CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
Diretoria de Pesquisa e Pós-Graduação
Programa de Mestrado em Modelagem Matemática e Computacional

Aguinaldo Alves Pinto

ALGORITMOS HÍBRIDOS APLICADOS AO PROBLEMA DE ROTEAMENTO DE VEÍCULOS COM JANELA DE TEMPO

Orientador: Prof. Dr. Sérgio Ricardo de Souza

Belo Horizonte - MG
2017

CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
Diretoria de Pesquisa e Pós-Graduação
Programa de Mestrado em Modelagem Matemática e Computacional

ALGORITMOS HÍBRIDOS APLICADOS AO PROBLEMA DE ROTEAMENTO DE VEÍCULOS COM JANELA DE TEMPO

Dissertação de Mestrado, submetida ao Programa de Pós-Graduação em Modelagem Matemática e Computacional, como parte dos requisitos exigidos para a obtenção do título de Mestre em Modelagem Matemática e Computacional.

Aluno: Aguinaldo Alves Pinto

Orientador: Prof. Dr. Sérgio Ricardo de Souza

Belo Horizonte - MG
2017

Pinto, Aguinaldo Alves
P659a Algoritmos híbridos aplicados ao problema de roteamento de veículos com janela de tempo. / Aguinaldo Alves Pinto. – – Belo Horizonte, 2017.
x, 96 f. : il.

Dissertação (mestrado) – Centro Federal de Educação Tecnológica de Minas Gerais, Programa de Pós-Graduação em Modelagem Matemática e Computacional, 2017.
Orientador: Prof. Dr. Sérgio Ricardo de Souza

Bibliografia

1. Otimização Combinatória. 2. Programação (Matemática). 3.Heurística. I. Souza, Sérgio Ricardo de. II. Centro Federal de Educação Tecnológica de Minas Gerais. III. Título

CDD 519.7

Elaboração da ficha catalográfica pela Biblioteca-Campus II / CEFET-MG

SERVIÇO PÚBLICO FEDERAL
MINISTÉRIO DA EDUCAÇÃO
CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
COORDENAÇÃO DO PROGRAMA DE PÓS-GRADUAÇÃO EM MODELAGEM MATEMÁTICA E COMPUTACIONAL

"ALGORITMOS HÍBRIDOS APLICADOS AO PROBLEMA DE ROTEAMENTO DE VEÍCULOS COM JANELA DE TEMPO"

Dissertação de Mestrado apresentada por **Aguinaldo Alves Pinto**, em 25 de agosto de 2017, ao Programa de Pós-Graduação em Modelagem Matemática e Computacional do CEFET-MG, e aprovada pela banca examinadora constituída pelos professores:

Prof. Dr. Sérgio Ricardo de Souza (Orientador)
Centro Federal de Educação Tecnológica de Minas Gerais

Prof. Dr. Carlos Alexandre Silva
Instituto Federal de Minas Gerais

Prof. Dr. Moacir Felizardo de França Filho
Centro Federal de Educação Tecnológica de Minas Gerais

Prof. Dr. Gray Farias Moita
Centro Federal de Educação Tecnológica de Minas Gerais

Visto e permitida a impressão,

Prof. Dr. José Geraldo Peixoto de Faria
Coordenador do Programa de Pós-Graduação em
Modelagem Matemática e Computacional

Agradecimento

Agradeço primeiramente a DEUS, a consciência divina e causa primária de todas as coisas, e, como tal, decorrem tudo que existe. Em segundo lugar, aos meus pais, Josê Pinto Filho(*in memoriam*) e Conceição Alves Pereira, que são meus maiores professores, que me ensinaram, não como fazer uma *"Heurística Construtiva"* ou uma *"Busca Local"*, mas os valores que me fazem trilhar os caminhos do bem. A minha família, em especial a minha irmã Almira, que me apoiou emocionalmente durante esta trajetória.

Agradeço ao Sérgio, pela oportunidade de trabalharmos juntos e de me desenvolver, não só em conhecimento, mas também como ser humano. Aos professores do programa, que proporcionaram uma expansão das fronteiras do conhecimento e mostraram o *"Caminho das Pedras"* para a pesquisa desenvolvida.

Agradeço também à Juliana, Daniel, José Maurício, Maria Amélia, Gustavo, Alexandre e tantos outros, que me auxiliaram nesta trajetória. São por causa deles que, muitas vezes no caminho, não desisti, levantei a cabeça e foi em frente.

Resumo

Nesta dissertação é tratado o Problema de Roteamento de Veículos com Janela de Tempo (PRVJT). O objetivo é atender um conjunto de clientes geograficamente distribuídos com um número de veículos limitado. É considerado um único depósito, a partir do qual os veículos partem e retornam após visitar todos os clientes de sua respectiva rota. Há uma janela de tempo associada ao depósito, que indica seu período de funcionamento, além de uma janela de tempo correspondente a cada cliente, que indica o intervalo de tempo para iniciar o atendimento. Todos os veículos possuem a mesma capacidade de carga em uma dada instância e há uma demanda correspondente a cada cliente. Esta dissertação propõe, para a solução de instâncias do PRVJT, uma hibridização das metaheurísticas *Iterated Greedy Search (IGS)* e *Greedy Randomized Adaptive Search Procedure (GRASP)*, integrada às heurísticas *Push-Forward Insertion Heuristic (PFIH)* e *Variable Neighborhood Descent (VND)*. Os testes computacionais realizados utilizam as 56 instâncias propostas por Solomon (1987) e comparam com os melhores resultados heurísticos da literatura para esta classe de instâncias do PRVJT.

PALAVRAS-CHAVE: Problema de Roteamento de Veículos com Janela de Tempo, IGS, GRASP, VND, PFIH, Metaheurísticas híbridas, Otimização.

Abstract

This dissertation addresses the Problem of Vehicle Routing with Time Window (VRPTW). The goal is to attend a set of geographically distributed customers with a limited number of vehicles. The problem to be dealt considers a single deposit, from which the vehicles depart and return after visiting all the clients of their respective route. There is a time window associated with the deposit, which indicates its period of operation, besides a time window corresponding to each customer, which indicates the time interval to start the service. All vehicles have the same load capacity and there is a demand associated with each customer. This dissertation proposes, in order to solve VRPTW, a hybridization of the metaheuristics Iterated Greedy Search (IGS) and Greedy Randomized Adaptive Search Procedure (GRASP), integrated with to the heuristics Push-Forward Insertion Heuristic (PFIH) and Variable Neighborhood Descent (VND). The computational tests performed use the 56 instances proposed by Solomon (1987) and compare with the best known results using heuristics in the literature for this class of instances of VRPTW.

Keywords: Vehicle Routing Problem with Time Window, Iterated Greedy Search, GRASP, VND, PFIH, Hybrid Metaheuristics, Optimization.

Lista de Tabelas

2.1 Informações dos Clientes para a Instância C101 9
2.2 Distâncias Euclidianas entre os pares de clientes 10

3.1 Panorama de metaheurísticas para a solução do PRVJT e de suas variações – Parte I 22
3.2 Panorama de metaheurísticas para a solução do PRVJT e de suas variações – Parte II 23
3.3 Panorama de aplicação do IGS na resolução de problemas combinatórios de diversas classes 29

6.1 Resultados IGS-VND1 para instância classe C1 69
6.2 Resultados IGS-VND1 para instância classe C2 70
6.3 Resultados IGS-VND1 para instância classe R1 70
6.4 Resultados IGS-VND1 para instância classe R2 71
6.5 Resultados IGS-VND1 para instância classe RC1 72
6.6 Resultados IGS-VND1 para instância classe RC2 72
6.7 Resultados IGS-VND2 para instância classe C1 74
6.8 Resultados IGS-VND2 para instância classe C2 74
6.9 Resultados IGS-VND2 para instância classe R1 75
6.10 Resultados IGS-VND2 para instância classe R2 76
6.11 Resultados IGS-VND2 para instância classe RC1 76
6.12 Resultados IGS-VND2 para instância classe RC2 77
6.13 Comparação Resultados Classe C1 77
6.14 Comparação Resultados Classe C2 78
6.15 Comparação Resultados Classe R1 78
6.16 Comparação Resultados Classe R2 79
6.17 Comparação Resultados Classe RC1 79
6.18 Comparação Resultados Classe RC2 80
6.19 Comparação dos resultados finais dos algoritmos 80

Lista de Figuras

2.1 Exemplo fictício do PCV mostrando a distância percorrida. 6
2.2 Exemplo fictício para o PRV. 6
2.3 Exemplo fictício para o PRV mostrando o cálculo da função objetivo. 7
2.4 Exemplo fictício para o PRVJT. 9
2.5 Exemplo para o PRVJT com distâncias e janela de tempo. 10
2.6 Exemplo para o PRVJT mostrando o cálculo do tempo de viagem entre dois clientes. 11

5.1 Solução de um exemplo fictício para o PRVJT. 41
5.2 Exemplo de estrutura vizinhança shift(1). 43
5.3 Exemplo de estrutura vizinhança shift(2). 43
5.4 Exemplo de estrutura vizinhança shift(3). 44
5.5 Exemplo de estrutura vizinhança Exchange(1) 45
5.6 Exemplo de estrutura vizinhança Exchange(2) 45
5.7 Exemplo de estrutura vizinhança Exchange(3) 45
5.8 Exemplo de estrutura vizinhança Shift(1,0). 46
5.9 Exemplo de estrutura vizinhança Shift(2,0). 47
5.10 Exemplo de estrutura vizinhança Shift(3,0). 48
5.11 Exemplo de estrutura vizinhança Swap(1,1). 48
5.12 Exemplo de estrutura vizinhança Swap(2,1). 49
5.13 Exemplo de estrutura vizinhança Swap(2,2). 50
5.14 Construção solução fictícia GRASP+PFIH. 53
5.15 Destruição de uma solução fictícia pelo IGS. 57
5.16 Reconstrução de uma solução fictícia pelo IGS. 59

6.1 Disposição Clientes Instância C101 67
6.2 Disposição Clientes Instância R101 67
6.3 Disposição Clientes Instância RC101 68

Lista de Algoritmos

4.1 *Push-Forward Insertion Heuristic* 32
4.2 *Greedy Randomized Adaptive Search Procedure* 34
4.3 *Variable Neighborhood Descent* 35
4.4 *Simulated Annealing* 36
4.5 *Iterated Greedy Search* 39
5.1 MovimentosVND2(s,*cont*) 51
5.2 buscaLocalVND 52
5.3 Construção_$GRASP_PFIH_VND$ 55
5.4 $GRASP_PFIH_VND$ 56
5.5 Destruição IGS 58
5.6 Reconstrução IGS 60
5.7 MovimentoVND1(s,*cont*) 62
5.8 $GRASP_PFIH_VND_IGS1$ 63
5.9 $GRASP_PFIH_VND_IGS2$ 64

Sumário

1 Introdução **1**
- 1.1 Objetivos . . . 2
- 1.2 Justificativa . . . 2
- 1.3 Organização da Dissertação . . . 3

2 Caracterização do Problema **5**
- 2.1 Problema de Roteamento de Veículos . . . 5
  - 2.1.1 Problema de Roteamento de Veículos com Janela de Tempo . . . 8
- 2.2 Exemplo de situação para o PRVJT . . . 9
- 2.3 Formulação Matemática . . . 11

3 Revisão Bibliográfica **14**
- 3.1 Resolução do problema PRVJT via Metaheurísticas . . . 14
- 3.2 Uma revisão de aplicações da metaheurística *Iterated Greedy Search* (IGS) . . . 23

4 Heurísticas Estudadas **31**
- 4.1 *Push-Forward Insertion Heuristic* (PFIH) . . . 31
- 4.2 *Greedy Randomized Adaptive Search Procedure* (GRASP) . . . 33
- 4.3 *Variable Neighborhood Descent* (VND) . . . 34
- 4.4 *Simulated Annealing* (SA) . . . 35
- 4.5 *Iterated Greedy Search* (IGS) . . . 37

5 Algoritmos Desenvolvidos **41**
- 5.1 Representação da Solução . . . 41
- 5.2 Estruturas de Vizinhança . . . 41
  - 5.2.1 Estruturas de vizinhança *Shift(k)* . . . 42
  - 5.2.2 Estruturas de vizinhança *Exchange(k)* . . . 44
  - 5.2.3 Estruturas de vizinhança *Shift(k,0)* . . . 46
  - 5.2.4 Estruturas de vizinhança *Swap(k,l)* . . . 47
- 5.3 Função de Avaliação . . . 49
- 5.4 Algoritmos Propostos . . . 50
  - 5.4.1 VND . . . 50
  - 5.4.2 GRASP+PFIH+VND . . . 52
  - 5.4.3 GRASP+PFIH+VND+IGS . . . 57
  - 5.4.4 Versões do Algoritmo GRASP+PFIH+VND+IGS . . . 61

6 Experimentos Computacionais **66**
6.1 Descrição das Instâncias 66
6.2 Análise dos Resultados 67
6.2.1 Resultados Algoritmo GRASP_PFIH_VND_IGS1 68
6.2.2 Resultados IGS-VND2 71
6.2.3 Comparação dos Resultados 75

7 Considerações Finais **81**

Referências **83**

Capítulo 1

Introdução

As organizações, em um mundo cada vez mais globalizado, têm buscado uma maior eficiência no gerenciamento de bens, serviços e na logística envolvida em suas tarefas diárias. Tendo em vista inúmeros problemas relativos às restrições, principalmente dos grandes centros urbanos, de entrega e coleta de produtos e prestação de serviços, surge a necessidade de se criar ferramentas que auxiliem na tomada de decisão, tanto no que diz respeito à logística quanto na redução de custos operacionais.

Neste cenário, o Problema de Roteamento de Veículos (PRV) e suas variações conseguem representar várias situações do mundo real, relacionando diversas restrições e apresentando soluções viáveis e de boa qualidade. O PRV originou-se de uma generalização do Problema do Caixeiro Viajante (PCV ou TSP, sigla a partir do nome em inglês *Traveling Salesman Problem*). O problema constitui-se na visita de um conjunto de cidades uma única vez, percorrendo a menor distância possível. Segundo Gutin e Punnen (2006), o PCV consiste na visita do vendedor partindo de uma cidade de origem, percorrendo todas as cidades e retornando à cidade inicial, o que varia de acordo com a ordem escolhida. Com isso, o PCV se torna um problema combinatório, cuja complexidade cresce exponencialmente. Assim, segundo Davendra (2010), este problema pertence à classe NP-Difícil devido a sua complexidade.

O PRV foi proposto inicialmente por Dantzig e Ramser (1959) e fundamentava-se no problema de despacho de caminhões para postos de combustíveis. Basicamente, tinha-se um conjunto de caminhões e um conjunto de postos de combustíveis os quais eles deveriam percorrer. Cada posto poderia ser visitado apenas por um caminhão e vice-versa. Diferentemente do PCV, esse problema necessitaria da formação de mais rotas e, conseqüentemente, deveriam ser adicionadas outras restrições próprias do problema. O objetivo passa a ser atender os postos de combustíveis com uma distância mínima percorrida e com o menor número de caminhões possíveis, em razão dos custos envolvidos serem elevados.

Ao longo de sua história, têm sido introduzidas diversas variações do PRV, que incluem, no modelo, elementos de questões reais do cotidiano. Uma destas variações é o PRV capacitado, que adiciona restrições relativas à demanda de cada cliente, ficando, cada veículo, limitado a uma carga máxima. Uma outra modificação do problema adiciona uma restrição temporal ao atendimento dos clientes, dando origem ao denominado Problema de Roteamento de Veículos com Janela de Tempo (PRVJT). Para Hartl et al. (2006), este tipo de abordagem é essencial para lidar

com limitações do mundo real, como entregas bancárias e postais, coleta de lixo industrial, serviços de patrulha de segurança, itinerário de ônibus escolar, gerenciamento de cadeias de suprimentos, dentre outras.

Para Solomon (1987), o PRVJT é uma variação do PRV que possui, como objetivos principais, a redução do número de veículos utilizados para atender a um conjunto de clientes e minimização da distância total percorrida. Com isso, espera-se conseguir soluções de boa qualidade, que venham reduzir custos operacionais. Outras características importantes do ponto de vista do modelo são em relação aos veículos, que possuem a mesma capacidade, e também quanto à janela de tempo, que determina um tempo inicial e um tempo máximo para o início do atendimento. Outra restrição é em relação à demanda associada a cada cliente, que não pode ultrapassar a carga máxima do veículo.

O problema PRVJT, segundo Savelsbergh (1985), é um problema combinatório NP-Difícil e possui ordem de complexidade exponencial, o que torna os métodos exatos, em geral, pouco eficientes para instâncias com maior número de clientes. Para Chi-Bin e Keng-Pin (2009), a utilização de heurísticas e metaheurísticas para esse problema apresenta bons resultados na literatura e se mostra como uma boa alternativa para métodos exatos. Esta dissertação fará um estudo do PRVJT propondo métodos híbridos, que combinam diversas técnicas relatadas na literatura para resolução do problema.

1.1 Objetivos

Esta dissertação tem, por objetivo geral, estudar a aplicação de algoritmos baseados em metaheurísticas ao Problema de Roteamento de Veículos com Janela de Tempo.

Os objetivos específicos são:

- Elaboração de algoritmos baseados nas metaheurísticas *Iterated Greedy Search (IGS)*, *Greedy Randomized Adaptive Search Procedure (GRASP)* e *Variable Neighborhood Descent (VND)* na resolução do PRVJT.
- Realizar comparações estatísticas entre diferentes técnicas utilizadas e sua contribuição para a literatura.

1.2 Justificativa

Para a resolução do PRVJT são abordadas inúmeras técnicas relatadas na literatura com o intuito de apresentar soluções de boa qualidade e que atendam as necessidades das organizações. Segundo Savelsbergh (1985), o PRV é um problema que vem sendo alvo de diversos estudos, devido a sua importância e capacidade de aderir a diversas necessidades das empresas. Umas das características de maior importância para o modelo são as que contém limitação temporal, pois refletem a necessidade de adequação do atendimento aos clientes a um determinado período, além de possibilitar uma melhor organização das entregas ao longo da rota. Este tipo de restrição acaba reduzindo o espaço de busca de soluções factíveis, o que ocasiona, na maioria das vezes, um custo computacional elevado.

Assim, Savelsbergh (1985) coloca que o PRVJT como sendo um problema NP-Difícil e com complexidade exponencial, cabendo aos pesquisadores dessa área desenvolver técnicas e ferramentas com o intuito de diminuir o custo computacional e apresentar soluções que atendam as necessidades de mercado. Qureshi et al. (2010) mostra que problemas envolvendo um maior número de restrições, quando aplicado para métodos exatos, tem uma eficácia até um certo número de clientes, já que os custos computacionais crescem exponencialmente com a quantidade de elementos do conjunto, sendo necessária aplicação de métodos heurísticos na resolução deste tipo de problema.

Segundo Dorigo et al. (2006), os algoritmos classificados como heurísticos são aqueles que exploram o espaço de busca em um tempo computacional reduzido e conseguem obter soluções de boa qualidade, sem a garantia de encontrar o ótimo global. Ainda segundo estes autores, os métodos heurísticos podem ser classificados com métodos construtivos e métodos de busca local. As heurísticas construtivas têm por finalidade montar uma solução que respeite as restrições e tenha uma melhor qualidade em relação ao valor da função objetivo do problema em questão. Já as heurísticas de busca local têm por objetivo modificar uma solução prévia, de modo a melhorar a exploração do espaço de busca, caminhando de forma mais eficiente para diferentes regiões do espaço de soluções e sair de ótimos locais.

Outra técnica amplamente aplicada em problemas combinatórios são as metaheurísticas. Para Hertz e Widmer (2003), elas têm a capacidade de se adaptar a uma gama de problemas e, por este motivo, apresentam estrutura mais genérica, cabendo ao seu usuário modificá-la de acordo com as características específicas necessárias à resolução do problema. Outro aspecto desta técnica está na capacidade de exploração das soluções, de forma a escapar de ótimos locais. As metaheurísticas podem ser divididas em duas categorias: populacionais e de trajetória. As classificadas como populacionais partem do conceito da montagem de diversas soluções distribuídas em diversos pontos do espaço de busca e, através de procedimentos de modificação e combinação entre elas, conseguir, ao final, um conjunto de soluções de boa qualidade. As de trajetória têm um princípio diferente: partem de uma única solução para percorrer o espaço de busca. Em ambas as técnicas, devido a sua capacidade de adequação e simplicidade de implementação, é possível combinar heurísticas construtivas e de busca local, acelerando a obtenção de uma solução ótima.

Assim, esta dissertação aborda a resolução do PRVJT via metaheurísticas de trajetória, combinando diversas técnicas, como as metaheurísticas *Greedy Randomized Adaptive Search Procedure (GRASP)* e *Iterated Greedy Search (IGS)* e as heurísticas *Push-Forward Insertion Heuristic (PFIH)* e *Variable Neighborhood Descent (VND)*. A principal contribuição desse trabalho se insere justamente na adaptação da metaheurística *Iterated Greedy Search (IGS)* à solução do PRVJT, abordagem ainda não relatada na literatura, pelos menos em trabalhos do conhecimento do autor da presente dissertação.

1.3 Organização da Dissertação

A presente dissertação está organizada da seguinte forma. O Capítulo 2 trata da caracterização do problema objeto de estudo desta dissertação. O Capítulo 3

aborda a revisão bibliográfica dos principais trabalhos relacionados. Já o Capitulo 4 aborda a fundamentação teórica, mostrando as heurísticas e metaheurísticas aplicadas na resolução do problema. O Capítulo 5 descreve os algoritmos elaborados para resolução do PRVJT e o Capítulo 6 mostra os resultados obtidos aplicando os algoritmos propostos através dos testes computacionais. Por fim, o Capítulo 7 faz as considerações finais desta dissertação.

Capítulo 2

Caracterização do Problema

Este capítulo trata do problema objeto de estudo desta dissertação. Na Seção 2.1 será abordado o Problema de Roteamento de Veículos (PRV). Na Seção 2.1.1 são descritas as características principais do Problema de Roteamento de Veículos com Janela de Tempo (PRVJT). A Seção 2.2 descreve um exemplo de solução baseado em dados retirados de uma instância. Por fim, a Seção 2.3 aborda a formulação matemática adotada para o problema nesta dissertação.

2.1 Problema de Roteamento de Veículos

Segundo Dantzig e Ramser (1959), o Problema de Roteamento de Veículos (PRV) é um problema de otimização combinatória que consiste na utilização de uma frota de veículos, na qual deve-se atender a um conjunto de clientes distribuídos em uma área geográfica conhecida. O PRV é considerado uma generalização do Problema do Caixeiro Viajante (PCV).

O PCV constitui-se de um conjunto de cidades nas quais o vendedor deve realizar as entregas da mercadoria e retornar à cidade de origem. Cada cidade possui uma localização geográfica e uma demanda de mercadorias. O objetivo do problema passa a ser organizar o itinerário do vendedor que irá fazer as entregas, de modo a reduzir a distância percorrida e os custos operacionais. A Figura 2.1 mostra um exemplo fictício do PCV. Nesta Figura, a solução $S = \{1, 6, 7, 5, 2, 4, 3\}$ representa a sequência de cidades visitados pelo vendedor. Ele parte de uma determinada cidade, percorre todas as demais cidades e retorna para a cidade inicial. Além da ordem de permutação dos clientes, o problema leva em conta a distância total percorrida, obtida pela soma da distância entre as cidades. O cálculo da função objetivo é realizado somando a distância euclidiana entre as cidades, respeitando a ordem de visitação estabelecida pelo vendedor.

Para Davendra (2010), o PCV pertence à classe de problemas NP-Difícil e, consequentemente, problemas derivados ou que possuam características semelhantes em sua formulação também são classificados como NP-Difícil. O PRV é um exemplo desta derivação. Ele foi proposto por Dantzig e Ramser (1959), baseado em um problema real de distribuição de combustíveis para uma rede de postos. Assim, uma frota de caminhões parte e retorna para o depósito, realizando as entregas nos clientes. Diferente do PCV, o PRV tem a necessidade de se formar mais de um itinerário para atender a um conjunto de clientes geograficamente distribuídos. Cada

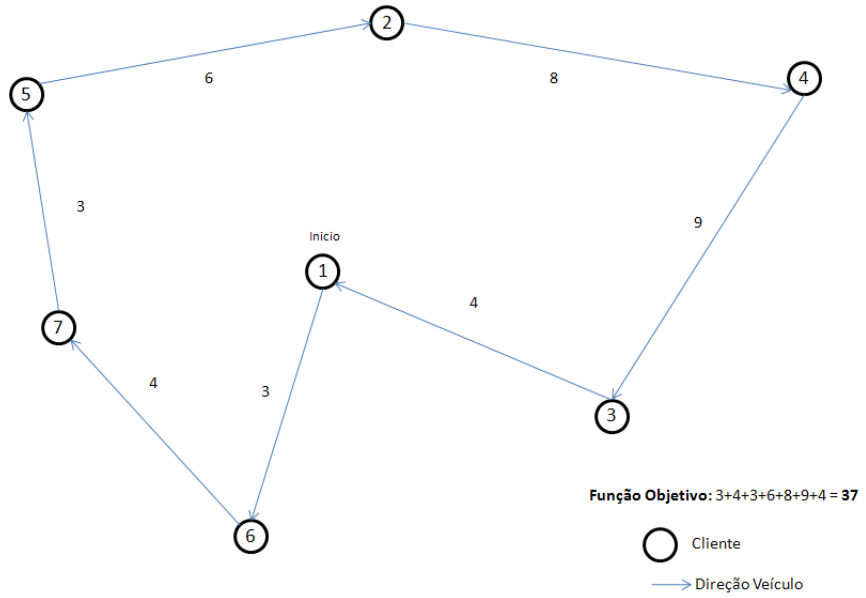

Figura 2.1: Exemplo fictício do PCV mostrando a distância percorrida.

rota é realizada por um único veículo e cada cliente tem uma demanda específica. A Figura 2.2 ilustra uma solução fictícia para o PRV.

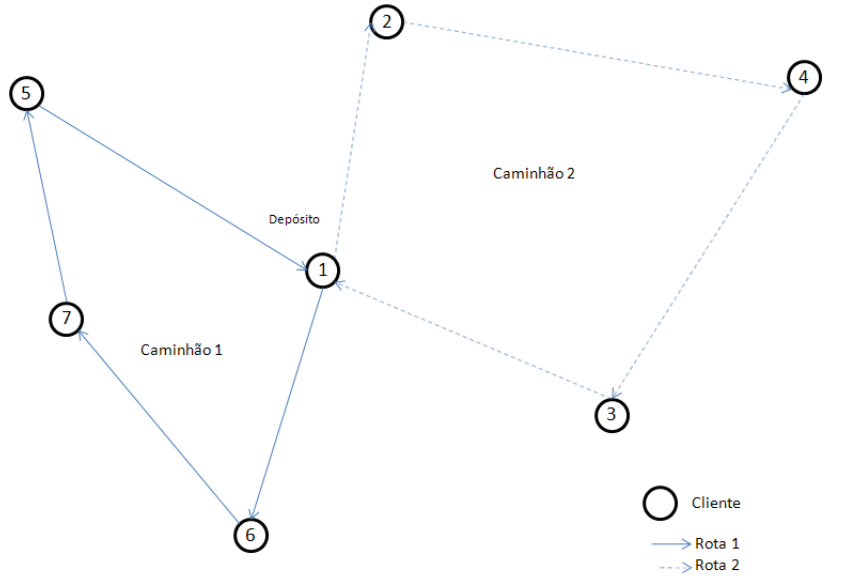

Figura 2.2: Exemplo fictício para o PRV.

Como pode-se observar na Figura 2.2, a solução é composta de duas rotas e cada uma representa um veículo. Cada veículo parte e retorna para o depósito ao final do trajeto e cada cliente tem uma demanda e uma localização específica. A Figura 2.3 ilustra o cálculo da função objetivo do problema.

Diferente do PCV, o PRV requer a necessidade de formação de diversas rotas, o que aumenta a complexidade e o custo computacional. Desta forma o problema passa a ter dois objetivos principais, que são (i) a redução do número de veículos e (ii) a redução da distância total percorrida. No exemplo da Figura 2.3, o valor da função

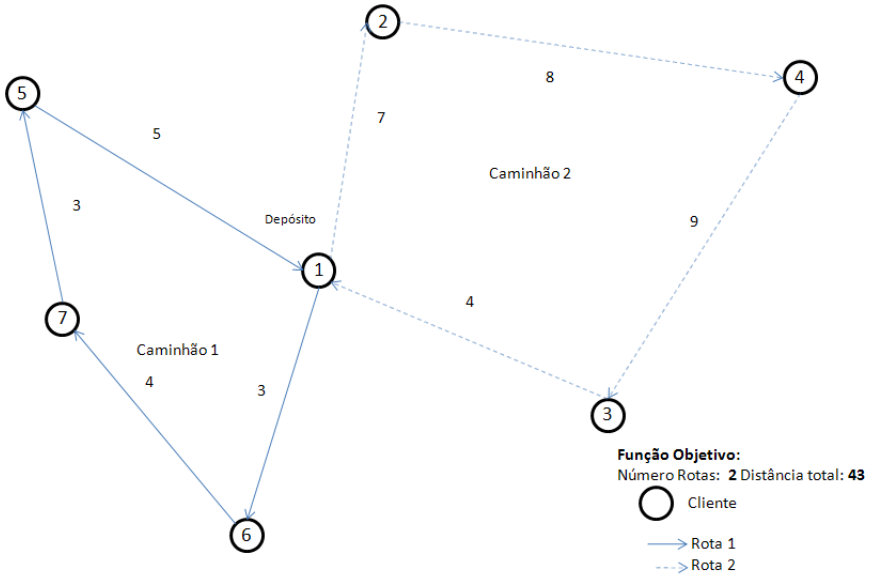

Figura 2.3: Exemplo fictício para o PRV mostrando o cálculo da função objetivo.

objetivo da solução $S = \{\{0, 6, 7, 5, 0\}, \{0, 2, 4, 3, 0\}\}$ é formada pelo número de veículos utilizados e pela soma das distâncias euclidianas entre os clientes. Assim, a solução fictícia S é dada por 2 rotas e uma distância total percorrida de 43 unidades. Como o problema possui dois objetivos, considerados de forma hierárquicos, busca-se, em primeiro lugar, reduzir o número de veículos e, posteriormente, minimizar a distância.

Do ponto de vista da modelagem computacional, o PRV pode ser formulado através de um grafo completo e não direcionado $G = (V, A)$, tendo um conjunto V de N vértices e um conjunto A de arestas, representados, respectivamente, por $V = \{0, 1, \ldots, N\}$ e $A = \{(i, j) \quad : \quad i, j \in V,\ i \neq j\}$. Cada vértice $i \in V$ representa um cliente, sendo que o vértice de índice $i = 0$ indica o depósito e os demais vértices formam o conjunto de clientes $C = \{1, 2, \ldots, N\}$. Assim, $V = \{0\} \cup C$. Cada cliente pode ser visitado apenas uma vez e possui uma demanda q_i relacionada a ele, além de apresentar um custo c_{ij}, que é retratado pela distância euclidiana entre dois vértices i e j.

Segundo Davendra (2010), existem algumas variações do problema abordados pela literatura de grande relevância para a pesquisa operacional nesta área. A seguir são relatadas algumas destas variações.

- PRV com Janela de Tempo e Frota Homogênea: nesta modalidade, além das informações do PRV, cada cliente possui restrição de janela de tempo e todos os veículos são iguais, tendo a mesma capacidade de carregamento;
- PRV com Janela de Tempo e Frota Heterogênea: nesta modalidade, além das informações do PRV, cada cliente possui restrição de janela de tempo e os veículos podem apresentar diferenças em relação à capacidade de carregamento;
- PRV com Múltiplos Depósitos: além das restrições do PRV clássico, os veículos podem partir e finalizar os atendimentos em vários depósitos, sendo que cada depósito possui sua própria frota;

- PRV com Múltiplas Janelas de Tempo: neste caso, cada cliente possui duas janelas de tempo. Cada janela tem um período de tempo que o veículo pode iniciar e finalizar o atendimento;
- PRV com Entrega Particionada: nesta modalidade, há a possibilidade de vários veículos fazerem a visita à determinado cliente;
- PRV com Múltiplos Compartimentos: esta restrição permite que cada veículo possa ter mais de um compartimento de carga. Esta abordagem permite dimensionar nos veículos cargas de diferentes tipos, como, por exemplo, peças organizadas em caixas e transporte de cargas líquidas, como combustíveis;
- PRV Periódico: nesta variação, todos os clientes devem ser atendidos em um número máximo de dias;
- PRV com Janelas de Tempo Flexíveis: considera-se para cada cliente uma penalidade para a violação da janela de tempo. Neste caso, a solução mantém a factibilidade;
- PRV com Limite de Distância: o problema PRV com uma restrição na qual as soluções criadas não podem ultrapassar uma distância máxima;
- PRV com Coleta e Entrega: Nesta modalidade, existem clientes apenas para coleta e clientes apenas para entrega. Os veículos saem carregados e retornam ao depósito vazios.

O PRVJT é uma das variações do PRV, que acrescenta uma restrição temporal ao modelo e é objeto de estudo dessa dissertação. No caso desta dissertação, o problema tratado possui frota homogênea.

2.1.1 Problema de Roteamento de Veículos com Janela de Tempo

O Problema de Roteamento de Veículos com Janela de Tempo (PRVJT), apresentado inicialmente por Solomon (1987), tem, como características principais, o atendimento de um grupo de clientes geograficamente distribuídos por uma frota de veículos homogênea com limitação temporal. Ela define um intervalo $[ti_i, tf_i]$, que delimita limite inferior (ti_i) e superior (tf_i) para que o veículo inicie o atendimento do cliente i. Caso o veículo chegue antes da janela de tempo inicial, ele deve aguardar até a abertura da mesma. Cada cliente possui um tempo de serviço Ts_i, que representa o período de tempo em que o veículo irá realizar o atendimento ao cliente i. Há uma janela de tempo $[T_i, T_f]$ associada ao depósito, que indica o tempo de funcionamento do mesmo. O exemplo da Figura 2.4 mostra a solução composta por $S = \{\{0, 6, 7, 5, 0\}, \{0, 2, 4, 3, 0\}\}$. Neste caso, o conjunto de clientes são atendidos por duas rotas, dadas por $r_1 = \{0, 6, 7, 5, 0\}$ e $r_2 = \{0, 2, 4, 3, 0\}$. Cada veículo parte e retorna ao depósito, representado pelo vértice 0. Os demais clientes são visitados apenas uma vez.

Segundo Chi-Bin e Keng-Pin (2009), há duas abordagens que se pode adotar quando se trabalha com restrições temporais: janelas de tempo rígidas e suaves. As

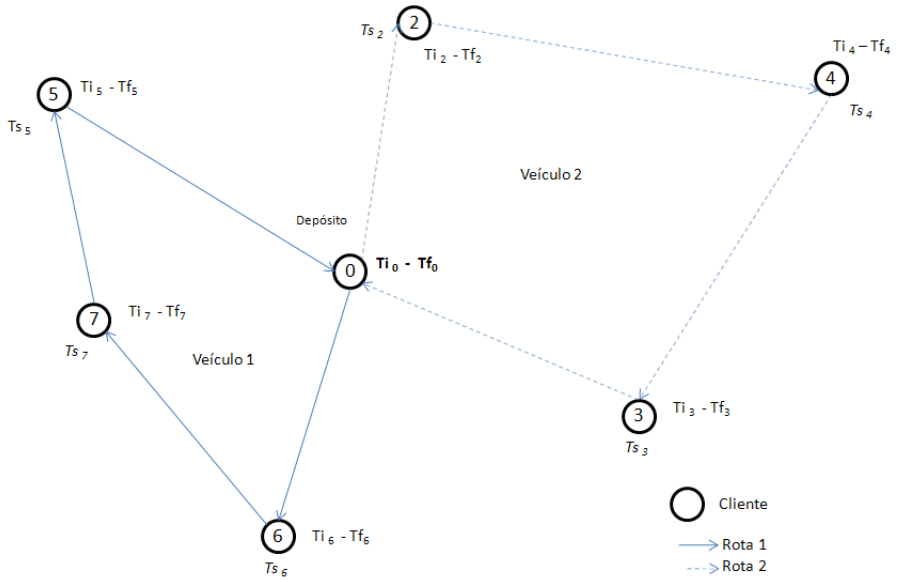

Figura 2.4: Exemplo fictício para o PRVJT.

janelas de tempo suaves possuem uma aceitação da violação da restrição temporal, de forma que a solução é penalizada e não é considerada inviável. Já as janelas de tempo rígidas consideram infactíveis as soluções que violem estas restrições. A abordagem rígida é adotada nesta dissertação.

2.2 Exemplo de situação para o PRVJT

Para exemplificar o problema, foram retiradas as informações de 10 clientes da instância C101, proposta por Solomon (1987). Estas informações estão exibidas na Tabela 2.1. Nesta tabela são mostradas as informações do cliente i, como: o número de identificação, as coordenadas X e Y no plano cartesiano, a demanda q_i, os valores da janela de tempo inicial ti_i e final tf_f e o tempo de serviço Ts_i. Na Tabela 2.2

Tabela 2.1: Informações dos Clientes para a Instância C101

Número	X	Y	q_i	ti_i	tf_i	Ts_i
0	40	50	0	0	1236	0
12	25	85	20	652	721	90
25	25	52	40	169	224	90
38	0	40	30	479	522	90
74	53	35	50	353	412	90
87	65	55	20	85	144	90

são exibidas as distâncias entre os clientes participantes do exemplo. O cliente de número zero representa o depósito e não apresenta demanda e tempo de serviço. A Figura 2.5 mostra graficamente as informações de janela de tempo e distância entre as cidades da rota.

Tabela 2.2: Distâncias Euclidianas entre os pares de clientes

Clientes	Valor
[0][38]	42,42
[38][74]	41,77
[74][25]	20,24
[25][0]	33,54
[12][0]	15,00
[87][12]	27,80
[0][87]	18,38

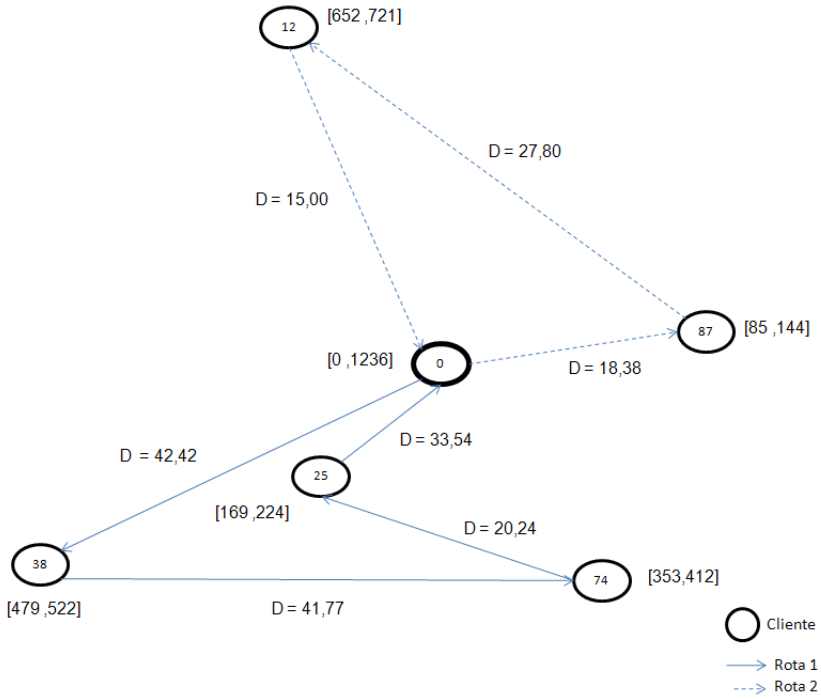

Figura 2.5: Exemplo para o PRVJT com distâncias e janela de tempo.

Na Figura 2.5, a solução $S = \{\{0, 38, 74, 25, 0\}, \{0, 87, 12, 0\}\}$ é composta de duas rotas: $r_1 = \{0, 38, 74, 25, 0\}$ e $r_2 = \{0, 87, 12, 0\}$. Para verificar a factibilidade entre os clientes são considerados, além da distância, o tempo de serviço para cada cliente. A Figura 2.6 mostra a forma de cálculo do tempo de viagem do veículo até os clientes respectivos. Para fins de exemplo, considere o cálculo feito para a formação da rota r_2:

- o veículo parte do depósito e vai em direção ao cliente 87;
- como o veículo chegou antes da abertura da janela de tempo, ele deve aguardar a abertura da mesma;
- é acrescido o tempo de serviço correspondente ao cliente, que refere-se ao tempo em que o veículo realiza a entrega;

- o veículo parte do cliente 87 em direção ao cliente 12;
- o veículo deve esperar a janela de tempo abrir para realizar a entrega da mercadoria;
- assim, é acrescido o tempo de serviço do cliente 12;
- o veículo retorna ao depósito antes do fechamento da janela de tempo.

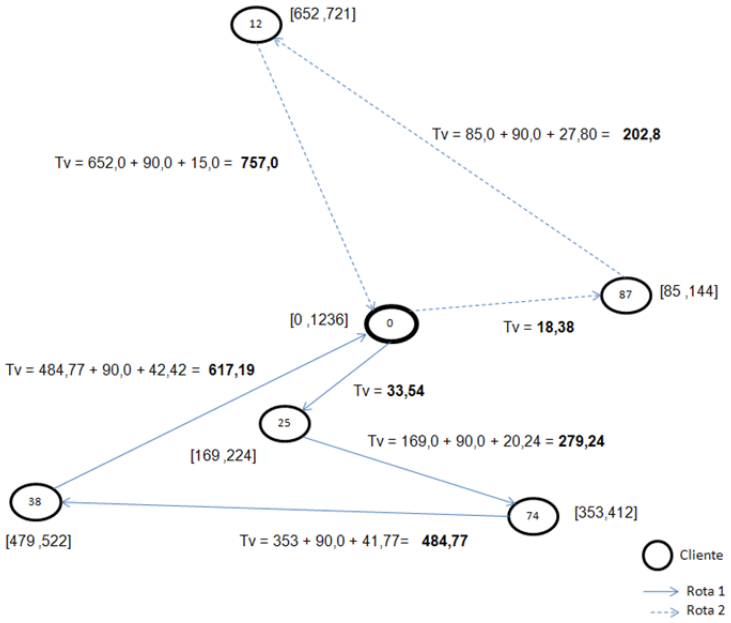

Figura 2.6: Exemplo para o PRVJT mostrando o cálculo do tempo de viagem entre dois clientes.

Outra verificação que é realizada é com relação à capacidade do veículo. Como se trata de uma frota homogênea, todos os veículos possuem capacidade igual. No caso da instância C101, esta capacidade é representada por 200 unidades. Para a rota r_2, a soma das demandas dos clientes é igual a 40 unidades, abaixo do limite de carga do veículo. A função objetivo da solução passa a ter valor igual a 2 em relação ao número de rotas e $199,15$ em relação à distância total percorrida. Na Figura 2.6, Tv representa o tempo de viagem entre os vértices.

2.3 Formulação Matemática

A formulação matemática descreve o problema de forma a melhorar a compreensão do mesmo. Assim, do ponto de vista computacional, o PRVJT é modelado como um grafo completo e não-direcionado $G = (V, A)$, em que o conjunto de vértices $V = \{0, 1, 2, \ldots, N\}$ representa os clientes e $A = \{(i, j) \quad : \quad i, j \in V, \ i \neq j\}$ representa o conjunto de arestas. Cada vértice i possui uma janela de tempo $[ti_i, tf_i]$, que representa o período de tempo no qual deve-se iniciar o atendimento. Caso o veículo chegue antes da abertura da janela de tempo, deve ser penalizado com o tempo de espera w_i. Além disso, é considerada uma demanda q_i e um tempo de

serviço ts_i relacionado a cada cliente. O custo de deslocamento entre dois vértices i e j é representado por d_{ij}, calculada através da distância euclidiana entre eles. Cada veículo tem capacidade Q máxima e pode compor apenas uma rota. O depósito é representado pelo índice $i = 0$ e mantém relacionado a ele uma janela de tempo $[Ti_i, Tf_i]$, que representa o intervalo de tempo na qual os veículos podem fazer as entregas. O veículo deve iniciar as entregas após o instante Ti_0 de abertura do depósito e retornar ao depósito antes do instante de tempo de fechamento Tf_0 do mesmo. O modelo matemático adotado foi proposto por Tan et al. (2001) e é dado por:

$$\min \quad \sum_{i=0}^{N} \sum_{j=0, j\neq i}^{N} \sum_{k=1}^{K} c_{ij} x_{ijk} \tag{2.1}$$

$$\text{suj. a} \quad \sum_{k=1}^{K} \sum_{j=1}^{N} x_{ijk} \leq K \qquad i = 0 \tag{2.2}$$

$$\sum_{j=1}^{N} x_{ijk} = \sum_{j=1}^{N} x_{jik} \leq 1 \qquad i = 0; \quad k \in \{1, ..., K\} \tag{2.3}$$

$$\sum_{k=1}^{K} \sum_{j=0, j\neq i}^{N} x_{ijk} = 1 \qquad i \in \{1, ..., N\} \tag{2.4}$$

$$\sum_{k=1}^{K} \sum_{j=0, j\neq i}^{N} x_{ijk} = 1 \qquad j \in \{1, ..., N\} \tag{2.5}$$

$$\sum_{i=1}^{N} q_i \sum_{j=0, j\neq i}^{N} x_{ijk} \leq Q_k \qquad k \in \{1, ..., K\} \tag{2.6}$$

$$Ti_0 = w_0 = Ts_0 = 0 \tag{2.7}$$

$$\sum_{k=1}^{K} \sum_{i=0, i\neq j}^{N} x_{ijk} \left(d_i + d_{ij} + ts_i + w_i\right) \leq tf_j \quad j \in \{1, ..., N\} \tag{2.8}$$

$$x_{ijk} \in \{0, 1\} \tag{2.9}$$

No modelo, a variável de decisão x_{ijk} pode assumir os valores 1 ou 0. Quando uma dada aresta (i, j) é atendida por um veículo de índice k, a variável x_{ijk} assume valor 1; caso contrário, recebe o valor 0. A variável d_{ij} exprime o tempo gasto na viagem entre dois clientes distintos, representado pela distância euclidiana entre clientes i e j. A variável w_i representa o tempo que o veículo esperou para a abertura da janela de tempo do cliente i. A expressão (2.1) mostra a função objetivo do problema: minimizar o custo total, representado pelo valor $c_{ij}x_{ijk}$, que consiste no somatório das distâncias percorridas.

A sentença (2.2) indica o número máximo de veículos que podem sair do depósito. A equação (2.3) garante que as rotas formadas iniciem e terminem no depósito central. As expressões (2.4) e (2.5) asseguram que cada par de clientes sejam visitados por um único veículo. Já a sentença (2.6) assegura que a capacidade do veículo seja respeitada. A equação (2.7) mostra que os tempos de viagem inicial, tempo de

espera e tempo de serviço são iniciados com o valor zero. A sentença (2.8) indica a condição de factibilidade entre dois clientes i e j, na qual o tempo de viagem para chegar ao vértice i, representado por d_i, mais a distância euclidiana entre os vértices i e j, representada por d_{ij}, mais o tempo de serviço ts_i gasto pelo veículo para realizar a entrega e o tempo de espera w_i, que representa o tempo que o veículo teve que esperar para a abertura da janela de tempo ti_i, devem ser menor ou igual à janela de tempo superior do cliente tf_j.

Capítulo 3

Revisão Bibliográfica

Este capítulo mostra, na Seção 3.1, aplicações de metaheurísticas para a solução do Problema de Roteamento de Veículos com Janela de Tempo (PRVJT) e variações em anos recentes. A Seção 3.2 relaciona aplicações da metaheurística *Iterated Greedy Search* a problemas diversos de Otimização Combinatória, mostrando a versatilidade desse método, apesar de sua recente proposição.

3.1 Resolução do problema PRVJT via Metaheurísticas

Diversos autores investigam o Problema de Roteamento de Veículos com Janela de Tempo, mostrando ser um problema de bastante destaque. Tanto algoritmos populacionais quanto de trajetória têm se mostrado eficientes na determinação de soluções para instâncias desse problema. Belmecheri et al. (2010) trabalha o Problema de Roteamento de Veículos com laços de linha mista, com regresso e janela de tempo (*Heterogeneous Vehicle Routing Problem with Mixed Linehauls and Backhauls with Time Windows* – HVRPMBTW). Os autores propõem o uso da metaheurística *Particle Swarm Optimization Heuristic* (PSO) e comparam com dois outros trabalhos, os quais aplicam a metaheurística *Ant Colony Optimization* (ACO) e o ambiente de programação matemática CPLEX. As comparações mostram um desempenho PSO superior ao ACO e ao CPLEX quando aplicados às instâncias de Solomon (1987).

Ma et al. (2010) elaboram um algoritmo genético (GA) para o problema *Vehicle Routing Problem with Soft Time Windows* (VRPSTW). É realizada uma comparação entre um GA tradicional e uma versão paralela desse metaheurística, em que são evoluídas duas populações. Na inicialização das populações, foram utilizadas a geração aleatória, e as heurísticas *Nearest Addition Method* (NAN) e *Sweep Algorithm*. O desempenho do GA de duas populações apresenta melhores resultados que o GA tradicional. Liu et al. (2010) aplica GA para o problema *Vehicle Routing Problem with Simultaneous Delivery and Pickup with Time Window* (VRPSDPTW) e apresenta uma melhoria no operador de recombinação, de modo a evitar ficar preso em ótimos locais. Já Wang (2010) apresenta um *Improved Genetic Algorithm* (IGA) para a resolução do PRVJT. O algoritmo IGA inicializa a população aplicando a metaheurística *Greed Randomized Adaptive Search Procedure* (GRASP). O meca-

nismo de seleção adotado utiliza um operador no qual os pais são escolhidos a partir do número de rotas de cada indivíduo, com o objetivo de herdar as rotas com maior número de clientes.

Mao e Deng (2010) desenvolvem, para a solução do PRVJT, um algoritmo evolucionário *Hybrid Evolutionary Algorithm* (HEA), baseado em GA, GRASP, PSO e *Expanding Neighborhood Search* (ENS). Na inicialização da população, é utilizada a metaheurística GRASP, juntamente com PSO; na fase de seleção, escolhe-se dois pais através do mecanismo de roleta. Esses pais passam pela fase de cruzamento e por um processo de clonagem, usando ENS, a fim de completar os descendentes; após essa etapa, é realizada a mutação e posteriormente a classificação utilizando PSO. Os resultados obtidos foram comparados com os da literatura, mostrando a eficiência dos algoritmos elaborados. Já Kim e Jeon (2010) desenvolve um IGA para o *Realistic Vehicle Routing Problem with Time Windows which considers traffic situations* (RVRPTWTS), uma variação do PRVJT considerando informações de tráfego das vias. O IGA parte de uma população inicial randômica que é classificada através do valor da funçao *Fitness* dos indivíduos. A evolução da população é feita tendo como referência um grupo de soluções que, após os processos de cruzamento e mutação, são comparadas, sendo selecionados os indivíduos que irão sobreviver à próxima iteração do método. O algoritmo desenvolvido, para análise de sua eficiência, é exposto a testes em instâncias baseadas em um caso real denominado *Barreled Drinking Water*, mostrando uma redução de custos em relação à situação real existente.

Quoc e Anh (2010) estuda o PRVJT e propõe um algoritmo combinando duas estratégias: programação por restrição (*Constraint Programming* - CP), com o objetivo de reduzir o número de rotas, e Busca Tabu (*Tabu Search* - TS), para reduzir a distância percorrida. Os testes realizados nas instâncias propostas por Solomon (1987) mostraram redução do número de rotas em 57% e 83% em relação à distância percorrida, comparados com os melhores resultados da literatura. Ma (2010) resolve o PRVJT através de um algoritmo baseado em colônia de formigas denominado *Improved Ant Colony Algorithm* (IACA), no qual aplica a estratégia de construção de caminho, que permite a troca de clientes durante o processo de busca. Para efeito de testes, foram desenvolvidas instâncias com características reais de até vinte clientes. Já Carabetti et al. (2010) utilizam ACO para o problema *Vehicle Routing Problem with Pickup and Delivery and Time Windows* (VRPPDTW), com estratégia de elitismo. Foram aplicados técnicas de busca local com diferentes estruturas de vizinhança. Testes realizados nas instâncias de Solomon (1987) adaptadas ao problema mostram a eficiência do método proposto.

Luo e Fu (2010) usam *Variable Neighborhood Tabu Search Algorithm* (VNTS) para o problema *Heterogeneous Fleet Vehicle Routing Problem with Time Windows* (HFVRPTW). Este algoritmo consiste em criar uma solução inicial utilizando o método denominado *GENIUS* e, em uma segunda fase, a técnica de Busca Tabu, de modo que os movimentos são modificados de acordo com o desempenho da melhor solução. A eficiência do algoritmo é demonstrada através da comparação com outras técnicas descritas na literatura, obtendo-se desempenho superior aos resultados relatados na maioria das instâncias. Já Jin e Fu (2010) desenvolvem um algoritmo genético híbrido (HGA) para o PRVJT, em que é utilizada a heurística com regra de prioridade para gerar a população inicial e o mecanismo *interchange* para

aumentar a diversidade da população. A avaliação do cromossomo é realizada utilizando um decodificador bi-direcional. Os testes feitos com instâncias de Solomon (1987) mostram a eficiência da ferramenta. Mengzhu et al. (2011) propõem um algoritmo genético híbrido para a solução do PRVJT utilizando as metaheurísticas Busca Tabu (TS) e *Simulated Annealing* (SA). Compara-se diferentes versões de várias metaheurísticas, através de instâncias de teste, sendo que a versão *Hybrid Genetic Tabu Search Simulated Annealing Algorithm* (GTSA) obteve os melhores resultados.

Kilic (2010) propõe um modelo baseado em programação possibilística para o PRVJT e utiliza em um algoritmo de Colônia de Formigas (*Ant Colony System* - ACS). Nesta proposta, o autor gera soluções através da metaheurística ACS e aplica o modelo de programação possibilística para encontrar soluções melhores. Para testar a eficiência do algoritmo desenvolvido, são utilizadas as instâncias de Solomon (1987) e de Li e Lim (2003) tendo 100 clientes, obtendo bons resultados. Já Schyns (2015) utiliza ACS para o problema *Dynamic Capacitated Vehicle Routing Problems with Time Windows, (partial) Split Delivery and Heterogeneous fleets* (DVRPTWSD). Nesta abordagem, além do aspecto inerente ao PRVJT, são acrescentados restrições em relação à diferença de capacidade dos veículos, à entrega fracionada da mercadoria, além das solicitações dinâmicas dos clientes durante a construção das soluções. Os testes adaptam as instâncias de Solomon (1987) para o problema em questão. Já Anxin (2011) desenvolve um algoritmo de Otimização por Nuvem de Partículas (*Improved Particle Swarm Optimization* - IPSO) para o PRVJT, com resultados relevantes para a literatura.

Yassen et al. (2012) propõem um algoritmo genético para o PRVJT com cruzamento de mais de dois pais. Para fins de testes, seleciona-se cinco instâncias de Solomon (1987) de diferentes classes e compara-se com os melhores resultados da literatura. Wei-Huai e Tsung-Che (2012) utiliza um algoritmo genético multiobjetivo (*Multiobjective Evolutionary Algorithm* - MEA) para o PRVJT, com operadores de cruzamento e mutação que levam em conta o conhecimento do domínio do problema. O operador de cruzamento indica uma troca entre as rotas formadas com o menor raio de distância. A eficácia da ferramenta mostra-se através dos testes de sete grupos de instâncias, comparando com os principais trabalhos da literatura. Blocho e Czech (2012) desenvolve um algoritmo paralelo denominado *Parallel EAX-based Algorithm* (EAX) para o PRVJT, aplicando-o às instâncias de maior dimensão desse problema, conhecidas como instâncias de Gehring e Homberger, apresentadas em Gehring e Homberger (1999). Já Blocho e Czech (2013) desenvolve um *Parallel Memetic Algorithm* (PMA) para o PRVJT, utilizando o algoritmo EAX na fase de busca local e um processo de reparação de soluções inviáveis, também explorando as instâncias de grande dimensão do problema.

Ruiz-Vanoye et al. (2012) propõem um algoritmo denominado *Transgenic Algorithm* (TA) para a solução do PRVJT. Trata-se de um algoritmo bio-inspirado, que se baseia na troca de genes horizontal e nas técnicas de agrupamento de soluções similares. Os testes com a classe de instâncias C, proposta por Solomon (1987), mostra-se com bons resultados, comparados com o algoritmo genético tradicional. Já Dhahri et al. (2013) aplicam diversas técnicas heurísticas para o PRVJT com restrição de disponibilidade e compara os resultados obtidos a partir da aplicação das técnicas nas instâncias de Solomon (1987).

Huang e Geng (2013) criam um *General Framework* (GF), aplicando diversas técnicas de busca local para o *Dynamic Vehicle Routing Problem with Time Windows* (DVRPTW). Os testes utilizam-se das instâncias de Solomon (1987) de forma adaptada, apresentando bons resultados. Já Yang (2013) propõem um algoritmo de Colônia de Formigas (*Ant Colony Algorithm* - ACA), com a introdução de perturbação caótica e função de penalidade. Os testes demonstram que a capacidade de exploração do espaço de busca do método utilizado é maior do que as metaheurísticas GA e ACA clássicas. Já Ruela et al. (2013) desenvolvem um algoritmo genético híbrido paralelo utilizando programação em nuvens *Parallel Hybrid Genetic Algorithm* (PHGA) para o PRVJT. A fim de mostrar a eficácia da ferramenta produzida, compara-se com diversos grupos de instâncias, obtendo resultados melhores do que da literatura em grande parte delas.

Baños et al. (2013) trabalham o PRVJT multiobjetivo no qual é considerado um balanceamento de carga de trabalho, além das restrições de janela de tempo, capacidade do veículo e distância total percorrida. Esse balanceamento é analisado pelas diferenças de distâncias percorridas entre a rota de maior e menor número de clientes e na distribuição das cargas entre os veículos de maior e menor capacidade, respectivamente. Nesta pesquisa é proposto um algoritmo denominado *Multi-start Multi-objective Evolutionary Algorithm with Simulated Annealing* (MMOEASA). Nele são combinadas as metaheurísticas *Multi-Start* (MS) e *Simulated Annealing* (SA). Para a solução inicial aplica-se a heurística *Time Window Based Insertion Heuristic* (TWHI), que leva em consideração os clientes que tem janela de tempo mais curtas para inserção na solução. Durante as iterações do método, são aplicados operadores de cruzamento e mutação, além de utilizar o operador de temperatura, na forma que é empregado no SA. Para medir o desempenho do MMOEASA compara-se os resultados obtidos com as metaheurísticas NSGA-II, proposta por Deb et al. (2000), e SPEA2, proposta por Zitzler et al. (2001). O método proposto traz, como contribuições, uma melhor exploração das soluções não-dominadas e das métricas de espaçamento, superando as obtidas pelo NSGA-II e SPEA2.

Hu et al. (2013) desenvolvem uma metaheurística denominado *Hybrid Chaos-particle Swarm Optimization Algorithm* (HPSO) para a resolução do PRVJT. Nela, para inicializar a nuvem de partículas, são geradas soluções aleatoriamente e realizada a decodificação através da heurística *Insertion Heuristic Algorithm* (IHA), sendo analisado o valor de *fitness* das soluções geradas. Logo após, é aplicada a metaheurística ACO e inserido o ótimo global encontrado no enxame de partículas. Para sair de ótimos locais, é aplicada mutação gaussiana, caso o algoritmo fique preso em uma região do espaço de busca em um determinado número de iterações. Neste caso, o enxame é reconstruído. Foram realizados testes com as instâncias de Solomon (1987) e comparados com o desempenho do PSO tradicional, na qual HPSO se mostrou mais eficiente. Já Zhenping et al. (2014) propõem um algoritmo denominado *Intelligent Water Drops Algorithm* (IWDA) para o PRVJT, aplicando em instâncias de teste.

Barbucha (2014) elabora um algoritmo denominado *Cooperative Population Learning Algorithm* (CPLA) para a resolução do PRVJT. Nele, são criados diversos agentes em paralelo, responsáveis por explorar o espaço de busca. Cada agente pode perturbar as soluções da população através de heurísticas e metaheurísticas. Na fase inicial, são criados indivíduos ou de forma randômica ou aplicando uma heurística

construtiva simples. Depois, é aplicado um estágio denominado Aprendizagem e Melhoria *Learning/Improvement Scheme*, quando os agentes são acionados e são mantidos na população as melhores soluções. Na fase denominada *Promotion Criterion* são definidas as soluções da população que irão sobreviver na próxima iteração. Os testes realizados pelo autor aplicam as instâncias propostas por Solomon (1987) de 100 clientes e compara os resultados obtidos com os melhores da literatura.

Melián-Batista et al. (2014) adaptam o PRVJT a um caso real de uma companhia localizada em Tenerife, Espanha. Assim, na construção das rotas, deve-se levar em consideração dois objetivos principais, que são a minimização da distância percorrida e a distribuição de trabalho entre os condutores dos veículos. Esta distribuição de trabalho tem relação direta com a extensão das rotas e com o tempo gasto para concluí-la. Um modelo linear inteiro misto bi-objetivo foi proposto e uma metaheurística denominada *Scatter Search Based Algorithm* (BioSS) foi desenvolvida. Na BioSS, na fase de criação da população inicial, é utilizada um algoritmo denominado *Parallel Route Building Algorithm* (PRBA), proposto por Potvin e Rousseau (1993) e, posteriormente uma série de buscas locais, com os objetivos de reduzir distâncias e balanceamento do tempo de viagem. Para testar a eficiência do algoritmo, duas versões do BioSS são comparadas com o NSGA-II através de testes nas instâncias de Solomon (1987), obtendo bons resultados.

Niu et al. (2014) implementaram um algoritmo evolucionário denominado *P-based Hybrid Evolutionary Algorithm* (PHEA) para a resolução do PRVJT. O PHEA é bio-inspirado e baseado em uma abordagem chamada *Membrane Computing*, em que são aplicadas regras que representam a separação e a comunicação entre as células, coletando e dispersando informações. Os operadores genéticos são aplicados para auxiliar a convergência de soluções de melhor qualidade. Os testes foram realizados utilizando as 56 instâncias de Solomon (1987) e a metaheurística mostrou-se uma ferramenta competitiva, quando têm seu resultados comparados com os resultados da literatura. Já de Armas e Melián-Batista (2015) desenvolveram uma adaptação da metaheurística *Variable Neighborhood Search* (VNS) em um problema denominado *Dynamic Rich Vehicle Routing Problem with Time Windows* (DRVRPTW), mostrando resultados competitivos com os melhores da literatura, tendo como base as instâncias de Solomon (1987).

Dayarian et al. (2015) estudam o *Multi-Period Vehicle Routing Problems* (MPVRP) e elabora um modelo matemático que divide o problema em sub-problemas e, ao mesmo tempo, propõem um algoritmo baseado em *Branch-and-Price Algorithm* (BPA). Küçükoğlu e Öztürk (2015) estudam o *Vehicle Routing Problem with Backhauls and Time Windows* (VRPBTW) e propõem um algoritmo híbrido denominado *Hybrid Meta-Heuristic Algorithm*(HMA). A eficiência do HMA, através de testes adotando-se as instâncias de Solomon (1987), mostra a relevância do trabalho para a literatura. Já Barkaoui et al. (2015) trabalham o DVRPTW com uma abordagem de satisfação do cliente em múltiplas visitas. Os autores propõem uma aplicação do tipo HGA para a solução de instâncias de Solomon (1987), adaptadas ao problema, com foco na satisfação do cliente, distância percorrida e número de rotas.

Já Wang et al. (2015b) fazem um estudo sobre o problema *Heterogeneous multi-type Fleet Vehicle Routing Problem with Time Windows and an Incompatible Loading Constraint* (HVRPTW-ILC). Nesta variação do PRVJT, cada veículo tem um va-

lor de capacidade próprio e são admitidos tipos diferentes de carga, podendo gerar uma restrição de incompatibilidade em relação à demanda do cliente. Os autores propõem um modelo matemático para o problema e duas heurísticas: *Ruin-recreate Heuristic Algorithm* (RR) e *Tabu Search* (TS). Adapta-se as instâncias clássicas de Solomon (1987) do PRVJT e compara-se com casos reais de empresas que atuam na China. Já Wang et al. (2015a) desenvolvem o algoritmo *Parallel Simulated Annealing* (pSA) para o problema *Vehicle Routing Problem in which Customers Require Simultaneous Pickup and Delivery of Goods During Specific Individual Time Windows* (VRPSPDTW). Nele, são levados em consideração clientes que têm coleta e entrega simultâneas, frota de veículos heterogênea e janela de tempo. Aplica-se um modelo linear de programação inteira, tendo, como objetivos, minimizar o custo do veículo e da viagem. Aplica-se testes nas instâncias de *Wang and Chen*, tendo 200, 400, 600, 800 e 1000 clientes, com boas contribuições para a literatura.

McNabb et al. (2015) apresentam *MAX-MIN Ant System* (MMAS) para o *Vehicle Routing Problem with Split Deliveries and Time Windows* (SDVRPTW). Nesta extensão do PRVJT, os clientes podem ter sua mercadoria entregue em partes, além da visita do veículo estar associada a uma restrição temporal. A fim de mostrar sua eficiência, comparam-se os desempenhos de vários operadores de busca local, adaptando as instâncias de Solomon (1987). Já Pace et al. (2015) tratam do problema denominado *Heterogeneous Fleet Capacitated Vehicle Routing Problem with Time Windows and Three-Dimensional Loading Constraints* (3L-HFCVRPTW), que combina aspectos de carregamento em três dimensões, transporte heterogêneo com limitação de capacidade e janela de tempo para entrega de mercadorias. Aplicam-se os algoritmos (SA) e *Iterated Local Search* (ILS) através de instâncias reais obtidas de indústrias, que realizam a entrega de placas de fibras. Os resultados mostram um desempenho melhor em buscas que procuram alterar as entregas em rotas distintas.

Yan et al. (2015) aplicam o *Two-step Solution Algorithm* (TSA) para o problema *Multi-trip Split delivery Vehicle Routing Problem with Time Windows* (MTSD-VRPTW). Nesta modalidade, as entregas podem ser feitas de forma fracionada, os veículos podem visitar os clientes mais de uma vez, respeitando-se as restrições impostas pela janela de tempo. Formula-se um modelo matemático, levando em consideração as interações entre janelas de tempo, entregas fracionadas e viagens múltiplas. Testa-se o algoritmo desenvolvido através de dados reais, mostrando ser uma ferramenta eficaz neste contexto. Já Zhou e Wang (2015) abordam o problema *Multi-Objective Vehicle Routing Problem with Time Windows* (MOVRPTW). Neste problema, são considerados 5 objetivos, que são o número de veículos, a distância total percorrida, o tempo de viagem da rota mais longa, o tempo de espera total e o tempo de atraso devido a chegadas tardias. Propõe-se um algoritmo denominado *Local Search-based Multiobjective Algorithm* (LSMOVRPTW), testado com dados reais e obtendo resultados superiores ao NSGA-II tradicional.

Jawarneh e Abdullah (2015) usam um algoritmo bio-inspirado baseado na metaheurística Otimização por Colônia de Abelhas para resolução do PRVJT. Nele, são inseridas informações sobre as soluções geradas, de forma a imitar as interações sociais e a dinâmica na resolução de problemas na colméia. Porém, a técnica *Bee Colony Optimisation* (BCO) é muito sensível tanto à diversidade da população inicial quanto aos parâmetros. Por isso, no artigo, compara-se o algoritmo BCO com a definição de parâmetros constante ao longo da execução e um algoritmo adaptativo,

em que eles são ajustados conforme o comportamento do algoritmo. Na geração da população inicial, comparam-se as heurísticas *Greedy Heuristic* (GE) e *Sequential Insertion Heuristic* (SIH), sendo que a SIH apresenta maior diversidade e, como consequência, um desempenho superior ao GE. Realiza-se testes através das instâncias de Solomon (1987), comparando com os melhores resultados heurísticos da literatura.

Bae e Moon (2016) apresentam um GA para o problema *Multi-depot Vehicle Routing Problem with Time Windows* (MDVRPTW). Nele, são considerados múltiplos depósitos e restrições do mundo real, através de entrega e instalação de produtos eletrônicos. Na formulação, são aplicados técnicas de programação não-linear, através de um método misto com programação inteira. Aplica-se, com o objetivo de mostrar o desempenho do GA, testes em três classes de instâncias propostas por Cordeau et al. (2001), com bons resultados. Já Brito et al. (2016) adapta um algoritmo VNS para o problema *Close-open Vehicle Routing Problem with Time Windows* (COVRPTW). É uma variação do PRVJT no qual há rotas que são abertas, ou seja, não têm a necessidade de voltar para o depósito e outras que são fechadas. Os autores combinam o GRASP para a construção da solução inicial e o VNS para explorar o espaço de busca, com diversas estruturas de vizinhança. Aplica-se as instâncias de Solomon (1987) para efeito de testes, mostrando a efetividade do método.

Haddadene et al. (2016) desenvolvem uma versão do algoritmo NSGA-II para o *Vehicle Routing Problem with Time Windows, Synchronization, and Precedence Constraints* (VRPTW-SP). Os autores adotam a abordagem de um problema real do tipo *Home Health Care* (HHC), que envolve a diminuição dos custos das rotas formadas para atender aos clientes. Assim, além das restrições de janela de tempo, são incluídas restrições de sincronização e de precedência entre os clientes. Os testes do algoritmo utilizam-se de dados reais de até 75 clientes, mostrando a eficiência do método abordado. Errico et al. (2016) atuam no problema *Vehicle Routing Problem with Hard Time Windows and Stochastic Service Times* (VRPTW-ST) com um algoritmo exato denominado *Branch-cut-and-Price algorithm*. No VRPTW-ST são consideradas a execução de serviços de forma dinâmica, não sendo permitidas violações na janela de tempo. Na modelagem do problema, são consideradas duas fases principais, que são a montagem da cobertura de viabilidade e a tentativa de reparar as rotas inviáveis. O algoritmo separa as soluções em grupos e definem os limites inferiores e superiores para realizar a busca. Os autores aplicam as instâncias de Solomon (1987) de até 50 clientes nos testes e mostram o desempenho da ferramenta desenvolvida.

de Grancy e Reimann (2016) comparam as metaheurísticas ACO e GRASP na resolução do *Vehicle Routing Problem with Time Windows and Multiple Service Workers* (VRPTWMS). O VRPTWMS trata de uma variação do PRVJT na qual são contados os custos com relação à alocação de trabalhadores, à minimização do tempo de entrega dos clientes e aos custos do veículo, representado pela distância percorrida. Compara-se os resultados do método exato proposto por Pureza et al. (2012) com os algoritmos ACO e GRASP, através de instâncias adaptadas ao problema, obtendo bons resultados. Já Li et al. (2016b), elaboram um algoritmo denominado *Hybrid Genetic Algorithm with Adaptive Local Search* (HGA-ALS) para o problema *Multi-depot Vehicle Routing Problem with Time Windows Under Shared Depot Resources* (MDVRPTWSDR). Nesta variação do PRVJT, há múltiplos

depósitos e recursos compartilhados entre eles. É considerada uma variação do MDVRPTW com limitação de recursos e rotas fechadas. O HGA-ALS introduz, após o processo de mutação e antes do cruzamento, uma busca local adaptativa que controla a intensidade da perturbação de acordo com a melhoria da função objetivo. Aplica-se, para efeito de testes, as instâncias de Cordeau et al. (2001) através de comparações com variações do MDVRPTW e MDVRPTWSDR, com resultados competitivos com os melhores da literatura.

Keskin e Çatay (2016) estudam o *Electric Vehicle Routing Problem with Time Windows* (EVRPTW), uma variação do problema PRVJT que considera veículos elétricos. Os autores propõem uma adaptação do algoritmo *Adaptive Large Neighborhood Search* (ALNS). Nela, são combinadas diversas técnicas para explorar o espaço de busca. Mostra-se, através dos testes nas instâncias de Azi et al. (2014), Goeke e Schneider (2015) e Hemmelmayr et al. (2012), ser uma técnica que traz resultados relevantes para a literatura. Já Penna et al. (2016) trabalha uma outra variação denominada *Electric Fleet Size and Mix Vehicle Routing Problem with Time Windows and Recharging Stations* (E-FSMVRPTW), que trata uma variação do PRVJT com frota de veículos elétricos e mistos, considerando a existência de estações de recarga. São propostas diversas técnicas, comparadas com os resultados da literatura, contribuindo para o estado da arte do problema.

Nguyen et al. (2016) tratam do problema *Vehicle Routing Problem with Time Windows with Stochastic Travel Times* (VRPTWST). Este problema traz o conceito de satisfação do cliente (*Satisficing Measure Approach* - SMA) que é a medida relacionando a capacidade do veículo, o tempo de serviço e o tempo de viagem, de forma a conseguir um equilíbrio entre essas restrições de forma mensurável. Os autores elaboram um algoritmo TS e comparam os resultados com as instâncias de Solomon (1987) modificadas para o problema, com resultados competitivos com os da literatura. Já Cruz-Chávez e Martínez-Oropeza (2016) desenvolvem um método de geração de indivíduos para algoritmos populacionais para resolução do PRVJT. Este processo é crucial para o bom desempenho de qualquer algoritmo deste tipo. Por isso, abordam duas técnicas, que são a *Clustering Algorithm*(CA) e a *Two-phase Algorithm*. A primeira proposta gera soluções parciais e as agrupa; a segunda, combina partes dessas soluções para formar indivíduos viáveis. O *Two-phase Algorithm* associa diversas técnicas de inserção de clientes de forma aleatória e, posteriormente, são analisados através do *Hamming Distance Method* adaptado ao problema, para garantir a diversidade de indivíduos na população. Utiliza-se as instâncias de Solomon (1987), para fins de teste, mostrando o desempenho do algoritmo proposto.

Liao et al. (2016) usam um algoritmo de duas fases para resolver o PRVJT. Nesta proposta, na primeira fase, é utilizada a metaheurística ACO aplicando *AS model*, em que são definidos "pré-caminhos", que são agrupamentos de clientes distribuídos em uma determinada região de busca. Posteriormente, cada segmento é otimizado, aplicando a heurística *Insertion Heuristic Algorithm* - IHA. A técnica mostra-se eficiente na organização dos clientes em relação à restrição temporal. Já Nalepa e Blocho (2016) investigam o PRVJT e desenvolvem um algoritmo denominado *Adaptive Memetic Algorithm* (AMA). A proposta desenvolve um novo esquema de seleção adaptativa, que melhoram tanto a busca em profundidade quanto a exploração de diversas regiões do espaço de busca. No que diz respeito à evolução da população, na fase de cruzamento, são utilizados *Assembly Crossover Operator* (EAX) e um

Tabela 3.1: Panorama de metaheurísticas para a solução do PRVJT e de suas variações – Parte I

Ano	Trabalho	Problema	Técnica
2010	Belmecheri et al. (2010)	HVRPMBTW	PSO
2010	Ma et al. (2010)	VRPSTW	GA
2010	Liu et al. (2010)	VRPSDPTW	GA
2010	Wang (2010)	PRVJT	IGA
2010	Mao e Deng (2010)	PRVJT	HEA
2010	Kim e Jeon (2010)	RVRPTWTS	HGA
2010	Quoc e Anh (2010)	PRVJT	CP e TS
2010	Ma (2010)	VRPPDTW	IACO
2010	Carabetti et al. (2010)	VRPPDTW	ACO + Elitismo
2010	Luo e Fu (2010)	HFVRPTW	VNTS
2010	Jin e Fu (2010)	PRVJT	GA
2010	Kilic (2010)	PRVJT	ACS
2011	Mengzhu et al. (2011)	PRVJT	GA+TS+AS
2011	Anxin (2011)	PRVJT	IPSO
2012	Yassen et al. (2012)	PRVJT	MEA
2012	Blocho e Czech (2012)	PRVJT	EAX
2012	Ruiz-Vanoye et al. (2012)	PRVJT	TA

procedimento para reparar soluções e outro que seleciona os melhores filhos da geração anterior. Mostra-se a eficácia do algoritmo através de testes, tendo, como base, as instâncias de Solomon (1987) e Gehring e Homberger (1999), obtendo resultados competitivos com o estado da arte do problema.

Pérez-Rodríguez e Hernández-Aguirre (2016) estudam o PRVJT em uma nova abordagem, na qual é combinado um algoritmo de distribuição e um modelo de simulação, denominado *Estimation of Distribution Algorithms* (EDA). A estimativa da melhor posição de um vértice é determinada por uma rede bayesiana, construindo um grafo com as informações dos clientes. Utiliza-se as instâncias de Solomon (1987) para fins de teste, apresentando resultados superiores aos reportados na literatura. Já Alvarez e Munari (2017) elaboram um método híbrido que mistura programação matemática e metaheurísticas, denominado *Exact Hybrid Method* (EHM) para o problema *Vehicle Routing Problem with Time Windows and Multiple Deliverymen* (VRPTWMD). Nesta variação do PRVJT, o tempo de serviço pode variar de acordo com o número de funcionários responsáveis pela entrega. O EHM, para resolver o problema em questão, combina *Branch-cut-and-Price* (BPC) e as metaheurísticas ILS e LNS. Compara-se o desempenho do BPC com o do EHM, através de instâncias adaptadas, mostrando que o EHM é uma ferramenta que apresenta um desempenho superior na grande maioria das instâncias testadas. Braaten et al. (2017) propõem quatro versões de um algoritmo baseado em ALNS para o problema *Robust Vehicle Routing Problem with Time Windows* (RVRPTW). Neste problema, são consideradas inúmeras situações que geralmente não são tratadas em problemas determinísticos. Desta forma, são inseridas variáveis que tornam o tempo de viagem incerta, limitando o tempo de atraso da rota. No algoritmo, são aplicadas diversas estruturas de vizinhança afim de explorar o espaço de busca. Mostra-se a eficiência da ferramenta desenvolvida através de testes nas instâncias de Agra et al. (2012), com importantes contribuições para o estado da arte do problema. Já Pierre e Zakaria (2017) resolve o PRVJT através do algoritmo multi-objetivo *Multi-Objective Genetic Algorithm* (MOGA). Nesta proposta é desenvolvida o operador de cruzamento

Tabela 3.2: Panorama de metaheurísticas para a solução do PRVJT e de suas variações – Parte II

Ano	Trabalho	Problema	Técnica
2013	Blocho e Czech (2013)	PRVJT	PMA
2013	Dhahri et al. (2013)	PRVJT	diversas técnicas
2013	Huang e Geng (2013)	DVRPTW	GF
2013	Yang (2013)	PRVJT	ACA
2013	Ruela et al. (2013)	PRVJT	PHGA
2013	Baños et al. (2013)	PRVJT	MMOEASA
2013	Hu et al. (2013)	PRVJT	HPSO
2014	Barbucha (2014)	PRVJT	CPLA
2014	Melián-Batista et al. (2014)	PRVJT	BioSS
2014	Niu et al. (2014)	PRVJT	PHEA
2014	Zhenping et al. (2014)	PRVJT	IWDA
2015	Wang et al. (2015b)	HVRPTW-ILC	RR e TS
2015	Wang et al. (2015a)	VRPSPDTW	pSA
2015	de Armas e Melián-Batista (2015)	DRVRPTW	VNS
2015	Schyns (2015)	DVRPTWSD	ACS
2015	Barkaoui et al. (2015)	DVRPTW	VNS
2015	McNabb et al. (2015)	SDVRPTW	MMAS
2015	Pace et al. (2015)	3L-HFCVRPTW	SA e ILS
2015	Yan et al. (2015)	MTSDVRPTW	TSA
2015	Zhou e Wang (2015)	MOVRPTW	LSMOVRPTW
2015	Jawarneh e Abdullah (2015)	PRVJT	BCO
2016	Bae e Moon (2016)	MDVRPTW	GA
2016	Brito et al. (2016)	COVRPTW	VNS
2016	Haddadene et al. (2016)	VRPTW-SP	NSGA-II
2016	Errico et al. (2016)	VRPTW-ST	*Branch-cut-and-Price*
2016	de Grancy e Reimann (2016)	VRPTWMS	ACO e GRASP
2016	Li et al. (2016b)	MDVRPTWSDR	HGA-ALS
2016	Keskin e Çatay (2016)	EVRPTW	ALNS
2016	Penna et al. (2016)	E-FSMVRPTW	diversas técnicas
2016	Nguyen et al. (2016)	VRPTWST	TS
2016	Cruz-Chávez e Martínez-Oropeza (2016)	PRVJT	*Two-phaseAlgorithm*
2016	Liao et al. (2016)	PRVJT	*Two-phase Algorithm*
2016	Nalepa e Blocho (2016)	PRVJT	AMA
2016	Pérez-Rodríguez e Hernández-Aguirre (2016)	PRVJT	EDA
2017	Alvarez e Munari (2017)	VRPTWMD	EHM
2017	Braaten et al. (2017)	RVRPTW	ALNS
2017	Pierre e Zakaria (2017)	PRVJT	MOGA

Stochastic Partially Optimize Cyclic Shift Crossover (SPOCSX), na qual os pais selecionados passam por um cruzamento estocástico, que procura manter a factibilidade. Compara-se os desempenhos do GA e do MOGA através de testes nas instâncias de Solomon (1987), com os melhores resultados relatados na literatura, com contribuições importantes para o problema.

As Tabelas 3.1 e 3.2 resumem os principais trabalhos relacionados à solução do PRVJT e suas variações, nos termos discutidos nesta revisão bibliográfica.

3.2 Uma revisão de aplicações da metaheurística *Iterated Greedy Search* (IGS)

A metaheurística *Iterated Greedy Search* (IGS) foi proposta por Ruiz e Stützle (2007) e é aplicada a diversos problemas combinatórios. Neste trabalho, os autores propõem uma solução para o problema de alocação de tarefas (*Permutation Flowshop Scheduling Problem* – PFSP). Basicamente, o algoritmo realiza, a partir

de uma solução inicial, dois processos principais: destruição e reconstrução. A destruição, no caso do PFSP, consiste na retirada de tarefas de forma aleatória. Já na fase de reconstrução, as tarefas retiradas são reinseridas, formando novamente uma solução completa. Uma busca local também é aplicada. Com isso, as soluções obtidas alcançaram bons resultados, mostrando ser uma ferramenta eficaz para a resolução do problema. O IGS, pela sua simplicidade de implementação e facilidade de adaptação a outros problemas, foi amplamente explorada nos anos recentes, o que a torna uma técnica relevante.

Ruiz e Stützle (2008) aplicam IGS para a resolução do *Sequence Dependent Setup Times Flowshop Problem with Makespan and Weighted Tardiness Objectives* (SDST-FSP). Nesta variação do PFSP, há introdução de restrições de tempo de preparação, tendo por objetivo reduzir o *makespan* (tempo de término da última tarefa) e emprego de penalidades por atraso. São desenvolvidas duas versões do IGS, a clássica, conforme proposta em Ruiz e Stützle (2007), e uma outra, na qual é aplicado uma busca local. Os resultados obtidos contribuíram para o estado da arte do problema.

Ying (2008) desenvolvem IGS para o problema *Non-permutation Flowshop Scheduling Problems* (NPFSP). Nesta versão, são comparados IGS com outras duas técnicas: *Simple Constructive Heuristics* (SCH) e *Multi-heuristic Desirability Ant Colony System* (MHD-ACS), proposta por Ying e Lin (2007). Os resultados são compatíveis com os melhores resultados da literatura. Já Pan et al. (2008) apresenta *Improved Iterated Greedy Algorithm* (IIGA) para o *No-wait Flow Shop Scheduling Problem* (NW-FSP). O IIGA combina a heurística construtiva *Nawaz-Enscore-Ham* (NEH) na fase inicial e uma busca local *Simple Local Search Algorithm* (SLSA), que foram incorporados ao IGS tradicional. O processo de escolha e re-inserção das tarefas se faz pelas metaheurísticas NEH e M-NEH e pela busca local *Insert-based Local Search* aplicada iterativamente ao longo do algoritmo. Os testes realizados foram comparados com outras metaheurísticas, mostrando a efetividade do método.

Ying et al. (2009) estudam o problema *Single-machine Tardiness Problem with Sequence Dependent Setup Times* (STP-SDST) e propõem IGS para sua resolução. Os autores discutem a variação do problema de agendamento de tarefas com tempo de preparação e elaboram uma técnica de solução usando IGS, através de uma construção inicial que leva em conta o custo aparente para inserir a tarefa (*Apparent Tardiness Cost with Setups* - ATCS), organizando de forma crescente e inserindo na solução conforme essa sequencia. Para o processo de destruição, são selecionadas tarefas diferentes de forma aleatória. Na fase de reconstrução, são realizadas tentativas de inserção a fim de diminuir o *makespan* da solução. Os processos são realizados de forma iterativa, com um critério de aceitação, que decide se há exploração do espaço de busca de piora ou não, determinado pelo operador de temperatura, proposto no algoritmo SA. Os resultados comparam o desempenho das principais metaheurísticas aplicadas ao problema e mostram a competitividade do algoritmo construído.

Ying e Cheng (2010) empregam IGS para o problema *Dynamic Parallel Machine Scheduling with Sequence-dependent Setup Times* (DPMS-SDST). A proposta consiste em, à partir uma solução inicial, realizar retirada de tarefas (destruição) e re-inserir na solução (reconstrução) e uma busca local com o objetivo de reduzir o *makespan*. O critério de aceitação baseado na temperatura também é aplicado nas iterações do método. Os testes realizados com o conjunto de instâncias de Ovacik e

Uzsoy (1995) apresentaram resultados superiores ao da literatura.

Fanjul-Peyro e Ruiz (2010) propõem um algoritmo denominado *Iterated Greedy Local Search* (IGLS) para o problema *Unrelated Parallel Machine Scheduling* (UPMS). O IGLS é uma modificação do IGS original, na qual foram empregadas diversas técnicas de busca local e várias versões de IGS combinados. Os resultados mostram que, na comparação com diversos grupos de instâncias, os algoritmos obtiveram bons resultados em um tempo computacional abaixo dos relatados na literatura. Já Ribas et al. (2011) abordam o *Flow Shops Scheduling Problem with Blocking* (FSPB) em uma versão do IGS. Como entrada do método, é utilizada a heurística *Improved NEH-Based Heuristic*. Posteriormente, a solução gerada é modificada através das fases de destruição e reconstrução. A busca local utiliza a heurística *Non-Exhaustive Descent Algorithm* (NEDA). A eficiência do algoritmo foi verificado através de testes em diversos grupos de instâncias e foram encontrados resultados melhores do que os relatados na literatura com instâncias de Kang et al. (1993)

Kang et al. (2011) aplicam o IGS no problema *Heterogeneous Computing* (EC), que trata da alocação de tarefas para múltiplos processadores. Nesta implementação, o IGS objetiva reduzir o custo de comunicação entre os processadores. Para realizá-lo, são aplicados, como solução inicial, a heurística *Greedy Constructive Heuristic* (GCH); na destruição, a retirada de tarefas aleatórias; na reconstrução, as tarefas retiradas são re-inseridas na solução, de forma que acarrete o menor custo na alocação de um processador e a busca local com a heurística *Rather Straight-forward Local Search Algorithm* (RSLS). Os testes realizados comparam o desempenho do IGS com outras três metaheurísticas presentes na literatura, obtendo os melhores resultados.

Tasgetiren et al. (2013b) desenvolvem uma pesquisa na solução do problema *No-idle Permutation Flow Shop Scheduling Problem* (NIPFS). Os autores propõem um método populacional denominado *Variable Iterated Greedy Algorithm with Differential Evolution* (VIG-DE). Nele, a quantidade de tarefas retiradas das soluções apresentadas e a probabilidade da utilização do IGS é controlada dinamicamente pela evolução diferencial, além do cruzamento e mutação aplicando a população durante a execução do método. Os testes realizados comparam o desempenho do VIG-DE com os melhores resultados da literatura e mostraram ser esta uma ferramenta eficiente para resolução do problema. Já Kang et al. (2013) investiga um problema de computação distribuída envolvendo a alocação de tarefas orientada à confiabilidade (*Reliability-Oriented Task Allocation in Distributed Computing Systems* - ROTA-CA). Nesta proposta são empregadas, na construção inicial, uma heurística construtiva GCH e uma busca local relacionada com a retirada de um processador e a realocação das tarefas em outro processador, de forma iterativa, até que seja encontrado um tempo inferior ao tempo corrente. Na fase de destruição, as tarefas são selecionadas de forma aleatória e retiradas de acordo com o parâmetro que determina a quantidade de tarefas a serem retiradas. Na fase de reconstrução, essas tarefas são re-inseridas, de forma a priorizar os processadores com menor quantidade de tarefas. Os resultados são comparados com o estado da arte do problema e verifica-se que obteve melhores resultados na maioria das instâncias testadas.

Rodriguez et al. (2013a) tratam do Problema de Agendamento de Máquinas Paralelas Não-Relacionadas com Minimização dos Tempos de Conclusão das Tarefas

(*Unrelated Parallel Machine Scheduling with Minimizing Total Weighted Completion Times problem* - UPMS). Na fase de destruição do IGS são aplicadas duas estratégias de seleção das tarefas: a primeira utiliza seleção uniforme e torneio binário; a segunda seleciona uma tarefa aleatoriamente. Na fase de reconstrução, várias heurísticas são adotadas, a fim de diminuir o tempo máximo de finalização das tarefas. Os testes realizados em vários grupos de instâncias foram comparados, sendo que o IGS apresenta os melhores resultados

Karabulut e Tasgetiren (2014a) utilizam o algoritmo *Variable Iterated Greedy* (VIG) para a resolução do *Traveling Salesman Problem with Time Windows* (TSPTW). Este problema consiste em, dado um conjunto de clientes, construir uma única rota que atenda a todos eles de forma a diminuir o custo total da viagem. O funcionamento do VIG se dá de forma a gerar um número muito grande de soluções vizinhas, baseada na ideia do VNS. Assim, são retirados clientes e re-inseridos na solução, variando a sua quantidade ao longo da execução do algoritmo. Os testes foram realizados nos grupos de instâncias Dumas et al. (1995), Gendreau et al. (1998), Ohlmann e Thomas (2007), Kahraman et al. (2010) e no conjunto de instâncias RC2 proposta por Solomon (1987), de forma adaptada. Em todas elas foram obtidos resultados competitivos aos melhores da literatura. Já Lozano et al. (2014) mostram uma versão da metaheurística IGS para o Problema de Alocação de Tarefas (*Flowshop Scheduling Problem* - FSP), em que apresentam uma técnica de desempate que leva em conta não só o valor do *makespan*, ou seja, o tempo de término da última tarefa alocada, mas também uma variação na solução, garantindo uma melhor exploração do espaço de busca. Os testes realizados comparam diferentes propostas apresentadas por Ruiz e Stützle (2007) e outros autores da literatura, mostrando a eficácia da ferramenta desenvolvida.

Lin et al. (2014) propõem a aplicação da metaheurística IGS para o Problema de Minimização do Tempo Total de Serviço na Alocação de Navios em um porto de forma dinâmica (*Discrete Dynamic Berth Allocation Problem* - DBAP). Neste contexto, o IGS recebeu adaptações que permitiram uma variabilidade na fase de destruição, sendo selecionado randomicamente o número de embarcações, limitando-se a um intervalo. A reconstrução é feita de forma sequencial, reinserindo as embarcações retiradas na melhor posição em relação à função objetivo. Na versão apresentada, não há busca local, apenas o critério de aceitação, que utiliza o operador de temperatura, tal como é utilizado na metaheurística SA. Já Pedraza et al. (2014) estudam o algoritmo IGS na aplicação de regras para a mineração de dados. Os estudos realizados pelos autores verificam diferentes alternativas para estabelecer as regras de classificação das informações com as que são colocadas no estado da arte do problema. Os resultados mostraram que IGS pode gerar sistemas de classificação com regras precisas e com níveis de interpretação, contribuindo para o estado da arte do problema. Já Shirvani et al. (2014) aplica IGS no Problema de Escala Cíclica Realista com produtos perecíveis em data de lançamento, data de vencimento e prazos (*Realistic Cyclic Scheduling Problem* - RCSB). Nesta proposta, os autores comparam os resultados obtidos com diferentes versões do IGS combinados com buscas locais distintas, mostrando a eficiência do IGS para o problema.

Yang e Sun (2015) tratam o *Electric Vehicles Battery Swap Stations Location Routing Problem* (BSS-EV-LRP). Para sua resolução, é desenvolvido uma heurística denominada *Sweep heuristic Iterated Greedy Adaptive Large Neighborhood Search*

(SIGALNS), que combinam *Modified Sweep Heuristic*, IGS, *Adaptive Large Neighborhood Search* e *Improvement Heuristic*. Essas quatro técnicas são combinadas de forma iterativa, sempre com o objetivo de diminuir o custo total de atendimento dos clientes e da melhor posição para inserir as estações para a troca e carregamento dos carros elétricos. O desempenho do SIGALNS foi comparado com a metaheurística *Tabu Search and modified Clarke and Wright Saving algorithm* (TS-MCWS) proposta pelos autores e obtendo bons resultados, comparados com melhores da literatura.

Mokhtari (2015) abordam o problema denominado *Simultaneous Order Scheduling and Resource Allocation* (SSRAP) e elaboram um algoritmo chamado *Two-layered Iterated Greedy Heuristic* (T-IG). Na proposta, o algoritmo parte de uma solução inicial e realiza a retirada de tarefas de forma aleatória. Na fase de reconstrução, as tarefas são re-inseridas empregando *Greedy Heuristic*. O critério de aceitação determina se o algoritmo irá continuar explorando a solução corrente ou retornar à solução anterior. A busca local é híbrida, na qual são empregados *Greedy Operations*, de forma a explorar melhor o espaço de busca. O desempenho foi mensurado através da comparação do algoritmo desenvolvido com um GA tradicional, mostrando bons resultados. Já Quevedo-Orozco e Ríos-Mercado (2015) propõem uma solução para o Problema dos Vértices p-*Center* Capacitado (*Capacitated Vertex p-Center Problem* - CpVP) empregando algoritmo *Iterated Greedy Local Search* (IGLS). Na abordagem, é aplicada a busca local *Variable Neighborhood Descent* (VND) e um procedimento denominado *Shake Method* (SM) dentro do critério de aceitação da solução, após as fases de destruição, reconstrução e busca local. O método SM realiza uma perturbação na solução caso ela tenha uma qualidade pior à solução anterior. Essa perturbação consiste em agrupar os vértices e apenas um subgrupo passa pelo processo de retirada dos vértices. Após essa fase, a solução é reconstruída e o método passa novamente pelas fases de destruição, reconstrução e busca local. Os testes realizados mostraram uma qualidade maior nas soluções geradas e no tempo de execução, comparados com técnicas semelhantes.

Nucamendi et al. (2015) mostram a aplicação da metaheurística IGS ao Problema de Roteamento de Veículos Centrado no Cliente, com Unidade de Demanda (*Unit Demand Customer-Centric Vehicle Routing Problem* - UDCCVRP). Neste trabalho, os autores desenvolvem duas estratégias para a fase de reconstrução: *Complete Evaluation* (CE) e *Random Selection* (RS). A técnica CE é obtida através da distância do cliente do centro do *cluster* e a técnica RS consiste na re-inserção randômica do cliente. Na busca local aplica-se uma heurística denominada *Local Search Improvement* (LSI), que combinam diversas estruturas de vizinhanças. Os resultados obtidos foram comparados com instâncias de 60 a 100 clientes, sendo IG-RS mais rápido computacionalmente do que IG-CE e ambos conseguindo bons resultados. Outra utilização é relatada em Hatami et al. (2015) sobre o Problema de Sequenciamento em Máquinas Paralelas Distribuídas com Tempo de Preparação (*Distributed Assembly Permutation Flowshop Scheduling Problem with Sequence Dependent Setup Times* - DAPFSP-SDST). Neste trabalho, são desenvolvidas duas heurísticas construtivas para realizar a entrada para o método, além da técnica de busca local *Variable Neighborhood Search* (VNS). Também são propostas duas técnicas para as fases de destruição e reconstrução e um critério diferente de aceitação, que substitui as diferenças entre os *makespans* das soluções por uma diferença percentual definida

como *Relative Percentage Difference* (RPD).

Vincent e Lin (2015) resolvem o problema *Time-Dependent Prize-Collecting Arc Routing Problem* (TD-PARP) através do IGS. O problema consiste em determinar o número de caminhões necessários para montar rota e maximizar o lucro. O IGS proposto parte de uma solução inicial gerada aleatoriamente e são escolhidos um determinado número de arcos entre um valor mínimo e máximo, definidos previamente. Posteriormente, os arcos retirados são re-inseridos na solução, mantendo a ordem de retirada. Após estas etapas, são aplicadas buscas locais. A fim de comparar o desempenho do IGS, os autores propõem duas buscas locais e comparam os resultados obtidos com variações do IGS, obtendo os melhores resultados da literatura em grande parte das instâncias testadas. Já Marcelo et al. (2015) comparam o IGS com outras técnicas para a resolução do Problema de Agendamento de Tarefas sem Tempo de Espera e com Tempos de Preparação Dependendo da Sequência (*No-Wait Flow Shop Scheduling with Sequence Dependent Setup Times* - NWFS-SDST). Foram encontrados bons resultados, comparados com a literatura.

Tasgetiren et al. (2017) estudam o problema *Blocking Flowshop Scheduling Problem with the Makespan Criterion* (BFSP) através do IGS. Como ponto de partida, é proposta uma heurística construtiva (*Constructive Heuristic*) gerando resultados semelhantes ao da literatura. Posteriormente, no processo de busca, são executados as fases de destruição, na qual são retiradas da solução um número de tarefas aleatoriamente e a reconstrução, em que as tarefas são re-inseridas, aplicando a heurística NEH. Na busca local, são elaboradas duas heurísticas, a *RIS Local Search* (RIS-LS) e a *Best Swap local search* (BS-LS), combinadas em diferentes versões IGS. Os testes realizados comparam diversas técnicas com os melhores resultados da literatura. Já Li et al. (2017) estuda *Two-sided Assembly Line Balancing Problem* (TALBP) e elaboram uma versão *Improved Iterated Greedy Algorithm* (IIGA) para sua resolução. O algoritmo parte de um solução obtida aplicando uma adaptação do NEH, combinado com a heurística *Ranked Positional Weight* (RPW). Também são aplicadas diversos métodos de busca local, além das fases de destruição e reconstrução propostas por Ruiz e Stützle (2007). Os testes realizados nos conjuntos de instâncias do problema e comparados com diversas outras técnicas relatadas em Özcan e Toklu (2009), Khorasanian et al. (2013), Yuan et al. (2015), Kim et al. (2000),Li et al. (2016a), Özbakır e Tapkan (2011), Simaria e Vilarinho (2008) e Simaria e Vilarinho (2009) mostram a eficiência do IIGA, contribuindo para o estado da arte do problema.

Arroyo e Leung (2017a) elabora uma versão IGS denominada *Effective Iterated Greedy Algorithm* (E-IG) para o problema *Scheduling Unrelated Parallel Batch Machines with Non-identical Capacities and Unequal Ready Times* (SUPBM-NICURT). Os autores primeiramente definem um *lower bound* para o problema utilizando programação inteira mista (MIP). Para sua resolução, são aplicados na solução inicial uma heurística construtiva denominada *H3-FF*, proposta por Arroyo e Leung (2017b). A destruição ocorre com a retirada de tarefas de forma aleatória e a reconstrução é realizada inserindo cada tarefa na solução, respeitando a ordem, através da heurística *Greedy Insertion Heuristic* (GIH). Os testes realizados compararam versões implementadas pelos autores de metaheurísticas da literatura (SA, GA, MMAS e SA) com o algoritmo proposto e mostrou bons resultados, contribuindo para o estado da arte do problema. Já Pan et al. (2017b) estudam o problema *Hybrid*

Tabela 3.3: Panorama de aplicação do IGS na resolução de problemas combinatórios de diversas classes

Ano	Trabalho	Problema	Técnicas Acessórias
2007	Ruiz e Stützle (2007)	PFSP	IGS
2008	Ruiz e Stützle (2008)	SDST-FSP	IGS
2008	Ying (2008)	NPFSP	IGS
2008	Pan et al. (2008)	NW-FSP	IIGS
2009	Ying et al. (2009)	STP-SDST	IGS + ATCS
2010	Ying e Cheng (2010)	DPMS-SDST	IGS
2010	Fanjul-Peyro e Ruiz (2010)	UPMS	IGLS
2011	Ribas et al. (2011)	FSP	IGS + NADA
2011	Kang et al. (2011)	EC	IGS + GCH + RSLS
2013	Tasgetiren et al. (2013b)	NIPFS	VIG-DE + IGS
2013	Kang et al. (2013)	ROTA-CA	IGS + GCH
2013	Rodriguez et al. (2013a)	UPMS	IGS
2014	Karabulut e Tasgetiren (2014a)	TSPTW	VIG
2014	Lozano et al. (2014)	FSP	IGS
2014	Lin et al. (2014)	DBAP	IGS + SA
2014	Pedraza et al. (2014)	Mineração Dados	IGS
2014	Shirvani et al. (2014)	RCSB	IGS
2015	Yang e Sun (2015)	BSS-EV-LRP	SIGALNS
2015	Mokhtari (2015)	SSRAP	T-IG
2015	Quevedo-Orozco e Ríos-Mercado (2015)	CpVP	IGS + VND + SM
2015	Nucamendi et al. (2015)	UDCCVRP	IGS-CE, IG-RS
2015	Hatami et al. (2015)	DAPFSP-SDST	IGS+VNS
2015	Vincent e Lin (2015)	TD-PARP	IGS
2015	Marcelo et al. (2015)	NWFS-SDST	IGS
2017	Tasgetiren et al. (2017)	BFSP	IGS + RIS-LS + BS-LS
2017	Li et al. (2017)	TALBP	IIGS + NEH-RPW
2017	Arroyo e Leung (2017a)	SUPBM-NICURT	E-IG
2017	Pan et al. (2017b)	HFDW	IGS
2017	Dubois-Lacoste et al. (2017)	PFSP	IGS + FINS
2017	Ribas et al. (2017)	BFSP	IGS + NADA
2017	Pan et al. (2017a)	SDST/HFS	IGS

Flowshops with Due Windows (HFDW). Para sua resolução, são utilizados os algoritmos IGS e ILS. Os algoritmos foram testados em diversas instâncias e as variações propostas se mostraram eficientes em grande parte delas.

Dubois-Lacoste et al. (2017) aplicam IGS para o problema de alocação de tarefas (PFSP). O algoritmo parte de uma solução inicial construída pela heurística NEH. Depois, a solução é refinada através da destruição (retirada de tarefas) e reconstrução (inserção de tarefas), além da busca local *First-improvement Insertion Neighborhood Scan* (FINS). Os testes realizados mostraram o potencial do método em diversos contextos do problema. Já Ribas et al. (2017) resolvem o problema BFSP via IGS, tendo como objetivo reduzir o *makespans* das soluções. Nele, é utilizada a heurística NEH como ponto de partida e, no processo de busca, a solução passa primeiramente por uma busca local através do algoritmo *Non-exhaustive Descent Algorithm* (NADA) antes de serem aplicadas as fases de destruição e reconstrução, diferentemente da proposta original relatada por Ruiz e Stützle (2007). Os testes realizados utilizaram um grupo de instâncias de Kang et al. (1993) e apresentaram resultados satisfatórios, além de reportar algumas soluções melhores que as relatadas na literatura. Já Pan et al. (2017a) apresentam nove algoritmos para o problema *Hybrid Flowshop Scheduling Problem with Sequence-Dependent Setup Times* (SDST/HFS). Os autores discutem várias técnicas de trajetória, dentre elas

o IGS. Na comparação com outros métodos, o IGS se mostrou mais eficiente que muitas metaheurísticas tradicionais, além da simplicidade de implementação.

A Tabela 3.3 resume os principais trabalhos relacionados à aplicação do IGS para a solução de problemas combinatórios de diversas classes, nos termos discutidos nesta revisão bibliográfica.

Capítulo 4

Heurísticas Estudadas

Este capítulo mostra a fundamentação teórica que deu suporte na elaboração dos algoritmos desenvolvidos, descritos no Capítulo 5. A Seção 4.1 descreve a heurística PFIH. Já as Seções 4.2, 4.3 e 4.4 mostram embasamento teórico das metaheurísticas GRASP, VND e SA, respectivamente. Por último, a Seção 4.5 descreve a metaheurística IGS na forma proposta pela literatura.

4.1 *Push-Forward Insertion Heuristic* (PFIH)

Dentro dos estudos desenvolvidos para a solução do PRVJT, diversas técnicas foram criadas ao longo do tempo, desde sua formulação. Um dos pontos chave para a resolução do problema é a inserção dos clientes de forma a contemplar os objetivos principais, que são, a redução da distância percorrida e do número de veículos utilizados. A heurística *Push-Forward Insertion Heuristic* (PFIH) foi introduzida por Solomon (1987) e aplicada a diversos problemas da literatura que trabalham com janela de tempo, como relatados em Ho et al. (2001), Tam e Kwan (2004b), Tam e Kwan (2004a), Ghoseiri e Ghannadpour (2010a), Ghoseiri e Ghannadpour (2010b) e Ombuki et al. (2006), dentre outros.

A heurística construtiva *PFIH* foi elaborada primeiramente para o PRVJT e leva em conta informações importantes na montagem de uma solução factível. Consiste em calcular o custo de inserção de um determinado cliente, com o objetivo de reduzir o número de veículos e a distância total percorrida, levando em consideração a distância que o cliente está do depósito, o limite superior da janela de tempo e o ângulo polar entre o cliente e o depósito.

Assim, a função proposta pelo autor calcula o custo de inserção de todos os clientes da rota e os ordena de forma crescente. Posteriormente, esses clientes são testados em todas as posições possíveis das rotas já formadas e aquela posição que tiver o menor valor da função objetivo e que não viole nenhuma restrição do problema, é selecionada. Caso o cliente não encontre nenhuma posição factível, ele é inserido em uma nova rota.A sequência dos clientes é determinada pela expressão (4.1):

$$C_i = -\alpha \cdot d_{0i} + \beta \cdot l_i + \gamma \cdot ((\frac{p_i}{360})d_{0i}) \tag{4.1}$$

Na equação (4.1), C_i é o custo de inserção do cliente i, d_{0i} é a distância do cliente

i em relação ao depósito, l_i é o limite da janela de tempo e p_i é o ângulo polar em relação às coordenadas cartesianas. Assim, a função considera tanto a distância do cliente em relação ao depósito como a janela de tempo, obtendo uma boa estimativa na montagem das rotas e na inserção de clientes geograficamente mais próximos do depósito, contribuindo para a redução da distância total percorrida. Os parâmetros α, β e γ foram definidos empiricamente por Solomon (1987) como $\alpha = 0.7$; $\beta = 0.1$; e $\gamma = 0.2$. O Algoritmo 4.1 mostra o pseudocódigo.

Algoritmo 4.1: *Push-Forward Insertion Heuristic*

```
   Entrada: c, n
   Saída: s
1  cont = 0;
2  s ← ∅;
3  CalculaCustoPFIH(c, n);
4  OrdenaClientesPFIH(c, n);
5  enquanto cont < n faça
6  |  se cont == 0 então
7  |  |  s = criaRota(c);
8  |  fim
9  |  se verificaFactibilidade(c, s) == true então
10 |  |  s = insereSolucao(c, s);
11 |  |  senão
12 |  |  |  s = criaRota(c);
13 |  |  fim
14 |  fim
15 |  cont++;
16 fim
17 retorna s;
```

O algoritmo 4.1 exibe o pseudocódigo do algoritmo PFIH tradicional. Recebe como entrada as informações dos clientes c, tais como as coordenadas no plano, janela de tempo, tempo de serviço e demanda; e o número de clientes n. As linhas 1 e 2 inicializam as variáveis *cont* que irá controlar a quantidade de clientes já inseridos na solução e s que representa a solução a ser construída pelo método. As linhas 3 e 4 correspondem ao cálculo do custo de inserção demonstrado na equação 4.1 e, baseado neste custo, ordenar os clientes de forma crescente, ou seja, aqueles que tem custo menor serem inseridos primeiro na solução. As linhas de 5 a 16 vão percorrer todos os clientes e inseri-los na solução. As linhas de 6 à 8 criam uma rota e inserem o primeiro cliente. Já as linhas de 9 a 14 é responsável por percorrer a solução já formada e calcular o custo de inserção do cliente que está sendo testado e, caso encontre uma posição ele é inserido nela. Neste procedimento, o cliente é sempre inserido na melhor posição encontrada. Caso não encontre uma posição factível, é criada uma nova rota e este cliente é inserido nela. O algoritmo finaliza a sua execução quando todos os clientes forem inseridos na solução.

4.2 *Greedy Randomized Adaptive Search Procedure* (GRASP)

Para a resolução de problemas combinatórios, muitas vezes métodos exatos ou heurísticos não são suficientes para obter soluções de boa qualidade. A idéia de se aplicar metaheurísticas surgiu com o intuito de combinar duas ou mais heurísticas de forma a obter uma melhora na qualidade das soluções, dentro de um tempo computacional viável. Seguindo essa linha, Feo e Resende (1995), propuseram uma metaheurística denominada *Greedy Randomized Adaptive Search Procedure (GRASP)* . Este procedimento é dividido em duas fases distintas: a de construção e de busca local.

A fase de construção é uma etapa fundamental, pois é o ponto de partida na qual o algoritmo explorará o espaço de busca. Caso seja aleatório, pode ocasionar um direcionamento para soluções de baixa qualidade e, caso seja muito "guloso", o algoritmo pode ficar caro computacionalmente. Por isso, foram sugeridas diversas formas de se construir soluções. Segundo Resende e Silva (2013), a escolha correta de parâmetros permite que o algoritmo direcione para soluções de melhor qualidade. A construção semi-gulosa aplica o conceito de avaliação da qualidade do elemento selecionado, através de uma medida na qual mostre a contribuição da inserção dele em determinada posição. Já a técnica de amostragem gulosa mede a qualidade da solução através da função objetivo e mantém as posições melhores em um conjunto de amostragem, selecionando sempre o melhor elemento para inserção na solução. Já a construção aleatória-gulosa mantém uma lista de candidatos com os melhores elementos e escolhe de forma aleatória um elemento deste grupo, melhorando a diversidade das soluções geradas.

Uma outra proposta de construção foi abordada por Resende e Werneck (2004) denominada construção gulosa proporcional, na qual são avaliados os elementos de forma gulosa e o candidato é selecionado de forma proporcional, ou seja, aquele candidato que tem uma melhor avaliação tem maiores chances de ser escolhido. Já Prais e Ribeiro (2000) aplicam a técnica denominada GRASP reativo, em que a função de avaliação, na qual são selecionados os elementos que irão ser inseridos, controla o grau de aleatoriedade baseando-se na qualidade das soluções geradas. A equação 4.2 mostra a forma de cálculo da função utilizada.

$$F_a = C_{min} + \alpha(C_{max} - C_{min}) \tag{4.2}$$

Na equação 4.2, considere F_a a função de avaliação utilizada como critério para selecionar a lista de candidatos, C_{min} e C_{max} como sendo o maior e o menor custo de inserção, respectivamente e α indicando o maior ou menor grau de aleatoriedade. O valor de α é definido como um número real $\alpha \in [0, 1]$, sendo que, quanto menor for o valor de α, maior o grau de aleatoriedade.

Segundo Fleurent e Glover (1999), uma outra forma de se construir soluções é chamada de construção com memória de longo prazo. Neste procedimento, um grupo de soluções chamadas de "elite" é armazenada e esse conjunto pode ser atualizado sempre que o método encontre soluções de melhor qualidade, respeitando o critério de diversidade. Já Bresina (1996) utiliza um procedimento estocástico para determinar a escolha dos candidatos denominada construção por amostragem viciada.

Independente da técnica selecionada para o procedimento de construção, é realizada uma busca local. Ela é necessária, pois permite a diversificação das soluções geradas e proporciona sair de ótimos locais. Com isso, a metaheurística GRASP se torna muito eficiente e de fácil implementação, além de possuir características que permitem aplicá-la a diversos problemas. O algoritmo 4.2 mostra o pseudocódigo do algoritmo em questão.

Algoritmo 4.2: *Greedy Randomized Adaptive Search Procedure*

```
   Entrada: c,n
   Saída: s*
1  s ← ∅;
2  s' ← ∅;
3  s* ← ∅;
4  cont = 0;
5  enquanto cont < n faça
6  |  s ← construcao();
7  |  s' ← buscaLocal();
8  |  se s' < s* então
9  |  |  s* ← s';
10 |  fim
11 fim
12 retorna s*;
```

Assim, o algoritmo 4.2 passa por uma série de iterações em que são realizadas uma construção e uma busca local. Em cada iteração, a melhor solução é armazenada, fazendo com que ela não se perca durante a exploração do espaço de busca. Estes procedimentos são executados de forma iterativa, até um número máximo de iterações.

4.3 *Variable Neighborhood Descent* (VND)

A heurística *Variable Neighborhood Descent* (VND) foi apresentada por Mladenović e Hansen (1997) e aplicada a diversos problemas, como em Gao et al. (2008), Vanchipura et al. (2014), Tlili et al. (2016), Zeng et al. (2016) e Sifaleras e Konstantaras (2017). Sua principal característica está na capacidade de se adaptar a diversas abordagens e a possibilidade de combinações com outras heurísticas e metaheurísticas. Uma heurística de busca local, segundo Dorigo et al. (2006), tem por objetivo modificar uma solução de forma a caminhar no espaço de busca, com custo computacional reduzido. A definição da forma na qual elas serão aplicadas durante a busca é fundamental para obtenção de bons resultados. Além disso, outro fator importante está em levar em consideração características do problema em questão.

O VND, por sua vez, é uma heurística de busca local que explora o espaço de busca através de estruturas de vizinhanças definidas de forma iterativa. Cada modificação da solução deve ser analisada indicando se houve melhora ou piora no valor da função objetivo. Assim, em caso de piora ou manutenção do mesmo valor, a heurística modifica a estrutura de vizinhança aplicada. No caso do PRVJT, o

VND enfoca em dois objetivos principais, que são redução do número de veículos e a distância total percorrida. Portanto, a modificação da solução só é efetivada se todas as restrições forem atendidas e houver melhora na função objetivo. O Algoritmo 4.3 mostra o pseudocódigo do VND padrão.

Algoritmo 4.3: *Variable Neighborhood Descent*

```
Entrada: s, n
Saída: s*
cont = 0;
s* ← ∅;
enquanto cont < n faça
    s' ← movimento(cont);
    se s' < s então
        s = s';
        cont = 0;
        senão
            cont++;
        fim
    fim
    se s' < s* então
        s* = s';
    fim
fim
retorna s*;
```

Assim, no algoritmo 4.3, são definidas as estruturas de vizinhança estabelecida em *movimento*(*cont*), realizando aa modificação correspondente ao movimento atual, estabelecido pela variável *cont*. Com isso, caso a solução corrente mantenha ou piore o valor da função objetivo, a estrutura de vizinhança é modificada. Esse procedimento é realizado até que todos os movimentos definidos sejam executados. Uma outra característica do VND é armazenar a melhor solução, para que ela não se perca no processo de exploração.

4.4 *Simulated Annealing* (SA)

O método *simulated annealing* (SA) foi proposto por Kirkpatrick et al. (1983) e inspirado no processo industrial que retira as impurezas do metal. Desde então, esta técnica vem sendo estudado e aplicados a diversos problemas de otimização combinatória como relatados em Bhowmik e Repaka (2016) , Brusco et al. (2017) e Chen et al. (2017), além de trabalhos em variações do PRVJT reportadas em Xu et al. (2013), Despaux e Basterrech (2014), Küçükoğlu et al. (2015), Afifi et al. (2016), Kirci (2016), dentre outras. Neste trabalho, como uma das técnicas aplicadas utiliza o operador de temperatura, é importante destacar seu funcionamento na sua proposta original.

A técnica SA pode ser entendida como sendo um processo que simula o recozimento de metais na qual passa por diversos estados sensíveis à temperatura. Fazendo

uma comparação com um problema combinatório, os vários estados do metal podem se relacionar com as soluções do espaço de busca do problema, a energia de cada estado, ou seja, o seu grau de pureza, pode ser comparado ao valor da função objetivo e o conceito de energia mínima (ou máxima dependendo do problema) como um ótimo local ou global.

O processo de busca acontece de forma a considerar soluções de piora de acordo com a temperatura estabelecida. Portanto, no início do método, são consideradas temperaturas mais elevadas. Com isso, soluções de piora têm uma maior probabilidade de serem selecionadas. Assim, conforme as iterações do método forem sendo realizadas, a temperatura vai diminuindo gradativamente. Com isso, permite-se uma melhor exploração do espaço de busca no início do método e uma tendência mais forte de encontrar uma solução ótima ao final. O Algoritmo 4.4 exibe o pseudocódigo SA tradicional para um problema combinatório.

Algoritmo 4.4: *Simulated Annealing*

Entrada: $f(.)$, $N(.)$, α, SA_{max}, T_0,s
Saída: s

```
1  s* ← s;
2  IterT ← 0;
3  T ← T_0;
4  enquanto T > 0 faça
5      enquanto IterT < SA_max faça
6          IterT ← IterT + 1;
7          s' ← GeraVizinho(N(.));
8          △ ← f(s') − f(s) ;
9          se △ < 0 então
10             s ← s';
11             se f(s') < f(s*) então
12                 s* ← s';
13             fim
14             senão
15                 x ← [0, 1];
16                 se x < exp^{−△/T} então
17                     s ← s';
18                 fim
19             fim
20         fim
21     fim
22     T ← α * T;
23     InterT ← 0 ;
24 fim
25 s ← s*;
26 retorna s;
```

O algoritmo possui alguns parâmetros iniciais, tais como:

- SA_{Max} : Número de iterações máxima em uma determinada temperatura;

- α : Constante que irá determinar qual será o fator de decréscimo da temperatura;
- N(.) : Estruturas de vizinhança aplicadas a uma solução no decorrer da busca do melhor vizinho;
- T_0 : Temperatura inicial;
- s : Solução inicial;
- f(.) : Valor da função objetivo solução.

O procedimento inicia a partir de uma solução inicial s na qual é atribuída a melhor solução corrente $s*$ e, posteriormente, essa solução passa por perturbações. Nas linhas 4 a 24 são realizadas modificações na solução corrente e avaliada o valor da função objetivo. Caso $\triangle$ for inferior a zero, significa que a perturbação ocasionou uma melhora na função objetivo, comparada com a solução anterior, portanto s' passa a ser solução corrente, do contrário, o algoritmo irá atender ao critério de acordo com a temperatura, da seguinte forma:

- Sorteia-se um número aleatório x entre zero e um;
- Verifica-se o valor da temperatura $\exp^{-\triangle/T}$, caso este valor seja inferior ao valor x , a solução s' passa a ser a solução corrente
- Caso o valor x seja inferior a temperatura, a solução corrente permanece inalterada.

Na linha 22 o algoritmo atualiza o valor da temperatura corrente de acordo com o parâmetro α definido, sendo de fundamental importância para a aplicação correta do método. A escolha da temperatura inicial também se torna determinante, pois, caso for definida com valor muito pequeno pode não ser suficiente para explorar de forma correta o espaço de busca e, caso tenha uma temperatura muito alta, pode ocasionar um tempo computacional muito grade, uma vez que o critério de parada do método usa a temperatura T até valor mínimo inferior a zero.

4.5 *Iterated Greedy Search* (IGS)

A metaheurística *Iterated Greedy Search (IGS)* é estudada pela primeira vez por Rubén e Thomas (2007), e propõe uma solução para o *Flowshop Scheduling Problem (FSP)*. O IGS compõe-se de três fases distintas:

(i) Destruição: uma solução é desmontada retirando certo número de elementos;

(ii) Reconstrução: re-inserção dos elementos retirados da solução obedecendo algum critério;

(iii) Busca local: A solução perturbada é refinada, podendo ser combinado com uma heurística ou metaheurística.

Na fase de destruição, segundo Rubén e Thomas (2007), são selecionadas tarefas de forma aleatória, retiradas da solução e armazenadas em um vetor. Logo após, é realizada uma reconstrução, onde os elementos retirados são re-inserções na solução obedecendo às restrições do problema. Logo após, é executada uma busca local. Assim que as fases descritas são executadas, é verificada a qualidade da solução gerada. Caso o valor da função objetivo do problema tenha melhorado, o método permanece com a solução corrente e armazena a melhor solução; caso contrário, ele realiza uma análise, aplicando o operador de temperatura descrito na Seção 4.4. Este conceito varia muito de problema para problema e deve ser definido como parâmetro do método. Com este operador, é possível evitar que o algoritmo fique preso em ótimos locais, permitindo uma melhor exploração do espaço de busca, além de um possível ótimo local.

A metaheurística IGS, diferente de outras metaheurísticas, possui um conjunto reduzido de parâmetros, o que simplifica sua implementação. Assim, a quantidade de tarefas retiradas, a temperatura e o critério de parada do algoritmo devem ser definidos antes da aplicação do método.

Rubén e Stützle (2008) emprega a metaheurística IGS no problema de sequenciamento de tarefas em máquinas paralelas com tempo de preparação (*Sequence Dependent Setup Times Flowshop Problem* – SDST). Neste trabalho, os autores utilizam uma heurística construtiva e uma busca local e empregam como entrada do método. Para a fase de destruição, foi selecionado um número de tarefas a serem removidas da solução aleatoriamente. Logo após, na fase de reconstrução, estas tarefas foram re-inseridas na posição onde apresenta o melhor valor da função objetivo . A busca local foi realizada através de uma hibridização denominada *Descent Local Search* (DLS), na qual são realizadas perturbações através de estruturas de vizinhança, até que se obtenha uma melhora na função objetivo. Esta técnica apresentou bons resultados, contribuindo para o estado da arte do problema.

Diversos autores, como Luis e Rubén (2010) e Rodriguez et al. (2013b), também adotam a metaheurística IGS para os problemas de máquinas paralelas não relacionadas (*Parallel Machine Scheduling Problem* – PMSP) e apresentam hibridizações na técnica de busca local, com o objetivo de melhorar a exploração do espaço de busca.

Já Yu e Lin (2015) elaboram uma versão da metaheurística IGS para o problema de roteamento de veículos com coleta de prêmios e dependência do tempo (*Time-Dependent Prize-Collecting Arc Routing Problem* – TD-PARP). Neste problema, o objetivo é visitar os clientes e recolher os prêmios, obtido pela soma total de prêmios menos o custo de viagem. Para que a metaheurística IGS seja aplicado ao problema, o autor mostra uma série de adaptações. Uma delas é a inclusão de uma aleatoriedade no número de arcos retirados na fase de destruição, que pode variar entre α_{min} e α_{max}. A busca local é aplicada com estrutura de vizinhança de troca entre os arcos e uma variação que, quando encontra uma solução melhor que a corrente, o método retorna ao início e é executado novamente até um critério de parada. Outra adaptação está na adoção de uma temperatura que não é constante e se comporta pela variação do valor da função objetivo anterior pela corrente, dando um comportamento semelhante a metaheurística *Simulated Annealing* (SA), mostrando a capacidade de adaptação para outros problemas.

Nucamendi et al. (2015) aplica IGS no Problema de Roteamento de Veículos

Centrado no Cliente, com unidade de demanda (*Unit Demand Customer-Centric Vehicle Routing Problem* – UDCCVRP). Este problema tem, por objetivo, diminuir o tempo de espera total dos clientes e possui limitações de carga e tempo de viagem. Nesta versão, a criação de uma solução inicial e da busca local são realizadas por uma série de hibridizações e de estratégias de perturbações, sendo a estratégia *Iterated Greedy - Complete Evaluation (IG-CA)* a que obteve os melhores resultados. A versão IG-CA coloca, como construção inicial, uma avaliação dos clientes pela proximidade dos centros de distribuição, mostrando que uma boa solução inicial é fundamental para desempenho do algoritmo IGS nos mais diversos problemas. Há adaptações do IGS em variações do Problema do Caixeiro Viajante, como em Tasgetiren et al. (2013a) e Karabulut e Tasgetiren (2014b), mostrando a eficiência do algoritmo na solução de diversos problemas.

Algoritmo 4.5: *Iterated Greedy Search*

Entrada: $maxIt$,T
Saída: $s*$

```
1  cont ← 0;
2  s* ← ∅;
3  s' ← ∅;
4  s'' ← ∅;
5  s''' ← ∅;
6  s ← solucaoInicial();
7  enquanto cont < maxIt faça
8      s' ← destruicao();
9      s'' ← reconstrucao();
10     s''' ← buscaLocal();
11     △ = f(s''') − f(s) ;
12     se △ < 0 então
13         s ← s''';
14         cont ← 0;
15         senão
16             x ← [0, 1];
17             se x < exp^{−△/T} então
18                 s ← s''';
19             fim
20             cont + +;
21         fim
22     fim
23     se f(s) > f(s*) então
24         s* ← s;
25     fim
26 fim
27 retorna s*;
```

O Algoritmo 4.5 mostra a metaheurística IGS. O algoritmo recebe como entrada $maxIt$, que representa o número de iterações sem melhora e T que estabelece a

temperatura utilizada no critério de aceitação de solução. As linhas de 1 a 6 inicializam as principais variáveis e a solução inicial s. As linhas 7 à 26 mostra as interações do método. As linhas 8 à 10 realizam os principais passos do método, que são a destruição, reconstrução e busca local. Já as linha 11 à 22 mostram o critério de aceitação da solução, que faz uma diferença entre o valor da função objetivo da solução anterior e a solução perturbada e, caso ela seja negativa, ou seja, ouve uma diminuição (no caso de minimização), ela passa a ser a solução correta; caso contrário, é sorteado um valor x de forma aleatória. Caso o valor seja menor que a expressão $exp^{-\triangle/T}$ a solução de piora é admitida, caso contrário, a solução corrente não é modificada.

Capítulo 5

Algoritmos Desenvolvidos

Neste capítulo, são apresentados os algoritmos desenvolvidos para a solução do Problema de Roteamento de Veículos com Janela de Tempo (PRVJT). A Seção 5.1 mostra como é representada uma solução computacionalmente. A Seção 5.2 apresenta as estruturas de vizinhanças elaboradas para o problema. A Seção 5.3 explica a função de avaliação adotada no trabalho e, por fim, a Seção 5.4 descreve os algoritmos desenvolvidos.

5.1 Representação da Solução

Para representar uma solução para o Problema de Roteamento de Veículos com Janela de Tempo (PRVJT), a informação é armazenada em uma matriz, em que cada linha representa uma rota da solução. Assim, cada posição da matriz possui um número que identifica o cliente que será visitado pelo veículo. Cada rota formada inicia e finaliza no depósito. A Figura 5.1 apresenta uma solução com os clientes $C = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$, utilizando quatro veículos.

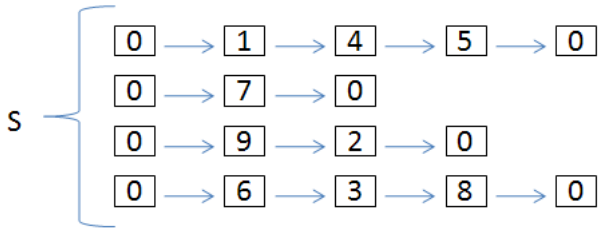

Figura 5.1: Solução de um exemplo fictício para o PRVJT.

Na Figura 5.1, cada linha apresenta a sequência de clientes que serão visitadas pelos veículos. O conjunto $S = \{\{0, 1, 4, 5, 0\}, \{0, 7, 0\}, \{0, 9, 2, 0\}, \{0, 6, 3, 8, 0\}\}$ mostra a ordem de visitação dos veículos, sendo que cada elemento de S exibe uma rota que parte e retorna para o depósito.

5.2 Estruturas de Vizinhança

As estruturas de vizinhança são definidas para determinado problema e podem variar de acordo com as características do mesmo. No caso do PRVJT, o espaço de

soluções é composto de modificações da ordem de visitação dos clientes dentro das rotas formadas. Esse procedimento faz com que as possibilidades de permutações entre eles cresçam exponencialmente. Por isso, as estruturas de vizinhança devem ser definidas de forma que o tempo computacional seja reduzido e, ao mesmo tempo, gere soluções factíveis e de boa qualidade.

Para explorar o espaço de busca do PRVJT são realizados movimentos de realocação e troca, sempre com o objetivo de reduzir o número de rotas e a distância total percorrida. São desenvolvidos 12 diferentes abordagens para explorar o espaço de busca. São estruturas de vizinhanças que modificam a ordem de visitação dos clientes dentro de uma mesma rota, denominados intra-rotas, e movimentos que modificam clientes localizados em rotas diferentes, denominados inter-rotas. As estruturas de vizinhança intra-rota têm, como objetivo, reduzir a distância total percorrida, encontrando posições dos clientes mais próximos e organizando melhor a viagem realizada pelo veículo. Já as estruturas denominadas de inter-rotas têm por objetivo reduzir tanto a distância quanto o número de rotas. A redução do número de rotas acontece pelo fato de, dependendo do número de clientes pertencente a uma determinada rota for realocada, essa rota pode ser eliminada, contribuindo para a redução do número de veículos necessários para atender a um conjunto de clientes.

Os movimentos intra-rotas implementados são os denominados *Exchange(k)* e *Shift(k)* e os movimentos inter-rotas são os do tipo *Shift(k,0)* e *Swap(k,l)*. Utiliza-se uma solução $S = \{\{0,1,4,15,6,20,0\},\{0,9,14,19,0\},\ \{0,2,8,5,17,7,12,18,0\},\ \{0,13,11,0\},\ \{0,10,3,16,0\}\}$ fictícia em todos os exemplos retratados nesta seção, a fim de mostrar o funcionamento de cada estrutura separadamente.

5.2.1 Estruturas de vizinhança *Shift(k)*

As estruturas de vizinhança *Shift(k)* são realizadas de forma a modificar uma determinada solução, alterando a ordem de visitação de apenas uma rota por vez. Sendo assim, o movimento é executado de forma a selecionar uma rota qualquer aleatoriamente e selecionar um, dois ou três clientes adjacentes e realizar a retirada desses clientes. Logo após esses procedimento, é aplicado um sorteio aleatório de outra posição na mesma rota de maneira que, os clientes retirados na primeira etapa são re-inseridos nesta posição. Caso a nova posição dos clientes atender a todas as restrições importas pelo problema e melhorar o valor da função objetivo, o movimento é aplicado, caso contrário, a solução não é modificada.

A Figura 5.2 retrata uma solução fictícia no qual é selecionada a rota $r_1 = \{0,1,4,15,6,20,0\}$ e a posição 2. Logo após, o cliente correspondente a posição 2, que no caso é o representado pelo número 4 é retirado da rota, modificado a rota $r_{1'} = \{0,1,15,6,20,0\}$. Posteriormente, é sorteado a posição 4, selecionada aleatoriamente, e re-inserida o cliente 4, respeitando as restrições do problema, na posição 4, obtendo assim a rota $r_{1''} = \{0,1,15,6,20,4,0\}$.

Já a Figura 5.3 seleciona a rota $r_3 = \{0,2,8,5,17,7,12,18,0\}$ e as posições 2 e 3. Desta forma, são retirados os clientes 8 e 5, obtendo a rota $r_{3'} = \{0,2,17,7,12,18,0\}$. O mesmo procedimento é realizado, selecionado a posição 4 , onde os clientes 8 e 5 são inseridos mantendo a ordem de visitação, modificando a rota $r_{3''} = \{0,2,17,7,8,5,12,18,0\}$.

A Figura 5.4 retira os clientes 15, 6 e 20 da rota $r_1 = \{0,1,4,15,6,20,0\}$, cor-

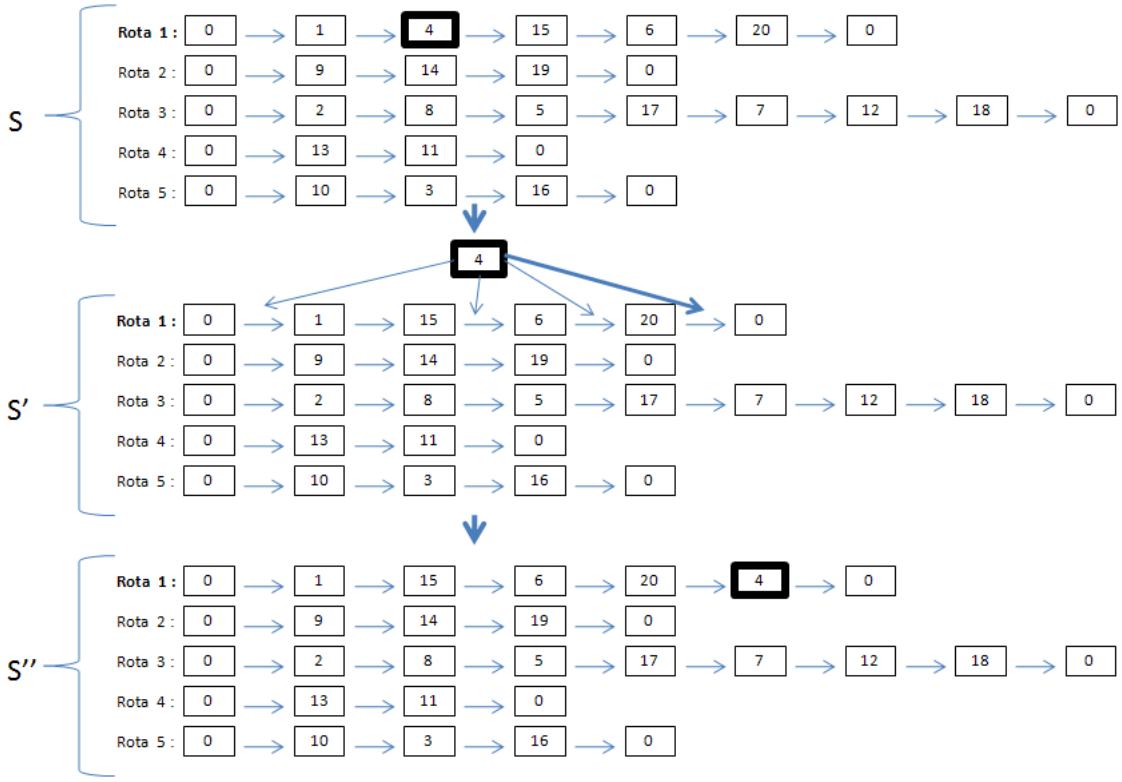

Figura 5.2: Exemplo de estrutura vizinhança shift(1).

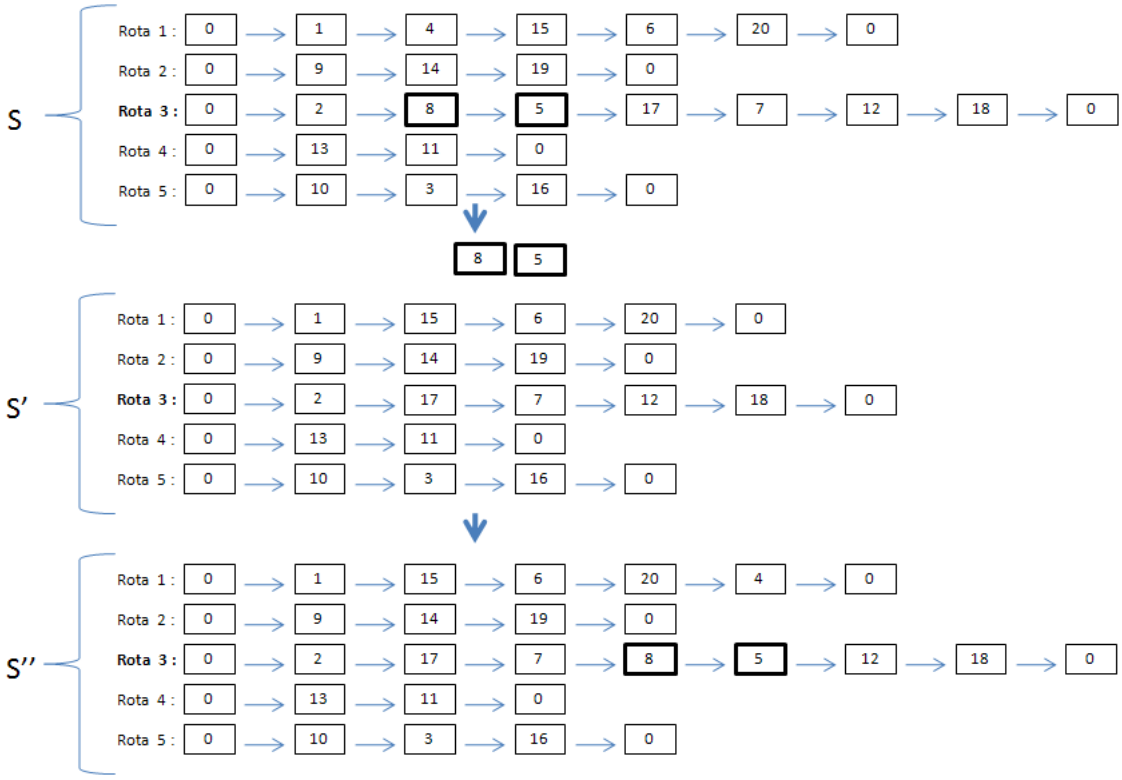

Figura 5.3: Exemplo de estrutura vizinhança shift(2).

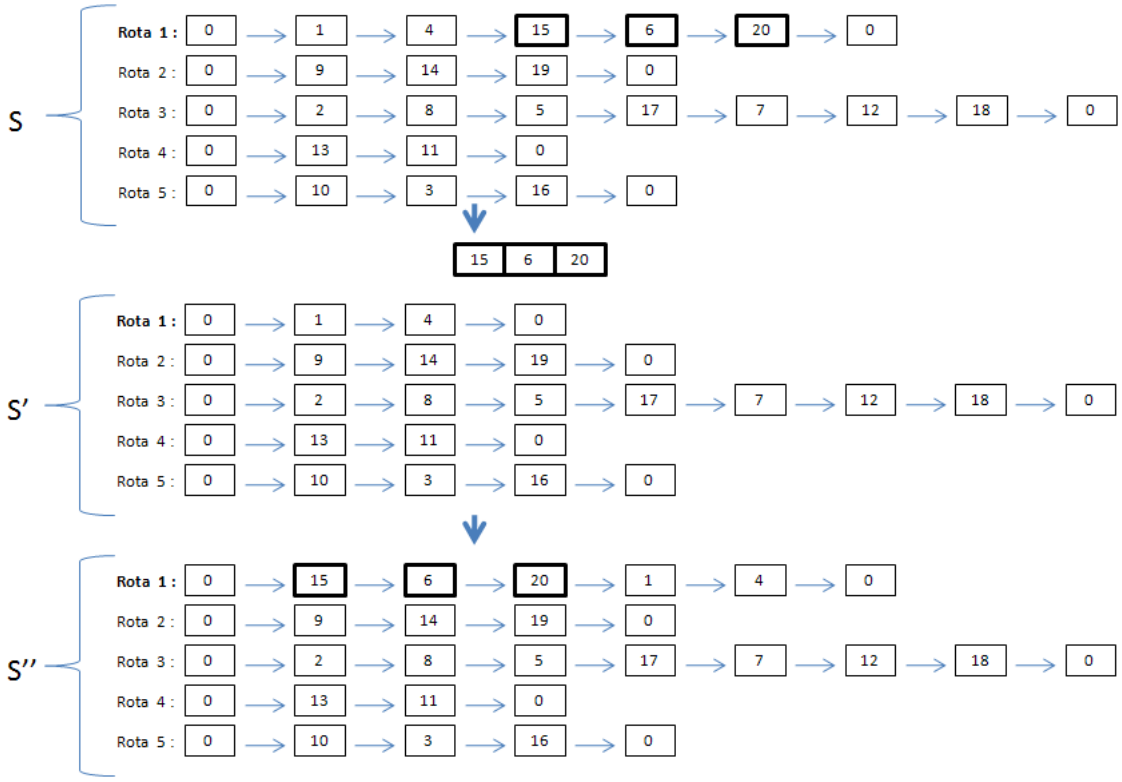

Figura 5.4: Exemplo de estrutura vizinhança shift(3).

respondentes as posições 3, 4 e 5, respectivamente, obtendo a rota $r_{1'} = \{0, 1, 4, 0\}$. Os clientes são re-inseridos no início da rota, na posição 0 de forma sequencial, modificando a rota $r_{1''} = \{0, 15, 6, 20, 1, 4, 0\}$.

5.2.2 Estruturas de vizinhança *Exchange(k)*

As estruturas de vizinhança *Exchange(k)* também parte do princípio de modificação da ordem de visitação dos clientes em uma única rota. É sorteado uma rota, sendo que todas as rotas que compõe a solução tem chances iguais de serem escolhidas e, dependendo da variação do método, são selecionados 2, 4 ou 6 posições desta rota e realizadas trocas entre os clientes. Da mesma forma, são verificadas as restrições de janela de tempo e distância percorrida. Somente se aa modificações sejam factíveis e houver uma redução no valor da distância percorrida, a estrutura de vizinhança é aplicada.

A Figura 5.5 mostra um exemplo em que a rota $r_5 = \{0, 10, 3, 16, 0\}$ é selecionada e as posições 1 e 2 sorteadas aleatoriamente. Assim, a ordem de visitação dos clientes 10 e 3 são trocados, repeitado as restrições, obtendo a rota $r_{5'} = \{0, 3, 10, 16, 0\}$. O movimento só é executado caso a distância total percorrida for menor.

Já na Figura 5.6 são sorteados as posições (1 e 2) e (3 e 5) respectivamente, da rota $r_1 = \{0, 1, 4, 15, 6, 20, 0\}$ e realizada a troca da ordem de visitação entre os clientes (1 e 4) e (15 e 20). Assim, obtém-se a rota $r_{1'} = \{0, 4, 1, 20, 6, 15, 0\}$, que será efetivamente realizada se satisfazer as restrições do problema e melhorar o valor da função objetivo.

A Figura 5.7 é selecionado a rota $r_3 = \{0, 2, 8, 5, 17, 7, 12, 18, 0\}$ e as posições (1 e

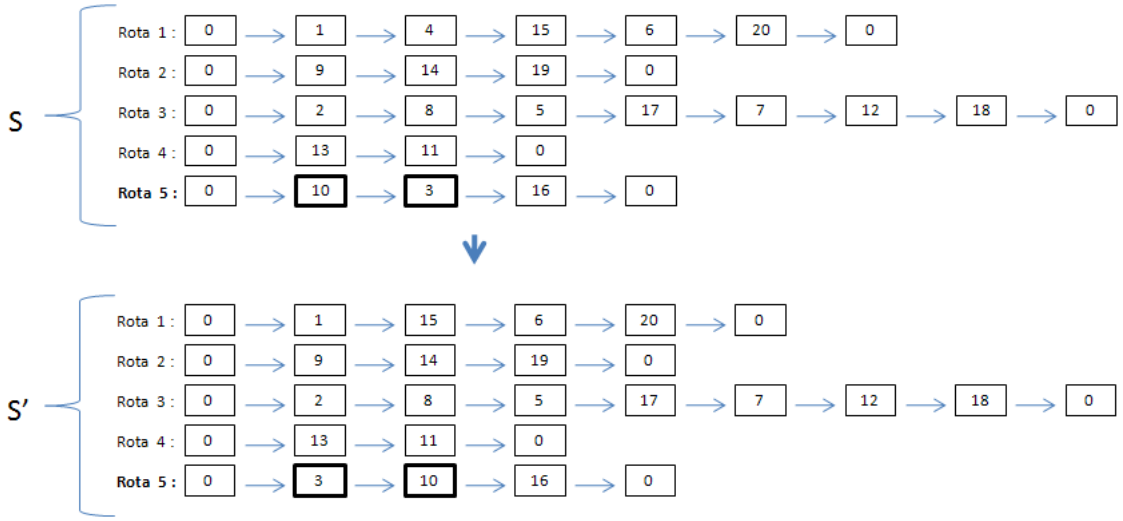

Figura 5.5: Exemplo de estrutura vizinhança Exchange(1)

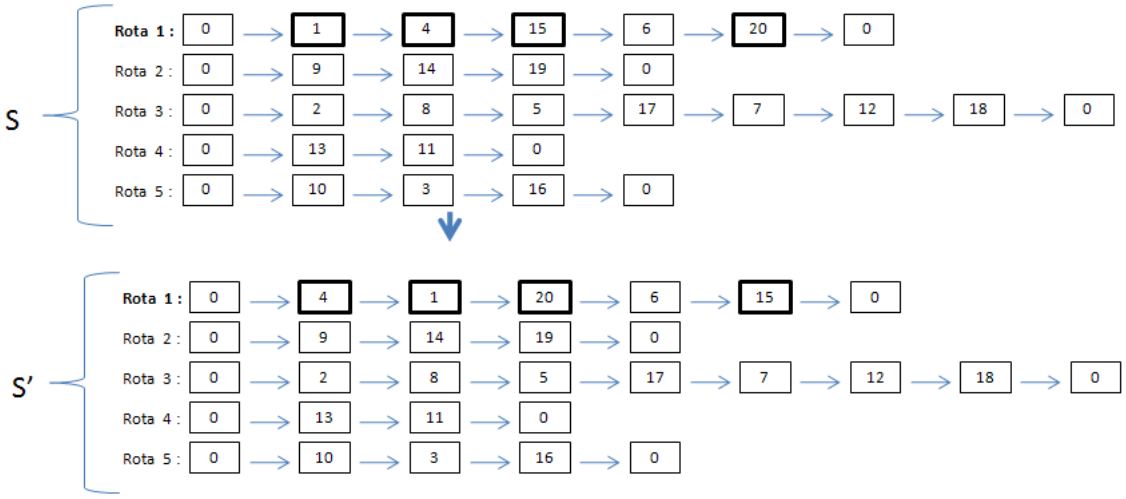

Figura 5.6: Exemplo de estrutura vizinhança Exchange(2)

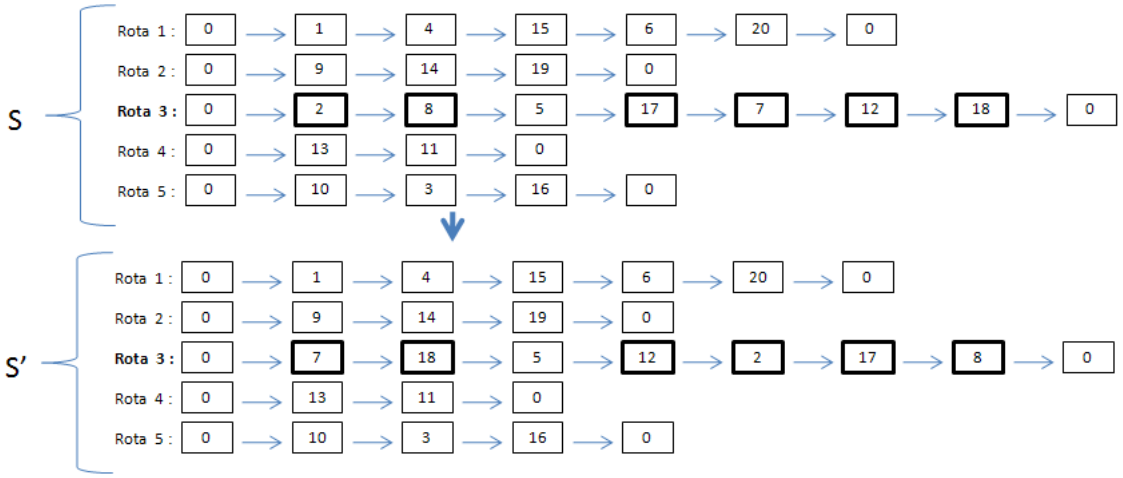

Figura 5.7: Exemplo de estrutura vizinhança Exchange(3)

5), (2 e 7) e (4 e 6). As trocas são realizadas entre os clientes 2, 8 e 17 pelos clientes 7, 18 e 12 respectivamente, modificando a rota $r_{3'} = \{0, 7, 18, 5, 12, 2, 17, 8, 0\}$.

5.2.3 Estruturas de vizinhança *Shift(k,0)*

As estruturas de vizinhança que modificam uma solução denominados como *Shift(k,0)*, são chamados de estruturas de vizinhança inter-rotas, ou seja, provocam uma modificação na solução relacionando duas rotas distintas. Cada variação do movimento relacionam sempre duas rotas e consistem em retirar determinado número de clientes adjacentes e re-inseri-los em outra rota, podendo varia em 1, 2 ou 3 clientes. Este movimento só é efetivado se as restrições de janela de tempo, carga e retorno ao depósito sejam satisfeitas, além de ter uma melhoria na função objetivo do problema. Este tipo de estrutura de vizinhança também auxilia na redução do número de rotas, uma vez que, caso seja sorteado uma rota com 1,2 ou 3 clientes e for encontrada uma posição factível em outra rota, tem a possibilidade dela ser eliminada.

Na variação *Shift(1,0)*, são escolhidos aleatoriamente uma rota e uma posição. O cliente que corresponde à posição selecionada é removido da rota. Posteriormente, é sorteada outra rota e posição, na qual é realizada a inserção do cliente retirado anteriormente. Caso as restrições de janela de tempo, carga e o valor da função objetivo forem melhores que a solução anterior, o movimento é feito; caso contrário, a solução não é afetada.

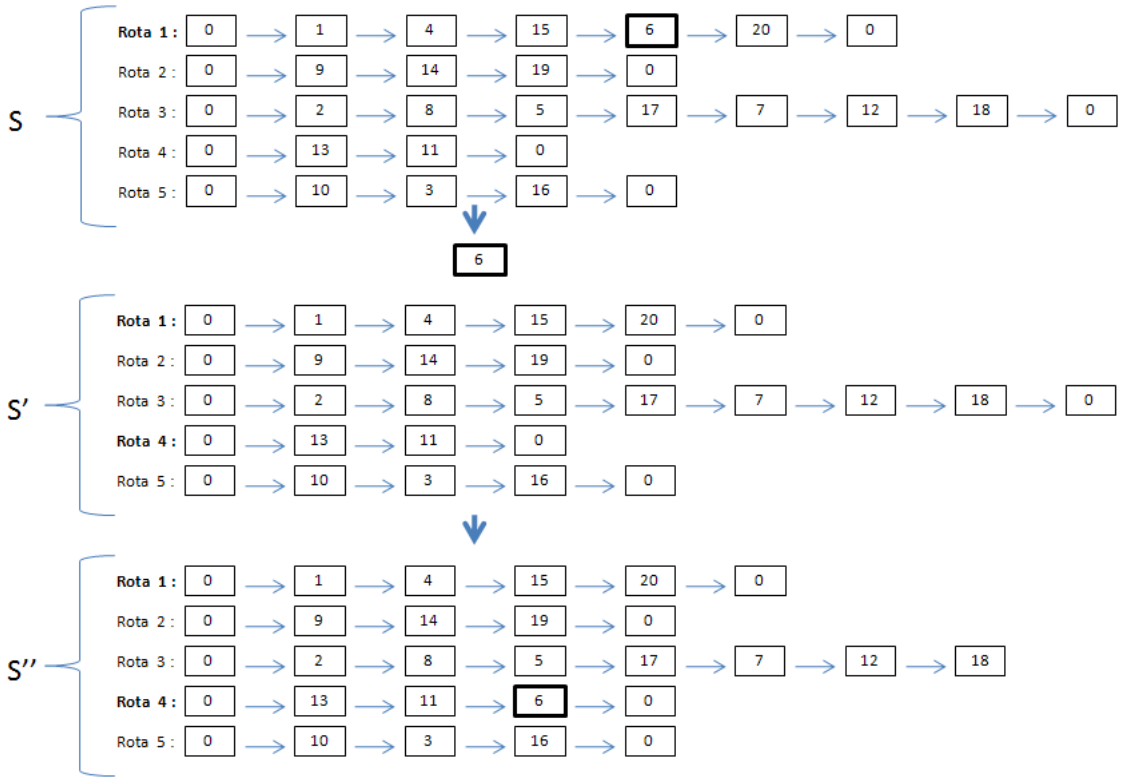

Figura 5.8: Exemplo de estrutura vizinhança Shift(1,0).

A Figura 5.8 são sorteadas as rotas $r_1 = \{0, 1, 4, 15, 6, 20, 0\}$ e $r_4 = \{0, 13, 11, 0\}$, para realizar o movimento. Seleciona-se a posição 4 da rota r_1 e retira-se o cliente 6

da rota, obtendo $r_{1'} = \{0, 1, 4, 15, 20, 0\}$. Depois, é sorteado a posição 2 da rota r_4 e é introduzido o cliente 6 na posição escolhida aleatoriamente, modificado a rota $r_{4'} = \{0, 13, 11, 6, 0\}$. A estrutura de vizinhança é aplicada somente se for factível e reduzir ou o número de rotas ou a distância total percorrida.

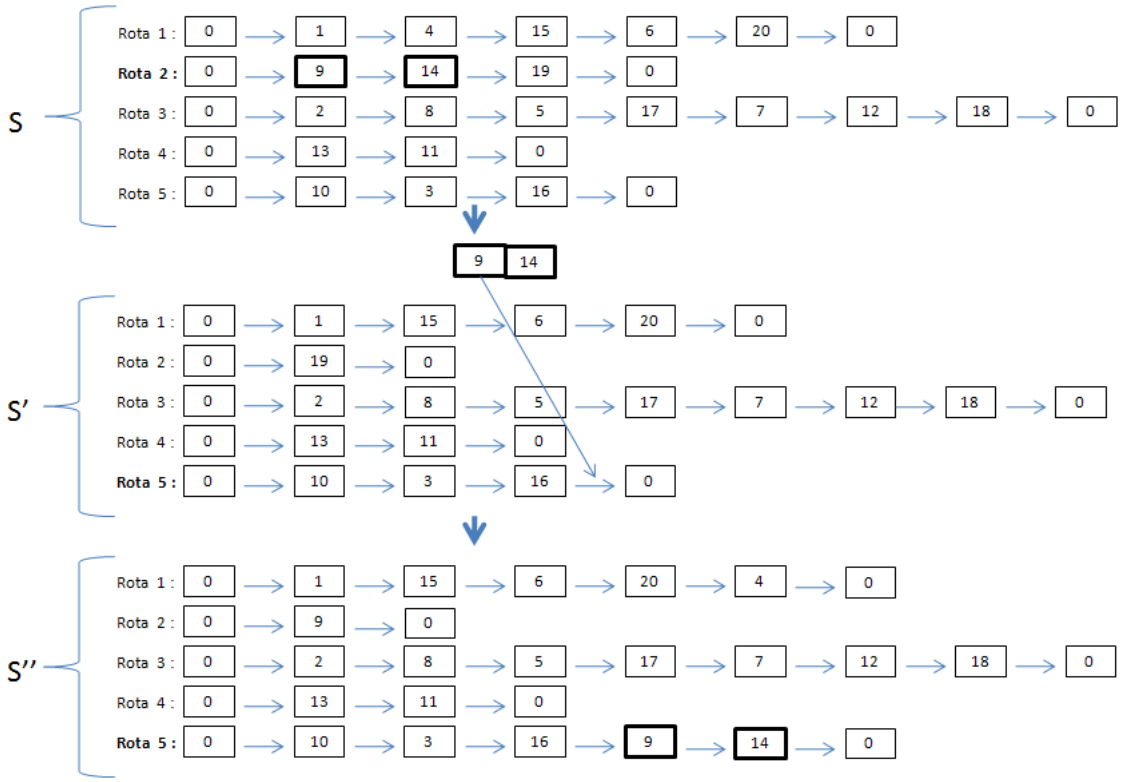

Figura 5.9: Exemplo de estrutura vizinhança Shift(2,0).

A Figura 5.9 mostra um exemplo de modificação da solução selecionando aleatoriamente as rotas $r_2 = \{0, 9, 14, 19, 0\}$ e $r_5 = \{0, 10, 3, 16, 0\}$. Posteriormente, são sorteadas as posições 0 e 1 da rota r_2, retirando os clientes 9 e 14. Assim, tem-se a rota modificada $r_{2'} = \{0, 19, 0\}$. A partir desta modificação, é escolhida a posição 3 da rota r_5 e inserido os clientes 9 e 14, respeitando a ordem de visitação, modificando a rota $r_{5'} = \{0, 10, 3, 16, 9, 14, 0\}$.

Já a Figura 5.10 escolhe as rotas $r_1 = \{0, 1, 4, 15, 6, 20, 0\}$ e $r_4 = \{0, 13, 11, 0\}$. Primeiramente, são sorteados as posição 3, 4 e 5 da rota r_1 e retirados os clientes 15, 6 e 20, obtendo a rota $r_{1'} = \{0, 1, 4, 0\}$. Depois é selecionada a posição 2 da rota r_4 e inseridos os clientes retirados da rota r_1, modificando a rota $r_{4'} = \{0, 13, 11, 15, 6, 20, 0\}$.

5.2.4 Estruturas de vizinhança *Swap(k,l)*

As estruturas de vizinhança *Swap(k,l)* são chamadas de inter-rotas, ou seja, envolvem mais de uma rota, e possuem a capacidade de realizar trocas entre clientes com o objetivo de reduzir a distância total percorrida. Neste movimento, são sorteados aleatoriamente duas rotas distintas. Depois, dependendo da variação do método, são sorteados 1 ou 2 posições em cada rota e realizada uma troca entre estes clientes, verificando a factibilidade. Caso o movimento consistir em uma modificação

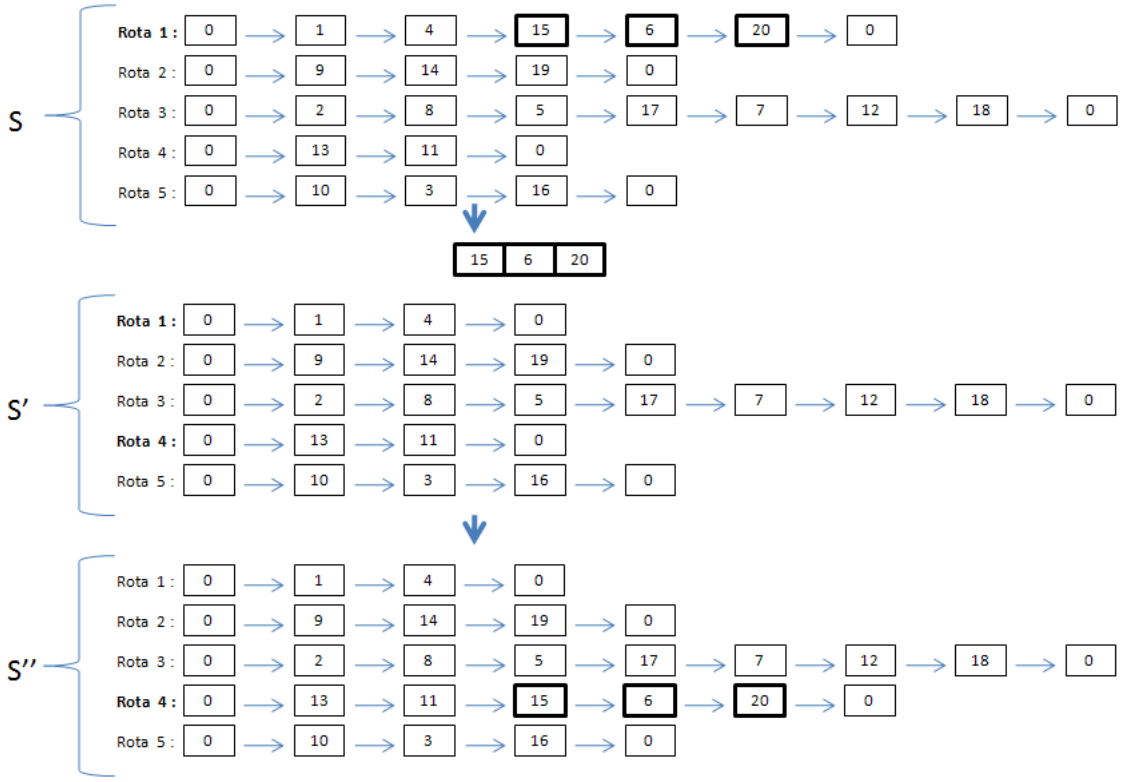

Figura 5.10: Exemplo de estrutura vizinhança Shift(3,0).

que reduz a distância percorrida, ele é efetivado, caso contrário, a solução segue sem alteração.

O movimento *Swap(1,1)* seleciona duas rotas e uma posição em cada rota de forma aleatória. Depois, os clientes das respectivas posições são trocados e verificadas as restrições de janela de tempo e carga. Caso seja factível, o movimento só é realizado se houver uma diminuição no valor da função objetivo.

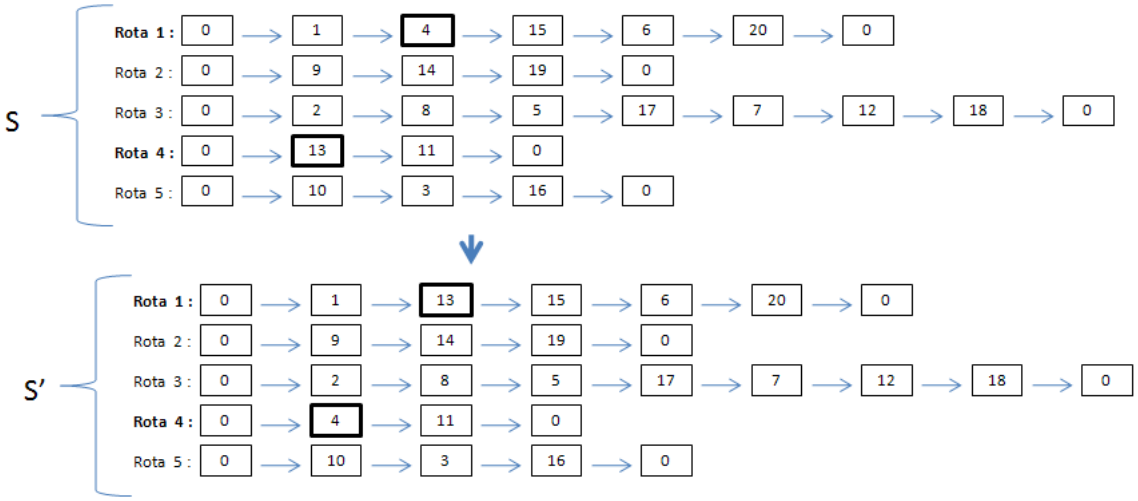

Figura 5.11: Exemplo de estrutura vizinhança Swap(1,1).

A Figura 5.11 exemplifica o movimento, selecionado as rotas $r_1 = \{0, 1, 4, 15, 6, 20, 0\}$ e $r_4 = \{0, 13, 11, 0\}$. Depois, são escolhidas as posições 1, pertencente a rota r_1 e a

posição 0 da rota r_4 e os clientes 4 e 13 são trocados de posição, obtendo as rotas modificadas $r_{1'} = \{0, 1, 13, 15, 6, 20, 0\}$ e $r_{4'} = \{0, 4, 11, 0\}$. Assim, o movimento só é aplicado se forem atendidas as restrições do problema e houver uma redução da distância total percorrida.

O movimento *Swap(2,1)* seleciona uma rota e dois clientes dessa rota, conservando a ordem de visitação. Depois, é selecionada outra rota e um cliente pertencente àquela rota, efetuando a troca. Caso o movimento seja permitido e a distância percorrida diminua, ele é executado; caso contrário, a solução não é alterada.

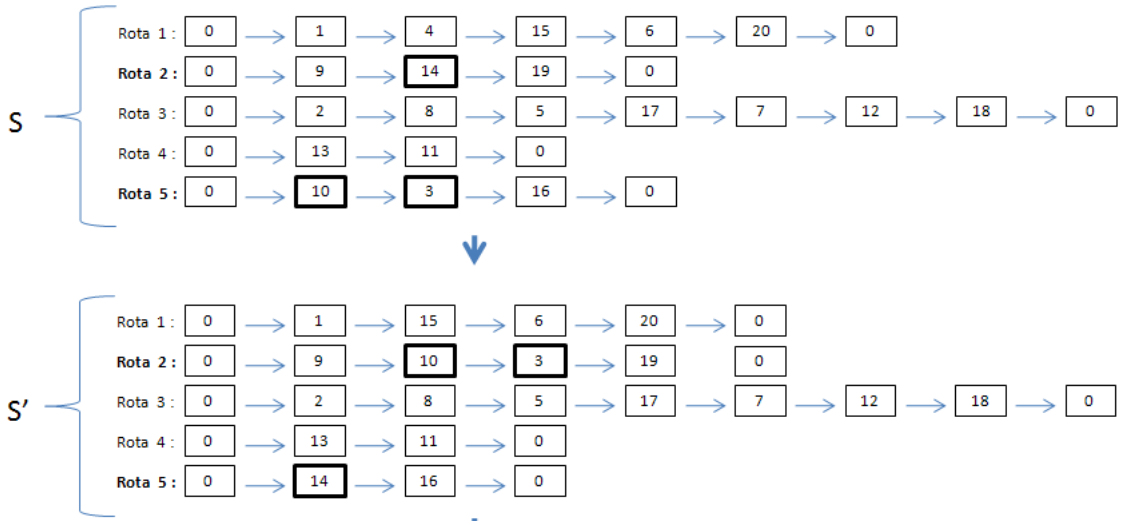

Figura 5.12: Exemplo de estrutura vizinhança Swap(2,1).

A Figura 5.12 mostra a seleção das rotas $r_2 = \{0, 9, 14, 19, 0\}$ e $r_5 = \{0, 10, 3, 16, 0\}$ e das posições 1 e 2 da rota r_2 e a posição 0 da rota r_5. Assim, os clientes 10 e 3 da rota r_5 são trocados pelo cliente 14 da rota r_2, modificando as rotas $r_2 = \{0, 9, 10, 3, 19, 0\}$ e $r_5 = \{0, 14, 16, 0\}$.

O movimento *Swap(2,2)*, por sua vez, realiza os mesmos procedimentos do *Swap(2,1)*, porém, na seleção da rota para fazer a troca, são escolhidos dois clientes adjacentes. Desse modo, os clientes são trocados e, caso seja um movimento de melhora, ele é executado.

A Figura 5.13 seleciona as rotas $r_3 = \{0, 2, 8, 5, 17, 7, 12, 18, 0\}$ e $r_5 = \{0, 10, 3, 16, 0\}$ e as posições 4 e 5 da rota r_3 e as posições 0 e 1 da rota r_5, fazendo a troca entres os clientes 7 e 12 da rota r_3 e os clientes 10 e 3 da rota r_5. Obtém-se as rotas modificadas $r_{3'} = \{0, 2, 8, 5, 17, 10, 3, 18, 0\}$ e $r_{5'} = \{0, 7, 12, 16, 0\}$.

5.3 Função de Avaliação

A função de avaliação é baseada na proposta de Ombuki et al. (2006), no qual o número de rotas e a distância total percorrida são consideradas, segundo a equação 5.1 mostrada a seguir:

$$FA = \lambda N + \delta D \tag{5.1}$$

A equação 5.1 é constituída de uma soma ponderada, em que D é a distância percorrida e N é o número de rotas formadas. Estas duas grandezas são somadas

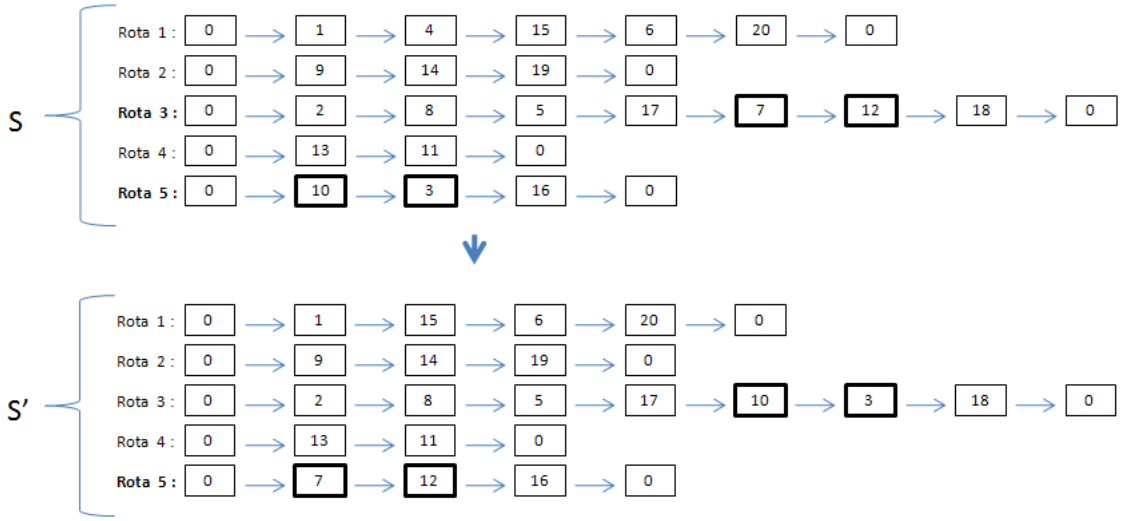

Figura 5.13: Exemplo de estrutura vizinhança Swap(2,2).

através de uma ponderação λ e δ, na qual são recomendados pelos autores os valores $\lambda = 100$ e $\delta = 0.001$. Com isso, como o problema tem objetivos hierárquicos, nos quais o número de veículos utilizados tem maior peso na solução do que a distância percorrida, atribui-se um valor bem superior à solução com menor número de rotas em detrimento de soluções com menor distância percorrida, porém com maior número de rotas.

5.4 Algoritmos Propostos

5.4.1 VND

Com o objetivo de explorar melhor o espaço de busca, a metaheurística VND (*Variable Neighborhood Descent*) é aplicada em diversos problemas da literatura como sendo uma boa alternativa para uma busca local. O VND consiste em, partindo de uma solução completa, executar estruturas de vizinhança a fim de encontrar o melhor vizinho, ou seja, são admitidas apenas modificações que irão melhorar o valor da função objetivo. Assim, sempre que um determinado movimento tiver encontrado uma solução de melhora, o método retorna a primeira estrutura de vizinhança, fazendo com que se caminhe no espaço de busca em direção a um ótimo local ou global. O processo finaliza quando todas as estruturas de vizinhanças definidas são percorridas. O algoritmo 5.1 mostra a ordem de execução de cada movimento, definida empiricamente.

O algoritmo 5.1 é um procedimento executado dentro do método VND 5.2 e recebe como parâmetros uma solução s e uma variável *cont* que representa qual a estrutura de vizinhança será executada. Todas as modificações da solução descritas na Seção 5.2 são utilizadas neste algoritmo e são aplicadas na ordem:

- Movimentos de modificação intra-rota: *Exchange(1)* e as variações *Exchange(2)*, *Exchange(3)*;
- Movimentos intra-rota: *Shift(3)*, *Shift(2)* e *Shift(1)*;

Algoritmo 5.1: MovimentosVND2(s,$cont$)

Entrada: s, $cont$
Saída: s'

```
1  se cont == 0 então
2  |  s' ← Exchange(1);
3  fim
4  se cont == 1 então
5  |  s' ← Exchange(2);
6  fim
7  se cont == 2 então
8  |  s' ← Exchange(3);
9  fim
10 se cont == 3 então
11 |  s' ← Shift(3);
12 fim
13 se cont == 4 então
14 |  s' ← Shift(2);
15 fim
16 se cont == 5 então
17 |  s' ← Shift(1);
18 fim
19 se cont == 6 então
20 |  s' ← Shift(1,0);
21 fim
22 se cont == 7 então
23 |  s' ← Shift(2,0);
24 fim
25 se cont == 8 então
26 |  s' ← Shift(3,0);
27 fim
28 se cont == 9 então
29 |  s' ← Swap(1,1);
30 fim
31 se cont == 10 então
32 |  s' ← Swap(2,1);
33 fim
34 se cont == 11 então
35 |  s' ← Swap(2,2);
36 fim
37 retorna s';
```

- Movimentos inter-rotas: *Shift(1,0)*, *Shift(2,0)* e *Shift(3,0)* e;
- Movimentos inter-rotas: *Swap(1,1)*, *Swap(2,1)* e *Swap(2,2)*

Algoritmo 5.2: buscaLocalVND

```
   Entrada: s, r
   Saída: s*
1  cont ← 0;
2  s* ← ∅;
3  enquanto cont < r faça
4  |  s' ← Movimento(s, cont);
5  |  se s' < s então
6  |  |  s ← s';
7  |  |  cont ← 0;
8  |  fim
9  |  se s' < s* então
10 |  |  s* ← s';
11 |  fim
12 |  se s' >= s então
13 |  |  cont + +;
14 |  fim
15 fim
16 retorna s*;
```

Já o algoritmo 5.2 recebe como entrada uma solução s e um número de estruturas de vizinhança r. Nas linhas 1 e 2 são inicializados as variáveis *cont* e $s*$, que representam, respectivamente, a estrutura de vizinhança corrente e a solução com melhor valor da função objetivo. Na linha 4, são executadas as estruturas de vizinhanças descritas em 5.1, a fim de modificar a solução corrente. Nas linhas 5 a 8 são comparados os valores das funções de avaliação de s', representando a solução modificada, e s representando a solução corrente. Caso houver uma melhora, há um retorno para a primeira estrutura de vizinhança, reiniciando a exploração,atribuindo a variável *cont* o valor 0. Já nas linhas 9 a 11 é verificado se a solução s' apresenta melhores resultados que a melhor solução encontrada até o momento. Assim, o algoritmo consegue armazenar a melhor solução durante a exploração do espaço de busca. Por fim, as linhas 12 a 14 verificam se s' é pior ou igual a solução anterior. Caso afirmativo, a estrutura de vizinhança representada por *cont* é modificada.

5.4.2 GRASP+PFIH+VND

A metaheurística GRASP é considerado um procedimento de busca adaptativa gulosa e randômica, na qual uma função adaptativa (FA) é utilizada para controlar o grau de aleatoriedade do método. Ele consiste, basicamente, de duas fases distintas: construção e bisca local. A fase de construção é realizada inserindo elementos na solução de forma a não gerar infactibilidades; já a fase de busca local permite que

seja aplicadas estruturas de vizinhanças para melhor explorar o espaço de busca. A solução que obteve o melhor valor da função objetivo é retornado.

Para a fase de construção do GRASP, foi adaptada a heurística PFIH proposta por Solomon (1987), na qual é realizada uma avaliação do custo de inserção descrita na equação 4.1. Desta forma, os clientes são ordenados de acordo com o valor deste custo. Após a ordenação dos clientes de forma crescente, o primeiro cliente é inserido na solução. Em seguida, os clientes restantes passam por uma função adaptativa descrita na expressão 5.1, que serve como critério para a montagem de uma lista restrita de candidatos. A função 5.1 é fundamental, pois o valor do parâmetro α garante o grau de aleatoriedade do método.

Após o cálculo da função adaptativa, os clientes que tiverem o custo menor ou igual ao valor da função adaptativa são selecionados para uma lista restrita de candidatos. O tamanho dessa lista é um parâmetro importante, pois, dependendo dele, as soluções formadas podem ser mais aleatórias ou mais gulosas. Com a lista de candidatos restrita formada, é selecionado um cliente aleatoriamente e inserido na solução. A Figura 5.14 ilustra a forma de construção de uma solução fictícia.

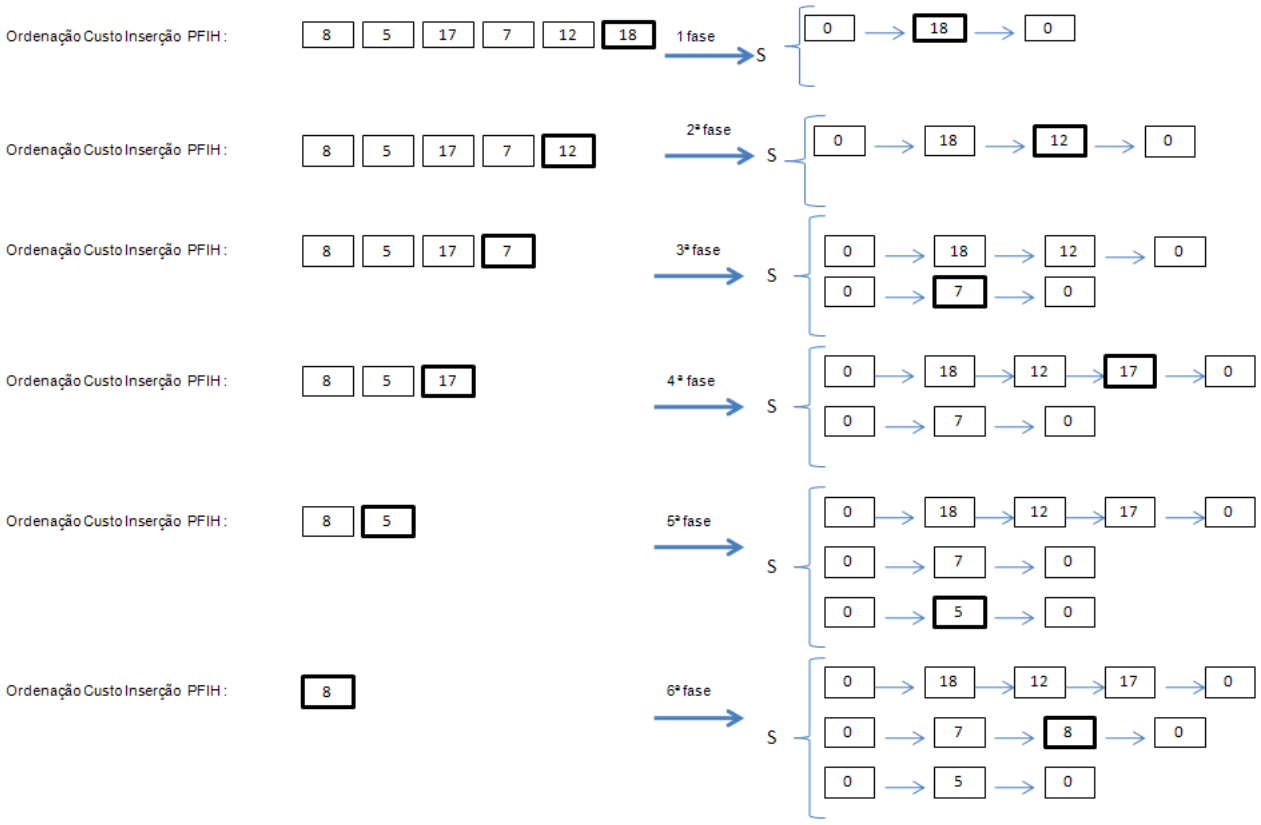

Figura 5.14: Construção solução fictícia GRASP+PFIH.

Na Figura 5.14 é relatada de forma resumida a forma de inserção de clientes segundo a proposta de hibridização com a heurística PFIH. Primeiramente, é calculado o custo de inserção de cada cliente, baseado na função de custo 4.1 e são ordenados de forma crescente, o que permite inserir clientes mais próximos do depósito no início do procedimento. Assim, tem-se a seguinte ordem de inserção: $C = 8, 5, 17, 7, 12, 18$. Logo após, o primeiro cliente é inserido na solução. Depois, o próximo cliente da

lista é inserido, testando todas as demais posições da solução. Caso não encontre nenhuma posição válida, uma nova rota é criada, caso contrário, é montada uma lista de candidatos restrita com as possíveis posições daquele elemento na solução, baseados nos valores da função de avaliação descrita em 5.1 e é sorteado uma posição desta lista de forma aleatória. Estas etapas são executadas até que todos os elementos contidos na lista C sejam inseridos na solução. Ao final, tem-se uma solução $S = \{\{0, 18, 12, 17, 0\}, \{0, 7, 8, 0\}, \{0, 5, 0\}\}$ com todos os clientes C inseridos nela. O procedimento pode ser resumido em:

- O cliente é inserido em todas as posições possíveis da solução;
- As posições em que não violarem as restrições de janela de tempo, capacidade do veículo e tempo de viagem, são armazenadas em uma lista de candidatos;
- As posições gravadas na lista de candidatas são classificadas de acordo com o valor da função objetivo e aquelas que obtiveram os melhores resultados irão compor uma lista restrita de candidatos, de acordo com a função avaliação 4.2
- Uma posição da lista restrita de candidatos é sorteada aleatoriamente e inserido o cliente;
- Caso nenhuma posição seja factível, é criada uma nova rota e o cliente é inserido nela.

O algoritmo 5.3 mostra o procedimento para a construção de uma solução, aplicando a PFIH como critério na escolha dos clientes a serem inseridos na solução. Ele recebe como entrada:

- c : Informações dos clientes;
- $tamLCR$: Tamanho da lista de candidatos restrita;
- n : Número de clientes da instância a ser testada.

Assim, nas linhas 1 a 4 são inicializadas as variáveis s e s' com conjunto vazio e as variáveis *cont* e *num* com valor 0. As linhas 5 e 6 calculam os custos dos clientes baseados na heurística PFIH e ordenam os clientes de forma crescente. A linha 7 cria uma nova rota e insere o primeiro cliente. As linhas de 8 a 24 são responsáveis pela inserção dos clientes na solução corrente s. A linha 10 calcula o valor da função adaptativa-gulosa do GRASP descrita em 5.1. A linha 11 é responsável por retornar o cliente que irá ser inserido na solução corrente. As linhas de 12 a 23 são responsáveis por percorrer a solução parcial s' e verificar se tem alguma posição factível. Caso tenha, é montada uma lista restrita de candidatos e o cliente é inserido em uma posição selecionada aleatoriamente desta lista; caso contrário, é criado uma nova rota e o cliente em questão é inserido nela. O procedimento finaliza retornando uma solução completa s.

Um algoritmo de busca local tem a finalidade de encontrar soluções de boa qualidade, em um tempo computacional viável. Neste contexto, uma busca local parte de uma solução inicial e executa uma serie de estruturas de vizinhanças definidas previamente e que permitem explorar de forma iterativa o espaço de busca. No

Algoritmo 5.3: Construção_$GRASP_PFIH_VND$

Entrada: c,$tamLCR$,n
Saída: s

```
1  s ← ∅;
2  s' ← ∅;
3  cont ← 0;
4  num ← 0;
5  calculaCustosPFIH(c);
6  ordenaCustosPFIH(c);
7  s = criaRota(s', c);
8  enquanto cont < n faça
9  |  num ← 0;
10 |  FA ← calculaFA();
11 |  LCR ← selecionaCliente(c, tamLCR, FA) ;
12 |  enquanto num < tamSol faça
13 |  |  s' ← insereSolucao(LCR, num);
14 |  |  se s' < s então
15 |  |  |  s ← s';
16 |  |  |  cont + +;
17 |  |  fim
18 |  fim
19 |  se LRC ⊄ s' então
20 |  |  s ← criaRota(s') ;
21 |  |  cont + +;
22 |  fim
23 fim
24 retorna s;
```

caso do GRASP, devido a sua grande versatilidade de combinação de outras técnicas, aplica-se a metaheurística VND descrita na seção 5.4.1. O pseudocódigo 5.4 descreve as fases do GRASP .

Algoritmo 5.4: $GRASP_PFIH_VND$

Entrada: c,$tamLCR$,n,it_{max},r
Saída: s*

```
1  cont ← 0;
2  s* ← ∅;
3  s' ← ∅;
4  s'' ← ∅;
5  enquanto cont < it_max faça
6  |  s' ← construcao_GRASP_PFIH_VND(c, tamLCR, n);
7  |  s'' ← busca_local_VND(s', r);
8  |  se s'' < s' então
9  |  |  cont = 0;
10 |  |  senão
11 |  |  |  cont++;
12 |  |  fim
13 |  fim
14 |  se s'' < s* então
15 |  |  s* ← s'';
16 |  fim
17 fim
18 retorna s*;
```

O pseudocódigo 5.4 mostra de que forma o GRASP foi aplicado no presente trabalho. Ele recebe como entrada:

- c : Informações dos clientes;
- $tamLCR$: Tamanho da lista de candidatos restrita;
- n : Número de clientes da instância a ser testada.
- it_{max}: Número de iterações máximas sem melhora
- r : Número de estruturas de vizinhança.

No algoritmo 5.4, considera-se, nas linhas 1 a 4 a inicialização da variável *cont* por 0 e das variáveis $s*$,s' e s'' pelo conjunto vazio, sendo que $s*$ representado a melhor solução e s' e s'' representado soluções modificadas ao longo do algoritmo. As linhas de 5 a 17 mostram as iterações do método. As linhas 6 e 7, mostram a execução da construção descrita no algoritmo 5.3 e a busca local implementada baseada no pseudocódigo descrito em 5.2. As linhas 8 a 13 verificam se a solução s'' é melhor que a solução construída anteriormente. Caso o valor seja menor, não é computada uma iteração sem melhora, senão, ela passa a ser considerada de piora. As linhas de 14 a 16 testam se a solução s'' é melhor do que a melhor solução $s*$, caso afirmativo, ela passa a ser a solução de melhora. O método finaliza retornando a melhor solução encontrada ao longo da busca.

5.4.3 GRASP+PFIH+VND+IGS

A metaheurística *Iterated Greedy Search (IGS)* foi proposta por Rubén e Thomas (2007) e adaptada para diversos problemas combinatórios relatados na literatura. Como não há uma aplicação do método para PRVJT, propõe-se uma série de modificações que levem em consideração o domínio do problema. O conceito em que se baseia a ideia do IGS parte do princípio de, à partir de uma solução completa, realizar perturbações mais intensas e permitir uma exploração do espaço de busca de forma mais rápida e com bons resultados. Suas ideias básicas podem ser resumidas em:

- Destruição: Nesta fase, um determinado número de elementos que compõe a solução são retirados, podendo seguir um critério ou de forma aleatória;
- Reconstrução : Esta fase, por sua vez, re-insere os elementos retirados da primeira fase, seguindo algum critério;
- Busca Local: Após a fase de destruição e reconstrução, é aplicado uma busca local na solução perturbada, podendo ser utilizada alguma hibridização com outra heurística ou metaheurística para melhor explorar o espaço de busca;
- Critério de Aceitação: Depois de realizar a perturbação e busca local, o método avalia se continua a busca na solução perturbada ou se descarta as modificações.

A adaptação do IGS para o PRVJT leva em consideração algumas características que o diferencia da proposta original. Uma delas é em relação as restrições de janela de tempo, que são o maior gargalo do problema, por gerar muitas soluções infactíveis. Neste caso, optou-se por trabalhar apenas com soluções factíveis, o que impactou na forma como é proposta as fases de destruição e reconstrução. Para a fase de destruição, opta-se por retirar todos os clientes de uma rota, selecionada aleatoriamente, para que se mantenha a factibilidade. A Figura 5.15 ilustra o funcionamento da fase de destruição.

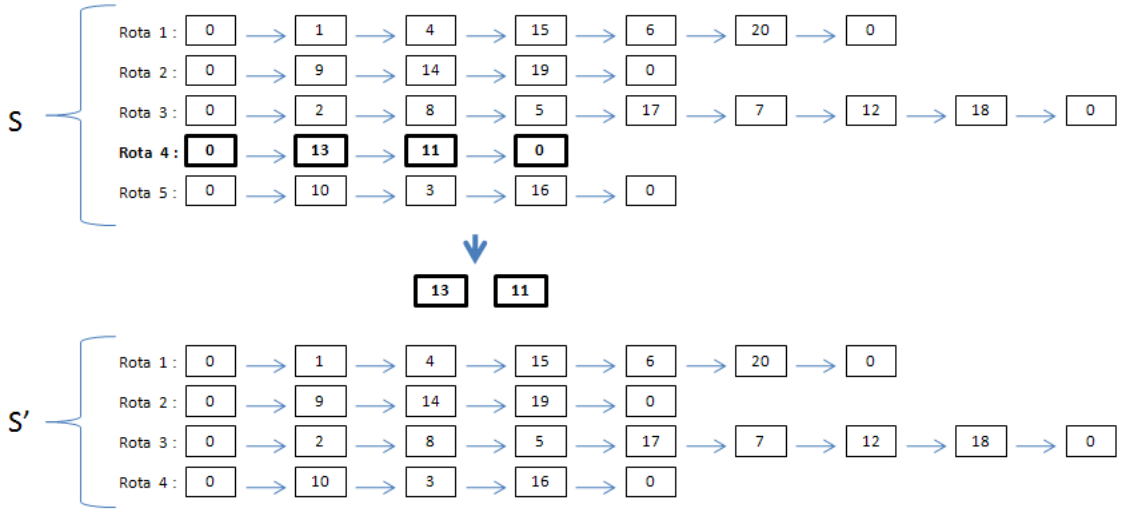

Figura 5.15: Destruição de uma solução fictícia pelo IGS.

Na Figura 5.15 mostra uma solução $s = \{\{0,1,4,15,6,20,0\},\{0,9,14,19,0\},\{0,2,8,5,17,7,12,18,0\},\{0,13,11,0\},\{0,10,3,16,0\}\}$ que passa pelo processo de destruição. O procedimento é realizado selecionando de forma aleatória a rota $r_4 = \{0,13,11,0\}$ e retirando os clientes 13 e 11, que irão compor uma lista de clientes. No final desta fase, tem-se uma solução parcial $s' = \{\{0,1,4,15,6,20,0\},\{0,9,14,19,0\},\{0,2,8,5,17,7,12,18,0\},\{0,10,3,16,0\}\}$. O algoritmo 5.15 mostra o pseudocódigo do procedimento.

Algoritmo 5.5: Destruição IGS

Entrada: *c*,*s*,*listaClientes*
Saída: s'

```
1  cont ← 0;
2  s' ← ∅;
3  r ← sorteioRota(s);
4  n ← s(r);
5  enquanto cont < n faça
6  |  listaClientes ← s(cont);
7  |  cont + +;
8  fim
9  cont ← 0;
10 enquanto cont < s.numRotas faça
11 |  se cont <> r então
12 |  |  s' ← s(cont);
13 |  fim
14 |  cont + +;
15 fim
16 retorna s';
```

O algoritmo 5.15 inicia recebendo três parâmetros:

- c : Informações dos clientes de uma determinada rota;
- s : Solução corrente
- *listaClientes*: Uma estrutura de dados que irá armazenar os clientes retirados. Esta estrutura irá ser utilizada na fase de reconstrução.

O procedimento 5.15 inicia, nas linhas 1 e 2, a variável *cont* com o valor 0 e a variável s' com conjunto vazio. A linha 3 faz o sorteio da rota que irá participar da destruição da solução, armazenado na variável r. A linha 4 armazena na variável n o número de clientes pertencente a rota que será retirada da solução. As linhas de 5 a 7 são responsáveis por retirar todos os elementos da rota e armazenar na variável *listaClientes*. A linha 9 inicializa a variável *cont* com 0. As linhas de 10 a 15 retiram a rota sorteada, de forma a gerar uma solução parcial s', que será retornada pelo algoritmo.

Já na fase de reconstrução, considera-se a inserção de elementos sem violar as restrições de janela de tempo, carga e funcionamento do depósito, sempre obtendo soluções factíveis. O procedimento passa a percorrer todas as posições que compõe a

solução parcial s', procurando posições factíveis para cada cliente que estão na lista de clientes retirados na fase de destruição. Caso encontrem alguma posição factível, o cliente é inserido nesta posição e o algoritmo passa a realizar a busca para outro clientes da lista, caso contrário, os clientes que não encontrarem nenhuma outra posição factível são armazenados e, ao final do processo de reconstrução, é criada uma nova rota e eles são inseridos nela, conservado a ordem de visitação. A Figura 5.16 ilustra a forma de reconstrução adotada.

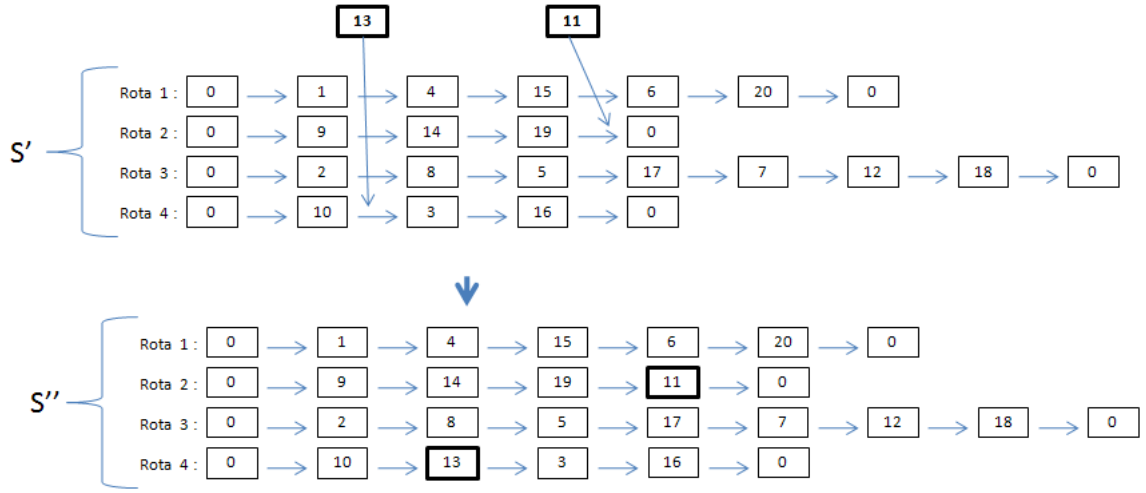

Figura 5.16: Reconstrução de uma solução fictícia pelo IGS.

Na Figura 5.16 retrata a inserção dos clientes na fase de reconstrução. Cada cliente que compõe a lista de clientes retirados na fase de destruição, representados pelos clientes $listaClientes = \{13, 11\}$, são testados em todas as posições da solução parcial $s' = \{\{0, 1, 4, 15, 6, 20, 0\}, \{0, 9, 14, 19, 0\}, \{0, 2, 8, 5, 17, 7, 12, 18, 0\}, \{0, 10, 3, 16, 0\}\}$. Como a fase de reconstrução preserva a factibilidade, as restrições são verificadas e, caso tenha soluções factíveis, o cliente é inserido na solução e retirado da lista de clientes. Assim, o processo de reconstrução acontece da seguinte forma:

- O cliente 11 é testado em todas as posições possíveis da solução s', verificando a factibilidade;
- Como o cliente 11 encontrou posição que atende as restrições na rota 2, da solução parcial s', no arco $[19, 0]$, o cliente é inserido nesta posição e é retirado da lista de clientes a serem inseridos;
- O cliente 13, por sua vez, também passa pelo mesmo procedimento anterior e encontra uma posição factível;
- O cliente 13 é inserido no arco $[10, 3]$ pertencente à rota 4, da solução s'.

Neste exemplo, após a fase de reconstrução, é obtida uma solução completa $s'' = \{\{0, 1, 4, 15, 6, 20, 0\}, \{0, 9, 14, 19, 11, 0\}, \{0, 2, 8, 5, 17, 7, 12, 18, 0\}, \{0, 10, 13, 3, 16, 0\}\}$. Caso algum cliente da lista de clientes não encontre posição factível, é criada uma

Algoritmo 5.6: Reconstrução IGS

Entrada: c,s',$listaClientes$
Saída: s"

```
1  cont ← 0;
2  s'' ← s';
3  listaClientesR ← ∅ ;
4  enquanto cont < listaClientes faça
5  |  se verificaFactibilidade(c, s'', listaClientes(cont)) == true então
6  |  |  s'' ← insereCliente(s'', listaClientes(cont));
7  |  |  senão
8  |  |  |  listaClientesR ← listaClientes(cont)/;
9  |  |  fim
10 |  fim
11 |  cont + +;
12 fim
13 se listaClientesR <> ∅ então
14 |  s'' ← criaRota(listaClientesR)
15 fim
16 retorna s'';
```

nova rota e eles são inseridos nela. A vantagem desse tipo de abordagem é ter a possibilidade de eliminação de uma rota completa, já que é o custo maior do problema. O algoritmo 5.6 descreve o pseudocódigo do processo de reconstrução.

O algoritmo 5.6 recebem como entrada:

- c : Informações dos clientes de uma determinada instância do problema;
- s' : Solução parcial obtida na fase de destruição;
- *listaClientes*: Uma estrutura de dados que contém os clientes que foram retirados na fase de destruição.

O pseudocódigo 5.6 inicializa, nas linhas 1 e 2, as variáveis *cont* com valor 0 e s'' com a solução parcial s' recebida como parâmetro. A linha 3 atribui conjunto vazio a variável *listaClientesR*, responsável por armazenar os clientes que, por ventura, não conseguirem encontrar uma posição factível na solução s''. As linhas de 4 a 12 percorrem a lista de clientes e verificam a factibilidade em todas as posições possíveis na solução s''. Caso eles encontrem, o cliente é inserido na solução, caso contrário, são armazenados na *listaCclientesR*. As linhas de 13 a 15 verificam se há algum cliente na *listaClientesR*, caso afirmativo, é criada uma nova rota, na qual todos os clientes da lista são inseridos nela, conservando a factibilidade. O algoritmo finaliza retornado a solução completa s''. Assim, as fases de destruição e reconstrução são realizadas de forma consecutiva e uma busca local é aplicada. Após estas etapas, o algoritmo estabelece um critério de aceitação, que pode variar de problema para problema. Um primeiro critério adotado na adaptação do método é aceitar apenas soluções de melhora, adotando este critério na versão GRASP_PFIH_VND_IGS1 descrita no algoritmo 5.8 . Outro critério adotado na

versão GRASP_PFIH_VND_IGS2, descrita em 5.9, é aplicado um operador de temperatura descrito na seção 4.4, sendo que a segunda versão apresentou melhores resultados. Na seção 5.4.4 são discutidas as versões propostas na hibridização com mataheurística IGS.

5.4.4 Versões do Algoritmo GRASP+PFIH+VND+IGS

O conceito de hibridização proposto leva em consideração o domínio do problema e as questões inerentes a várias técnicas presentes na literatura, envolvendo tanto heurísticas como metaheurísticas. A combinação de heurísticas, como a PFIH, com outras metaheurísticas tem por objetivo gerar soluções com maior qualidade, com um tempo computacional reduzido. Seguindo esta linha, a hibridização GRASP_PFIH_VND_IGS1 relatada em 5.8 combina, os algoritmos GRASP_PFIH_VND tratado em 5.4 como solução inicial,a metaheurística VND descrita em 4.3 com busca local, e as implementações da fase de destruição descritas no algoritmo 5.5 e reconstrução mostradas em 5.6. As estruturas de vizinhança usadas na busca local são mostradas a seguir:

- Movimentos de modificação intra-rota: *Exchange(1)*;
- Movimentos intra-rota: *Shift(3)*, *Shift(2)* e *Shift(1)*;
- Movimentos inter-rotas: *Shift(1,0)*, *Shift(2,0)* e *Shift(3,0)* ;
- Movimentos inter-rotas: *Swap(2,1)* e *Swap(2,2)* e;
- Estratégia Eliminação de Rota: *eliminaRota()*

Tanto as estruturas de vizinhanças inter-rotas quanto as intra-rotas são aplicadas seguindo a ordem apresentada e da forma como é descrita na seção 5.2. Para intensificar a busca foi aplicada uma técnica denominada *eliminaRota*, na qual é selecionada uma rota aleatoriamente e é feita uma tentativa de re-inserção dos clientes desta rota nas demais rotas da solução. Este movimento só é realizado caso consiga re-inserir todos os clientes da rota, do contrário, a solução não é modificada. O algoritmo 5.7 descreva o pseudocódigo das estruturas de vizinhança aplicadas na busca local.

O algoritmo possui como entrada s, que representa a solução que ira receber a modificação e *cont* representando qual a estrutura de vizinhança será aplicada a solução. As linhas 1 a 30 são responsáveis por executar as modificações correspondentes, produzindo um vizinho s'. Ao final, o algoritmo returna a solução modificada s'. Na versão GRASP_PFIH_VND_IGS1 descrita no pseudocódigo 5.8, mostra, como parâmetros de entrada:

- c : Informações dos clientes de uma determinada rota;
- $tamLCR$: Tamanho da lista de candidatos restrita,parâmetro para metaheurística GRASP_PFIH_VND descrita em 5.4;
- n : Número de clientes da instância a ser testada.

Algoritmo 5.7: MovimentoVND1(s,*cont*)

Entrada: s, *cont*
Saída: s'

```
1  se cont == 0 então
2  |  s' = Exchange(1);
3  fim
4  se cont == 1 então
5  |  s' = Shift(3);
6  fim
7  se cont == 2 então
8  |  s' = Shift(2);
9  fim
10 se cont == 3 então
11 |  s' = Shift(1);
12 fim
13 se cont == 4 então
14 |  s' = Shift(1,0);
15 fim
16 se cont == 5 então
17 |  s' = Shift(2,0);
18 fim
19 se cont == 6 então
20 |  s' = Shift(3,0);
21 fim
22 se cont == 7 então
23 |  s' = Swap(2,1);
24 fim
25 se cont == 8 então
26 |  s' = Swap(2,2);
27 fim
28 se cont == 9 então
29 |  s' = eliminaRota;
30 fim
31 retorna s';
```

- it_{max} : Número de iterações máximas sem melhora, parâmetro da metaheurística GRASP_PFIH_VND descrita em 5.4;
- r : Número de estruturas de vizinhança como entrada para a busca local VND descrita em 5.2;
- $maxIt$: Número máximo de iterações sem melhora, utilizada no algoritmo GRASP_PFIH_VND_IGS1

As linhas 1 a 5 inicializa a variável *cont* com valor 0 e as variáveis $s*$,s',s'',s''' com conjunto vazio. A linha 6 executa o algoritmo 5.4 e retorna uma solução completa, que é armazenada na solução corrente s. As linhas de 7 a 17 realizam o processo de busca. A linha 8 executa o método de destruição, conforme descrito no algoritmo 5.5, retornando uma solução parcial s'. A linha 9 executa o procedimento de reconstrução, descrito em 5.6, retornando uma solução completa s''. A linha 10 executa uma busca local, descrita em 5.2 com as estruturas de vizinhanças mostradas no algoritmo 5.7. As linhas 11 a 16 são responsáveis pelo critério de aceitação adotado, no qual apenas soluções de melhora são admitidas. Assim, se a solução s''' tiver o valor da função objetivo menor que a melhor solução $s*$, ela passa a ser a solução melhor, caso contrário uma iteração sem melhora é computado. O procedimento termina quando a busca fizer $maxIt$ iterações sem melhora. O procedimento retorna a melhor solução encontrada.

Algoritmo 5.8: $GRASP_PFIH_VND_IGS1$

Entrada: c,$tamLCR$,n,it_{max},r,$maxIt$
Saída: $s*$

```
1  cont ← 0;
2  s* ← ∅;
3  s' ← ∅;
4  s'' ← ∅;
5  s''' ← ∅;
6  s ← GRASP_PFIH_VND(c, tamLCR, n, it_max, r);
7  enquanto cont < maxIt faça
8  |  s' ← destruicao(c, s, listaClientes);
9  |  s'' ← reconstrucao(c, s', listaClientes);
10 |  s''' ← buscaLocalVND(s'', r);
11 |  se f(s''') < f(s*) então
12 |  |  s* ← s''';
13 |  |  cont = 0;
14 |  |  senão
15 |  |  |  cont + +;
16 |  |  fim
17 |  fim
18 fim
19 retorna s*;
```

Já a versão GRASP_PFIH_VND_IGS2, descrita no algoritmo 5.9, trabalha um conceito importante durante a exploração das soluções: operador de temperatura.

Este operador foi proposto por Kirk Patrick et al. (1983) e aplicado na metaheurística *Simulated Annealing* (SA) e exemplificado na seção 4.4. Ele faz uma análise das soluções geradas e pode aceitar uma solução de piora, dependendo do estado da temperatura corrente. Caso ela seja muito alta, a probabilidade de aceitar soluções piores é maior, caso contrário, menor. Assim, no início da busca, são admitidas soluções de piora e, conforme a busca vai evoluindo, esta probabilidade vai diminuindo. Esta técnica tem se mostrado eficiente em diversos problemas de otimização combinatória e, por esse motivo, é aplicada ao método, porém mantendo a mesma temperatura ao longo de toda busca, sendo definida por parâmetro.

Algoritmo 5.9: $GRASP_PFIH_VND_IGS2$

Entrada: c,$tamLCR$,n,it_{max},r,T,$maxIt$
Saída: $s*$

```
1  cont ← 0;
2  s* ← ∅;
3  s' ← ∅;
4  s'' ← ∅;
5  s''' ← ∅;
6  s ← GRASP(c, tamLCR, n, it_max, r);
7  enquanto cont < maxIt faça
8      s' ← destruicao(c, s, listaClientes);
9      s'' ← reconstrucao(c, s', listaClientes);
10     s''' ← buscaLocalVND(s'', r);
11     △ ← f(s''') − f(s);
12     se △ < 0 então
13         s ← s''';
14         cont ← 0;
15         senão
16             x ← [0, 1] se x < exp^(−△/T) então
17                 s ← s''';
18                 cont ← 0;
19             fim
20             cont + +;
21         fim
22     fim
23     se f(s) < f(s*) então
24         s* ← s;
25     fim
26 fim
27 retorna s*;
```

No algoritmo 5.9, as linhas 1 a 5 inicializam as variáveis principais. A linha 6 retorna a melhor solução encontrada pela metaheurística GRASP descrita em 5.4. As linhas de 7 a 27 descrevem o procedimento propriamente dito. A linha 8 indica o processo de destruição, que é implementado selecionando uma rota aleatoriamente e retirando todos os seus clientes, obtendo uma solução parcial s' e uma lista de

clientes retirados da solução. A linha 9 indica o procedimento de re-inserção, que recebe como entrada a solução s', as informações dos clientes c e uma lista de clientes retirados da solução corrente. O procedimento é realizado fazendo tentativas de re-inserir os clientes observando as restrições, porém sem verificar o valor da função objetivo. Assim, caso algum cliente não seja inserido na solução, é criado uma nova rota e eles são colocados nela na mesma ordem em que foram retirados na fase de destruição. A linha 10 é executada a busca local VND descrita na Seção 5.4.1. Nas linhas de 12 à 25 é estabelecido o critério de aceitação de uma solução de piora, que funciona da seguinte forma:

- $\triangle$ - representa a variação da função objetivo da solução corrente s e a solução modificada s''';
- Caso a variação $\triangle$ seja negativa, significa que houve uma melhora na solução e por isso s''' passa a ser a solução corrente;
- Caso a variação $\triangle$ seja positiva, mostra que houve uma piora na solução. Neste caso é definido um valor aleatório entre $[0, 1]$ e, caso esse valor for menor que a expressão $exp^{-\triangle/T}$ essa solução se torna a solução corrente e o processo de busca continua a explorá-la por mais interações.

O mecanismo de aceitação de uma solução de piora pode auxiliar na exploração do espaço de busca. As linhas 23 à 25 garantem que a melhor solução encontrada não se perca. O algoritmo finaliza a busca quando houver um número $maxIt$ de iterações sem melhora.

Capítulo 6

Experimentos Computacionais

Neste capítulo são mostrados os experimentos computacionais dos algoritmos desenvolvidos descritos na Seção 5. A Seção 6.1 discutem as características principais das instâncias utilizadas nos testes. A Seção 6.2 faz uma análise dos resultados obtidos.

6.1 Descrição das Instâncias

A fim de mostrar a efetividade da ferramenta computacional desenvolvida foram utilizadas as instâncias propostas por Solomon (1987) de 100 clientes. Essas instâncias são amplamente aplicadas na literatura e tidas como referência, não só para o PRVJT, mas para inúmeras outras variações do problema que envolvem janela de tempo.

As instâncias são divididas em três grupos e duas classes principais. As instâncias C, R e RC têm relação com a disposição dos clientes distribuídas no plano cartesiano. As classes têm relação com a duração correspondente a janela de tempo de cada cliente.

As instâncias C, os clientes estão distribuídos em *clusters*, ou seja, estão agrupados em distâncias geográficas mais próximas. Além disso, elas são agrupadas nas classes $C1$ e $C2$. A classe $C1$ apresenta um período de tempo de atendimento a cada cliente reduzido e a carga máxima relacionada com o veículo equivale a 200 unidades. A classe $C2$, por sua vez, apresenta janela de tempo mais longas e carga máxima de 700 unidades. A Figura 6.1 mostra a distribuição dos clientes da instância $C101$ em um plano cartesiano.

Já as instâncias R possui seus clientes espalhados aleatoriamente com distâncias mais longas. Também são divididas em duas classes: $R1$ e $R2$. A classe $R1$ possuí restrições de janela de tempo mais curtas e carga máxima de 200 unidades por veículo. Já a classe $R2$ tem restrições de carga de 100 unidades em relação ao veículo e janelas de tempo largas. A Figura 6.2 mostra a distribuição dos clientes da instancia $R101$ no plano cartesiano.

O grupo de instâncias RC têm características tanto de clientes agrupados em *clusters* quanto distribuídos geograficamente de forma aleatória. A classe $RC1$ tem restrições de carga de 200 unidades e janelas de tempo menores. A classe $RC2$ possuí janelas de tempo largas e carga máxima de 1000 unidades. A Figura 6.3 mostra a disposição dos clientes na instância $RC101$.

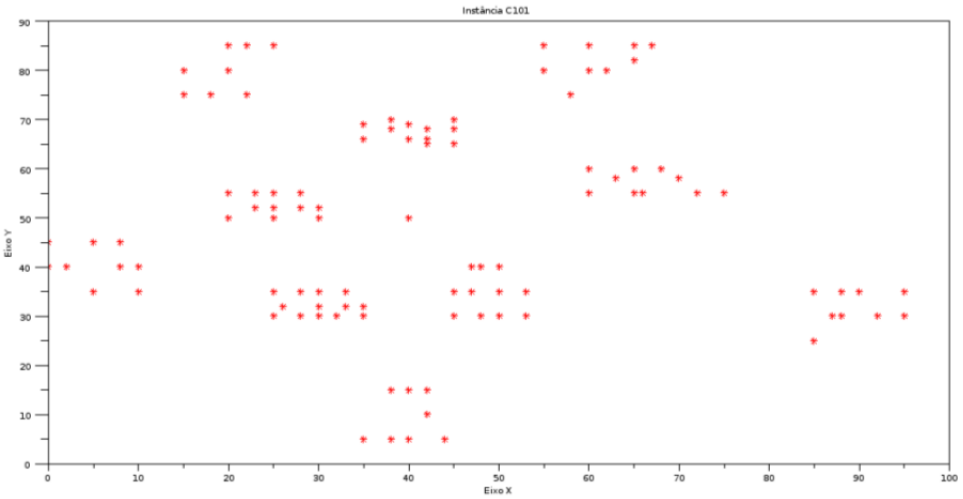

Figura 6.1: Disposição Clientes Instância C101

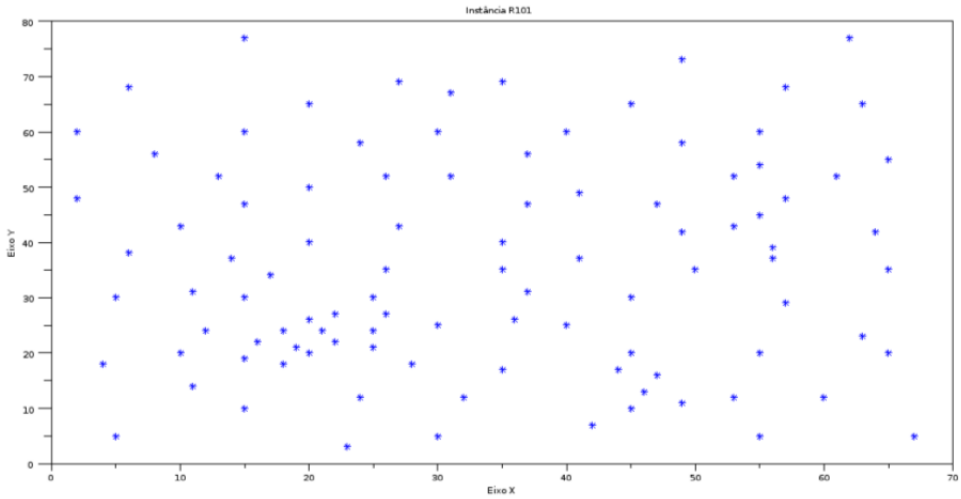

Figura 6.2: Disposição Clientes Instância R101

Em relação ao tempo de serviço, as instâncias C possuem 90 unidades de tempo, enquanto as instâncias R e RC utilizam 10 unidades. As instâncias que apresentam janela de tempo curtas indicam que necessitam de um número maior de veículos para serem atendidos, uma vez que o tempo de deslocamento de um cliente i para cliente j influência diretamente no número de veículos e na factibilidade de uma dada rota. Já as instâncias que tem janelas mais longas, tem a possibilidade de atender mais clientes no intervalo de tempo correspondente aos clientes i e j, ocasionando uma redução do número de veículos para atendê-los.

6.2 Análise dos Resultados

Os testes realizados foram aplicados no sistema operacional Linux, distribuição *Linux Mint versão 16* em um notebook *Intel Core i7* 2.66 GHz e memória RAM 8

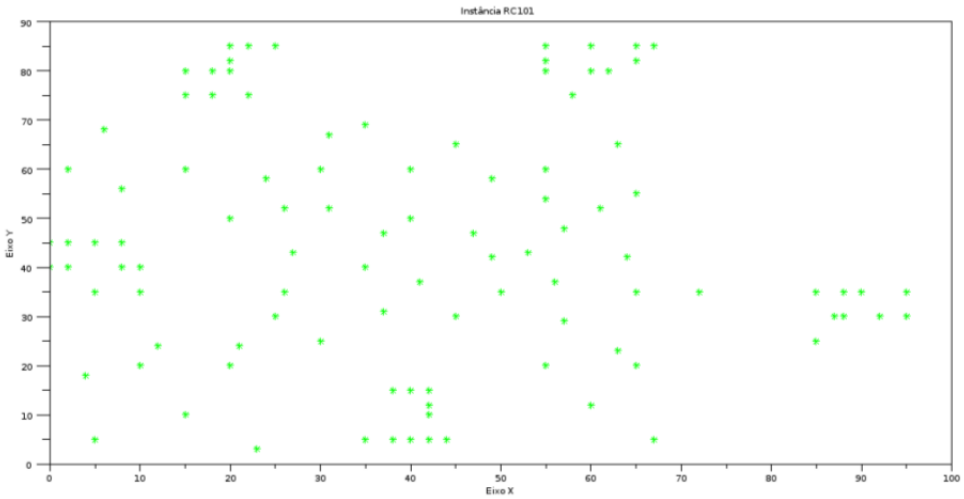

Figura 6.3: Disposição Clientes Instância RC101

GB. Os algoritmos foram implementados no ambiente *Netbeans 8.1* em linguagem C++ e testadas nas 56 instâncias propostas por Solomon (1987) e executadas 30 vezes para cada instância. A seguir são discutidos os resultados obtidos.

6.2.1 Resultados Algoritmo GRASP_PFIH_VND_IGS1

Os testes foram realizados nas 56 instâncias propostas por Solomon (1987), com 30 execuções para cada instância, conservando os mesmos parâmetros. Para os parâmetros $tamLCR$, it_{max} e α, como entrada para o algoritmo $GRASP_PFIH_VND$, com os valores 4, 600 e 0,3, respectivamente, obtidos empiricamente. Já para o algoritmo $GRASP_PFIH_VND_IGS1$ com os parâmetros r, representado o número de estruturas de vizinhanças e $maxIt$ representando iterações máximas sem melhora, com os valores 9 e 100 respectivamente, obtidos empiricamente. Para melhor compreensão das tabelas apresentadas nesta seção, considere:

- MRL : Melhores resultados heurísticos relatados pela literatura;
- MR_IGS_VND1 : Melhores resultados obtidos pela execução do algoritmo $GRASP_PFIH_VND_IGS1$;
- NV : Número de veículos utilizados na solução;
- DT : Distância total percorrida pelo veículo em uma determinada instância;
- DP_NV : Desvio padrão em relação ao número de veículos, considerando as 30 execuções;
- DP_DT : Desvio padrão em relação a distância total percorrida;
- DM_DT : Desvio padrão médio em relação a distância total percorrida.

Os resultados foram reunidos em seis grupos ($C1$,$C2$,$R1$,$R2$,$RC1$ e $RC2$) e comparados com os melhores resultados heurísticos da literatura (MRL) disponível no endereço *http://www.sintef.no/projectweb/top/vrptw/ solomon-benchmark/100-customers/*. Foram relacionados valores dos desvios padrões calculados nas amostras de 30 execuções de cada instância.

Na Tabela 6.1, relacionam os resultados para as instâncias do tipo $C1$. Nelas, pode-se observar, em relação ao número de veículos totais utilizados, 6 rotas ou $6,66\%$ acima dos melhores resultados heurísticos da literatura, enquanto a distância total ficou $8437,0$ ou 113% distantes dos melhores resultados. Para as instâncias $C101$, $C102$, $C104$ e $C109$, foram encontrados os mesmos valores das melhores soluções em relação ao número de veículos. Com relação aos desvios, observa-se que os valores obtidos estão próximos da média amostral, sem muitas variações, já os desvios das distâncias indicam uma dispersão maior, ficando no intervalo [$130,75$ e $227,15$].

Tabela 6.1: Resultados IGS-VND1 para instância classe C1

	MRL		MR_IGS_VND1			
instâncias	NV	DT	NV	DT	DP_NV	DP_DT
C101	10	828,94	10	1206,68	0	223,85
C102	10	828,94	10	1751,46	0	205,48
C103	10	828,06	11	1873,48	0	227,15
C104	10	824,78	10	1798,05	0	130,75
C105	10	828,94	11	2047,07	0	160,57
C106	10	828,94	11	1830,00	0	131,13
C107	10	828,94	11	1525,94	0	203,17
C108	10	828,94	12	2048,75	0	109,31
C109	10	828,94	10	1810,99	0	174,14
Total	90	7455,42	96	15892,42	–	–

Já a Tabela 6.2 mostram os resultados correspondentes a classe $C2$. Com relação ao número total de veículos, o algoritmo obteve $33,33\%$ ou 8 veículos acimas dos melhores resultados e $163,63\%$ ou $7721,33$ unidades acima da literatura em relação a distância percorrida. Já em relação aos desvios, observa-se que as instâncias $C201$, $C203$ e $C204$, atingiu os valores $0,18$, $0,34$ e $0,25$, indicando uma variação nos resultados em relação a média. Os desvios em relação a distância percorrida, ficou no intervalo [$108,32$ e $298,55$].

A Tabela 6.3 retratam os melhores resultados obtidos para o conjunto de instâncias $R1$. Os valores atingidos ficaram $24,48\%$ ou 35 veículos acima dos melhores resultados heurísticos relatados na literatura e $42,21\%$ ou $5988,08$ unidades distantes dos melhores resultados heurísticos da literatura como relação a distância percorrida. Os desvios em relação ao número de veículos ficaram no intervalo [0 e $0,77$] e em relação a distância percorrida, entre [$83,25$ e $129,09$], mostrando uma dispersão em relação a média.

A Tabela 6.4 exibem os resultados das instâncias da classe $R2$. Os resultados, em relação ao número total de veículos, obtiveram $33,33\%$ ou 10 rotas acima dos

Tabela 6.2: Resultados IGS-VND1 para instância classe C2

	MRL		MR_IGS_VND1			
instâncias	NV	DT	NV	DT	DP_NV	DP_DT
C201	3	591,56	4	1210,83	0,18	184,28
C202	3	591,56	4	1713,53	0	298,55
C203	3	591,17	4	1697,24	0,34	298,89
C204	3	590,6	4	1681,24	0,25	266,18
C205	3	588,88	4	1239,1	0	256,25
C206	3	588,49	4	1896,05	0	147,79
C207	3	588,29	4	1752,94	0	108,31
C208	3	588,32	4	1249,27	0	214,97
Total	24	4718,87	32	12440,2	–	–

Tabela 6.3: Resultados IGS-VND1 para instância classe R1

	MRL		MR_IGS_VND1			
instâncias	NV	DT	NV	DT	DP_NV	DP_DT
R101	19	1650,8	21	2045,89	0,25	48,7
R102	17	1486,12	18	1868,39	0	33,18
R103	13	1292,68	14	1669,74	0,5	70,93
R104	9	1007,31	14	1563,97	0,58	90,11
R105	14	1377,11	17	1884,62	0,77	79,75
R106	12	1252,03	15	1832,77	0,67	99,31
R107	10	1104,66	13	1628,25	0,71	89,52
R108	9	960,88	12	1424,87	0,59	126,5
R109	11	1194,73	14	1783,39	0,66	129,09
R110	10	1118,84	13	1562,3	0,5	83,25
R111	10	1096,72	14	1634,53	0,45	66
R112	9	982,14	13	1613,38	0,4	86,19
Total	143	14524,02	178	20512,1	–	–

melhores resultados heurísticos da literatura e 81,98% ou 8574,72 unidades superiores aos obtidos pelos melhores resultados heurísticos da literatura. Com relação aos desvios, as instâncias $R201$, $R206$ e $R207$ não apresentaram variações em relação ao número de veículos, as demais instâncias variaram no intervalo [0,18 e 0,64]. O desvio em relação a distância percorrida ficaram entre os valores [47,52 e 478,68].

Tabela 6.4: Resultados IGS-VND1 para instância classe R2

	MRL		MR_IGS_VND1			
instâncias	NV	DT	NV	DT	DP_NV	DP_DT
R201	4	1252,37	5	2096,45	0	45,04
R202	3	1191,7	4	1990,44	0,64	478,68
R203	3	939,5	3	1705,23	0,3	318,55
R204	2	825,52	3	1378,74	0,52	278,52
R205	3	994,42	4	1888,27	0,18	74,11
R206	3	906,14	4	1795,45	0	47,52
R207	2	890,61	3	1547,22	0	48,53
R208	2	726,82	3	1440,54	0,4	322,83
R209	3	909,16	4	1758,5	0,18	130,65
R210	3	939,37	4	1848,16	0,3	307,18
R211	2	885,71	3	1587,04	0,18	117,81
Total	30	10461,32	40	19036,04	–	–

A Tabela 6.5 relacionam os resultados obtidos da classe de instâncias $RC1$. Elas obtiveram 28 ou 30,43% de veículos acima dos melhores resultados heurísticos relatados na literatura em relação ao número de rotas totais e, 3875,44 ou 35% superiores aos melhores resultados heurísticos da literatura com relação a distância total percorrida pelos melhores resultados heurísticos da literatura. Já o desvio padrão apresenta variação entre [0,61 e 0,82] associado ao número de veículos utilizados e o intervalo entre [81,92 e 157,66] relacionado a distância total percorrida.

A Tabela 6.6 retratam os resultados das instâncias da classe $RC2$. O número de veículos utilizados ficaram 30,76% ou 8 veículos distantes dos melhores resultados heurísticos da literatura e a distância total percorrida ficaram 83,34% ou 7462,28 unidades superiores aos obtidos pelos melhores resultados heurísticos reportados pela literatura. O desvio das instâncias $RC201$ e $RC205$ não apresentou variação em relação a media de valores da amostra. Já as demais instâncias ficaram no intervalo [0,18 e 0,81] em relação ao número de veículos. O desvio envolvendo a distância percorrida ficaram compreendidas entre [30,68 e 422,45].

6.2.2 Resultados IGS-VND2

Para verificar o desempenho do algoritmo $GRASP_PFIH_VND_IGS2$, foram testadas as 56 instâncias propostas por Solomon (1987), com 30 execuções para cada instância. Para o ajuste de parâmetros, foram obtidos so seguintes valores empiricamente:

Tabela 6.5: Resultados IGS-VND1 para instância classe RC1

	MRL		MR_IGS_VND1			
instâncias	NV	DT	NV	DT	DP_NV	DP_DT
RC101	14	1696,94	18	2132,17	0,69	95,05
RC102	12	1554,75	16	2017,91	0,81	81,91
RC103	11	1261,67	13	1756,7	0,62	101,13
RC104	10	1135,48	13	1618,23	0,73	82,57
RC105	13	1629,44	17	2143,93	0,82	103,14
RC106	11	1424,73	16	2012,15	0,57	97,56
RC107	11	1230,48	14	1692,37	0,69	157,66
RC108	10	1139,82	13	1575,29	0,61	114,43
Total	92	11073,31	120	14948,75	–	–

Tabela 6.6: Resultados IGS-VND1 para instância classe RC2

	MRL		MR_IGS_VND1			
instâncias	NV	DT	NV	DT	DP_NV	DP_DT
RC201	4	1406,94	5	2397,74	0	51,12
RC202	3	1365,65	4	2136,35	0,48	634,45
RC203	3	1049,62	4	1928,77	0,81	422,45
RC204	3	798,46	4	1715,95	0,3	265,6
RC205	4	1297,65	5	2372,56	0	30,68
RC206	3	1146,32	4	2050,54	0,18	59,82
RC207	3	1061,14	4	2060,3	0,25	94,33
RC208	3	828,14	4	1753,99	0,43	235,46
Total	26	8953,92	34	16416,2	–	–

- $tamLCR$: Tamanho da lista de candidatos restrita, que serve de parâmetro para o algoritmo $GRASP_PFIH_VND$, com o valor de 3 clientes;
- α : Parâmetro da metaheurística $GRASP_PFIH_VND$, na função adaptativa gulosa da construção da solução através $GRASP$ com o valor de $0,3$;
- it_{max} : Número de iterações máxima sem melhora, sendo passado como parâmetro do algoritmo $GRASP_PFIH_VND$, com o valor de 100 iterações sem melhora.
- r : Parâmetro do algoritmo $GRASP_PFIH_VND_IGS2$, indicando o número de estruturas de vizinhanças presentes na busca local, configurado com o número 12, indicando a execução de 12 movimentos na metaheurística VND.
- $maxIt$: Parâmetro que configura o número de iterações sem melhora do algoritmo $GRASP_PFIH_VND_IGS2$, com o valor de 30 iterações sem melhora.

Para melhor compreensão dos resultados, considere que MRL indica os melhores resultados heurísticos relatados na literatura, MR_IGS_VND2, por sua vez, indicando os melhores resultados obtidos pelo algoritmo $GRASP_PFIH_VND_IGS2$ dentre as 30 execuções realizadas. Já DP_NV exibe o desvio padrão em relação ao número de veículos e DP_DT o desvio padrão calculado pelas distâncias percorridas, nas 30 execuções relatadas em cada instância.

A Tabela 6.7 exibem os resultados para as instâncias da classe $C1$. Nelas, pode-se observar que, em relação ao número de veículos, todas as instâncias tiveram os resultados iguais aos melhores resultados heurísticos reportados pela literatura, o que é expresso também pelo desvio padrão igual a 0. Já em relação a distância percorrida, as instâncias $C103$ e $C104$ mostraram valores próximos, mas, em relação a distância total percorrida, ficaram $101,2\%$ ou $7572,48$ unidades distantes dos melhores resultados da literatura. Já os desvios em relação as distâncias percorridas da amostra permaneceram entre os valores [$46,05$ e $106,59$].

A Tabela 6.8 relacionam as instâncias do grupo $C2$. O número de veículos utilizados ficaram, para todas as instâncias, os mesmos valores em relação aos melhores resultados heurísticos relatados na literatura. Já em relação a distância total percorrida, obtiveram $108,5\%$ ou $5122,21$ unidades acima dos melhores resultados heurísticos relatados na literatura, sendo que o desvio varia entre o intervalo [$28,46$ e $59,22$].

A Tabela 6.9 mostram os resultados das instâncias da classe $R1$. Em relação ao número total de veículos, os resultados obtiveram $6,99\%$ ou 11 veículos acima dos melhores resultados heurísticos relatas na literatura e $11,07\%$ ou $1756,82$ unidades distantes dos melhores resultados heurísticos da literatura no que diz respeito a distância total percorrida. Já os desvios, em relação a distância total percorrida, apresentam os valores entre [$75,71$ e $183,55$] e entre [0 e $0,52$] em relação ao número de veículos utilizados.

A Tabela 6.10 reportam os melhores resultados obtidos no grupo de instâncias da classe $R2$. A distância total percorrida ficaram $32,08\%$ ou $3922,72$ unidades acima dos melhores resultados da literatura e $6,66\%$ ou 3 veículos distantes dos melhores resultados heurísticos relatados na literatura. Já os desvios da amostra,

Tabela 6.7: Resultados IGS-VND2 para instância classe C1

	MRL		MR_IGS_VND2			
instâncias	NV	DT	NV	DT	DP_NV	DP_DT
C101	10	828,94	10	1976,91	0	88,45
C102	10	828,94	10	1804,98	0	65,21
C103	10	828,06	10	828,25	0	73,99
C104	10	824,78	10	911,03	0	46,05
C105	10	828,94	10	2080,55	0	67,56
C106	10	828,94	10	2180,78	0	57,59
C107	10	828,94	10	1925,81	0	106,59
C108	10	828,94	10	1855,02	0	91,87
C109	10	828,94	10	1464,57	0	72,87
Total	90	7455,42	90	15027,9	–	–

Tabela 6.8: Resultados IGS-VND2 para instância classe C2

	MRL		MR_IGS_VND2			
instâncias	NV	DT	NV	DT	DP_NV	DP_DT
C201	3	591,56	3	891,17	0	34,03
C202	3	591,56	3	1636,88	0	37,65
C203	3	591,17	3	1262,21	0	48,78
C204	3	590,6	3	874,52	0	59,22
C205	3	588,88	3	1053,84	0	32,55
C206	3	588,49	3	1338,9	0	36,18
C207	3	588,29	3	1545,49	0	35,19
C208	3	588,32	3	1238,07	0	28,46
Total	24	4718,87	24	9841,08	–	–

Tabela 6.9: Resultados IGS-VND2 para instância classe R1

	MRL		MR_IGS_VND2			
instâncias	NV	DT	NV	DT	DP_NV	DP_DT
R101	19	1650,8	19	1744,12	0	75,71
R102	17	1486,12	17	1496,74	0,18	94,39
R103	13	1292,68	13	1338,78	0,5	153,89
R104	9	1007,31	11	1186,23	0,52	150,16
R105	14	1377,11	14	1448,32	0	82,02
R106	12	1252,03	13	1353,73	0,18	125,64
R107	10	1104,66	11	1323,75	0,49	183,55
R108	9	960,88	10	1292,44	0,43	124,12
R109	11	1194,73	12	1304,48	0	117,22
R110	10	1118,84	12	1272	0	155,18
R111	10	1096,72	11	1233,39	0,46	158,72
R112	9	982,14	11	1286,86	0,49	145,93
Total	143	14524,02	154	16280,84	–	–

as instâncias $R201$, $R205$, $R209$, $R210$ e $R211$ não apresentaram desvio em relação a média no que diz respeito ao número de veículos, porém, com relação a distância total percorrida, ficaram no intervalo [14, 06 e 44, 53].

A Tabela 6.11 exibem os resultados referentes as instâncias do grupo $RC1$. Neles, em relação ao número de veículos totais utilizados, apresentam 8, 07% ou 8 veículos acima dos melhores resultados heurísticos da literatura e 6, 85% ou 1147, 83 unidades distantes dos melhores resultados heurísticos da literatura, em relação a distância total percorrida. Em relação aos desvios, observa-se que, para o número de veículos da amostra, ficou compreendido entre [0, 25 e 0, 5] e os desvios em relação a distância percorrida variaram entre [214, 82 e 316, 81].

A Tabela 6.12 reportam os resultados para a instâncias da classe $RC2$. O número de veículos totais ficaram 3, 85% ou 1 rota acida dos melhores resultados heurísticos da literatura e 34, 36% ou 3076, 74 unidades acima dos resultados relatados pela literatura com relação a distância total percorrida. Os desvios variaram entre [0 e 0, 5] com relação ao número de veículos utilizados e, em relação a distância total percorrida, entre [23, 3 e 58, 44].

6.2.3 Comparação dos Resultados

Os algoritmos propostos para a resolução do $PRVJT$ foram comparados tanto com os melhores resultados da literatura, quanto em relação ao desempenho das duas versões propostas. O algoritmo $GRASP_PFIH_VND_IGS1$, representado na tabela pela sigla IGS_VND1, se comportou de forma mais adversa, tendo obtido valores distantes da literatura. O algoritmo $GRASP_PFIH_VND_IGS2$, representado pela sigla IGS_VND2 na tabela, já mostrou um desempenho melhor em relação ao número de veículos utilizados, porém, em relação a distância percorrida, ficou distante em relação aos melhores resultados heurísticos da literatura, mas com

Tabela 6.10: Resultados IGS-VND2 para instância classe R2

	MRL		MR_IGS_VND2			
instâncias	NV	DT	NV	DT	DP_NV	DP_DT
R201	4	1252,37	4	1567,61	0	14,96
R202	3	1191,7	3	1437,48	0,49	32,49
R203	3	939,5	3	1439,26	0,47	40,9
R204	2	825,52	2	837,89	0,44	44,53
R205	3	994,42	3	1350,09	0	18,38
R206	3	906,14	3	1414,88	0,5	35,5
R207	2	890,61	3	1331,26	0,44	30,26
R208	2	726,82	3	1066,65	0,44	38,92
R209	3	909,16	3	1230,27	0	24,51
R210	3	939,37	3	1494,08	0	28
R211	2	885,71	3	1214,57	0	21,37
Total	30	10461,32	33	14384,04	–	–

Tabela 6.11: Resultados IGS-VND2 para instância classe RC1

	MRL		MR_IGS_VND2			
instâncias	NV	DT	NV	DT	DP_NV	DP_DT
RC101	14	1696,94	14	1707,62	0,4	247,68
RC102	12	1554,75	12	1611,98	0,5	292,45
RC103	11	1261,67	12	1408,17	0,49	257,03
RC104	10	1135,48	11	1311,1	0,25	219,81
RC105	13	1629,44	13	1697,04	0,44	316,81
RC106	11	1424,73	12	1446,63	0,25	214,82
RC107	11	1230,48	12	1345,69	0,37	229,24
RC108	10	1139,82	14	1692,91	0,5	271,46
Total	92	11073,31	100	12221,14	–	–

Tabela 6.12: Resultados IGS-VND2 para instância classe RC2

	MRL		MR_IGS_VND2			
instâncias	NV	DT	NV	DT	DP_NV	DP_DT
RC201	4	1406,94	4	1758,4	0	23,3
RC202	3	1365,65	4	1702,07	0,5	58,44
RC203	3	1049,62	3	1400,58	0,44	47,71
RC204	3	798,46	3	1316,99	0,44	50,67
RC205	4	1297,65	4	1689,44	0	46,14
RC206	3	1146,32	3	1519,32	0	27,05
RC207	3	1061,14	3	1417,26	0	38,99
RC208	3	828,14	3	1226,6	0	37,71
Total	26	8953,92	27	12030,66	–	–

desempenho superior a primeira versão do algoritmo. Os resultados comparativos entre as versões IGS_VND1 e IGS_VND2 são discutidas a seguir.

A Tabela 6.13 comparam os desempenhos dos algoritmos em relação as instâncias da classe $C1$. Com relação ao número de veículos totais a versão IGS_VND2 obteve desempenho superior, visto que a versão IGS_VND1 ficou $6,66\%$ ou 6 veículos a mais que a soluções obtidas pela primeira versão. Já em relação a distância percorrida, o algoritmo IGS_VND2 encontrou, considerando as distâncias totais, $5,75\%$ ou $864,52$ unidades a menos em relação ao obtido pela primeira versão. No que diz respeito aos desvios, a segunda versão do algoritmo têm valores idênticos com relação ao número de veículos utilizados, porém a variação da solução permaneceu mais próxima aos valores médios, gerando variações reduzidas em relação a versão IGS_VND1.

Tabela 6.13: Comparação Resultados Classe C1

	MRL		MR_IGS_VND1				MR_IGS_VND2			
Instâncias	NV	DT	NV	DT	DP_NV	DP_DT	NV	DT	DP_NV	DP_DT
C101	10	828,94	10	1206,68	0	223,85	10	1976,91	0	88,45
C102	10	828,94	10	1751,46	0	205,48	10	1804,98	0	65,21
C103	10	828,06	11	1873,48	0	227,15	10	828,25	0	73,99
C104	10	824,78	10	1798,05	0	130,75	10	911,03	0	46,05
C105	10	828,94	11	2047,07	0	160,57	10	2080,55	0	67,56
C106	10	828,94	11	1830	0	131,13	10	2180,78	0	57,59
C107	10	828,94	11	1525,94	0	203,17	10	1925,81	0	106,59
C108	10	828,94	12	2048,75	0	109,31	10	1855,02	0	91,87
C109	10	828,94	10	1810,99	0	174,14	10	1464,57	0	72,87
Total	90	7455,42	96	15892,42	–	–	90	15027,9	–	–

A Tabela 6.14 retratam os resultados obtidos pelas instâncias da classe $C2$. O algoritmo IGS_VND2 obteve, comparando os resultados com a primeira versão, uma melhora de $33,33\%$ ou 8 rotas, igualando a obtida pelos melhores resultados

heurísticos reportados pela literatura. Já em relação a distância total percorrida, a segunda versão do algoritmo alcançou uma melhoria de 2599, 12 unidades ou 26, 41% em relação a versão IGS_VND1. Os desvios do algoritmo IGS_VND2 também se mantiveram uma menor variação, tanto no número de veículos utilizados quanto na distância total percorrida.

Tabela 6.14: Comparação Resultados Classe C2

	MRL		MR_IGS_VND1				MR_IGS_VND2			
Instâncias	NV	DT	NV	DT	DP_NV	DP_DT	NV	DT	DP_NV	DP_DT
C201	3	591,56	4	1210,83	0,18	184,28	3	891,17	0	34,03
C202	3	591,56	4	1713,53	0	298,55	3	1636,88	0	37,65
C203	3	591,17	4	1697,24	0,34	298,89	3	1262,21	0	48,78
C204	3	590,6	4	1681,24	0,25	266,18	3	874,52	0	59,22
C205	3	588,88	4	1239,1	0	256,25	3	1053,84	0	32,55
C206	3	588,49	4	1896,05	0	147,79	3	1338,9	0	36,18
C207	3	588,29	4	1752,94	0	108,31	3	1545,49	0	35,19
C208	3	588,32	4	1249,27	0	214,97	3	1238,07	0	28,46
Total	24	4718,87	32	12440,2	–	–	24	9841,08	–	–

A Tabela 6.15 reportam os resultados obtidos das instâncias do tipo $R1$. Com relação ao número de veículos, o algoritmo IGS_VND1 teve um resultado inferior a segunda versão, com uma diferença de 4378, 85 unidades ou 27, 14% em relação a distância percorrida e 25 ou 16, 33% em relação ao número de rotas formadas. Com relação aos desvios, em ambos os algoritmos, houve uma variação grande entre elas, indicando uma boa exploração do espaço de busca.

Tabela 6.15: Comparação Resultados Classe R1

	MRL		MR_IGS_VND1				MR_IGS_VND2			
Instâncias	NV	DT	NV	DT	DP_NV	DP_DT	NV	DT	DP_NV	DP_DT
R101	19	1650,8	21	2045,89	0,25	48,7	18	1612,61	0	75,71
R102	17	1486,12	18	1868,39	0	33,18	17	1480,66	0,18	94,39
R103	13	1292,68	14	1669,74	0,5	70,93	13	1338,78	0,5	153,89
R104	9	1007,31	14	1563,97	0,58	90,11	11	1186,23	0,52	150,16
R105	14	1377,11	17	1884,62	0,77	79,75	14	1448,32	0	82,02
R106	12	1252,03	15	1832,77	0,67	99,31	13	1353,73	0,18	125,64
R107	10	1104,66	13	1628,25	0,71	89,52	11	1323,75	0,49	183,55
R108	9	960,88	12	1424,87	0,59	126,5	10	1292,44	0,43	124,12
R109	11	1194,73	14	1783,39	0,66	129,09	12	1304,48	0	117,22
R110	10	1118,84	13	1562,3	0,5	83,25	12	1272	0	155,18
R111	10	1096,72	14	1634,53	0,45	66	11	1233,39	0,46	158,72
R112	9	982,14	13	1613,38	0,4	86,19	11	1286,86	0,49	145,93
Total	143	14524,02	178	20512,1	–	–	153	16133,25	–	–

A Tabela 6.16 comparam o desempenho das duas versões do algoritmo para o grupo de instâncias $R2$. O algoritmo IGS_VND2 apresentou resultados superiores ao IGS_VND1, de 5217, 73 unidades ou 37, 75% melhor que a primeira versão

e 25% ou 8 veículos menor em relação a primeira versão. Os desvios da amostra tiveram valores mais altos no algoritmo IGS_VND1 do que na segunda versão, mostrando que o IGS_VND2 obteve resultados mais próximos da média.

Tabela 6.16: Comparação Resultados Classe R2

	MRL		MR_IGS_VND1				MR_IGS_VND2			
Instâncias	NV	DT	NV	DT	DP_NV	DP_DT	NV	DT	DP_NV	DP_DT
R201	4	1252,37	5	2096,45	0	45,04	4	1567,61	0	14,96
R202	3	1191,7	4	1990,44	0,64	478,68	3	1437,48	0,49	32,49
R203	3	939,5	3	1705,23	0,3	318,55	3	1439,26	0,47	40,9
R204	2	825,52	3	1378,74	0,52	278,52	2	711,59	0,44	44,53
R205	3	994,42	4	1888,27	0,18	74,11	3	1350,09	0	18,38
R206	3	906,14	4	1795,45	0	47,52	3	1414,88	0,5	35,5
R207	2	890,61	3	1547,22	0	48,53	3	1331,26	0,44	30,26
R208	2	726,82	3	1440,54	0,4	322,83	3	1066,65	0,44	38,92
R209	3	909,16	4	1758,5	0,18	130,65	3	1230,27	0	24,51
R210	3	939,37	4	1848,16	0,3	307,18	3	1494,08	0	28
R211	2	885,71	3	1587,04	0,18	117,81	2	775,14	0	21,37
Total	30	10461,32	40	19036,04	–	–	32	13818,31	–	–

A Tabela 6.17 mostram os resultados dos algoritmos IGS_VND1 e IGS_VND2 para o grupo de instâncias $RC1$. O desempenho da versão IGS_VND2 foi superior, comparando resultados totais, tanto em relação ao número de veículos, tendo obtido uma diferença de 20 veículos ou 20% melhor que a primeira versão, quanto em relação a distância total percorrida, tendo uma redução de $3116,73$ unidades ou $26,34\%$ em relação a versão IGS_VND2. Com relação aos desvios das amostras, o algoritmo IGS_VND1 teve uma variação entre $[81,91$ e $157,66]$ em relação a distância percorrida e $[0,61$ e $0,81]$. Já o algoritmo IGS_VND2 obteve $[214,82$ e $316,81]$ em relação a distância e $[0,25$ e $0,5]$ no que diz respeito ao número de veículos, tendo uma variação menor, comparado com a primeira versão do algoritmo.

Tabela 6.17: Comparação Resultados Classe RC1

	MRL		MR_IGS_VND1				MR_IGS_VND2			
Instâncias	NV	DT	NV	DT	DP_NV	DP_DT	NV	DT	DP_NV	DP_DT
RC101	14	1696,94	18	2132,17	0,69	95,05	14	1630,29	0,4	247,68
RC102	12	1554,75	16	2017,91	0,81	81,91	12	1490,39	0,5	292,45
RC103	11	1261,67	13	1756,7	0,62	101,13	12	1408,17	0,49	257,03
RC104	10	1135,48	13	1618,23	0,73	82,57	11	1311,1	0,25	219,81
RC105	13	1629,44	17	2143,93	0,82	103,14	13	1506,84	0,44	316,81
RC106	11	1424,73	16	2012,15	0,57	97,56	12	1446,63	0,25	214,82
RC107	11	1230,48	14	1692,37	0,69	157,66	12	1345,69	0,37	229,24
RC108	10	1139,82	13	1575,29	0,61	114,43	14	1692,91	0,5	271,46
Total	92	11073,31	120	14948,75	–	–	100	11832,02	–	–

A Tabela 6.18 reportam os melhores resultados para as instâncias do grupo RC2. O algoritmo IGS_VND2, comparado a versão IGS_VND1, obteve uma melho-

ria na distância total percorrida de 4385,44 unidades ou 36,45% e de 7 veículos ou 25,92% com relação ao número de veículos utilizados. Os desvios do algoritmo IGS_VND2 se comportaram com uma faixa de valores menor que a primeira versão, obtendo melhores resultados.

Tabela 6.18: Comparação Resultados Classe RC2

	MRL		MR_IGS_VND1				MR_IGS_VND2			
Instâncias	NV	DT	NV	DT	DP_NV	DP_DT	NV	DT	DP_NV	DP_DT
RC201	4	1406,94	5	2397,74	0	51,12	4	1758,4	0	23,3
RC202	3	1365,65	4	2136,35	0,48	634,45	4	1702,07	0,5	58,44
RC203	3	1049,62	4	1928,77	0,81	422,45	3	1400,58	0,44	47,71
RC204	3	798,46	4	1715,95	0,3	265,6	3	1316,99	0,44	50,67
RC205	4	1297,65	5	2372,56	0	30,68	4	1689,44	0	46,14
RC206	3	1146,32	4	2050,54	0,18	59,82	3	1519,32	0	27,05
RC207	3	1061,14	4	2060,3	0,25	94,33	3	1417,26	0	38,99
RC208	3	828,14	4	1753,99	0,43	235,46	3	1226,6	0	37,71
Total	26	8953,92	34	16416,2	–	–	27	12030,66	–	–

A Tabela 6.19 mostra um resumos dos resultados, comparando com os melhores da literatura com as versões IGS_VND1 e IGS_VND2. Fazendo uma comparação entre os desempenhos dos algoritmos, percebe-se que o algoritmo IGS_VND2 obteve os melhores resultados, tanto no número de veículos totais quanto na distância percorrida. A segunda versão do algoritmo obteve uma redução de 74 veículos ou 17,37% e de 20562,49 unidades ou 26,13% de redução em relação à distância total percorrida. Já na comparação IGS_VND2 com os melhores resultados heurísticos da literatura, ele ficou distante, em relação ao número de veículos, em 21 rotas ou 5,18% e em 21496,36 unidades ou 37,59% em relação à distância total percorrida. Em relação ao desvio médio da distância total percorrida (DM_DT), observa-se valores mais próximos à média para IGS_VND2, mostrando uma melhor exploração do espaço de busca, comparando com IGS_VND1.

Tabela 6.19: Comparação dos resultados finais dos algoritmos

	MRL		MR_IGS_VND1			MR_IGS_VND2		
Instâncias	NV	DT	NV	DT	DM_DT	NV	DT	DM_DT
C1	90	7455,42	96	15892,42	36,45	90	15027,9	14,11
C2	24	4718,87	32	12440,2	58,07	24	9841,08	7,50
R1	143	14524,02	178	20512,1	19,91	153	16133,25	27,36
R2	30	10461,32	40	19036,04	130,85	32	13818,31	7,76
RC1	92	11073,31	120	14948,75	15,93	100	11832,02	28,28
RC2	26	8953,92	34	16416,2	165,25	27	12030,66	9,49
Total	405	57186,86	500	99245,71	–	426	78683,22	–

Capítulo 7

Considerações Finais

Esta dissertação abordou o Problema de Roteamento de Veículos com Janela de Tempo (PRVJT). O PRVJT é uma variação do problema de roteamento de veículos clássico que incluem restrições de janela de tempo tanto em relação ao cliente e quanto ao depósito, uma demanda relacionada a cada cliente, uma carga máxima associada ao veículo que faz a entrega, um tempo de serviço que representa um período temporal que o veículo gasta para realizar a entrega, tendo como objetivos reduzir a distância total percorrida e o número de veículos utilizados. Foram apresentados diversos trabalhos encontrados na literatura reportando várias abordagens para a resolução do problema em questão.

Para a resolução do PRVJT, diversos trabalhos na literatura tem mostrado que os algoritmos baseados em heurísticas e metaheurísticas como sendo uma alternativa na obtenção de solução de boa qualidade e que atendam as necessidades das organizações. Outra técnica de bastante destaque diz respeito a hibridização, que é a combinação de uma ou mais heurísticas com o intuito de explorar o espaço de busca de forma mais intensa, obtendo melhores resultados. Seguindo esta linha, foram realizadas diversas combinações no intuito de dar suporte a técnicas que ainda não foram aplicadas para o problema, como é o caso do *Iterated Greedy Search*(IGS).

Para a solução inicial do IGS, foram adaptadas a função de cálculo do custo de inserção de clientes da heurística *Push-Forward Insertion Heuristic (PFIH)* na fase de construção da metaheurística *Greedy Randomized Adaptive Search Procedure (GRASP)*, além da heurística *Variable Neighborhood Descent (VND)* na fase de busca local. Para a busca local VND, foram elaboradas diversas estruturas de vizinhanças, levando em consideração o domínio do problema.

O primeiro algoritmo híbrido proposto, denominado $GRASP_PFIH_VND_IGS1$, descrita no algoritmo 5.8, combina a proposta da solução inicial descrita no pseudocódigo 5.4 ($GRASP_PFIH_VND$), juntamente com uma busca local VND com 8 estruturas de vinhanças e uma estratégia de eliminação de rota. Além disso, o IGS adotou o critério de aceitação de uma solução apenas se ela tivesse melhora, ou seja, caso uma solução apresentasse o valor da função objetivo maior que a anterior, a modificação era descartada. Esta abordagem teve resultados, em termos percentuais de $23,45\%$ distantes dos melhores resultados heurísticos mostrados na literatura e $73,54\%$ em relação a distância total percorrida.

A segunda versão desenvolvida, chamada $GRASP_PFIH_VND_IGS2$, descrita no pseudocódigo 5.9, parte da mesma solução inicial, porém aplicando 12 es-

truturas de vizinhança. Além dessa modificação, foram modificados os critérios de aceitação, usando o operador de temperatura proposto na metaheurística *Simulated Annealing (SA)*, modificando-o para uma temperatura constante. Os resultados, em termos percentuais, ficou $5,18\%$ acima dos melhores resultados heurísticos da literatura em relação ao número de veículos e $37,58\%$ no que diz respeito a distância percorrida.

Em termos comparativos, o algoritmo $GRASP_PFIH_VND_IGS2$ alcançou resultados totais mais próximos da literatura em todos os conjunto de instâncias testadas, porém ainda distantes dos melhores resultados heurísticos relatados. Isto se deve a decisões de projeto que foram cruciais, como a inclusão de mais estruturas de vizinhança, o próprio formato da fase de destruição e reconstrução e a aplicação do operador de temperatura no critério de aceitação, que permite que a busca aconteça em soluções de piora. Como a hibridização requer um conjunto de parâmetros muito grade, o que pode ser feito em trabalhos futuros é um estudo de formas mais eficientes de calibrar estes parâmetros de forma auto-adaptativo, o desenvolvimento de outras técnicas para a fase de destruição e reconstrução, levando em contra a factibilidade entre os clientes e combinação de outras heurísticas mais simples, como a primeira de melhora, na busca local.

Com base nas análises dos resultados, pode-se concluir que a proposta de hibridação da heurística IGS se mostrou promissora, porém requer a aplicação de diferentes abordagens, para a diminuição tanto do número de veículos quanto da distancia percorrida, sendo necessária a utilização de outras hibridizações que permitam a exploração do espaço de busca, além de outras abordagens envolvendo a fase de destruição e reconstrução, de forma adaptativa, proporcionando uma redução dos valores encontrados de forma a aproximarem dos melhores resultados heurísticos da literatura.

Referências Bibliográficas

Afifi, Sohaib; Dang, Duc-Cuong e Moukrim, Aziz. (2016). Heuristic solutions for the vehicle routing problem with time windows and synchronized visits. *Optimization Letters*, v. 10, n. 3, p. 511–525.

Agra, Agostinho; Christiansen, Marielle; Figueiredo, Rosa; Hvattum, Lars Magnus; Poss, Michael e Requejo, Cristina. (2012). Layered formulation for the robust vehicle routing problem with time windows. *International Symposium on Combinatorial Optimization*, p. 249–260. Springer, (2012).

Alvarez, Aldair e Munari, Pedro. (2017). An exact hybrid method for the vehicle routing problem with time windows and multiple deliverymen. *Computers & Operations Research*, v. 83, p. 1–12.

Anxin, Ye. (2011). Study of the vehicle routing problem with time windows based on improved particle swarm optimization algorithm. *Computer Science and Service System (CSSS), 2011 International Conference on*, p. 4053–4057, (2011).

Arroyo, José Elias e Leung, Joseph. (2017)a. An effective iterated greedy algorithm for scheduling unrelated parallel batch machines with non-identical capacities and unequal ready times. *Computers & Industrial Engineering*, v. 105, p. 84–100.

Arroyo, José Elias e Leung, Joseph. (2017)b. Scheduling unrelated parallel batch processing machines with non-identical job sizes and unequal ready times. *Computers & Operations Research*, v. 78, p. 117–128.

Azi, Nabila; Gendreau, Michel e Potvin, Jean-Yves. (2014). An adaptive large neighborhood search for a vehicle routing problem with multiple routes. *Computers & Operations Research*, v. 41, p. 167–173.

Bae, Heechul e Moon, Ilkyeong. (2016). Multi-depot vehicle routing problem with time windows considering delivery and installation vehicles. *Applied Mathematical Modelling*, v. 40, n. 13–14, p. 6536–6549.

Baños, Raúl; Ortega, Julio; Gil, Consolación; Márquez, Antonio L e De Toro, Francisco. (2013). A hybrid meta-heuristic for multi-objective vehicle routing problems with time windows. *Computers & Industrial Engineering*, v. 65, n. 2, p. 286–296.

Barbucha, Dariusz. (2014). A cooperative population learning algorithm for vehicle routing problem with time windows. *Neurocomputing*, v. 146, p. 210–229.

Barkaoui, Mohamed; Berger, Jean e Boukhtouta, Abdeslem. (2015). Customer satisfaction in dynamic vehicle routing problem with time windows. *Applied Soft Computing*, v. 35, p. 423–432.

Belmecheri, Farah; Prins, Christian; Yalaoui, Farouk e Amodeo, Lionel. (2010). Particle swarm optimization to solve the vehicle routing problem with heterogeneous fleet, mixed backhauls, and time windows. *IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW)*, (2010).

Bhowmik, Arka e Repaka, Ramjee. (2016). Estimation of growth features and thermophysical properties of melanoma within 3-d human skin using genetic algorithm and simulated annealing. *International Journal of Heat and Mass Transfer*, v. 98, p. 81–95.

Blocho, Miroslaw e Czech, Zbigniew J. (2012). A parallel eax-based algorithm for minimizing the number of routes in the vehicle routing problem with time windows. *High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), 2012 IEEE 14th International Conference on*, p. 1239–1246. IEEE, (2012).

Blocho, Miroslaw e Czech, Zbigniew J. (2013). A parallel memetic algorithm for the vehicle routing problem with time windows. *P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2013 Eighth International Conference on*, p. 144–151. IEEE, (2013).

Braaten, Simen; Gjønnes, Ola; Hvattum, Lars Magnus e Tirado, Gregorio. (2017). Heuristics for the robust vehicle routing problem with time windows. *Expert Systems with Applications*, v. 77, p. 136–147.

Bresina, John L. (1996). Heuristic-biased stochastic sampling. *AAAI/IAAI*, volume 1, p. 271–278, (1996).

Brito, Julio; Expósito, Airam e Moreno, José A. (2016). Variable neighbourhood search for close–open vehicle routing problem with time windows. *IMA Journal of Management Mathematics*, v. 27, n. 1, p. 25–38.

Brusco, Michael; Stolze, Hannah J; Hoffman, Michaela e Steinley, Douglas. (2017). A simulated annealing heuristic for maximum correlation core/periphery partitioning of binary networks. *PloS one*, v. 12, n. 5.

Carabetti, Eduardo Goecking; deSouza, Sergio Ricardo; Fraga, Marcelo Caramuru Pimentel e Gama, Pedro Henrique Antonacci. (2010). An application of the ant colony system metaheuristic to the vehicle routing problem with pickup and delivery and time windows. *Neural Networks (SBRN), 2010 Eleventh Brazilian Symposium on*, p. 176–181. IEEE, (2010).

Chen, Haihong; Wu, Changchun; Zuo, Lili; Diao, Feng; Wang, Li; Wang, Dapeng e Song, Baoqiang. (2017). Optimization of detailed schedule for a multiproduct pipeline using a simulated annealing algorithm and heuristic rules. *Industrial & Engineering Chemistry Research*, v. 56, n. 17, p. 5092–5106.

Chi-Bin, Cheng e Keng-Pin, Wang. (2009). Solving a vehicle routing problem with time windows by a decomposition technique and a genetic algorithm. *Expert Systems with Applications*, v. 36, p. 7758–7763.

Cordeau, Jean-François; Laporte, Gilbert e Mercier, Anne. (2001). A unified tabu search heuristic for vehicle routing problems with time windows. *Journal of the Operational research society*, v. 52, n. 8, p. 928– – 936.

Cruz-Chávez, Marco Antonio e Martínez-Oropeza, Alina. (2016). Feasible initial population with genetic diversity for a population-based algorithm applied to the vehicle routing problem with time windows. *Mathematical Problems in Engineering*, v. 2016.

Dantzig, George B e Ramser, John H. (1959). The truck dispatching problem. *Management Science*, v. 6, p. 80–91.

Davendra, Donald. (2010). *Traveling Salesman Problem, Theory and Applications*. InTech.

Dayarian, Iman; Crainic, Teodor Gabriel; Gendreau, Michel e Rei, Walter. (2015). A branch-and-price approach for a multi-period vehicle routing problem. *Computers & Operations Research*, v. 55, p. 167–184.

de Armas, Jesica e Melián-Batista, Belén. (2015). Variable neighborhood search for a dynamic rich vehicle routing problem with time windows. *Computers & Industrial Engineering*, v. 85, p. 120–131.

de Grancy, Gerald Senarclens e Reimann, Marc. (2016). Vehicle routing problems with time windows and multiple service workers: a systematic comparison between aco and grasp. *Central European Journal of Operations Research*, v. 24, n. 1, p. 29–48.

Deb, Kalyanmoy; Agrawal, Samir; Pratap, Amrit e Meyarivan, Tanaka. (2000). A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii. *International Conference on Parallel Problem Solving From Nature*, p. 849–858, (2000).

Despaux, François e Basterrech, Sebastián. (2014). A study of the multi-trip vehicle routing problem with time windows and heterogeneous fleet. *Intelligent Systems Design and Applications (ISDA), 14th International Conference on*, p. 7–12. IEEE, (2014).

Dhahri, Amine; Zidi, Kamel e Ghedira, Khaled. (2013). Vehicle routing problem with time windows under availability constraints. *Advanced Logistics and Transport (ICALT), 2013 International Conference on*, p. 308–314. IEEE, (2013).

Dorigo, Marco; Birattari, Mauro e Stutzle, Thomas. (2006). Ant colony optimization. *IEEE computational intelligence magazine*, v. 1, n. 4, p. 28–39.

Dubois-Lacoste, Jérémie; Pagnozzi, Federico e Stützle, Thomas. (2017). An iterated greedy algorithm with optimization of partial solutions for the makespan permutation flowshop problem. *Computers & Operations Research*, v. 81, p. 160–166.

Dumas, Yvan; Desrosiers, Jacques; Gelinas, Eric e Solomon, Marius M. (1995). An optimal algorithm for the traveling salesman problem with time windows. *Operations research*, v. 43, n. 2, p. 367–371.

Errico, Fausto; Desaulniers, Guy; Gendreau, Michel; Rei, Walter e Rousseau, L-M. (2016). A priori optimization with recourse for the vehicle routing problem with hard time windows and stochastic service times. *European Journal of Operational Research*, v. 249, n. 1, p. 55–66.

Fanjul-Peyro, Luis e Ruiz, Rubén. (2010). Iterated greedy local search methods for unrelated parallel machine scheduling. *Iterated greedy local search methods for unrelated parallel machine scheduling*, v. 207, n. 1, p. 55–69.

Feo, T.A. e Resende, M.G.C. (1995). Greedy randomized adaptive search procedures. *Journal of Global Optimization*, v. 6, p. 109–133.

Fleurent, Charles e Glover, Fred. (1999). Improved constructive multistart strategies for the quadratic assignment problem using adaptive memory. *Informs journal on computing*, v. 11, n. 2, p. 198–204.

Gao, Jie; Sun, Linyan e Gen, Mitsuo. (2008). A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems. *Computers and operations research*, v. 35, n. 19, p. 2892–2907.

Gehring, Hermann e Homberger, Jörg. (1999). A parallel hybrid evolutionary metaheuristic for the vehicle routing problem with time windows. Miettinen, K.; Mäkelä, M. e Toivanen, J., editors, *Proceedings of EUROGEN99 – Short Course on Evolutionary Algorithms in Engineering and Computer Science*, p. 57–64. University of Jyväskylä, (1999).

Gendreau, Michel; Hertz, Alain; Laporte, Gilbert e Stan, Mihnea. (1998). A generalized insertion heuristic for the traveling salesman problem with time windows. *Operations Research*, v. 46, n. 3, p. 330–335.

Ghoseiri, Keivan e Ghannadpour, Seyed Farid. (2010)a. A hybrid genetic algorithm for multi-depot homogenous locomotive assignment with time windows. *Applied Soft Computing*, v. 10, n. 1, p. 53–65.

Ghoseiri, Keivan e Ghannadpour, Seyed Farid. (2010)b. Multi-objective vehicle routing problem with time windows using goal programming and genetic algorithm. *Applied Soft Computing*, v. 10, n. 4, p. 1096–1107.

Goeke, Dominik e Schneider, Michael. (2015). Routing a mixed fleet of electric and conventional vehicles. *European Journal of Operational Research*, v. 245, n. 1, p. 81–99.

Gutin, Gregory e Punnen, Abraham. (2006). *The Traveling Salesman Problem and Its Variations*, volume 12. Springer Science & Business Media.

Haddadene, SR Ait; Labadie, N e Prodhon, C. (2016). Nsgaii enhanced with a local search for the vehicle routing problem with time windows and synchronization constraints. *IFAC-PapersOnLine*, v. 49, n. 12, p. 1198– 1203.

Hartl, Richard F; Hasle, Geir e Janssens, Gerrit K. (2006). Special issue on rich vehicle routing problems. *Central European Journal of Operations Research*, v. 14, p. 103–104.

Hatami, Sara; Ruiz, Rubén e Andres-Romano, Carlos. (2015). Heuristics and metaheuristics for the distributed assembly permutation flowshop scheduling problem with sequence dependent setup times. *International Journal of Production Economics*, v. 169, p. 76–88.

Hemmelmayr, Vera C; Cordeau, Jean-François e Crainic, Teodor Gabriel. (2012). An adaptive large neighborhood search heuristic for two-echelon vehicle routing problems arising in city logistics. *Computers & operations research*, v. 39, n. 12, p. 3215–3228.

Hertz, Alain e Widmer, Marino. (2003). Guidelines for the use of meta-heuristics in combinatorial optimization. *European Journal of Operational Research*, v. 151, n. 2, p. 247–252.

Ho, Wee-Kit; Lim, Andrew e Oon, Wee-Chong. (2001). Maximizing paper spread in examination timetabling using a vehicle routing method. *Tools with Artificial Intelligence, Proceedings of the 13th International Conference on*, p. 359–366, (2001).

Hu, Wenbin; Liang, Huanle; Peng, Chao; Du, Bo e Hu, Qi. (2013). A hybrid chaos-particle swarm optimization algorithm for the vehicle routing problem with time window. *Entropy*, v. 15, n. 4, p. 1247–1270.

Huang, Zhaohe e Geng, Kaifeng. (2013). Local search for dynamic vehicle routing problem with time windows. *Instrumentation and Measurement, Sensor Network and Automation (IMSNA), 2013 2nd International Symposium on*, p. 841–844. IEEE, (2013).

Jawarneh, Sana e Abdullah, Salwani. (2015). Sequential insertion heuristic with adaptive bee colony optimisation algorithm for vehicle routing problem with time windows. *PloS one*, v. 10, n. 7.

Jin, Fenghe e Fu, Yaping. (2010). Hybrid genetic algorithm for vehicle routing problem with time windows. *Management and Service Science (MASS), 2010 International Conference on*, p. 1–4, (2010).

Kahraman, Cengiz; Engin, Orhan; Kaya, İhsan e Öztürk, R Elif. (2010). Multiprocessor task scheduling in multistage hybrid flow-shops: A parallel greedy algorithm approach. *Applied Soft Computing*, v. 10, n. 4, p. 1293–1300.

Kang, Qinma; He, Hong e Song, Huimin. (1993). Task assignment in heterogeneous computing systems using an effective iterated greedy algorithm. *Journal of Systems and Software*, v. 84, n. 6, p. 985 – –992.

Kang, Qinma; He, Hong e Song, Huimin. (2011). Task assignment in heterogeneous computing systems using an effective iterated greedy algorithm. *Journal of Systems and Software*, v. 84, n. 6, p. 985 – –992.

Kang, Qinma; He, Hong e Wei, Jun. (2013). An effective iterated greedy algorithm for reliability-oriented task allocation in distributed computing systems. *Journal of parallel and distributed computing*, v. 73, n. 8, p. 1106 – –1115.

Karabulut, Korhan e Tasgetiren, M. Fatih. (2014)a. A variable iterated greedy algorithm for the traveling salesman problem with time windows. *Information Sciences*, v. 279, p. 338–395.

Karabulut, Korhan e Tasgetiren, M. Fatih. (2014)b. A variable iterated greedy algorithm for the traveling salesman problem with time windows. *Information Sciences*, v. 279, p. 383–395.

Keskin, Merve e Çatay, Bülent. (2016). Partial recharge strategies for the electric vehicle routing problem with time windows. *Transportation Research Part C: Emerging Technologies*, v. 65, p. 111–127.

Khorasanian, Danial; Hejazi, Seyyed Reza e Moslehi, Ghasem. (2013). Two-sided assembly line balancing considering the relationships between tasks. *Computers & Industrial Engineering*, v. 66, n. 4, p. 1096–1105.

Kilic, Sezgin. (2010). A possibilistic programming approach for the vehicle routing problem with time windows. *Fuzzy Systems (FUZZ), 2010 IEEE International Conference on*, p. 1–7. IEEE, (2010).

Kim, Ki Tae e Jeon, Geonwook. (2010). A vehicle routing problem considering traffic situation with time windows by using hybrid genetic algorithm. *IEEE Industrial Engineering (CIE-40)*, p. 1–6, (2010).

Kim, Yeo Keun; Kim, Yeongho e Kim, Yong Ju. (2000). Two-sided assembly line balancing: a genetic algorithm approach. *Production Planning & Control*, v. 11, n. 1, p. 44–53.

Kirci, Pinar. (2016). An optimization algorithm for a capacitated vehicle routing problem with time windows. *Sādhanā*, v. 41, n. 5, p. 519–529.

Kirk Patrick, S.; Gelatt, C. D. e Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, v. 220, n. 4598, p. 671–680.

Kirkpatrick, Scott; Gelatt, C Daniel e Vecchi, Mario P. (1983). Optimization by simulated annealing. *science*, v. 220, n. 4598, p. 671–680.

Küçükoğlu, İlker; Ene, Seval; Aksoy, Aslı e Öztürk, Nursel. (2015). A memory structure adapted simulated annealing algorithm for a green vehicle routing problem. *Environmental Science and Pollution Research*, v. 22, n. 5, p. 3279–3297.

Küçükoğlu, İlker e Öztürk, Nursel. (2015). An advanced hybrid meta-heuristic algorithm for the vehicle routing problem with backhauls and time windows. *Computers & Industrial Engineering*, v. 86, p. 60–68.

Li, Dashuang; Zhang, Chaoyong; Shao, Xinyu e Lin, Wenwen. (2016)a. A multi-objective tlbo algorithm for balancing two-sided assembly line with multiple constraints. *Journal of Intelligent Manufacturing*, v. 27, n. 4, p. 725–739.

Li, Haibing e Lim, Andrew. (2003). Local search with annealing-like restarts to solve the vrptw. *European journal of operational research*, v. 150, n. 1, p. 115–127.

Li, Jian; Li, Yang e Pardalos, Panos M. (2016)b. Multi-depot vehicle routing problem with time windows under shared depot resources. *Journal of Combinatorial Optimization*, v. 31, n. 2, p. 515–532.

Li, Zixiang; Tang, Qiuhua e Zhang, LiPing. (2017). Two-sided assembly line balancing problem of type i: Improvements, a simple algorithm and a comprehensive study. *Computers & Operations Research*, v. 79, p. 78–93.

Liao, Linling; Cai, Xiushan; Huang, Huadong e Liu, Yanhong. (2016). Vehicle routing with time windows based on two-stage optimization algorithm. *Control and Decision Conference (CCDC), 2016 Chinese*, p. 4741–4745, (2016).

Lin, Shih-Wei; Ying, Kuo-Ching e Wan, Shu-Yen. (2014). Minimizing the total service time of discrete dynamic berth allocation problem by an iterated greedy heuristic. *Scientific world journal*, v. 2014, p. 1–12.

Liu, Zhen-hua; Li, Nai-liang; Mi, Xiao-wei; Zhang, Bai-yu e Ma, Hong-zhan. (2010). Improvement research on vehicle routing problem with simultaneous delivery and pickup with time windows for barreled water. *IEEE 17Th International Conference on Industrial Engineering and Engineering Management*, p. 1347–1350, (2010).

Lozano, Manuel; Glover, Fred; García-Martínez, Carlos; Rodríguez, Francisco Javier e Martí, Rafael. (2014). abu search with strategic oscillation for the quadratic minimum spanning tree. *IIE Transactions*, v. 46, n. 4, p. 414–428.

Luis, Fanjul-Peyro e Rubén, Ruiz. (2010). Iterated greedy local search methods for unrelated parallel machine scheduling. *European Journal of Operational Research*, v. 207, n. 1, p. 55–69.

Luo, Wei e Fu, Zhuo. (2010). A variable neighborhood tabu search algorithm for the heterogeneous fleet vehicle routing problem with time windows. *Logistics Engineering and Intelligent Transportation Systems (LEITS), 2010 International Conference on*, p. 1–4. IEEE, (2010).

Ma, Weimin; Hu, Yonghuang e Zhou, Yang. (2010). An improved double-population genetic algorithm for vehicle routing problem with soft time windows. *EEE International Conference on Software Engineering and Service Sciences (ICSESS)*, p. 133–136, (2010).

Ma, Xin. (2010). Vehicle routing problem with time windows based on improved ant colony algorithm. *IEEE 2010 2nd International Conference on Information Technology and Computer Science (ITCS 2010)*, p. 94–97, (2010).

Mao, Yong e Deng, Yanfang. (2010). Solving vehicle routing problem with time windows with hybrid evolutionary algorithm. *EEE 2010 Second Global Congress on Intelligent Systems (GCIS)*, p. 335–339, (2010).

Marcelo, Seido Nagano; Hugo, Hissashi Miyata e Daniella, Castro Araujo. (2015). A constructive heuristic for total flowtime minimization in a no-wait flowshop with sequence-dependent setup times. *Journal of Manufacturing Systems*, v. 36, p. 224–230.

McNabb, Marcus E; Weir, Jeffery D; Hill, Raymond R e Hall, Shane N. (2015). Testing local search move operators on the vehicle routing problem with split deliveries and time windows. *Computers & Operations Research*, v. 56, p. 93–109.

Melián-Batista, Belén; De Santiago, Alondra; AngelBello, Francisco e Alvarez, Ada. (2014). A bi-objective vehicle routing problem with time windows: A real case in tenerife. *Applied Soft Computing*, v. 17, p. 140–152.

Mengzhu, Shi; Qingguo, Lin; Qiang, Zhang e Huanhong, Zhao. (2011). Hybrid genetic tabu search simulated annealing algorithm and its application in vehicle routing problem with time windows. *Artificial Intelligence, Management Science and Electronic Commerce (AIMSEC), 2011 2nd International Conference on*, p. 4022–4025. IEEE, (2011).

Mladenović, Nenad e Hansen, Pierre. (1997). Variable neighborhood search. *Computers and Operations Research*, v. 24, n. 11, p. 1097–1100.

Mokhtari, Hadi. (2015). Designing an efficient bi-criteria iterated greedy heuristic for simultaneous order scheduling and resource allocation: a balance between cost and lateness measures. *Neural Computing and Applications*, v. 26, n. 5, p. 1085–1101.

Nalepa, Jakub e Blocho, Miroslaw. (2016). Adaptive memetic algorithm for minimizing distance in the vehicle routing problem with time windows. *Soft Computing*, v. 20, n. 6, p. 2309–2327.

Nguyen, Viet Anh; Jiang, Jun; Ng, Kien Ming e Teo, Kwong Meng. (2016). Satisficing measure approach for vehicle routing problem with time windows under uncertainty. *European Journal of Operational Research*, v. 248, n. 2, p. 404–414.

Niu, Yunyun; He, Juanjuan; Wang, Zhongyu e Xiao, Jianhua. (2014). A p-based hybrid evolutionary algorithm for vehicle routing problem with time windows. *Mathematical Problems in Engineering*, v. 2014.

Nucamendi, S; Cardona-Valdes, Y e Acosta, F Angel-Bello. (2015). Minimizing customers waiting time in a vehicle routing problem with unit demands. *Journal of Computer and Systems Sciences International*, v. 54, n. 6, p. 866– – 881.

Ohlmann, Jeffrey W e Thomas, Barrett W. (2007). A compressed-annealing heuristic for the traveling salesman problem with time windows. *INFORMS Journal on Computing*, v. 19, n. 1, p. 80–90.

Ombuki, Beatrice; Ross, Brian J e Hanshar, Franklin. (2006). Multi-objective genetic algorithms for vehicle routing problem with time windows. *Applied Intelligence*, v. 24, n. 1, p. 17–30.

Ovacik, Irfan M e Uzsoy, Reha. (1995). Rolling horizon procedures for dynamic parallel machine scheduling with sequence-dependent setup times. *International journal of production research*, v. 33, n. 11, p. 3173 – –3192.

Özbakır, Lale e Tapkan, Pınar. (2011). Bee colony intelligence in zone constrained two-sided assembly line balancing problem. *Expert Systems with Applications*, v. 38, n. 9, p. 11947–11957.

Özcan, Uğur e Toklu, Bilal. (2009). A tabu search algorithm for two-sided assembly line balancing. *The International Journal of Advanced Manufacturing Technology*, v. 43, n. 7, p. 822–829.

Pace, Shannon; Turky, Ayad; Moser, Irene e Aleti, Aldeida. (2015). Distributing fibre boards: A practical application of the heterogeneous fleet vehicle routing problem with time windows and three-dimensional loading constraints. *Procedia Computer Science*, v. 51, p. 2257–2266.

Pan, Quan-Ke; Gao, Liang; Li, Xin-Yu e Gao, Kai-Zhou. (2017)a. Effective metaheuristics for scheduling a hybrid flowshop with sequence-dependent setup times. *Applied Mathematics and Computation*, v. 303, p. 89–112.

Pan, Quan-Ke; Ruiz, Rubén e Alfaro-Fernández, Pedro. (2017)b. Iterated search methods for earliness and tardiness minimization in hybrid flowshops with due windows. *Computers & Operations Research*, v. 80, p. 50–60.

Pan, Quan-Ke; Wang, Ling e Zhao, Bao-Hua. (2008). An improved iterated greedy algorithm for the no-wait flow shop scheduling problem with makespan criterion. *The International Journal of Advanced Manufacturing Technology*, v. 38, n. 7, p. 778–786.

Pedraza, Juan A; García-Martínez, Carlos; Cano, Alberto e Ventura, Sebastián. (2014). Classification rule mining with iterated greedy. Polycarpou, M; DeCarvalho, ACPLF; Pan, JS; Wozniak, M; Quintian, H e Corchado, E, editors, *International Conference on Hybrid Artificial Intelligence Systems*, volume 8480 of *Polycarpou, M and DeCarvalho, ACPLF and Pan, JS and Wozniak, M and Quintian, H and Corchado, E*, p. 585–595, (2014).

Penna, Puca Huachi Vaz; Afsar, H Murat; Prins, Christian e Prodhon, Caroline. (2016). A hybrid iterative local search algorithm for the electric fleet size and mix vehicle routing problem with time windows and recharging stations. *IFAC-PapersOnLine*, v. 49, n. 12, p. 955–960.

Pérez-Rodríguez, Ricardo e Hernández-Aguirre, Arturo. (2016). Simulation optimization for the vehicle routing problem with time windows using a bayesian network as a probability model. *The International Journal of Advanced Manufacturing Technology*, v. 85, n. 9–12, p. 2505–2523.

Pierre, Djamalladine Mahamat e Zakaria, Nordin. (2017). Stochastic partially optimized cyclic shift crossover for multi-objective genetic algorithms for the vehicle routing problem with time-windows. *Applied Soft Computing*, v. 52, p. 863–876.

Potvin, Jean-Yves e Rousseau, Jean-Marc. (1993). A parallel route building algorithm for the vehicle routing and scheduling problem with time windows. *European Journal of Operational Research*, v. 66, n. 3, p. 331–340.

Prais, Marcelo e Ribeiro, Celso C. (2000). Reactive grasp: An application to a matrix decomposition problem in tdma traffic assignment. *Informs journal on computing*, v. 12, n. 3, p. 164–176.

Pureza, Vitoria; Morabito, Reinaldo e Reimann, Marc. (2012). Vehicle routing with multiple deliverymen: Modeling and heuristic approaches for the vrptw. *European Journal of Operational Research*, v. 218, n. 3, p. 636– – 647.

Quevedo-Orozco, Dagoberto R e Ríos-Mercado, Roger Z. (2015). Improving the quality of heuristic solutions for the capacitated vertex p-center problem through iterated greedy local search with variable neighborhood descent. *Computers & Operations Research*, v. 62, p. 133–144.

Quoc, Nguyen Huu Bao e Anh, Duong Tuan. (2010). Combining constraint programming and tabu search to solve vehicle routing problem with time windows. *IEEE Communication Technologies, Research, Innovation, and Vision for the Future (RIVF)*, p. 1–4, (2010).

Qureshi, Ali Gul; Taniguchi, Eiichi e Yamada, Tadashi. (2010). Exact solution for the vehicle routing problem with semi soft time windows and its application. *Procedia-Social and Behavioral Sciences*, v. 2, n. 3, p. 5931–5943.

Resende, Mauricio GC e Werneck, Renato F. (2004). A hybrid heuristic for the p-median problem. *Journal of heuristics*, v. 10, n. 1, p. 59–68.

Resende, MGC e Silva, RMA. (2013). Grasp: Procedimentos de busca gulosos, aleatórios, e adaptativos. *Metaheurística em Pesquisa Operacional*, v. 1, p. 20.

Ribas, Imma; Companys, Ramon e Tort-Martorell, Xavier. (2011). An iterated greedy algorithm for the flowshop scheduling problem with blocking. *Omega*, v. 39, n. 3, p. 293 – –301.

Ribas, Imma; Companys, Ramon e Tort-Martorell, Xavier. (2017). Effective metaheuristics for scheduling a hybrid flowshop with sequence-dependent setup times. *Expert Systems with Applications*, v. 303, p. 89–112.

Rodriguez, Francisco J; Lozano, Manuel; Blum, Christian e GarcíA-MartíNez, Carlos. (2013)a. An iterated greedy algorithm for the large-scale unrelated parallel machines scheduling problem. *Computers and operations research*, v. 40, n. 7, p. 1829–1841.

Rodriguez, Francisco J; Lozano, Manuel; Blum, Christian e GarcíA-MartíNez, Carlos. (2013)b. n iterated greedy algorithm for the large-scale unrelated parallel machines scheduling problem. *Computers and Operations Research*, v. 40, n. 7, p. 1829–1841.

Rubén, Ruiz e Stützle, Thomas. (2008). An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. *European Journal of Operational Research*, v. 187, n. 3, p. 1143–1159.

Rubén, Ruiz e Thomas, Stützle. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, v. 177, n. 3, p. 2033–2049.

Ruela, André Siqueira; Guimaraes, Frederico Gadelha; Oliveira, Ricardo Augusto Rabelo; Neves, Brayan; Amorim, Vicente Peixoto e Fraga, Larissa Maiara. (2013). A parallel hybrid genetic algorithm on cloud computing for the vehicle routing problem with time windows. *Systems, Man, and Cybernetics (SMC), 2013 IEEE International Conference on*, p. 2467–2472. IEEE, (2013).

Ruiz, Rubén e Stützle, Thomas. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, v. 177, n. 3, p. 2033–2049.

Ruiz, Rubén e Stützle, Thomas. (2008). An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. *European Journal of Operational Research*, v. 187, n. 3, p. 1143–1159.

Ruiz-Vanoye, Jorge A; Díaz-Parra, Ocotlán; Cocón, Felipe; Buenabad-Arias, Angeles e Saenz, Ana Canepa. (2012). A transgenic algorithm for the vehicle routing problem with time windows. *Nature and Biologically Inspired Computing (NaBIC), 2012 Fourth World Congress on*, p. 138–143. IEEE, (2012).

Savelsbergh, Martin WP. (1985). Local search in routing problems with time windows. *Annals of Operations research*, v. 4, n. 1, p. 285–305.

Schyns, M. (2015). An ant colony system for responsive dynamic vehicle routing. *European Journal of Operational Research*, v. 245, n. 3, p. 704 – 718.

Shirvani, Nargess; Ruiz, Rubén e Shadrokh, Shahram. (2014). Cyclic scheduling of perishable products in parallel machine with release dates, due dates and deadlines. *International Journal of Production Economics*, v. 279, p. 383–395.

Sifaleras, Angelo e Konstantaras, Ioannis. (2017). Variable neighborhood descent heuristic for solving reverse logistics multi-item dynamic lot-sizing problems. *Computers & Operations Research*, v. 78, p. 385–392.

Simaria, Ana S e Vilarinho, Pedro M. (2008). 2-antbal: An ant colony optimisation algorithm for balancing two-sided assembly lines. *Computers & Industrial Engineering*, v. 36, n. 5-6, p. 582–588.

Simaria, Ana S e Vilarinho, Pedro M. (2009). 2-antbal: An ant colony optimisation algorithm for balancing two-sided assembly lines. *Computers & Industrial Engineering*, v. 56, n. 2, p. 489–506.

Solomon, M. M. (1987). Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, v. 35, p. 254–265.

Tam, Vincent e Kwan, MC. (2004)a. Adapting the large neighborhood search to effectively solve pickup and delivery problems with time windows. *IEEE International Conference on Tools with Artificial Intelligence*, p. 519–525. IEEE, (2004)a.

Tam, Vincent e Kwan, MC. (2004)b. Improving the initialization and repair heuristics to effectively solve the pickup and delivery problems with time windows. *IEEE International Conference on Tools with Artificial Intelligence*, p. 519–525. IEEE, (2004)b.

Tan, Kay Chen; Lee, Loo Hay; Zhu, QL e Ou, Ke. (2001). Heuristic methods for vehicle routing problem with time windows. *Artificial intelligence in Engineering*, v. 15, n. 3, p. 281–295.

Tasgetiren, M. Fatih; Buyukdagli, Ozge; Kizilay, Damla e Karabulut, Korhan. (2013)a. A populated iterated greedy algorithm with inver-over operator for traveling salesman problem. Panigrahi, Bijaya Ketan; Suganthan, Ponnuthurai Nagaratnam; Das, Swagatam e Dash, Shubhransu Sekhar, editors, *Proceedings of the 4th International Conference on Swarm, Evolutionary, and Memetic Computing (SEMCCO 2013)*, volume 8297 of *Lecture Notes in Computer Science*, p. 1–12. Springer-Verlag New York, Inc., (2013)a.

Tasgetiren, M. Fatih; Kizilay, Damla; Pan, Quan-Ke e Suganthan, PN. (2017). Iterated greedy algorithms for the blocking flowshop scheduling problem with makespan criterion. *Computers & Operations Research*, v. 77, p. 111–126.

Tasgetiren, M. Fatih; Pan, Quan-Ke; Suganthan, Ponnuthurai N e Buyukdagli, Ozge. (2013)b. A variable iterated greedy algorithm with differential evolution for the no-idle permutation flowshop scheduling problem. *Computers & Operations Research*, v. 40, n. 7, p. 1729–1743.

Tlili, Takwa; Yahyaoui, Hiba e Krichen, Saoussen. (2016). An iterated variable neighborhood descent hyperheuristic for the quadratic multiple knapsack problem. Lee, R, editor, *Software engineering, artificial intelligence, networking and parallel/distributed computing*, volume 612 of *Studies in Computational Intelligence*, p. 245–251, (2016).

Vanchipura, Rajesh; Sridharan, R. e Babu, A. Subash. (2014). mprovement of constructive heuristics using variable neighbourhood descent for scheduling a flow shop with sequence dependent setup time. *Journal of manufacturing systems*, v. 33, n. 2, p. 65–75.

Vincent, F Yu e Lin, Shih-Wei. (2015). Iterated greedy heuristic for the time-dependent prize-collecting arc routing problem. *Computers & Industrial Engineering*, v. 90, p. 54–66.

Wang, Chao; Mu, Dong; Zhao, Fu e Sutherland, John W. (2015)a. A parallel simulated annealing method for the vehicle routing problem with simultaneous pickup and delivery and time windows. *Computers & Industrial Engineering*, v. 83, p. 111–122.

Wang, Wenjun. (2010). mproved genetic algorithm for vehicle routing problem with time windows. *International Conference on Intelligent Computing and Cognitive Informatics*, p. 203–206, (2010).

Wang, Zheng; Li, Ying e Hu, Xiangpei. (2015)b. A heuristic approach and a tabu search for the heterogeneous multi-type fleet vehicle routing problem with time windows and an incompatible loading constraint. *Computers & Industrial Engineering*, v. 89, p. 162–176.

Wei-Huai, Hsu e Tsung-Che, Chiang. (2012). A multiobjective evolutionary algorithm with enhanced reproduction operators for the vehicle routing problem with time windows. *Evolutionary Computation (CEC), 2012 IEEE Congress on*, p. 1–8. IEEE, (2012).

Xu, Yingcheng; Wang, Li e Yang, Yuexiang. (2013). Dynamic vehicle routing using an improved variable neighborhood search algorithm. *Journal of Applied Mathematics*, v. 2013.

Yan, Shangyao; Chu, James C.; Hsiao, Fei-Yen e Huang, Han-Jheng. (2015). A planning model and solution algorithm for multi-trip split-delivery vehicle routing and scheduling problems with time windows. *Computers & Industrial Engineering*, v. 87, p. 383–393.

Yang, Hai. (2013). Study on application of improved chaos ant colony algorithm in vehicle routing problem with time windows. *Innovative Computing Technology (INTECH), 2013 Third International Conference on*, p. 379–382. IEEE, (2013).

Yang, Jun e Sun, Hao. (2015). Battery swap station location-routing problem with capacitated electric vehicles. *Computers & Operations Research*, v. 55, p. 217–232.

Yassen, Esam Taha; Ayob, Masri; Nazri, Mohd Zakree Ahmad e Sabar, Nasser R. (2012). Multi-parent insertion crossover for vehicle routing problem with time windows. *Data Mining and Optimization (DMO), 2012 4th Conference on*, p. 103–108. IEEE, (2012).

Ying, Kuo-Ching. (2008). olving non-permutation flowshop scheduling problems by an effective iterated greedy heuristic. *The International Journal of Advanced Manufacturing Technology*, v. 38, n. 3, p. 148– – 354.

Ying, Kuo-Ching e Cheng, Hui-Miao. (2010). Dynamic parallel machine scheduling with sequence-dependent setup times using an iterated greedy heuristic. *Expert Systems with Applications*, v. 37, n. 4, p. 2848 – –2852.

Ying, Kuo-Ching e Lin, Shih-Wei. (2007). Multi-heuristic desirability ant colony system heuristic for non-permutation flowshop scheduling problems. *The International Journal of Advanced Manufacturing Technology*, v. 33, n. 7-8, p. 793– –802.

Ying, Kuo-Ching; Lin, Shih-Wei e Huang, Chien-Yi. (2009). Sequencing single-machine tardiness problems with sequence dependent setup times using an iterated greedy heuristic. *Expert Systems with Applications*, v. 36, n. 3, p. 7087– – 7092.

Yu, Vincent F. e Lin, Shih-Wei. (2015). Iterated greedy heuristic for the time-dependent prize-collecting arc routing problem. *Computers and Industrial Engineering*, v. 90, p. 54–66.

Yuan, Biao; Zhang, Chaoyong e Shao, Xinyu. (2015). A late acceptance hill-climbing algorithm for balancing two-sided assembly lines with multiple constraints. *Journal of Intelligent Manufacturing*, v. 26, n. 1, p. 159–168.

Zeng, Zhizhong; Yu, Xinguo; He, Kun; Huang, Wenqi e Fu, Zhanghua. (2016). Iterated tabu search and variable neighborhood descent for packing unequal circles into a circular container. *European journal of operational research*, v. 250, n. 2, p. 615–627.

Zhenping, Li; Fei, Zhao e Hongwei, Liu. (2014). Intelligent water drops algorithm for vehicle routing problem with time windows. *Service Systems and Service Management (ICSSSM), 2014 11th International Conference on*, p. 1–6. IEEE, (2014).

Zhou, Ying e Wang, Jiahai. (2015). A local search-based multiobjective optimization algorithm for multiobjective vehicle routing problem with time windows. *IEEE Systems Journal*, v. 9, n. 3, p. 1100– – 1113.

Zitzler, Eckart; Laumanns, Marco; Thiele, Lothar e others,. Spea2: Improving the strength pareto evolutionary algorithm, (2001).