maior de vizinhanças, optamos por implementar o BVNS. Nas seções a seguir, retornamos ao problema central deste trabalho, apresentando a abordagem realizada.

[2] Resultados utilizando RVNS podem ser vistos em [21] e [22]
[3] Veja por exemplo [19, 24, 25]

Algoritmo 13: BVNS

Requer: P, k_{max}, t_{max}
Fornece: P
$t \leftarrow 0$
while $t \leqslant t_{max}$ **do**
  $e \leftarrow cputime$
  $k \leftarrow 1$
  while $k < k_{max}$ **do**
    $P' \leftarrow Shake(P, k)$
    $P'' \leftarrow FirstImprovement(P')$
    NeighborhoodChange(P, P'', k)
  end
  $t \leftarrow t + cputime - e$
end

Algoritmo 14: GVNS

Requer: $P, k_{max}, k'_{max}, t_{max}$
Fornece: P
$t \leftarrow 0$
while $t \leqslant t_{max}$ **do**
  $e \leftarrow cputime$
  $k \leftarrow 1$
  while $k < k_{max}$ **do**
    $P' \leftarrow Shake(P, k)$
    $P'' \leftarrow VND(P', k'_{max})$
    NeighborhoodChange(P, P'', k)
  end
  $t \leftarrow t + cputime - e$
end

3.2.4 Representação de uma solução

Assim como em outros problemas combinatórios, neste trabalho optou-se por representar computacionalmente as soluções do PCVCPJT atráves de vetores de números inteiros, nos quais cada índice armazenado representa uma cidade (cliente, nó, etc.). Como é de praxe, o termo n é usado para representar o número total de vértices, incluindo o depósito. O vetor solução é representado por S, e contém todas as cidades candidatas a fazer parte da rota. Esse vetor S é composto por outros dois, a saber, R que contém a rota solução, e NR, que contém os demais nós. Desse modo, S é a junção dos dois vetores, vindo primeiro os elementos de R, seguidos dos elementos de NR:

$$S = (R, NR)$$

O tamanho de R, é o tamanho da solução, que não necessariamente é igual ao número de cidades n, já que no PCVCPJT, assim como no PCVCP, nem sempre todas as cidades fazem parte da rota. A ordenção dos itens no vetor R, e consequentemente no

vetor S, indica a ordem de visitação das cidades pelo caixeiro viajante. Como exemplo, imagine um problema contendo 6 cidades, além do depósito, que associamos ao nó 1.

Figura 6 – Representação de um solução de tamanho 5 e $n = 7$

A Figura 6 representa, através de um vetor, uma solução com tamanho igual 5, embora n seja igual a 7. A solução é composta pelos arcos $(i, i + 1)$, com $i = 1, \ldots, 5$. Os nós 6 e 7 (em azul) não fazem parte da rota obtida como solução. A Figura 7 mostra a representação gráfica dessa solução. Os quadros vizinhos aos nós indicam a penalidade, seguida do prêmio, $(\pi_i; p_i)$.

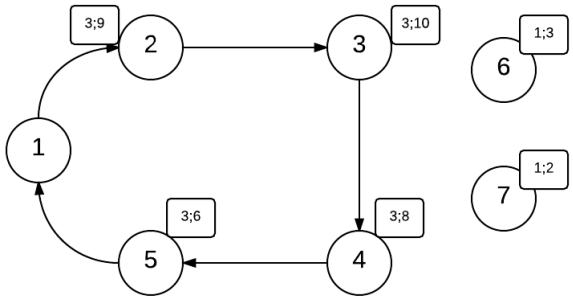

Figura 7 – Representação gráfica de um solução de tamanho 5 e $n = 7$

Observe que na Figura 7 não há um quadro indicando o prêmio e penalidade do nó 1. Nos problemas deste trabalho, os dados para o nó 1, que representa o depósito, são: $p_1 = 0$ e $\pi_1 = \infty$. Assim, é sempre melhor inclur o depósito na rota do que ser penalizado por não incluí-lo. Do ponto de vista heurístico, isso é desnecessário, já que a rota é sempre construída com início e término nele.

Além disso, ao resolvermos um PCVCPJT, consideramos que os dados dos clientes incluem uma penalidade grande o suficiente para que o algoritmo exato não escolha excluir algum deles apenas por melhoria da função objetivo.

3.2.5 Alguns movimentos adicionais

Uma vez que nosso objetivo é atender o máximo de clientes possível, complementamos o algoritmo incluindo alguns movimentos que colaboram para a obtenção de soluções mais completas - que contenham mais nós. Os movimentos aqui apresentandos, definem novas estruturas de vizinhanças, que estabelecem relações entre os conjuntos R e NR, quando $NR \neq \varnothing$. São eles:

- **M1** Inserir entre um par de nós da rota um vértice que está fora dela.
- **M2** Trocar um vértice que faça parte da rota por um que não faça parte da rota.

Se a estrutura N_1 determina a aplicação do movimento **M1**, então para uma dada solução R, $N_1(R)$ contém todas as soluções que podem ser obtidas por inserir um nó de NR entre um par de nós de R. Já se for N_2 baseada no movimento **M2**, $N_2(R)$ contém todas as soluções obtidas por trocar um nó de R por um de NR. Observe que não adotamos nenhum movimento que diminui o tamanho da rota solução (como remoção de um nó, por exemplo). Desta forma, uma vez que a rota atinge um tamanho, ela não passará a ter um número menor de clientes. Optamos por fazer isso pelo fato de que pode ser muito difícil recuperar a factibilidade das janelas de tempo após excluir um nó que já ocupava uma posição factível. Ainda assim, com esses movimentos adicionais, nossa busca ganha um pouco de flexibilidade e eficiência. Esses movimentos também são úteis caso a solução inicial obtida contenha poucos clientes, aumentando o número de clientes que serão atendidos ou reestruturando a rota de modo que isso venha a ser possível.

3.3 O Método Proposto

Dado um PCVCPJT, algoritmo proposto para sua resolução é composto por três etapas, sendo as duas primeiras baseadas em heurísticas e a última em uma meta-heurística. A primeira etapa consiste na aplicação das heurísticas de construção abaixo.

- Heurísticas de ordenação: por ordem não decrescente de a_i; por ordem não decrescente de b_i; por ordem não decrescente de $\frac{a_i + b_i}{2}$.
- Heurísticas de inserção: priorizando menor custo de inserção; priorizando urgência de atendimento.
- Heurística do vizinho mais próximo.

Na segunda etapa, aplica-se a versão para o problema com janelas de tempo do método GENI (2.1.1), desvinculado de sua fase de pós otimização US. Por fim, a terceira etapa consiste na utilização do procedimento BVNS, no qual implementamos um controle inteligente para a exploração das vizinhanças.

Uma vez que pode não ser possível atender todos os clientes por conta de restrições de tempo, na primeira etapa do método, cada heurística de construção procede da forma como foi apresentada, porém passa a fornecer R (vetor que representa a rota produzida) e NR (vetor contendo os nós que ela foi incapaz de inserir na rota). Dentre os percursos fornecidos por todas as heurísticas de construção, escolhemos o mais barato.

Apesar de o procedimento GENI também ser uma heurística de construção, ela é aplicada na segunda etapa, separadamente das demais. Assim como as outras, ela também é adaptada para forncecer R e NR. Após sua aplicação, comparamos o resultado obtido nessa etapa com o resultado obtido na etapa anterior, mantendo o melhor deles. A fase US da heurística GENIUS é omitida em nosso método, deixando a tarefa de melhoria da solução a cargo da meta-heurística VNS.

A terceira e última etapa do algoritmo proposto é regida pelo VNS Básico, onde são exploradas vizinhaças cujo espaço de soluções, como vimos anteriormente, são compostos por caminhos que podem ser obtidos a partir de movimentos de uma heurística de refinamento. Considerando R como a solução corrente, as vizinhanças são:

- $N_1(R)$ - Espaço de todas as soluções obtidas a partir de um movimento **M1** aplicado a R.
- $N_2(R)$ - Espaço de todas as soluções obtidas a partir de um movimento **M2** aplicado a R.
- $N_3(R)$ - Espaço de todas as soluções obtidas a partir de R por reinserção de um de seus arcos.
- $N_4(R)$ - Espaço de todas as soluções obtidas a partir de R por troca de dois de seus nós.
- $N_5(R)$ - Espaço de todas as soluções obtidas a partir de R por uma aplicação 2-opt.
- $N_6(R)$ - Espaço de todas as soluções obtidas a partir de R por uma aplicação 3-opt.

Dessa forma, em nosso algoritmo, temos $k_{max} = 6$. Incluímos os movimentos **M1** e **M2**, para contemplar os casos em que as heurísticas de construção não fornecem uma solução de tamanho n.

É importante ressaltar que, no que se refere a relação entre conjuntos, as vizinhanças 1, 2 e 3 não possuem relação alguma com as vizinhanças 4, 5 e 6, o que não condiz com o aninhamento de vizinhanças que é característico da VNS. Essa mudança de vizinhança é mais comum em algoritmos chamados *Adaptive Large Neighborhood Search* (ALNS).

Considere dados um número natural n, uma matriz de custos c de ordem n, um vetor de prêmios p, de números não negativos p_i, um vetor de penalidades π, de valores não negativos π_i e janelas de tempo $J_i = [a_i, b_i]$, com $0 \geqslant a_i > b_i$, para $i = 1, \ldots, n$. O método proposto está resumido no Algortimo 15.

Algoritmo 15: Algoritmo Proposto

Requer: c, n, a, b, p, π
Garante: R, NR
Aplique as heurísticas de construção:
  Heurísticas de ordenação
  Heurísticas de inserçao
  Heurística do vizinho mais próximo
Aplique a heurística GENI adaptada.
Defina R como a melhor rota construída e NR como a lista dos nós ainda não alocados em R.
Aplique o método BVNS adaptado.

O Algoritmo 16 representa o procedimento VNS como usado no método proposto.

Algoritmo 16: BVNS adaptado

Requer: R, z, NR, it_{max}, t_{max}
Garante: R, z
$t \leftarrow 0$; $it \leftarrow 0$; $k_{max} \leftarrow 6$
$e \leftarrow cputime$
while $t \leqslant t_{max}$ *and* $it \leqslant it_{max}$ **do**
  $it \leftarrow it + 1$
  if $NR = \varnothing$ **then**
    $k \leftarrow 3$
  end
  $k \leftarrow 1$ **while** $k \leqslant k_{max}$ **do**
    $R' \leftarrow Shake(R, k)$
    $R'' \leftarrow FirstImprovement(R')$
    $NeighborhooodChange(R, R'', k)$
  end
  $t \leftarrow cputime - e$
end

As tecnicas de descida incluídas no *FirstImprovement* são aquelas que exploram as vizinhanças $N_k(R)$ $(k = 1, 2, \ldots, 6)$, descritas nesta seção. Note que, uma vez que todos os nós estejam inseridos na rota, os movimentos **M1** e **M2** tornam-se impraticáveis, de modo que, a partir desse momento, as vizinhanças $N_1(P)$ e $N_2(P)$ não podem mais ser exploradas. Além disso, a função *NeighborhoodChange* também foi adaptada para retornar ao valor $k = 3$ quando houver melhoria e NR for vazio.

4 Experimentos Computacionais

O algoritmo proposto foi codificado na versão 7.10 do MatLab, de 2010. Para resolução da formulação (1.34) - (1.40), fez-se uso do software GLPK, versão 4.50 (DRAFT, 2013), visando encontrar a solução ótima para o PCVCPJT. Os testes foram realizados em um microcomputador com sistema operacional Windows 7 Professional 64-bit, Processador Intel(R) Core(TM) i7-4790 (8 CPUs de 3.6 GHz) e 16 GB de Memória RAM. Assim como em [28], adotamos os parâmetros $\alpha = 0,2$ e o prêmio mínimo sendo 75% do prêmio total (soma de todos os prêmios). Testes preliminares mostraram que os valores $\theta = 4$, $p_1 = 8$ e $p_2 = 6$ conduzem a heurística GENI a fornecer soluções iniciais de melhor qualidade, de modo que adotamos estes valores para prosseguir com os testes. Além disso, para o BVNS adaptado, adotamos $t_{max} = 2n$ segundos e $it_{max} = n$. Esses testes preliminares foram realizados para todos os problemas apresentados neste capítulo, onde p_1 e p_2 variavam no conjunto $\{1, 2, \ldots, 10\}$ e θ em $\{1, 2, \ldots, 6\}$. Os valores para t_{max} e it_{max} foram reduzidos conforme mostrava-se desnecessário uma quantidade maior de tempo ou iterações, isto é, a solução estagnava, de modo que iterar mais ou permitir um tempo maior de execução do algoritmo não nos conduzia a melhores resultados.

O intuito dos testes é validar a abordagem proposta como um método heurístico eficiente para solucionar o PCVCPJT, principalmente para instâncias maiores, para as quais a resolução exata é inviável. Os problemas para testes estão divididos em quatro grupos:

1. Problemas do Caixeiro Viajante com Janelas de Tempo, propostos por Ascheuer [33].
2. Problemas de Roteamento de Veículos com Janelas de Tempo (PRVJT), propostos por Solomon [32].
3. Problemas do Caixeiro Viajante com Coleta de Prêmios, propostos por Chaves [28].
4. Problemas do Caixeiro Viajente com Coleta de Prêmios e Janelas de Tempo, propostos neste trabalho.

Os problemas dos grupos 1 e 2 estão disponíveis em http://lopez-ibanez.eu/tsptw-instances. Os do grupo 3, em http://www.lac.inpe.br/ lorena/instancias.html. Ao que é do nosso conhecimento, não há uma biblioteca pública de problemas-teste do PCVCPJT. Por esse motivo, para avaliar o método heurístico proposto neste trabalho, cada problema do tipo 4 é obtido por meio da junção de um problema do grupo 1 ou 2 com um do grupo

3. Uma descrição precisa de como estes problemas são obtidos é feita juntamente com a apresentação das Tabelas 4 e 5.

Nas tabelas apresentadas neste capítulo, a coluna 'Problema' indica a instância à qual os dados na mesma linha se referem. A coluna 'n', representa o número de nós da instância, incluindo o depósito. Além disso, os tempos são dados em segundos. As colunas abaixo da frase 'Algoritmo proposto' contêm os resultados obtidos pelo algoritmo proposto neste trabalho, enquanto as que estão abaixo de 'Método exato' se referem aos resultados obtidos pelo GLPK.

4.1 Comparação dos resultados

As duas primeiras tabelas apresentam resultados de testes que confirmam a eficiência do algoritmo proposto, com respeito à qualidade das soluções. Na Tabela 1, comparamos as melhores soluções conhecidas para dez instâncias do primeiro grupo com as soluções encontradas pelo método proposto. No mesmo endereço eletrônico onde constam essas instâncias, podem ser encontradadas as informações sobre suas soluções. Podemos notar que dos valores obtidos, 5 são iguais aos das melhores soluções. Dos outros 5, rbg048a é o que apresenta a maior perda, sendo esta aproximadamente 0,46% do melhor valor encontrado. Dessa forma, concluímos que para essa amostra, o método produz boas soluções para variados tamanhos de instâncias, obtendo soluções iguais ou muito próximas das melhores conhecidas.

Tabela 1

Problema	n	Melhor solução conhecida	Algoritmo proposto	
			Solução	Tempo (s)
rbg010a	11	671	671	0,5
rbg021.7	20	4479	4479	18,5
rbg031a	32	1863	1863	80,1
rbg038a	39	2480	2484	101,3
rbg048a	49	9383	9426	255,8
rbg055a	56	3761	3761	175,4
rbg067a	68	4625	4625	230,9
rbg086a	87	8400	8413	991,3
rbg092a	93	7160	7191	1390,0
rbg125a	126	7936	7951	2088,2

Tabela 1 – Resultados computacionais para os problemas-teste 1.

A Tabela 2 contém informações sobre os problemas do segundo grupo. A coluna Solução 1, se refere aos valores das soluções reportadas por Potvin *et al.* [35]. Os valores contidos na coluna Solução 2, foram obtidos por Gendreau como resultado da aplicação

Tabela 2

				Algoritmo proposto			
Problema	n	Solução 1	Solução 2	Solução	Ganho Sol. 1 (%)	Ganho Sol. 2 (%)	Tempo (s)
rc 201.1	20	465,53	444,54	444,54	4,5	0,0	17,5
rc 201.2	26	739,79	712,91	711,53	3,8	0,2	55,3
rc 201.3	32	839,76	795,44	795,44	5,3	0,0	92,5
rc 201.4	26	803,55	793,64	793,64	1,2	0,0	44,1
rc 202.1	33	844,97	772,18	772,18	8,6	0,0	126,7
rc 202.2	14	328,28	304,14	304,14	7,3	0,0	4,2
rc 202.3	29	878,65	839,58	839,58	4,4	0,0	83,7
rc 202.4	28	852,73	793,03	793,03	7,0	0,0	50,5
rc 203.1	19	481,13	453,48	453,48	5,7	0,0	14,7
rc 203.2	33	843,22	784,16	784,16	7,0	0,0	88,7
rc 203.3	37	911,18	842,25	817,53	10,3	2,9	95,3
rc 203.4	15	330,33	314,29	314,29	4,8	0,0	6,7
rc 204.1	45	923,86	897,09	878,76	4,9	2,0	127,4
rc 204.2	33	686,56	679,26	663,19	3,4	2,4	81,1
rc 204.3	34	455,03	460,24	455,03	0,0	1,1	51,3
rc 205.1	14	381,00	343,21	343,21	9,9	0,0	3,3
rc 205.2	27	796,57	755,93	768,10	3,6	-1,6	48,2
rc 205.3	35	909,37	825,06	825,06	9,3	0,0	176,6
rc 205.4	28	797,20	762,41	773,20	3,0	-1,4	50,3
rc 206.1	4	117,85	117,85	117,85	0,0	0,0	0,0
rc 206.2	37	850,47	842,17	855,90	-0,6	-1,6	100,1
rc 206.3	25	652,86	591,20	574,42	12,0	2,8	30,1
rc 206.4	38	893,34	845,04	837,53	6,2	0,9	136,7
rc 207.1	34	797,67	741,53	732,69	8,1	1,2	79,8
rc 207.2	31	721,39	718,09	702,33	2,6	2,2	68,2
rc 207.3	33	750,03	684,40	682,40	9,0	0,3	83,8
rc 207.4	6	119,64	119,64	119,64	0,0	0,0	0,0
rc 208.1	38	812,23	799,19	793,61	2,3	0,7	84,8
rc 208.2	29	584,14	543,41	544,07	6,9	-0,1	56,2
rc 208.3	36	691,50	660,15	641,31	7,3	2,9	79,8

Tabela 2 – Resultados computacionais para os problemas-teste 2.

de seu método GENIUS para o PCVJT. Os valores de ambas as soluções podem ser encontrados em [10]. Assim como Gendreau, encaramos os problemas do grupo 2 como PCVJTs, considerando que há apenas um veículo disponível e traçando uma única rota para ele. As colunas Ganho Sol. 1 e Ganho Sol. 2, se referem à melhoria da solução obtida por nosso algoritmo em relação aos valores obtidos por Potvin *et al.* (Solução 1) e Gendreau (Solução 2), respectivamente. Dos resultados para 30 instâncias contidos nessa tabela, em relação à Solução 1, obtivemos 3 iguais e apenas um pior. Os demais apresentam uma melhoria significativa, com um ganho médio de aproximadamente 6,1%. Em relação à Solução 2, obtivemos 12 resultados melhores, dos quais 6 apresentam ganho igual ou superior a 2%, e 4 piores, sendo 3 com perda acima de 1%. Dessa forma, podemos concluir que nosso método produz soluções tão boas quanto as de Gendreau, sendo na maior parte soluções de mesmos valores, ou de valores razoavelmente melhores.

Gostaríamos de deixar claro para o leitor que seria possível incorporar ao algoritmo proposto a fase US, logo após ao procedimento GENI, de modo que o método GENIUS fosse empregado na forma proposta por Gendreau. No entanto, não adotamos essa estratégia porque, testes preliminares mostraram que a implementação da fase US no MatLab torna o algoritmo computacionalmente muito caro.

Uma vez que os problemas dos grupos 1 e 2 não possuem prêmios nem penalidades, definimos em todas as instâncias desses grupos as mesmas penalidades e prêmios para todos os clientes, sendo $\pi_i = p_i = 0$, para cada nó i. A escolha desses valores se dá pelo fato de que, para esses grupos de problemas, apenas os custos de travessia são considerados para calcular o valor de uma solução, de modo que as duas últimas parcelas da função objetivo (1.34) permanecem sempre nulas.

A Tabela 3 apresenta os resultados dos testes realizados para sete problemas do terceiro grupo. Esses testes têm por objetivo comprovar a robustez do método, pois embora incluam prêmios, tais problemas não possuem janelas de tempo. Desse modo, para realizar tais testes, inserimos em cada instância, para cada nó i, a mesma janela de tempo $J_i = [0, 10000]$. Essa escolha para as janelas de tempo permite que todos os clientes sejam atendidos a qualquer instante. Como a formulação original desses problemas não força as soluções ótimas a conter todos os clientes e, por outro lado, o algoritmo proposto tende a inserir o máximo de clientes que não infrigem as janelas de tempo, para podermos comparar as soluções obtidas pelo método proposto com as fornecidas para o modelo (1.13)-(1.21) pelo GLPK, fez-se necessário a mudança das penalidades. Dessa maneira, para cada instância do grupo 3, fizemos $\pi_i = 10000$, $\forall i \in V$. Com esses valores de penalização, excluir um nó da rota torna-se impossível para o GLPK.

Mesmo se tratando de problemas que não possuam janelas de tempo, os resultados da Tabela 3 nos levam a concluir que o método proposto é estável, enquanto o método exato consome um tempo elevado para 2 dos 3 problemas de tamanho $n = 31$. Além

Tabela 3

		Método exato		Algoritmo proposto	
Problema	n	Solução ótima	Tempo (s)	Solução	Tempo (s)
pv10	11	2982	0,2	2982	1,9
pv20	21	2762	10,7	2762	18,6
pv30a	31	4102	3147,0	4117	66,7
pv30b	31	2687	96,1	2687	63,9
pv30c	31	3245	3737,5	3245	79,9
pv50a	51	-	>10000,0	4430	170,1
pv50b	51	-	>10000,0	3972	150,2

Tabela 3 – Resultados computacionais para os problemas-teste 3.

disso, conseguimos obter, ainda neste caso, soluções ótimas ou próximas da otimalidade. Note que para os problemas de tamanho $n = 51$, o GLPK já não foi capaz de encontrar solução, enquanto o nosso algoritmo o fez em tempo hábil.

4.2 Resultados para o PCVCPJT

Para entender como procedemos com a produção dos problemas do tipo 4, suponha que P12 seja um problema de tamanho n pertencente ao grupo 1 ou 2, e P3 um problema do grupo 3, de tamanho igual ou maior que n. O problema resultante, digamos P4, salvo menção explícita em contrário, contém:

- A matriz de distâncias $c = (c_{ij})$, do problema P12.
- Janelas de tempo do problema P12.
- O vetor de prêmios composto pelos n primeiros prêmios contidos no vetor de prêmios de P3.
- Penalidades $\pi_i = max\{w_i; i = 1, \ldots, n\}$, para cada $i = 1, \ldots, n$, sendo w_i o maior elemento do vetor v_i, que por sua vez é o vetor obtido por somar o vetor linha i de c, com o transposto do vetor coluna i de c.

A escolha da penalidade é feita de forma que, dada uma solução P para o problema $P4$, se P' é obtido por remover um nó de P, então P' gera um valor de função objetivo maior do que o gerado por P. Além disso, como os problemas do grupo 1 e 2 são factíveis, fez-se necessário algumas alterações em suas janelas de tempo para forçar que algum cliente não possa ser atendido. Para isso, em cada instância, escolhemos dois clientes e deixamos suas janelas de tempo próximas o suficiente para que, obrigatoriamente, o algoritmo precise escolher apenas um deles para incluir na rota solução. A seguir,

descrevemos os problemas que foram obtidos a partir da junção de instâncias do grupo 2 (aquelas com prefixo "rc") com instâncias do grupo 3 (aquelas com prefixo "v"). A Tabela 4 contém os resultados para esses problemas.

- p4 - Junção de rc_206.1 com v10. Alterações: $b_2 = 45$ e $b_4 = 40$.
- p6 - Junção de rc_207.4 com v10. Alterações: $b_2 = 35$ e $b_3 = 35$.
- p14 - Junção de rc_202.2 com v20. Alterações: $b_3 = 35$ e $b_6 = 35$.
- p15 - Esse problema herdou a matriz c do problema p14, que foi completada com a última linha e última coluna da matriz do problema rc_203.4. As janelas de tempo também foram herdadas de rc_203.4, enquanto os prêmios provêm de v20. Alterações: $b_2 = 40$, $a_{11} = 0$ e $b_{11} = 40$.
- p19 - Junção de rc_2031 com v20. Alterações: $b_{11} = 31$ e $b_{18} = 34$.

Tabela 4

		Método exato		Algoritmo proposto	
Problema	n	Solução ótima	Tempo (s)	Solução	Tempo (s)
p4	4	193,30	0,0	193,30	0,1
p6	6	178,22	0,0	178,22	0,2
p14	14	433,92	179,0	433,92	6,4
p15	15	456,54	4828,6	456,54	15,5
p19	19	-	>10000.0	595,49	11,0

Tabela 4 – Testes para junção de problemas dos grupos 2 e 3.

Observando a Tabela 4, podemos ter a certeza de que o método proposto encontrou a solução ótima para os 4 primeiros problemas. O GLPK obteve mais rapidamente as soluções para os dois primeiros problemas, no entanto, com uma diferença irrisória de tempo, que não excede 0,2 segundos. À medida que os problemas crescem, podemos notar - da instância p14 em diante - que o tempo de execução do algoritmo proposto torna-se significativamente menor que o do método exato, e que a diferença entre esses tempos fica cada vez maior. Veja que, para o problema p19, o GLPK não pôde encontrar uma solução, enquanto o algoritmo proposto o fez em apenas 11 segundos, chegando a atingir, nesse caso, um tempo de execução inferior a $\frac{1}{900}$ do tempo do GLPK.

Vejamos, agora, a descrição dos problemas obtidos a partir da junção de instâncias do grupo 1 (aquelas com prefixo "rbg"), com as do grupo 3. A Tabela 5 contém os resultados para esses problemas.

- p11 - Junção de rbg010a com v20. Alterações: $b_2 = 60$, $a_9 = 0$ e $b_9 = 60$.

- p20 - Junção de rbg021.7 com v20. Alterações: $a_4 = 0$, $b_4 = 45$, $a_{12} = 0$ e $b_{12} = 45$.
- p30 - Obtido a partir de p32, por remover as informações referentes aos nós 31 e 32.
- p32 - Junção de rbg031a com v50a. Alterações: $b_2 = 45$, $a_{10} = 0$ e $b_{10} = 45$.
- p39 - Junção de rbg038a com v50a. Alterações: $b_2 = 40$, $a_{10} = 0$ e $b_{10} = 40$.

Tabela 5

		Método exato		Algoritmo proposto	
Problema	n	Solução ótima	Tempo (s)	Solução	Tempo (s)
p11	11	772	0,0	772	2,8
p20	20	7970	11,0	7970	41,5
p30	30	1847	477,4	1847	76,9
p32	32	1975	2451,0	1975	137,8
p39	39	-	>10000,0	2584	114,1

Tabela 5 – Testes para junção de problemas dos grupos 1 e 3.

Observando a Tabela 5, podemos ter a certeza de que o método proposto encontrou a solução ótima para os 4 primeiros problemas. O GLPK obteve mais rapidamente as soluções para os dois primeiros problemas. Novamente, o crescimento de n acarreta em uma vantagem de tempo para o algoritmo proposto sobre o método exato, como podemos ver a partir de p30. Além disso, a diferença entre o tempo de execução do GLPK e do método proposto, torna-se cada vez maior. Veja que, para o problema p39, o GLPK não pôde encontrar uma solução, enquanto o algoritmo proposto o fez em 114,1 segundos, chegando a atingir, nesse caso, um tempo de execução inferior a $\frac{1}{87}$ do tempo do GLPK.

Em todos os problemas das Tabelas 4 e 5, o tamanho da solução encontrada é $n - 1$. Observa-se que, exatamente como se esperava, um dos clientes para os quais restringimos as janelas de tempo deixou de ser atendido. As duas últimas tabelas confirmam que o método heurístico apresentado neste trabalho obteve sucesso em sua aplicação, gerando boas sluções para os problemas propostos. Em todos os casos em que foi possível para o software GLPK encontrar a solução ótima, nosso método também a encontrou. Nota-se que não demora para que o crescimento do tamanho dos problemas acarretem numa ótima vantagem de tempo para o nosso algoritmo, e mesmo quando o GLPK não pôde encontrar uma solução, o algoritmo proposto continuou conseguindo soluções em tempo hábil.

5 Considerações Finais

O elaboração do modelo proposto para o PCVCPJT tem como base dois problemas de grande aplicação prática: PCVJT e PCVCP. Ambos os problemas são de muita importância no comércio mundial, principalmente para empresas ligadas ao transporte rodoviário. Por juntar as características desses dois problemas, o PCVCPJT pertence à classe de problemas NP-completos e, portanto, exige um alto esforço computacional. Dessa forma, faz-se necessário a escolha de um bom método heurístico para resolvê-lo.

Algumas das heurísticas mais adotadas para a resolução do PCV e suas variantes são exibidas neste trabalho. Também apresentamos a meta-heurística VNS, que vem sendo bastante utilizada nos últimos anos (ver [28], [16] e [27], por exemplo).

Propusemos neste trabalho um algoritmo heurístico que combina várias heurísticas de construção, o método GENI e a meta-heurística VNS. O método GENI é incorporado na fase de construção para aumentar as chances de se obter uma boa solução inicial. A melhoria da solução é realizada pelo modelo adaptado do BVNS, apresentado ao final do Capítulo 4, que é composto por vizinhanças obtidas a partir de movimentos baseados em heurísticas de refinamento, apresentadas na seção 3.1.1.2, bem como nos movimentos adicionais apresentados na seção 3.2.5.

A robustez do método proposto é comprovada com os experimentos computacionais exibidos no capítulo anterior, no qual se observa que ele consegue resolver as instâncias de quatro grupos de problemas. Além disso, ele tem se mostrado eficiente, produzindo, na vasta maioria dos casos, solução de mesmo valor que a fornecida pelo GLPK, quando este foi capaz de encontrá-la. Sua eficiência se estende ao quesito tempo, uma vez que, com o crescimento do tamanho das instâncias, nota-se um aumento mais leve e controlado no tempo que o algoritmo aqui apresentado leva para resolvê-las, enquanto o tempo do método exato geralmente cresce exponencialmente.

Muito ainda pode ser explorado na linha de pesquisa desenvolvida neste trabalho. Trocar ou acrescentar novas heurísticas de construção na primeira etapa do método para tentar obter melhores soluções iniciais é uma alternativa. Implementar o algoritmo em outra linguagem, como C, por exemplo, pode deixar o método mais rápido, tornando viável a ideia de incorporar a fase US ao final do procedimento GENI. Além disso, estruturas de vizinhanças adicionais podem ser exploradas, tomando por base outras heurísticas de refinamento - or-exchange ([34, 37, 36]) ou k-opt ([12, 13]) com k=4, por exemplo - ou movimentos como excluir um vértice da rota e nela inserir um vértice que está fora, não necessariamente na mesma posição do que foi excluído.

No mais, mesmo que não venha a deixar o método mais ágil, combinar o algoritmo proposto com outras meta-heurísticas pode aumentar o potencial do método para produzir soluções que atinjam ou estejam mais próximas da otimalidade.

Referências

[1] Balas, E., The Asymmetric Assignment Problem and Some New Facets of the Traveling Salesman Polytope on a Directed Graph, "SIAM Journal on Discrete Mathematics, Vol. 2, No. 4: pp. 425-451, 1989.

[2] Flood, M. M., The Traveling Salesman Problem. Opns. Res., Vol. 4, No 1: pp. 61-75, 1956.

[3] G. Dantzig, R. Fulkerson, and S. Johnson, "Solution of a Large-Scale Traveling-Salesman Problem,"J. Opes. Res. Soc. Am., Vol. 2, pp. 393-410, 1954.

[4] E.K. Baker. An exact algorithm for the time-constrained traveling salesman problem. Opns. Res., Vol. 31, pp. 938–945, 1983.

[5] Balas, E., The Prize Collecting Traveling Salesman Problem. Networks, Vol. 19, pp. 621-636, 1989.

[6] Garey, M.R. and Johnson, D.S. **Computers and intractability: a guide to the theory of NP-completeness**. New York, W. H. Freeman, 1979.

[7] W. W. R. Ball, **Mathematical Recreations and Essays**. 4. ed. Macmillan, New York, 1939.

[8] Gendreau, M., A. Hertz, and G. Laporte. A Tabu Search Heuristic for the Vehicle Routing Problem. Mgmt. Sci., Vol. 40, pp. 1276-1290, 1994.

[9] Gendreau, M., A. Hertz, and G. Laporte. New Insertion and Postoptimization Procedures for the Travaeling Salesman Problem. Opns. Res., Vol. 43, pp. 367-371, 1992.

[10] Gendreau, M., A. Hertz, G. Laporte, and M. Stan. A Generalized Insertion Heuristic for the Traveling Salesman Problem with Time Windows. Operations Research and Management Science, Vol. 46, pp. 330-335, 1996.

[11] Glover, F. and Kochenberger, G. A. **Handbook of Metaheuristics**. Kluwer Aademic Publisher, Boston, 2003.

[12] Lin, S. Computer Solutions of the Traveling Salesman Problem. J. Bell System Technical., Vol. 44, pp.2245-2269, 1965.

[13] Croes, G. A method for solving traveling salesman problem. Opns. Res., Vol. 6, pp. 791-812, 1958.

[14] Papadimitriou, C. **Computational Complexity**. Reading: Addison-Wesley, 1994.

[15] Hansen, P., and Mladenović, N. A Tutorial on Variable Neighborhood Search. Les Cahiers du GERAD, 2003.

[16] Hansen, P., and Mladenović, N. Variable Neighborhood Search: Methods and Applications. European J. Oper. Res., Vol. 175, pp. 367-407, 2010.

[17] Hansen, P., and Mladenović, N. Variable Neighborhood Search: Principles and Applications. Opns. Res., Vol. 130, pp. 449-467, 2001(a).

[18] Hansen, P., and Mladenović, N. First vs. best improvement: An empirical study. Discrete Applied Mathematics, Vol. 154, pp. 802-817, 2006.

[19] Brimberg, J., Hansen, P., Mladenović, N., and Taillard, É. Improvements and Comparison of heuristics for solving the multisource Weber problem. Opns. Res., Vol. 48, pp. 444-460, 2000.

[20] Hansen, P., and Mladenović, N. J-Means: a new local search heuristic for minimum sum-of-squares clustering. Pattern Recognition, Vol. 34, pp. 405-413, 2001(b).

[21] Mladenović, N., Petrović, J., Kovačević-Vujčić, V., and Čangalović, M. Solving spread spectrum radar polyphase code design problem by tabu search and variable neighborhood search. European Journal of Operational Research, Vol. 151, pp. 389-399, 2003(b).

[22] Hansen, P., and Mladenović, N., and Pérez-Brito, D. Variable neighborhood decomposition search. Journal of Heuristics, Vol. 7, pp. 335-350, 2001.

[23] Mladenović, N., and Hansen, P. Variable Neighborhood Search. Computers and Operations Search, Vol. 24, pp. 1097-1100, 1997.

[24] Andreatta, A., Ribeiro, C. Heuristics for phylogeny problem. Journal of Heuristics, Vol. 8, pp. 429-447, 2002.

[25] Canuto, S., Resende, M., and Ribeiro, C. Local search with perturbations for the prize-collecting Steiner tree problem in graphs. Networks, Vol. 31, pp. 201-206, 2001.

[26] Vieira, H. P. Meta-Heuristica para a solução de Problemas de Roteamento de Veículos com Janela de Tempo. Dissertação (Mestrado em Matemática Aplicada) - Instituo de Matemática, Estatística e Computação Científica, Universidade Estadual de Campinas, 2009.

[27] Pedro, O. R. Uma abordagem de Busca Tabu para o Problema do Caixeiro Viajante com Coleta de Prêmios. Dissertação (Mestrado em Engenharia Elétrica) - Universidade Federal de Minas Gerais, 2013.

[28] Chaves, A. A., Biajoli, F. L., Mine, O. M. and Souza, M. J. F. Metaheurísticas híbridas para resolução do problema do caixeiro viajante com coleta de prêmios. Produção, Vol. 17, pp. 263-272, 2007.

[29] Miller, C. E., Tucker, A., W. and Zemlin, R. A. Integer programming formulations of traveling salesman problem. J. Assoc. Comput. Mach., Vol. 7, pp. 326-329, 1960.

[30] Desrochers, M. and Laporte, G. Improvements and extensions to the Miller-Tucker-Zemlin subtour elimination constraints. Operations Research Letters, Vol. 10, pp. 27-36, 1991.

[31] Chaves, A. A. Modelagens Exata e Heurística para Resolução do Problema do Caixeiro Viajante com Coleta de Prêmios. Monografia (Bacharelado em Ciência da Computação) - Instituto de Ciências Exatas e Biológicas, Universidade Federal de Ouro Preto, 2003.

[32] Solomon, M. M. Algorithms for the Vehicle Routing and Scheduling Problem with Time Windows Constraints. Opns. Res., Vol. 35, pp. 254-265, 1987.

[33] Ascheuer, N. Hamiltonian Path Problems in the On-line Optimization of Flexible Manufacturing Systems. PhD Thesis, Technische Universităt Berlin, Berlin, Germany, 1995.

[34] Ascheuer, N., Fischetti, M. and Grótschel, M. Solving the Asymetric Travelling Salesman Problem with Time Windows by branch-and-cut. Math. Program., Ser. A 90: 475–506, 2001.

[35] Potvin, J-Y., Kervahut, T., Garcia, B-L. and Rousseau, J-M. The Vehicle Routing Problem - Part I: Tabu Search. INFORMS J. on Computing, Vol. 8, pp. 158-164, 1996.

[36] Or, I. Traveling Salesman-Type Combinatorial Problems and their relation to the logistics of regional blood banking. PhD Thesis. Dept. of Industrial Engineering and Management Science, Northwestern University, Evanston, 1976.

[37] Savelsbergh, M. W. P. Local Search in routing problems with time windows. Annals of Operations Research, Vol. 4, pp. 285-305, 1985.