Universidade Federal da Bahia
Instituto de Computação
Bacharelado em Sistemas de Informação

Trabalho de Conclusão de Curso

Meta-heurísticas hibridizadas com variações de busca local aplicadas ao Problema do Caixeiro Viajante com Coleta de Prêmios

Matheus Carvalho do Carmo

Salvador - Bahia

29 de Novembro de 2023

Meta-heurísticas hibridizadas com variações de busca local aplicadas ao Problema do Caixeiro Viajante com Coleta de Prêmios

Matheus Carvalho do Carmo

Trabalho de Conclusão de Curso apresentado como requisito parcial para obtenção do título de Bacharel em Sistemas de Informação.

Orientador(a): Prof. Dr. Islame Felipe da Costa Fernandes.

Salvador - Bahia
29 de Novembro de 2023

Ficha catalográfica elaborada pela Biblioteca Universitária de Ciências e Tecnologias Prof. Omar Catunda, SIBI – UFBA.

C287	Carmo, Matheus Carvalho do
	Meta-heurísticas hibridizadas com variações de busca local aplicadas ao do problema do caixeiro viajante com coleta de prêmios/ Matheus Carvalho Carmo. – Salvador, 2023.
	182 f.: il.color.
	Orientador: Prof. Dr. Islame Felipe da Costa Fernandes.
	Trabalho de Conclusão de Curso (Graduação)– Universidade Federal da Bahia. Instituto de Computação, 2023.
	1. Otimização. 2. Computação. 3. Algoritmos Híbridos. I. Fernandes, Islame Felipe da Costa. II. Universidade Federal da Bahia. III. Título.
	CDU 004

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DA BAHIA
INSTITUTO DE COMPUTAÇÃO
COLEGIADO DO CURSO DE SISTEMAS DE INFORMAÇÃO
Av. Milton Santos s/n – Campus Universitário de Ondina, Ondina – Salvador- Bahia
CEP 40170-110

"META-HEURÍSTICAS HIBRIDIZADAS COM VARIAÇÕES DE BUSCA LOCAL APLICADAS AO PROBLEMA DO CAIXEIRO VIAJANTE COM COLETA DE PRÊMIOS"

Matheus Carvalho do Carmo

Trabalho de Conclusão de curso apresentado como requisito parcial para obtenção do Título de Bacharel em Sistemas de Informação.

Banca Examinadora:

Prof. Dr. Islame Felipe da Costa Fernandes (DCC/IC) (Orientador)

Prof. Dr. Rafael Augusto de Melo (DCC/IC)

Prof. Dr. Roberto Freitas Parente (DCI/IC)

À minha família e amigos que me deram todo suporte
e todos professores que fizeram parte da minha trajetória.

Agradecimentos

Agradeço profundamente meus pais, que sempre me sustentaram com amor, dedicação e se esforçaram ao máximo para me proporcionar uma educação de qualidade. A eles, minha eterna gratidão por cada sacrifício, por cada ensinamento e por acreditarem incessantemente em mim, mesmo nos momentos mais difíceis.

Estendo minha gratidão a toda minha família, que direta ou indiretamente influenciou minha trajetória e esteve ao meu lado em cada etapa desta jornada.

À minha irmã e aos meus amigos, por estarem sempre presentes, nos momentos de alegria e nas adversidades, e por serem fonte constante de apoio e motivação.

Um agradecimento especial aos meus professores, que não apenas transmitiram conhecimento, mas também moldaram meu caráter e minha perspectiva de mundo. Agradeço por cada lição, por cada desafio proposto e por me prepararem não só para a vida acadêmica, mas para a vida como um todo.

Minha profunda gratidão ao meu orientador, Islame Felipe da Costa Fernandes. Sua paciência, sabedoria e orientação foram fundamentais para a realização deste trabalho. Agradeço por acreditar em meu potencial e por me guiar nesses últimos semestres.

Resumo

O Problema do Caixeiro Viajante com Coleta de Prêmios (PCVCP) é uma variante do tradicional Problema do Caixeiro Viajante (PCV). No PCVCP, o caixeiro percorre um ciclo hamiltoniano em um sub-conjunto de vértices e coleta um prêmio em cada vértice visitado. O objetivo é minimizar a soma da distância percorrida e das penalidade associadas aos vértices não visitadas, sujeito a coletar uma quantidade mínima de prêmios. O PCVCP pertence à classe NP-difícil e possui aplicações em situações teóricas e práticas, tais como produção de lâminas e tiras de aço. Nos últimos anos, pesquisas têm demonstrado o potencial de meta-heurísticas híbridas em encontrar soluções de alta qualidade para problemas NP-difíceis. Tais técnicas combinam e exploram as melhores características de meta-heurísticas individuais. O objetivo deste trabalho é desenvolver meta-heurísticas híbridas para o PCVCP. O foco é hibridizar variações de busca local com algoritmos genéticos e GRASP. Foram desenvolvidos oito algoritmos híbridos, considerando quatro variações de busca local: 2-opt, VNS, VND, e uma combinação de VNS-VND. Os resultados obtidos revelam perspectivas significativas sobre o desempenho e as características das meta-heurísticas hibridizadas, com destaque para o algoritmo memético com VNS-VND, cuja análise sugere potenciais benefícios em termos de qualidade da solução e tempo de execução.

Palavras-Chaves: Problema do Caixeiro Viajante com Coleta de Prêmios, Algoritmos Híbridos, Meta-heurísticas, GRASP, VNS, VND.

Abstract

The Prize-Collecting Traveling Salesman Problem (PCTSP) is a variant of the traditional Traveling Salesman Problem (TSP). In PCTSP, the salesman traverses a Hamiltonian cycle within a subset of vertices and collects a prize at each visited vertex. The objective is to minimize the sum of the traveled distance and penalties associated with unvisited vertices, subject to collecting a minimum amount of prizes. PCTSP belongs to the NP-hard class and finds applications in both theoretical and practical scenarios, such as the production of steel sheets and strips. In recent years, research has demonstrated the potential of hybrid metaheuristics in finding high-quality solutions for NP-hard problems. Such techniques combine and exploit the best features of individual metaheuristics. The aim of this research is to develop hybrid metaheuristics for PCTSP. The focus is on hybridizing variations of local search with genetic algorithms and GRASP. Eight hybrid algorithms were developed, considering four local search variations: 2-opt, VNS, VND, and a combination of VNS-VND. The results obtained reveal significant insights into the performance and characteristics of hybrid metaheuristics, with particular emphasis on the memetic algorithm with VNS-VND, whose analysis suggests potential benefits in terms of solution quality and execution time.

Keywords: Prize-Collecting Traveling Salesman Problem, Hybrid Algorithms, Metaheuristics, GRASP, VNS, VND.'

Lista de Figuras

1 Exemplo de instância simétrica para o PCVCP 21
2 Exemplo de solução viável para uma instância simétrica para o PCVCP . . 21
3 Adaptado de Talbi (2002) 25

4 Demonstração do 2-opt. 28
5 Exemplo de Crossover de dois pontos para um Problema do Caixeiro Viajante. 37

6 Média de tempo de execução dos algoritmos para instâncias simétricas. . . 56
7 Média de tempo de execução dos algoritmos para instâncias assimétricas. . 57

Lista de Tabelas

1 Resumo das variáveis 19
2 Resumo dos parâmetros 19

3 Comparação dos p-valores entre os algoritmos para a instância 10.1. 42
4 Comparação dos p-valores entre os algoritmos para a instância 10.2. 43
5 Comparação dos p-valores entre os algoritmos para a instância 20.1. 43
6 Comparação dos p-valores entre os algoritmos para a instância 20.2. 44
7 Comparação dos p-valores entre os algoritmos para a instância 50.1. 45
8 Comparação dos p-valores entre os algoritmos para a instância 50.2. 45
9 Comparação dos p-valores entre os algoritmos para a instância 100.1. 46
10 Comparação dos p-valores entre os algoritmos para a instância 100.2. 46
11 Comparação dos p-valores entre os algoritmos para a instância 200.1. 47
12 Comparação dos p-valores entre os algoritmos para a instância 200.2. 47
13 Comparação dos p-valores entre os algoritmos para a instância 10.1. 49
14 Comparação dos p-valores entre os algoritmos para a instância 10.2. 49
15 Comparação dos p-valores entre os algoritmos para a instância 20.1. 50
16 Comparação dos p-valores entre os algoritmos para a instância 20.2. 50
17 Comparação dos p-valores entre os algoritmos para a instância 50.1. 51
18 Comparação dos p-valores entre os algoritmos para a instância 50.2. 51
19 Comparação dos p-valores entre os algoritmos para a instância 100.1. 52
20 Comparação dos p-valores entre os algoritmos para a instância 100.2. 52
21 Comparação dos p-valores entre os algoritmos para a instância 200.1. 53
22 Comparação dos p-valores entre os algoritmos para a instância 200.2. 53
23 Resultados dos testes de Mann-Whitney para instâncias simétricas. 54
24 Resultados dos testes de Mann-Whitney para instâncias assimétricas. 55

Lista de Algoritmos

1 Busca local 2-Opt 28
2 Variable Neighborhood Search (VNS) 31
3 Variable Neighborhood Descent (VND) 33
4 Algoritmo Memético 35
5 Algoritmo GRASP 39

Sumário

1 Introdução **13**

1.1 Contextualização e Motivação . . . 13

1.2 Objetivos . . . 15

1.2.1 Gerais . . . 15

1.2.2 Específicos . . . 15

1.3 Metodologia de Pesquisa . . . 15

1.4 Contribuição . . . 16

1.5 Estrutura do Trabalho . . . 16

2 Fundamentação Teórica **17**

2.1 Problema do caixeiro viajante . . . 17

2.2 Problema do Caixeiro Viajante com Coleta de Prêmios . . . 18

2.3 Trabalhos correlatos . . . 22

2.4 Hibridização de Meta-heurísticas . . . 23

3 Algoritmos desenvolvidos **27**

3.1 Busca Local . . . 27

3.1.1 2-opt . . . 27

3.1.2 VNS . . . 29

3.1.3 VND . . . 32

3.1.4 VNS-VND . . . 33

3.2 Algoritmo Memético . . . 34

3.3 GRASP . . . 38

4 Experimentos **40**

4.1 Metodologia . . . 40

4.1.1 Conjunto de instâncias . . . 40

4.1.2 Ajustes dos parâmetros . . . 41

4.1.3 Testes estatísticos . . . 41

4.2 Comparação dos algoritmos entre si . . . 41

4.2.1 Resultados para instâncias simétricas 42
4.2.2 Resultados para instâncias assimétricas 48
4.3 Comparação dos melhores algoritmos entre si 54
4.4 Comparação das médias de tempo entre algoritmos 56

5 Conclusão **58**

Capítulo 1

Introdução

Este capítulo apresenta o panorama geral do trabalho. A Seção 1.1 contextualiza o cenário e a importância do estudo. A Seção 1.2 apresenta os objetivos deste trabalho. A Seção 1.3 descreve a metodologia utilizada. A Seção 1.4 apresenta a contribuição para o campo acadêmico e prático. A Seção 1.5 apresenta a estrutura do trabalho.

1.1 Contextualização e Motivação

Em um mundo crescentemente interconectado, a otimização de rotas e planejamento logístico é crucial. Desde a entrega de encomendas até a programação de tarefas de manutenção em locais distantes, a necessidade de traçar rotas eficientes é fundamental não apenas para reduzir custos, mas também para melhorar a eficiência operacional.

O clássico Problema do Caixeiro Viajante (PCV) consiste em, dado um conjunto de cidades, encontrar a rota de menor custo que passe por todos as cidades uma única vez e retorna à cidade de partida. Embora seja muitas vezes discutido em um contexto teórico, o PCV tem aplicações práticas valiosas. Ele é frequentemente encontrado em logística, onde empresas de transporte e entregas buscam otimizar rotas para economizar tempo e recursos [40]. Além disso, o PCV aparece em áreas como planejamento de circuitos eletrônicos [31] e fiação de computador [40]. Diversas técnicas têm sido desenvolvidas para resolver o PCV, variando desde abordagens exatas [37], como o método de planos de corte, até heurísticas e meta-heurísticas, como algoritmos genéticos [1] [65] e otimização por colônia de formigas [61]. As heurísticas são estratégias aplicadas para encontrar soluções suficientemente boas em tempo hábil para um determinado problema. Já as meta-heurísticas estão um nível acima, e são adaptáveis para diferentes problemas. A escolha do método geralmente depende do tamanho da instância do problema e da precisão desejada para a solução. Uma revisão abrangente e atualizada destas e outras técnicas podem ser encontradas nos *surveys* [59, 58].

O Problema do Caixeiro Viajante com Coleta de Prêmios (PCVCP) ou *Prize Collecting Traveling Salesman Problem* (PCTSP) como é referido em inglês, é uma variante do clássico PCV. Esta variação foi introduzida por Egon Balas em 1989 [3], onde o objetivo é minimizar o custo total, que é a soma da distância percorrida e das penalidades por cidades não visitadas, enquanto se cumpre a meta de coletar um prêmio total mínimo nas cidades visitadas. Esta variante pode ser generalizada ao problema original, portanto, também se trata de um problema NP-difícil [22]. Alguns dos usos práticos do PCVCP são a produção de lâminas de aço em fábricas [3] [4] e no planejamento de um laminador de tiras a quente [46].

Além das abordagens tradicionais para o Problema do Caixeiro Viajante que são adaptáveis para o PCVCP, tem-se observado um crescente interesse nos algoritmos híbridos. Estes algoritmos combinam diferentes técnicas de otimização, como heurísticas, meta-heurísticas e até métodos exatos, visando explorar as vantagens complementares de cada abordagem.

Os algoritmos genéticos hibridizados com busca local são chamados de algoritmos meméticos. Esses algoritmos oferecem uma abordagem que combina a eficácia dos algoritmos genéticos com a precisão da busca local. O poder dos algoritmos meméticos reside em sua capacidade de explorar o espaço de soluções de uma forma mais ampla, ao mesmo tempo em que se aprofunda em regiões promissoras.

O GRASP é uma meta-heurística que se destaca pela combinação de uma fase de construção gulosa aleatória com uma fase de busca local. O GRASP é conhecido por oferecer um equilíbrio entre explorar novas soluções e aproveitar soluções já encontradas para melhorá-las. Os estudos de Melo e Martinhon [50] apontaram que a hibridização do GRASP com diferentes técnicas de busca local são robustas.

A busca local desempenha um papel fundamental de explotação nos algoritmos, permitindo a obtenção de uma solução melhor em uma determinada vizinhança de soluções. Heurísticas como 2-opt, VNS (*Variable Neighborhood Search*) e VND (*Variable Neighborhood Descent*) são técnicas de busca local que podem ser aplicadas para aprimorar soluções iniciais. O 2-opt consiste na estratégia de inverter vértices da solução para obter uma solução de menor custo. O VNS aplica diferentes estratégias de vizinhança, onde uma pequena parte da solução é alterada a fim de escapar de um valor ótimo local. O VND é uma variante do VNS onde as estratégias aplicadas são variações de busca local.

Este trabalho investiga oito algoritmos híbridos aplicados ao PCVCP, quatro destes algoritmos consistem em hibridização de algoritmo genético com variações de busca local, os outros quatro consistem no GRASP com variações de busca local. Blum et. al. [8] faz um levantamento sobre essas e outras meta-heurísticas híbridas utilizadas em otimização combinatória.

1.2 Objetivos

A Seção 1.2.1 apresenta os objetivos gerais do trabalho e a Seção 1.2.2 os objetivos específicos.

1.2.1 Gerais

Avaliar a eficácia de meta-heurísticas hibridizadas com variações de busca local aplicadas ao Problema do Caixeiro Viajante com Coleta de Prêmios.

1.2.2 Específicos

1. Desenvolver quatro algoritmos meméticos: memético com 2-opt, memético com VNS, memético com VND e memético com VNS e VND.
2. Adaptar e desenvolver quatro algoritmos GRASP: GRASP com 2-opt, GRASP com VNS, GRASP com VND e GRASP com VNS e VND.
3. Analisar a eficácia e eficiência de custo e tempo entre algoritmos desenvolvidos.

1.3 Metodologia de Pesquisa

Neste trabalho, adotamos uma abordagem metodológica focada na hibridização de meta-heurísticas e no estudo aprofundado do estado da arte referente ao PCVCP.

A fase inicial da pesquisa consistiu em uma revisão da literatura dos temas relacionados a este trabalho. Esse estudo foi crucial para compreender as abordagens existentes, suas limitações e oportunidades para inovações. Analisamos diferentes técnicas e algoritmos que foram previamente propostos, com atenção especial às meta-heurísticas e suas possíveis combinações.

Com base no conhecimento adquirido do estado da arte, focamos na hibridização de meta-heurísticas na resolução do PCVCP. Ao hibridizar meta-heurísticas com estratégias de busca local, esperávamos alcançar uma sinergia que permitiria uma exploração mais eficiente do espaço de soluções e, consequentemente, resultados superiores.

Após a fase de desenvolvimento, os algoritmos híbridos propostos foram implementados e testados. Foram utilizadas instâncias disponíveis na literatura do PCVCP para avaliar a eficácia dos nossos algoritmos, comparando-os entre si com respeito à qualidade de solução e tempo de processamento. Empregamos os testes estatísticos de Kruskal-Wallis e Mann-Whitney afim de obter relevância estatística nas análises.

1.4 Contribuição

A principal contribuição deste trabalho é a proposição e análise de meta-heurísticas hibridizadas com busca local aplicadas ao PCVCP. O trabalho se diferencia propondo o estudo da eficácia de custo e tempo do algoritmo memético combinado com as técnicas de busca local VNS e VND. Ao analisar as características dos algoritmos meméticos com as capacidades de refinamento das buscas VNS e VND, este trabalho demonstra o poder das técnicas híbridas no contexto da otimização combinatória.

1.5 Estrutura do Trabalho

Este trabalho está dividido em 5 capítulos, O Capítulo 2 apresenta a fundamentação teórica. O Capítulo 3 apresenta os algoritmos que foram desenvolvidos para este trabalho. O Capítulo 4 disserta sobre os experimentos e testes feitos a fim de comparar os algoritmos. O Capítulo 5 apresenta as considerações finais, suas contribuições e possíveis trabalhos futuros.

Capítulo 2

Fundamentação Teórica

Este capítulo apresenta a base teórica do problema e da abordagem analisada neste trabalho. A Seção 2.1 examina o Problema do Caixeiro Viajante, explorando suas origens e literatura recente. A Seção 2.2 define formalmente o Problema do Caixeiro Viajante com Coleta de Prêmios, apresentando seu modelo matemático. A Seção 2.3 apresenta o estado da arte do Problema do Caixeiro Viajante com Coleta de Prêmios. A Seção 2.4 apresenta a base teórica para técnicas de hibridização de meta-heurísticas.

2.1 Problema do caixeiro viajante

O Problema do Caixeiro Viajante (PCV), ou *Traveling Salesman Problem* (TSP) [62, 39], é um dos problemas estudados em otimização combinatória e pesquisa operacional. O PCV consiste em determinar a rota de menor custo que, partindo de uma cidade de origem, percorre todos as cidades do conjunto e retorna à cidade inicial, garantindo que cada cidade seja visitada uma única vez.

Formalmente, consideramos um grafo completo não direcionado ponderado $G = (V, A)$, onde V é o conjunto de vértices e A é o conjunto de arestas entre os vértices. Considera-se que cada aresta $(i, j) \in A$ possui um peso c_{ij} representando a distância ou custo entre os vértices i e j. O objetivo é encontrar um ciclo hamiltoniano de custo mínimo em G, onde o custo consiste no somatório dos pesos das arestas do ciclo. O PCV tem aplicabilidade em diversas áreas, como logística, planejamento e ciência da computação [49]. Este problema é provado ser um problema NP-difícil [22], o que significa que não se conhece a solução em tempo polinomial.

No caso do Problema do Caixeiro Viajante Assimétric, o grafo é direcionado, ou seja, a matriz de custos não necessariamente obedece à propriedade $c_{ij} = c_{ji}$ para todas os vértices i e j. Isso significa que o custo da aresta do vértice i para o vértice j pode ser diferente do custo de viajar de j para i. Essa característica torna o Problema do Caixeiro

Viajante Assimétrico particularmente desafiador, pois a ausência de simetria na matriz de custos pode levar a uma combinação significativamente maior de rotas possíveis a serem consideradas.

O PCV tem sido abordado por diversos algoritmos, que podem ser categorizados em exatos e heurísticos [37]. Os métodos exatos, como o método *branch and bound* [5] e o método de programação dinâmica [12], buscam a solução ótima. Já os algoritmos heurísticos, que englobam o algoritmo genético [35], o algoritmo de colônia de formigas [70], o algoritmo de Lin-Kernighan [43], o algoritmo de recozimento simulado [68], e a otimização por enxame de partículas [67], bem como suas versões híbridas [10, 64, 41], oferecem soluções em um tempo computacional viável. Além disso, certos algoritmos heurísticos, como o de colônia de formigas, podem ter convergência lenta e tender a soluções locais, comprometendo a eficiência da otimização [70].

O Algoritmo Genético [26] é uma das ferramentas utilizáveis na resolução do PCV e suas variações. Em estudos recentes, Lo et al. [45] adaptaram o algoritmo para o *Multiple TSP*. Os estudos de operadores como o *crossover* proposto em [34] e o *Edge Assembly Crossover* de [33], evidencia o constante aprimoramento na área. Além disso, estratégias de inicialização, exemplificadas pelo método de permutação gananciosa [44] e uma técnica inspirada no algoritmo *k-means* [17], demonstram a busca por eficiência. A vasta literatura, incluindo contribuições notáveis como [54, 30] refletem a contínua evolução e interesse no PCV e no Algoritmo Genético, tratando de variações multi-objetivos e predição de trajetória.

2.2 Problema do Caixeiro Viajante com Coleta de Prêmios

No Problema do Caixeiro Viajante com Coleta de Prêmios (PCVCP), o caixeiro viajante é responsável por coletar um prêmio em cada cidade visitada e pagar uma penalidade para cada cidade não visitada, além dos custos de deslocamento entre as cidades. O objetivo é minimizar a soma dos custos da viagem e das penalidades, com uma rota que permita coletar um prêmio mínimo pré-estabelecido. Cada cidade só pode ser visitada no máximo uma vez, e a rota deve terminar na mesma cidade de origem.

A formulação apresentada nesta seção foi proposta por [3]. Seja $G = (V, A)$ um grafo completo direcionado ponderado, onde V é o conjunto de nós representando as cidades e A é o conjunto de arestas representando os percursos entre as cidades. Denota-se por (i, j) a aresta que sai do vértice i e chega no vértice j. O caixeiro viaja entre as cidades i e j com um custo c_{ij}.

Seja y_i igual a 1 se a cidade i é incluída na rota e 0 caso contrário. Seja x_{ij} o

vetor de incidência da aresta (i, j) na rota, ou seja, $x_{ij} = 1$ se (i, j) está na solução, e $x_{ij} = 0$ caso contrário. O caixeiro recebe um prêmio b_i em cada cidade i que ele visita e é penalizado por uma quantia p_i por cada cidade i que ele não visita. O somatório dos prêmios coletados pelo caixeiro deve ser maior ou igual que b_o. Seja s é o ponto de partida e chegada do caixeiro. Seja u_i uma variável que recebe um valor inteiro positivo, que denota a posição do vértice i na rota. Por convenção, $u_s = 1$.

A Tabela 1 resume as variáveis e a Tabela 2 resume os parâmetros do problema.

Tabela 1: Resumo das variáveis

Notação	Significado
x_{ij}	1 se a aresta (i, j) está na rota, 0 caso contrário
y_i	1 se a cidade i está na rota, 0 caso contrário
u_i	Variável inteira positiva denotando a posição do vértice i na rota
s	Vértice de partida e chegada do caixeiro

Tabela 2: Resumo dos parâmetros

Notação	Significado
V	Conjunto de nós representando as cidades
A	Conjunto de arestas representando os percursos entre as cidades
c_{ij}	Custo de viagem entre as cidades i e j
p_i	Penalidade por não visitar a cidade i
b_i	Prêmio recebido ao visitar a cidade i
b_o	Quantidade mínima de prêmios a ser coletada

A formulação pode ser escrita como:

$$\min \sum_{i \in V} \sum_{j \in V j \neq i} c_{ij} x_{ij} + \sum_{i \in V} p_i (1 - y_i) \tag{1}$$

sujeito a:

$$\sum_{j \in V - \{i\}} x_{ij} = y_i \quad \forall i \in V \tag{2}$$

$$\sum_{i \in V - \{j\}} x_{ij} = y_j \quad \forall j \in V \tag{3}$$

$$\sum_{i \in V} b_i y_i \geq b_o \tag{4}$$

$$y_s = 1 \tag{5}$$

$$\sum_{i \in V - \{j\}} x_{ij} - \sum_{i \in V - \{j\}} x_{ji} = 0 \quad \forall j \in V - \{s\} \tag{6}$$

$$1 \leq u_i \leq |V| \quad \forall i \in V \tag{7}$$

$$u_s = 1 \tag{8}$$

$$u_i - u_j + (|V| - 1) x_{ij} \leq |V| - 2 \tag{9}$$

$$y_i \in \{0, 1\}, \ i \in V; \ x_{ij} \in \{0, 1\}, \ (i, j) \in A \tag{10}$$

A expressão (1) define a função objetivo de minimização, sendo a soma do custo total da rota a partir das arestas incidentes e das penalidades por cidade não visitada. Expressões de (2) a (10) definem as restrições do PCVCP. A restrição (2) garante que para todo vértice i pertencente a solução, apenas uma aresta da solução é iniciada em i. Restrição (3) garante que para todo vértice j pertencente à solução, apenas uma aresta da solução é terminada em j. Restrição (4) garante que o somatório de bônus coletado seja maior ou igual ao valor mínimo b_o preestabelecido. As restrições 5 à 9 foram baseadas no trabalho de Miller et. al. [51] e não estão presentes na formulação de Balas [3], mas é comum em trabalhos envolvendo o PCV adição de restrições que eliminem possíveis sub-rotas. Restrição (10) indica os tipos das variáveis.

A restrição (5) garante que s esteja na rota. A restrição (6) garante a continuidade do ciclo, ou seja, se o caixeiro chega a um vértice j, então ele deve sair de j. A restrição (7) define os limites para as variáveis u_i. A restrição (8) especifica que o vértice s deve

ser o primeiro da rota. A restrição (9) força que, se o caixeiro trafega na aresta (i, j) (ou seja, se $x_{ij} = 1$), então os vértices i e j diferem em uma posição na rota.

O grafo da Figura 1 representa uma instância simétrica para o PCVCP, onde s é o ponto inicial e final, b é o bônus de cada vértice, p é a penalidade de cada vértice, e a cota mínima do bônus é 50. A Figura 2 representa uma solução viável para a instância da Figura 1. A linha vermelha representa a rota percorrida na solução, onde o caixeiro começa e termina no vértice s e coleta o bônus mínimo necessário, com um custo de rota equivalente a soma das distancia e da penalidade do vértice não visitado.

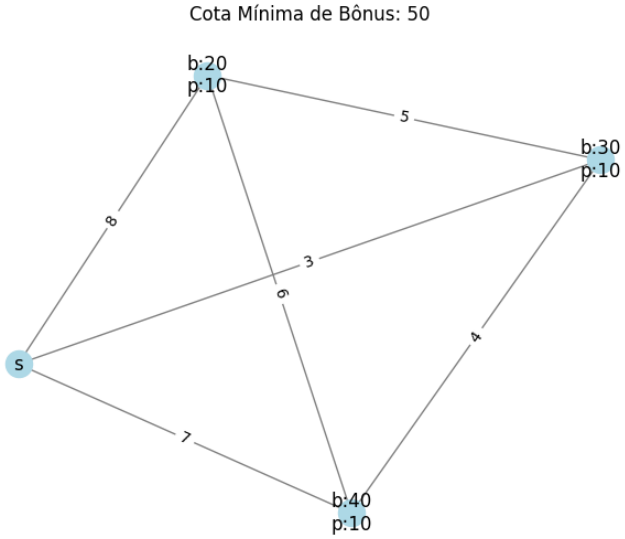

Figura 1: Exemplo de instância simétrica para o PCVCP

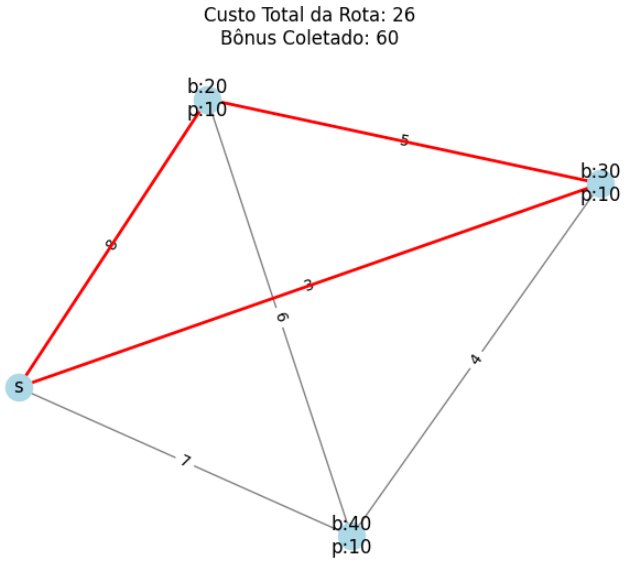

Figura 2: Exemplo de solução viável para uma instância simétrica para o PCVCP

2.3 Trabalhos correlatos

O PCVCP foi originalmente investigado por Balas e Martin [5] em 1985, focando na programação de produção de uma fábrica de laminação de aço. O desgaste dos rolos de laminação tornou crucial a ordenação das chapas de aço. O objetivo é selecionar blocos de aço de um inventário, atendendo a um requisito mínimo de peso, e sequenciá-los de forma eficiente. A sequência de processamento dos blocos é essencial devido ao impacto no desgaste dos rolos e outros fatores operacionais. O PCVCP ajuda a integrar a escolha dos blocos e a ordem de processamento, visando minimizar custos operacionais ou desgaste do equipamento.

Ao longo dos anos, várias pesquisas abordaram diferentes aspectos do PCVCP. Fischetti e Toth [21] propuseram novas formulações matemáticas para o problema, usando a Relaxação Lagrangeana. Na década de 1990, Dell'Amico et al. [16, 15] forneceram uma síntese dos resultados até então, introduzindo novos limites inferiores para variações do PCVCP. Em contraste, Lopez et al. [46] modelaram uma produção de laminação usando uma abordagem diferente do PCVCP, aplicando uma técnica chamada de Canibalização e utilizando conceitos da Busca Tabu.

Gomes et al. [29] e Melo e Martinhon [50] realizaram estudos comparativos entre diferentes versões de uma heurística baseada no método GRASP [19], combinado com os métodos VNS e VND [52]. O primeiro estudo mostrou que a versão com filtro obteve os melhores resultados, enquanto no segundo, foi proposto o conceito de distribuição de soluções progressivas (GRASP-Progressivo) como uma fase de construção, que apresentou resultados médios promissores. Os resultados desses estudos sugerem que a combinação de heurísticas com métodos de busca local pode ser uma abordagem promissora para resolver o problema do caixeiro viajante com coleta de prêmios.

Even e Shahar [6] consideraram uma variação do PCVCP com janelas de tempo. Pedro, Saldanha e Camargo [56], que desenvolveram uma heurística híbrida para o problema, tendo a fase de construção baseada no procedimento GENIUS [23] e a busca local 2-opt [42], mostrando melhor desempenho em comparação com Chaves et al. [13].

Mais recentemente, Gimadi e Tsidulko [24] apresentou algoritmos assintoticamente ótimos, e Chan et al. [11] apresentou um esquema de aproximação de tempo polinomial (PTAS) unificado para o PCVCP e para o *Steiner tree problem* [69].

Muitos problemas de otimização combinatória são variações ou extensões do PCVCP, com nuances que os distinguem uns dos outros. O Orienteering Problem (OP) e o Selective TSP (sTSP), por exemplo, têm como foco a maximização do valor coletado durante a rota, com um teto no custo total. Enquanto o OP foi inicialmente explorado por [27] e [28], o sTSP teve seus fundamentos estabelecidos por [38].

O Profitable Tour Problem (PTP), introduzido por [16], se assemelha ao PCVCP,

mas com uma característica distinta: não exige um valor mínimo a ser coletado ao longo do percurso. Isso significa que, no PTP, rotas que não visitam nenhum vértice são soluções aceitáveis. Assim, o valor ótimo do PTP pode se basear apenas nas penalidades dos vértices, ou optar por incluir o vértice raiz.

Ainda no universo das variações, temos o QTSP [2], que é essencialmente um PCVCP em que todos as penalidades são nulas, e variações mais complexas desta, como o *Quota travelling salesman problem with passengers, incomplete ride and collection time* [63] que incorpora passageiros, rotas incompletas e tempo de coleta ao problema. Já o *Resource Constrained Travelling Salesman Problem* (RCTSP), proposto por [57], adiciona uma camada extra de complexidade ao PCVCP ao limitar os recursos disponíveis durante a rota, visando sempre o menor custo possível.

2.4 Hibridização de Meta-heurísticas

Problemas de otimização combinatória surgem em diversos campos, desde o planejamento de produção até o gerenciamento de redes. Estes problemas são notáveis por sua formulação direta, mas solução desafiadora, sendo muitos classificados como NP-difíceis. A relevância prática desses problemas impulsionou o desenvolvimento de várias técnicas de solução, divididas em algoritmos exatos e heurísticos. Algoritmos exatos garantem encontrar a solução ótima, enquanto algoritmos heurísticos, são utilizados quando a rapidez é prioritária, oferecendo soluções de alta qualidade em tempos computacionais viáveis.

Na otimização por meta-heurísticas, os conceitos de exploração e explotação são fundamentais para entender como esses algoritmos navegam pelo espaço de busca para encontrar soluções ótimas ou satisfatórias. O avanço de estudos e pesquisas na área resultou em uma nova categoria de algoritmos como uma ferramenta na busca por soluções próximas do ótimo para problemas de otimização. Esses algoritmos são conhecidos como meta-heurísticas, um termo introduzido por Glover [25], derivado do prefixo grego "meta", indicando um nível além ou superior, e "heuriskein", que traduz a ideia de encontrar, conforme Blum e Roli [9].

Exploração refere-se à capacidade do algoritmo de investigar diversas regiões do espaço de busca. Isso significa olhar além do entorno imediato de soluções conhecidas para descobrir novas áreas que podem conter soluções melhores. Na prática, isso pode envolver a geração de soluções candidatas bastante diferentes das atuais. Meta-heurísticas que promovem a exploração ajudam a evitar que o algoritmo fique preso em ótimos locais.

Explotação, por outro lado, é o processo de intensificar a busca em torno de soluções promissoras. Isso significa fazer ajustes finos nas melhores soluções encontradas para alcançar o ótimo local ou global. Na prática, a explotação pode envolver o uso de busca local

para realizar pequenas alterações em uma solução candidata, refinando-a para melhorar sua qualidade.

Meta-heurísticas híbridas combinam diferentes estratégias de otimização para equilibrar entre exploração e explotação. Por exemplo, um algoritmo pode usar uma abordagem mais exploratória no início do processo de otimização para garantir uma boa diversificação e, em seguida, passar para estratégias mais explotatórias à medida que boas soluções são identificadas.

De acordo com Raidl [60], a hibridização algorítmica visa unir o melhor de diversas estratégias de otimização para capturar a sinergia entre elas. Embora híbridos possam ter eficácia variável dependendo do problema específico, certos tipos de hibridização têm se mostrado amplamente bem-sucedidos, fornecendo orientações valiosas para futuras inovações.

Essa abordagem sinérgica permite que as meta-heurísticas hibridizadas tirem proveito da diversificação global e da intensificação local, buscando equilibrar a exploração e explotação do espaço de busca. A literatura, incluindo trabalhos de referência como os de Talbi [66] e Blum e Roli [9], sugere que a hibridização pode levar a um desempenho significativamente melhorado em comparação com abordagens singulares, principalmente em problemas complexos e de alta dimensionalidade. Em Blum [8], a pesquisa evoluiu de uma abordagem orientada a algoritmos para uma mais orientada a problemas específicos, resultando na criação de algoritmos híbridos eficazes, fruto da combinação de diferentes técnicas de otimização, incluindo algoritmos exatos e meta-heurísticas.

A hibridização de meta-heurísticas também tem demonstrado ser valiosa no tratamento de problemas de otimização multiobjetivo. O trabalho de Fernandes [20] oferece uma revisão da literatura atual para problemas multiobjetivo, Em seu estudo fundamental sobre meta-heurísticas híbridas, Talbi [66] propõe uma taxonomia que destaca as questões de design e de implementação para o desenvolvimento de algoritmos híbridos. A Taxonomia é separada em duas partes: a hierárquica e a horizontal. A hierárquica, organiza os algoritmos por níveis. A horizontal categoriza as classes obtidas da hierarquia. Seu diagrama é apresentado na Figura 3.

Na primeira camada da divisão hierárquica, podem ser separadas entre baixo nível e alto nível. No baixo nível, existe uma relação de composição entre meta-heurísticas onde uma dada função de uma meta-heurística é executada por outra meta-heurística. No alto nível as meta-heurísticas são independentes e não há uma influência direta no funcionamento de uma sobre a outra. Na segunda camada existe a divisão entre revezamento e co-evolucionária. No revezamento, as meta-heurísticas são aplicadas sequencialmente e conectadas de forma que a saída de uma serve como entrada para outra. Na co-evolucionária, as meta-heurísticas atuam em paralelo, onde cada um realiza uma busca

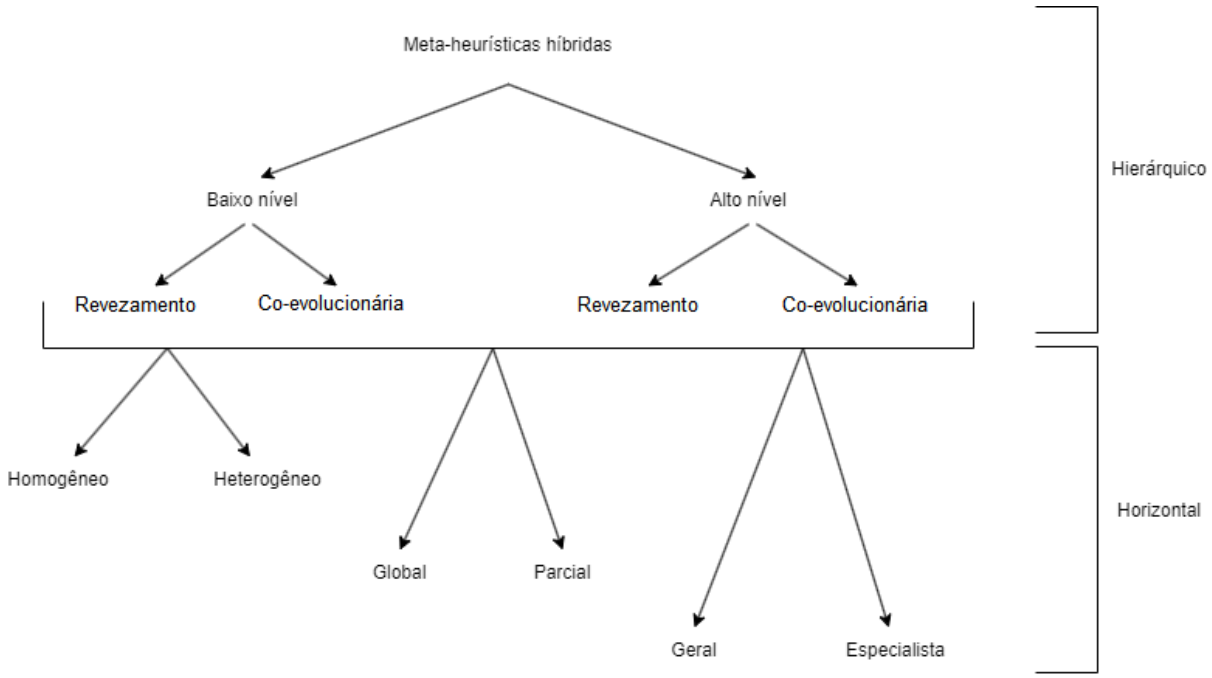

Figura 3: Adaptado de Talbi (2002)

em um espaço de solução diferente.

Existem então, quatro classes finais nessa hierarquia: baixo nível por revezamento, baixo nível co-evolucionária, alto nível por revezamento e alto nível co-evolucionária.

A hibridização de baixo nível por revezamento é aquela em que uma meta-heurística é inserida dentro de outra meta-heurística de solução única, ou seja, não populacional. A ideia é que a solução que está em um ótimo local obtido através da meta-heurística mais interna, sofra uma pertubação na meta-heurística mais externa para que possa procurar um ótimo local em outra vizinhança. Um clássico exemplo desse tipo de hibridização é a combinação de *simulated annealing* com busca local para resolver o problema do caixeiro viajante [48].

A hibridização de baixo nível co-evolucionária também possui uma meta-heurística inserida em outra, porém a meta-heurística mais externa é populacional, como por exemplo o algoritmo genético [55] ou a colonia de formigas [18]. O algoritmo populacional nessa hibridização tem a função de encontrar ótimos globais e variar as áreas que serão alvo de explotação da meta-heurística mais interna.

A hibridização de alto nível por revezamento é composta por meta-heurísticas independentes, conectadas em *pipeline*, onde a primeira meta-heurística é responsável pela exploração e fornece a entrada para a meta-heurística seguinte executar o processo de explotação.

A hibridização de alto nível co-evolucionária é composta por meta-heurísticas que funcionam de forma independente em paralelo, cada uma buscando um ótimo, porém, se

comunicando de forma colaborativa para obter melhores resultados. Talbi [66] exemplifica esse tipo de hibridização com meta-heurísticas populacionais conectadas de forma que elas podem repassar subpopulações entre si.

A classificação horizontal de hibridizações de Talbi [66] distingue-se em homogêneas ou heterogêneas, globais ou parciais e gerais ou especialistas. As homogêneas utilizam a mesma meta-heurística com diferentes parâmetros. Já as heterogêneas combinam diferentes meta-heurísticas, como em hibridizações que integram algoritmos genéticos e busca tabu. Outra distinção é entre hibridizações globais, onde todos os algoritmos exploram o espaço de pesquisa completo para uma exploração mais aprofundada, e parciais, onde o problema é dividido e cada sub-problema é abordado por um algoritmo dedicado, mantendo comunicação para cumprir as restrições e alcançar uma solução viável globalmente. Além disso, as hibridizações gerais envolvem diferentes algoritmos trabalhando no mesmo problema de otimização para encontrar uma solução única. Em contraste, as hibridizações especialistas combinam algoritmos que resolvem diferentes aspectos ou problemas distintos, contribuindo cada um para uma parte da solução final do sistema.

Capítulo 3

Algoritmos desenvolvidos

Este capítulo detalha os algoritmos desenvolvidos no decorrer deste trabalho. A Seção 3.1 apresenta os algoritmos de busca local. A Seção 3.2 apresenta o Algoritmo Memético, que combina busca local e algoritmos genéticos para explorar e explorar o espaço de soluções de maneira eficiente. A Seção 3.3 apresenta o GRASP, ou *Greedy Randomized Adaptive Search Procedure*, um algoritmo iterativo que constrói soluções de maneira aleatória, mas guiada por um critério de ganância.

3.1 Busca Local

Esta seção descreve os algoritmos de busca local que foram utilizados. A Seção 3.1.1 apresenta o 2-opt. A Seção 3.1.2 apresenta o VNS, ou *Variable Neighborhood Search*, que explora sistematicamente diferentes vizinhanças da solução atual em busca de um ótimo local. A Seção 3.1.3 apresenta o VND, ou *Variable Neighborhood Descent*, uma versão do VNS que realiza uma busca local intensiva em várias estruturas de vizinhança. Por fim, a Seção 3.1.4 apresenta o VNS-VND, que combina os elementos do VNS e VND.

3.1.1 2-opt

O algoritmo 2-opt [42] é uma técnica de busca local utilizada para melhorar rotas em problemas de roteamento, como o Problema do Caixeiro Viajante e suas variações. A estratégia envolve pegar dois vértices em uma rota e inverter os segmentos que os conectam. A Figura 4 ilustra um exemplo onde os vértices 1 e 5 são invertidos na rota. Se a rota resultante possuir um custo menor, a troca é aceita, caso contrário, é rejeitada. Este processo é repetido para todos os pares de vértices da rota até que não sejam possíveis mais melhorias. Esta técnica tem provado ser eficaz na melhoria de soluções iniciais e é frequentemente utilizada como parte de meta-heurísticas mais complexas.

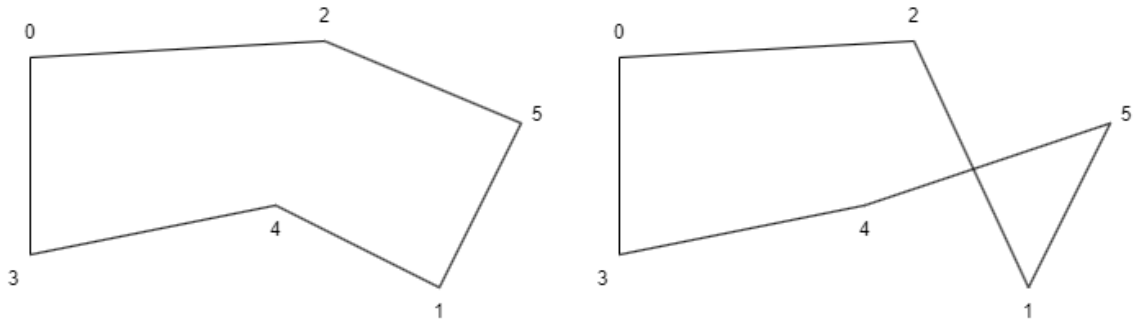

Figura 4: Demonstração do 2-opt.

Algorithm 1: Busca local 2-Opt

Input: *route*

Output: *route*

```
1  best_route ← route;
2  best_cost ← route_cost(best_route);
3  improved ← true;
4  best_route_len ← tamanho da best_route;
5  while improved do
6      improved ← false;
7      for i ← 1 to best_route_len − 2 do
8          for j ← i + 1 to best_route_len − 1 do
9              new_route ← swap_2(best_route, i, j);
10             new_cost ← route_cost(new_route);
11             if new_cost < best_cost then
12                 best_cost ← new_cost;
13                 best_route ← new_route;
14                 improved ← true;
15 return best_route;
```

o Algoritmo 1 detalha o procedimento de busca local com 2-opt. Linha 1 guarda a rota recebida como parâmetro em *best_route*. Linha 2 calcula o custo dessa rota e guarda seu valor como *best_cost*. Linha 3 define o valor inicial de *improved* como verdadeiro para que a execução possa entrar no laço da linha 5. A linha 4 calcula o tamanho da melhor rota e guarda o valor em *best_route_len*. Linha 5 marca o início do laço mais externo e define a condição de parada como *improved* sendo falso. Linha 6 altera o valor de *improved* para falso, para que ele só se torne verdadeiro quando for obtida uma rota melhor. Linhas 7 e 8 estabelecem uma iteração entre os nós da rota, de modo que possa ser trabalhado com dois nós i e j, considerando não repetir a mesma dupla de nós. Linha

9 aplica a troca de aresta sobre a *best_route* a partir de dois nós i e j e guarda esta nova rota em *new_route*. A linha 10 calcula o custo da *new_route* e guarda este valor em *new_cost*. A linha 11 checa se o valor de *new_cost* é menor que o valor de *best_cost*, caso verdadeiro, as linhas 12 a 14 são executadas. As linhas 12 e 13 guardam o valor de *new_cost* e *new_route* como *best_route* e *best_route* respectivamente. Linha 14 indica que houve melhoria na rota alterando o valor de *improved* para verdadeiro. Linha 15 retorna a melhor rota obtida, *best_route*.

3.1.2 VNS

Proposto inicialmente por Mladenovic e Hansen [52], o *Variable Neighborhood Search* (VNS) é uma meta-heurística que consiste na mudança sistemática de vizinhança durante a busca. Em vez de restringir a busca a uma única vizinhança, como é comum em muitos algoritmos de busca local, o VNS alterna entre diferentes vizinhanças a fim de escapar de ótimos locais e encontrar melhores soluções. O processo envolve três etapas principais: seleção de uma solução inicial, perturbação dessa solução utilizando uma vizinhança, e aplicação de uma busca local na solução perturbada. Se a nova solução for melhor, ela é aceita como a solução corrente. Este processo é repetido até que um critério de parada seja atingido.

No contexto dos algoritmos meta-heurísticos, as estratégias de vizinhança são cruciais para explorar o espaço de soluções de maneira eficiente. Estas estratégias permitem que a solução atual seja sutilmente alterada, originando uma solução vizinha. Para o presente trabalho, foram adotadas cinco estratégias de vizinhança, propostas pelo estudo de Chaves et. al. [13]. Estas estratégias foram escolhidas por sua capacidade de proporcionar uma ampla gama de modificações na rota. As vizinhanças diferem da 2-opt pela possibilidade de aumentar ou reduzir o tamanho da rota, proporcionando uma exploração mais abrangente no espaço de busca. As estratégias de vizinhança adotadas são detalhadas a seguir:

n_1 **: Remoção do Vértice com Maior Economia:** Usando o método DROP-Step [29], seleciona-se um vértice que faz parte da rota e apresenta a maior economia em caso de remoção. Este vértice é então removido da solução.

n_2 **: Adição do Vértice com Maior Economia:** Empregando o método ADD-Step [29], todos os vértices que ainda não estão na rota são avaliados, e aquele que proporciona a inserção mais econômica é selecionado e incluído na solução.

n_3 **: Troca de Dois Vértices:** Dois vértices que fazem parte da solução são selecionados e suas posições são trocadas entre si.

n_4 **: Remoção Aleatória de Vértice:** Um vértice que faz parte da solução é escolhido aleatoriamente e removido.

n_5 **: Adição Aleatória de Vértice:** Escolhe-se aleatoriamente um vértice para adicionar à solução.

Considerando r é o nó a ser removido, que o nó anterior a ele na rota é i, e que o nó posterior a ele é j, a economia de remoção é calculado pela Fórmula 11

Considerando r é o nó a ser inserido, i é o nó anterior a onde ele vai ser posicionado na rota, e que j é o nó posterior a onde ele vai ser posicionado, a economia de remoção é calculado pela Fórmula 12

$$\text{economia de remoção} = c_{ir} + c_{rj} - c_{ij} - p_r \tag{11}$$

$$\text{economia de inserção} = c_{ij} + p_r - c_{ir} - c_{rj} \tag{12}$$

A condição de parada utilizada também foi proposta no trabalho de Chaves et. al. [13]. É definido um tempo máximo (MAX_TIME) e caso o tempo de execução sem melhorias ultrapasse esse tempo, o laço é encerrado.

O Algoritmo 2 apresenta o funcionamento do VNS. A linha 1 guarda a rota recebida como parâmetro como melhor rota em *best_route*. Linha 2 calcula o custo dessa rota e guarda seu valor em *best_cost*. Linha 3 inicia o contador de tempo sem melhorias *no_improvement_time* com 0. Linha 4 define as funções de vizinhança $neighborhood\_functions$ como uma lista das funções. Linha 5 atribui a *neighboorhood_length* o tamanho da lista de vizinhanças. A linha 6 captura a hora com precisão de milissegundos no momento da execução e guarda este valor em *start_time*. A Linha 7 inicia o laço externo, que será executado enquanto o tempo de execução sem melhorias (*no_improvement_time*) for menor que o tempo máximo definido (MAX_TIME). Linha 8 inicializa o valor de k, onde k é responsável por iterar sobre as funções de vizinhança. Linha 9 marca o início do laço interno, que será executado enquanto o valor de k for menor que o tamanho da lista de vizinhanças (*neighboorhood_length*). A linha 10 seleciona a k-ésima vizinhança, aplica essa função na *best_route*, e guarda o resultado em *new_route*. Linha 11 calcula o prêmio coletado da rota *new_route*, e guarda este valor em *bonus_collected*. Linha 12 aplica a busca local 2-opt sobre a rota *new_route* e guarda a nova rota em *new_route*. Linha 13 calcula o custo da *new_route* e guarda esse valor em *new_cost*. Linha 14 checa se o valor de *new_cost* é menor que *best_cost* e se *bonus_collected* é maior ou igual que a *quota*, caso ambas condições sejam verdadeiras, as linhas 15 a 18 são executadas, caso contrário, a linha 20 é executada. Esta checagem descarta soluções que não atingiram a cota mínima de bônus coletado ou soluções piores que a melhor encontrada até então. As linhas 15 e 16 guardam o valor de *new_cost* em *best_cost* e a rota *new_route* em *best_route* respectivamente. A linha 17 recomeça o valor de k em 0. A Linha 18 redefine *start_time* com a hora atual do momento da execução. Linha 20 acrescenta uma unidade ao valor

Algorithm 2: Variable Neighborhood Search (VNS)

Input: *quota*, *route*

Output: *route*

```
1  best_route ← route
2  best_cost ← route_cost(best_route)
3  no_improvement_time ← 0
4  neighborhood_functions ← {n1, n2, n3, n4, n5}
5  neighboorhood_length ← 5
6  start_time ← hora atual
7  while no_improvement_time < MAX_TIME do
8      k ← 0
9      while k < neighboorhood_length do
10         new_route ← neighborhood_functions[k](best_route)
11         bonus_collected ← calculate_bonus_collected(new_route)
12         new_route ← 2_opt(new_route)
13         new_cost ← route_cost(new_route)
14         if new_cost < best_cost and bonus_collected ≥ quota then
15             best_cost ← new_cost
16             best_route ← new_route
17             k ← 0
18             start_time ← hora atual
19         else
20             k ← k + 1
21         current_time ← hora atual
22         no_improvement_time ← current_time − start_time
23 return best_route
```

de k para que o algoritmo utilize a próxima função de vizinhança na próxima iteração do laço interno. Linha 21 define *current_time* com a hora atual do momento da execução. Linha 22 atualiza o valor de *no_improvement_time* com a diferença entre *current_time* e *start_time*. Linha 23 retorna a melhor rota obtida, a *best_route*.

3.1.3 VND

O *Variable Neighborhood Descent* (VND), foi proposto por Hertz e Mittaz [32] como uma técnica derivada do VNS, mas com foco na descida. Em vez de perturbar a solução atual, o VND explora sistematicamente as vizinhanças em ordem, procurando por melhorias locais. Quando uma melhoria é encontrada em uma vizinhança, o algoritmo reinicia a busca na primeira vizinhança. Se não houver melhorias em uma dada vizinhança, o algoritmo passa para a próxima. O processo é repetido até que não sejam encontradas melhorias em nenhuma das vizinhanças. Por causa de sua natureza determinística, o VND é muitas vezes mais intensivo em termos de busca, mas pode fornecer soluções de alta qualidade.

Para uma melhor distinção entre VNS e VND, pode-se chamar uma vizinhança no contexto do VND de refinamento. Baseado nos estudos de Melo e Martinhon [50], foram escolhidos três refinamentos específicos para o VND. Estes refinamentos, aqui denotados por r, são estratégias para melhorar a solução atual. A descrição dos refinamentos selecionados é detalhada a seguir:

r_1: **DropAdd**: Esta estratégia primeiro retira vértices, priorizando aqueles com economia positiva de remoção como referenciado no DROP-Step [29] Posteriormente, adiciona vértices que oferecem uma economia positiva na inserção, o chamado ADD-step [29].

r_2: **2-opt**: Esta abordagem avalia todas as possíveis trocas entre duas arestas, optando pela troca que proporciona o maior benefício na função de avaliação.

r_3: **AddDrop**: Esta técnica primeiramente insere o vértice com a maior economia de inserção, ADD-step [29]. Em seguida, retira o vértice que tem a maior economia de remoção, DROP-step [29].

A cada iteração do VND, a solução é refinada utilizando-se o r-ésimo refinamento, sendo $r = \{r_1, r_2, r_3\}$. Se, em algum momento, uma melhoria na solução é encontrada, o algoritmo retorna ao primeiro refinamento para garantir que todas as possibilidades de otimização sejam exploradas.

O Algoritmo 3 detalha o funcionamento do VND. A linha 1 guarda a rota recebida como parâmetro como melhor rota em *best_route*. Linha 2 calcula o custo dessa rota e guarda seu valor em *best_cost*. Linha 3 inicializa o valor de k com 0, onde k é responsável por iterar sobre as funções de refinamento. Linha 4 inicializa $refinements\_functions$

Algorithm 3: Variable Neighborhood Descent (VND)

Input: *quota*, *route*

Output: *route*

```
1  best_route ← route
2  best_cost ← route_cost(best_route)
3  k ← 0;
4  refinements_functions ← {r1, r2, r3}
5  refinements_length ← 3
6  while k < refinements_length do
7      new_route ← refinements_functions[k](best_route)
8      new_cost ← route_cost(new_route)
9      if new_cost < best_cost then
10         best_cost ← new_cost
11         best_route ← new_route
12         k ← 0
13     else
14         k ← k + 1
15 return best_route
```

como uma lista de refinamentos definidos acima. Linha 5 atribui a $refinements\_length$ o tamanho da lista de refinamentos em $refinements\_functions$. A Linha 6 inicia o laço, que será executado enquanto o valor de k for menor que o tamanho da lista de refinamentos. A linha 7 seleciona a função de refinamento que está no índice k, e aplica essa função na *best_route*, o resultado desse função é guardado em *new_route*. Linha 8 calcula o custo da *new_route* e guarda esse valor em *new_cost*. Linha 9 checa se o valor de *new_cost* é menor que *best_cost*, caso seja verdadeiro, as linhas 10 a 12 são executadas, caso contrário, a linha 14 é executada. As linhas 10 e 11 guardam o valor de *new_cost* em *best_cost* e a rota *new_route* em *best_route* respectivamente. A linha 14 recomeça o valor de k em 0. Linha 15 retorna a melhor rota obtida, a *best_route*.

3.1.4 VNS-VND

O VNS-VND combina as estratégias de busca do VNS e do VND. Inicialmente, utiliza-se o VNS para explorar o espaço de soluções e introduzir diversidade. Uma vez que uma solução promissora é encontrada pelo VNS, o VND é então aplicado para intensificar a busca ao redor dessa solução. Isso permite ao algoritmo explorar mais profundamente regiões promissoras do espaço de soluções, combinando a capacidade exploratória do VNS com a intensificação do VND. O VNS-VND é, portanto, uma meta-heurística de dois níveis

que busca equilibrar exploração e explotação para encontrar soluções de alta qualidade em um tempo computacional razoável.

Baseado no trabalho de Chaves et. al [13], utilizamos o VNS detalhado no Algoritmo 2, substituindo a busca local utilizada na linha 12 pelo VND, apresentado no Algoritmo 3.

3.2 Algoritmo Memético

Os algoritmos meméticos são uma classe de algoritmos de otimização inspirados na teoria da evolução e nas ideias de memética, que se referem à evolução de ideias e comportamentos culturais. Eles operam sobre uma população de soluções candidatas, aplicando operadores genéticos como seleção, *crossover* e mutação para explorar o espaço de solução de um problema.

A distinção entre algoritmos meméticos e genéticos vem da sua incorporação de busca local, onde cada indivíduo ou solução está sujeito a um processo de melhoria individual. Isso significa que, além de herdar características de seus pais, cada solução é refinada por meio de métodos de otimização locais.

O algoritmo memético representa a solução como a sequência de nós que constituem a rota. Cada solução tem seu *fitness*, que indica a qualidade da solução, quanto maior o fitness, melhor avaliada é aquela solução [26]. Baseado em Moscato et al. [53], em problemas de minimização o *fitness* pode ser calculado como o inverso da função objetivo para que os menores custos tenham os melhores *fitness*. Para o nosso problema de minimização, o *fitness* consiste no valor calculado de acordo com a fórmula (13), onde o custo da rota é definido pela expressão (1).

$$\frac{1}{\text{custo da rota}} \tag{13}$$

O processo inicia-se com uma população de soluções geradas aleatoriamente, as quais podem variar em tamanho e foram criadas seguindo uma distribuição uniforme. Essas soluções são então avaliadas com base em sua qualidade ou aptidão. As soluções são então submetidas a um método de seleção para que sejam selecionados os pais da próxima reprodução. Os pais são submetidos a um procedimento chamado *crossover* que combina suas soluções a fim de formar novos indivíduos. Estes novos indivíduos estão sujeitos a sofrer uma mutação. Mutação são pequenas alterações destinadas a manter a diversidade na população. As novas soluções são submetidas a uma busca local para melhorar ainda mais suas qualidades. As novas soluções geradas então competem com a população anterior para estarem na próxima geração. A população é então atualizada

com os novos indivíduos. Este ciclo de geração, avaliação, *crossover*, mutação e busca local continua até que um critério de parada seja atingido.

Descrevemos o passo a passo do Algoritmo 4 deixando em aberto qual busca local será utilizada, para então aplicarmos as variações (2-opt, VNS, VND e VNS-VND) deste trabalho.

A Condição de Parada Geral foi definida como uma quantidade máxima de avaliações da função objetivo, chamada de MAX_COST_CALLS. *route_cost_call* será o contador das avaliações.

Adotaram-se os seguintes parâmetros: uma probabilidade de *crossover* ($CROSSOVER\_RATE$), uma probabilidade de mutação ($MUTATION\_RATE$) e um tamanho de população (POP_SIZE). Para a seleção dos pais, empregou-se o método de seleção por torneio. Estas escolhas basearam-se no trabalho de Krasnogor et al. [36].

Algorithm 4: Algoritmo Memético

Input: *quota*

Output: *route*

```
1  population ← init_population(POP_SIZE)
2  while condição de parada não for atingido do
3      for i ← 0 to POP_SIZE − 1 do
4          parent_1 ← tournament_selection(population)
5          parent_2 ← tournament_selection(population)
6          random_number ← Número aleatório no intervalo [0, 1)
7          if random_number < CROSSOVER_RATE then
8              child_1, child_2 ← crossover(parent_1, parent_2)
9          else
10             child_1 ← parent_1
11             child_2 ← parent_2
12         random_number ← Número aleatório no intervalo [0, 1)
13         if random_number < MUTATION_RATE then
14             child_1 ← mutation(child_1)
15             child_2 ← mutation(child_2)
16         new_route ← busca_local(child_1, quota)
17         Adiciona new_route à new_population;
18         new_route ← busca_local(child_2, quota)
19         Adiciona new_route à new_population;
20     population← population ∪ new_population;
21     Atualizar a population com base no fitness de cada solução;
22 return population[0];
```

O Algoritmo 4 detalha o funcionamento do algoritmo memético. A linha 1 cria a população inicial e guarda seus valores em *population*. A linha 2 marca o início do laço mais externo, que será executado enquanto a condição geral de parada não for atingida. A linha 3 marca o início do laço mais interno, que será uma iteração do tamanho da população. Linhas 4 e 5 selecionam dois pais, *parent_1* e *parent_2*, através da seleção por torneio. Linha 6 guarda em *random_number* um número real aleatório no intervalo [0, 1). Linha 7 checa se o número aleatório gerado na linha anterior é menor que a taxa de *crossover*, caso seja verdadeiro, a linha 8 é executada, caso contrário, as linhas 10 e 11 são executadas. A linha 8 aplica o *crossover* nos pais selecionados e guarda os dois filhos gerados em $child_1$ e $child_2$. As linhas 10 e 11 guardam os valores de *parent_1* e *parent_2* em $child_1$ e $child_2$ respectivamente. Linha 12 guarda em *random_number* um número real aleatório no intervalo [0, 1). Linha 13 checa se o número aleatório gerado na linha anterior é menor que a taxa de mutação, se for verdadeiro, as linhas 14 e 15 são executadas. As linhas 14 e 15 aplicam a mutação nos filhos, atualizando seu valor. Linha 16 aplica uma algoritmo de busca local em *child_1* e guarda a nova rota em *new_route*. Linha 17 adiciona *new_route* à *new_population*, que é a população a ser gerada após a iteração da população anterior, chamada *population*. Linhas 18 e 19 replicam o processo das linhas 16 e 17, porém aplicando a busca local no *child_2*., chamada *population*. Linha 20 une a *population* e a *new_population* e guarda essa união em *population*. Linha 21 aplica a atualização de população, ordenando-a a partir do $fitness$. Linha 22 retorna a melhor solução, sendo a primeira rota da *population* ordenada por $fitness$.

A população inicial é criada de maneira aleatória com distribuição uniforme. Para cada solução, um nó aleatório que não esteja presente na mesma, é adicionado até que a cota seja atingida.

O método de seleção utilizado foi o torneio com tamanho 2. O método seleciona dois indivíduos aleatoriamente, com distribuição uniforme, e escolhe aquele de melhor *fitness*.

O método de *crossover* é uma técnica de reprodução que combina os genes de dois pais para criar descendentes. No contexto dos algoritmos genéticos, é um método para gerar novas soluções candidatas (filhos), a partir das características de dois indivíduos selecionados (pais). Este método promove a exploração do espaço de soluções ao misturar as informações genéticas dos pais, com a esperança de criar descendentes que herdem as boas características de ambos.

O *crossover* utilizado, como ilustra a Figura 5, consiste em escolher aleatoriamente com distribuição uniforme, dois índices ao longo do vetor que representa a solução, considerando que os índices devem diferir um do outro e devem ser menores que o tamanho da menor solução entre os pais. O menor índice escolhido representa o ponto inicial de

crossover, enquanto o maior índice representa o ponto final de *crossover*. Esses pontos dividem cada pai em três blocos.

Para gerar o filho 1, o primeiro bloco do pai 1 é copiado diretamente para o filho 1. Após isto, o filho 1 recebe em sequência cada nó do bloco do meio do pai 2, com exceção daqueles que já estavam presentes no primeiro bloco do pai 1. Por fim, é adicionado ao filho 1 os nós que ainda lhe faltam na ordem em que aparecem no pai 1.

Essa mesma lógica é aplicada ao filho 2, onde o primeiro bloco é copiado do pai 2, seguido pelos nós do bloco do meio do pai 1 (exceto aqueles já presentes), e, por fim, os nós restantes do pai 2 são adicionados na ordem original.

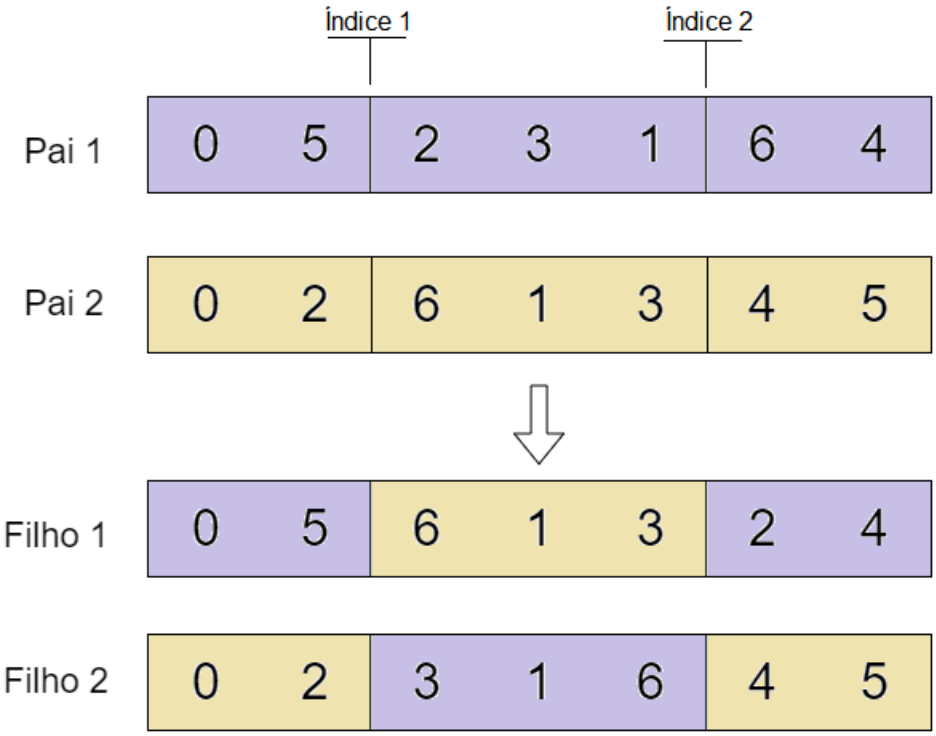

Figura 5: Exemplo de Crossover de dois pontos para um Problema do Caixeiro Viajante.

O método de mutação utilizado foi uma simples inversão de duas arestas aleatórias da solução, o mesmo procedimento utilizado na busca local 2-opt detalhada no Algoritmo 4, possibilitando uma variação genética e possivelmente um novo espaço de soluções.

A atualização da população é feita através de uma estratégia elitista onde a população é combinada com a nova população gerada de filhos, e ranqueada com base no *fitness* das soluções, eliminando as piores soluções e retornando ao seu tamanho original. *population*[0] representa a melhor solução da população ranqueada.

3.3 GRASP

O GRASP (*Greedy Randomized Adaptive Search Procedure*), inicialmente proposto por Feo et al. [19], é uma meta-heurística *multi-start* que consiste em duas fases. Na primeira fase, chamada de fase de construção, uma solução é construída utilizando um método que combina abordagem gulosa e randômica. Em vez de selecionar sempre o candidato de melhor economia (que seria uma abordagem puramente gulosa), o GRASP introduz um componente de aleatoriedade na seleção, usando o parâmetro α. Este parâmetro determina o tamanho da Lista Restrita de Candidatos (LRC), de onde será escolhida de forma aleatoriamente uniforme o próximo nó da solução. Na segunda fase é executada busca local sobre a solução recém-criada. O processo se repete até que um critério de parada seja atingido. A combinação da construção aleatória com a refinada busca local permite ao GRASP explorar uma variedade de rotas possíveis para o problema, frequentemente identificando soluções de alta qualidade em um tempo computacional aceitável.

Para o PCVCP, a fase de construção foi adaptada do estudo de Chaves et. al. [13], de modo que os vértices com as melhores economias de inserção, conforme detalhadas na seção 12, são adicionados à lista de candidatos. A diferença para o algoritmo original é que neste trabalho, antes de iniciar a fase gulosa, a rota já possuidora de um nó inicial, recebe um segundo nó aleatório, para que a economia de inserção seja baseada na posição imediatamente anterior a esse último nó.

Enquanto o prêmio total coletado for menor que a *quota* e a melhor economia calculada for positiva, as economias para cada nó são calculadas. Para a construção da LRC, dois valores, $e_{\max}$ e $e_{\min}$, são definidos como os valores de economia para o melhor e o pior candidato na lista, respectivamente. A partir destes, um limite é calculado para definir quais candidatos entrarão na LRC, de acordo com a expressão (14).

$$e_{\text{limite}} = e_{\max} - \alpha(e_{\max} - e_{\min}) \tag{14}$$

Apenas os candidatos com economia maior ou igual a e_{limite} são incluídos na LRC. Posteriormente, um elemento é selecionado aleatoriamente desta lista restrita para ser adicionado à solução em construção. O valor de α influencia diretamente a aleatoriedade da rota construída, quanto próximo de 0, a abordagem é quase gulosa, pois a LRC se inclina mais para os candidatos de melhor economia. Por outro lado, um α perto de 1 permitirá uma seleção mais diversificada e exploratória, já que a LRC conterá uma variedade mais ampla de candidatos, visto que o e_{limite} terá o mesmo valor que $e_{\min}$.

Similarmente ao que foi feito no algoritmo memético, no GRASP descrevemos o algoritmo deixando em aberto qual busca local será aplicada, para então decorrer sobre as mesmas quatro variações de busca local. A condição de parada utilizada foi a mesma

condição geral de parada do memético, detalhado no Algoritmo 3.2.

Algorithm 5: Algoritmo GRASP

Input: $graph, quota$

Output: $route$

```
1 best_cost ← ∞
2 while condição geral de parada não for atingido do
3     route ← grasp_constructor(graph, quota, α)
4     route ← busca_local(route, quota)
5     cost ← route_cost(route)
6     if cost < best_cost then
7         best_cost ← cost
8         best_route ← route
9 return best_route
```

O Algoritmo 5 detalha o funcionamento do GRASP. Linha 1 inicializa o valor do melhor custo, *best_cost*. Por se tratar de um problema de minimização, é iniciado com um valor alto o suficiente para que o primeiro custo calculado sobrescreva esse valor. Linha 2 marca o início do laço e define sua condição de parada. Linha 3 executa a fase de construção do GRASP a partir do grafo do problema, o valor da cota (*quota*) e do valor de α, o resultado dessa construção é guardado em *route*. Linha 4 aplica busca local na rota construida na linha anterior e guarda a nova rota em *route*. Linha 5 calcula o custo dessa rota e guarda o valor em *cost*. Linha 6 checa se o valor de *cost* é menor que o $best_cost$, caso verdadeiro, as linhas 7 e 8 são executadas. As linhas 7 e 8 guardam o valor de *cost* e *route* em *best_cost* e *best_route* respectivamente. Linha 9 retorna a melhor rota resultante, *best_route*.

Para as nossa análise, foram desenvolvidos quatro algoritmos utilizando o GRASP e substituindo a fase de busca local pelo 2-opt, VNS, VND e VNS-VND, cada uma representando um algoritmo diferente.

Capítulo 4

Experimentos

Os algoritmos foram implementados em C, e compilados com compilador gcc versão 11.3.0., processador AMD Ryzen 5 3600 6-Core 3.59 GHz, Sistema Operacional Windows 10 Pro. Os códigos e experimentos estão disponíveis em `http://github.com/MatheusCCarmo/pcvcp_code`.

Neste capítulo, serão apresentados os experimentos realizados para comparar a eficácia dos algoritmos GRASP e meméticos com diferentes estratégias de busca local. O objetivo é avaliar o desempenho das abordagens híbridas hora apresentadas, comparando-as entre si, além de lançar luz sobre o impacto que diferentes técnicas de busca local podem exercer sobre a qualidade das soluções.

A Seção 4.1 descreve a metodologia utilizada nos experimentos, as instâncias e as configurações de cada algoritmo. Em seguida, a seção 4.2 apresenta os resultados obtidos e comparações de desempenho.

4.1 Metodologia

Esta seção apresenta a metodologia aplicada neste estudo. A Seção 4.1.1 apresenta as instâncias utilizadas. A Seção 4.1.2 apresenta os parâmetros dos algoritmos. A Seção 4.1.3 apresenta os testes estatísticos utilizados.

4.1.1 Conjunto de instâncias

Foram utilizadas 10 instâncias simétricas e 10 instâncias assimétricas de tamanhos entre 10 e 200 nós. As instâncias foram propostas por [47] para estudos sobre o *Quota traveling salesman with passengers and collection time*, e adaptadas nesse trabalho para o PCVCP, de modo que os dados que informavam o delay(tempo para coletar o bônus) no trabalho original, foram utilizados como penalidade para cidades não visitadas. As informações envolvendo matriz de tempo, passageiros entre outras que não seriam de

interesse ao estudo do PCVCP foram descartados. No trabalho original existem grupos diferentes de instâncias, agrupados de acordo com a forma com que foram gerados, porém, a diferença diz respeito ao tempo de viagem, atributo que não existe para o problema do PCVCP. Para este trabalho foram utilizadas instâncias do grupo 1 e do grupo 2. Ambos randômicos com distribuição uniforme, criados a fim de evitar que os casos de testes fossem tendenciosos.

4.1.2 Ajustes dos parâmetros

Para os algoritmos meméticos, foram utilizadas as probabilidades de *crossover* de 0,8, de mutação de 0,05, o tamanho da população de 50 e para seleção dos pais, foi utilizado o método de seleção por torneio com tamanho 2. Esses parâmetros foram baseados em [36].

Para os algoritmos VNS e VNS-VND, foi utilizado um tempo máximo de 200 segundos como MAX_TIME, e quanto aos GRASPs, foi utilizado um valor α de $0,2$, ambos baseados em [13]. A condição geral de parada para o algoritmo memético e para o GRASP foi definido em MAX_COST_CALLS com o valor de 10^6 avaliações da função objetivo.

4.1.3 Testes estatísticos

Cada instância foi executada 30 vezes de modo independente. A condição de parada foi de 10^6 avaliações da função objetivo. Serão analisadas a qualidade de solução e o tempo de computação. A qualidade das soluções foi submetida ao teste de Kruskal-Wallis [14], com nível de significância 0,05. O Kruskal-Wallis é um teste apropriado para comparar mais de duas amostras. Ele executa em duas fases. A primeira analisa se tem diferença estatísticas entre todas as amostras fornecidas. Se sim, então ele executa a segunda fase, a qual consiste em comparar par a par cada algoritmo.

Em seguida, O algoritmo memético e o algoritmo GRASP que apresentaram os melhores resultados foram comparados entre si para cada instância através do teste de Mann-Whitney [14], apropriado para comparar duas amostras, mantendo o nível de significancia de 0,05. O código dos testes foi disponibilizado pelo PISA [7]

4.2 Comparação dos algoritmos entre si

Esta seção compara os algoritmos entre si. Seção 4.2.1 apresenta resultados das instâncias simétricas. Seção 4.2.2 apresenta resultados para as assimétricas.

4.2.1 Resultados para instâncias simétricas

Memético	**2-opt**	**VND**	**VNS**	**VNS-VND**
2-opt	-	1,00	1,00	1,00
VND	0,00	-	0,50	0,50
VNS	0,00	0,50	-	0,50
VNS-VND	0,00	0,50	0,50	-
GRASP	**2-opt**	**VND**	**VNS**	**VNS-VND**
2-opt	-	1,00	1,00	1,00
VND	0,00	-	1,00	1,00
VNS	0,00	0,00	-	1,00
VNS-VND	0,00	0,00	0,00	-

Tabela 3: Comparação dos p-valores entre os algoritmos para a instância 10.1.

No contexto dos testes de Kruskal-Wallis aplicados para comparar os algoritmos A e B, a hipótese nula (H_0) afirma que não há diferenças significativas no desempenho entre os algoritmos. A hipótese alternativa é que o algoritmo A supera o algoritmo B

Interpretamos o p-valor da seguinte forma: um p-valor inferior a 0,05 fornece evidência estatisticamente significativa de que o algoritmo A supera o algoritmo B. Em contraste, um p-valor maior ou igual a 0,95 sugere que ao algoritmo B é estatisticamente superior ao algoritmo A. Entretanto, para valores de p-valor no intervalo [0,05; 0,95], a diferença entre os algoritmos A e B não é considerada estatisticamente conclusiva. Nas tabelas desta seção, o algoritmo A é representado nas linhas e o algoritmo B nas colunas.

As tabelas 3 e 4 apresentam os resultados para as instâncias simétrica 10.1 e 10.2 respectivamente. Ambas mostram resultados semelhantes para algoritmos meméticos, onde os algoritmos baseados em VNS, VND e VNS-VND não foram conclusivamente melhor entre si, dado o p-valor de 0,50. Isso pode estar ligado ao fato das instâncias serem pequenas, fazendo com que esses algoritmos obtenham a mesma solução. Por outro lado, todos se saíram melhor que o 2-opt. Isso acontece por que os outros algoritmos conseguem se desprender de um ótimo local de forma mais eficiente, visto que utilizam diferentes estratégias para causar uma pertubação na solução, enquanto o 2-opt aplica apenas a estratégia de trocar arestas.

O algoritmo GRASP que se saiu melhor foi o VNS-VND. Por utilizar estratégias de exploração e explotação do VNS e do VND, é natural que obtenha resultados melhores, desde que a instância seja grande o suficiente para que os outros algoritmos não convertam para a mesma solução. Foi percebida a diferença que na instância 10.2 não houve uma

Memético	**2-opt**	**VND**	**VNS**	**VNS-VND**
2-opt	-	1,00	1,00	1,00
VND	0,00	-	0,50	0,50
VNS	0,00	0,50	-	0,50
VNS-VND	0,00	0,50	0,50	-
GRASP	**2-opt**	**VND**	**VNS**	**VNS-VND**
2-opt	-	1,00	1,00	1,00
VND	0,00	-	0,50	1,00
VNS	0,00	0,50	-	1,00
VNS-VND	0,00	0,00	0,00	-

Tabela 4: Comparação dos p-valores entre os algoritmos para a instância 10.2.

disparidade significativa na qualidade do VNS em relação ao VND, enquanto na instância 10.1 o VNS se mostrou estatisticamente superior que o VND. Sabendo que o VNS possui um foco maior em exploração e o VND um foco maior em explotação, é possível que na hibridização com GRASP, a exploração seja mais importante para instâncias deste tamanho.

O impacto da escolha da busca local no GRASP foi maior que o impacto no memético. Isso indica que o construtor do GRASP não gerou boas soluções e os operadores de *crossover* e mutação do algoritmo memético foram mais eficientes em gerar boas soluções para as instâncias 10.1 e 10.2.

Memético	**2-opt**	**VND**	**VNS**	**VNS-VND**
2-opt	-	1,00	1,00	1,00
VND	0,00	-	0,86	0,98
VNS	0,00	0,14	-	0,86
VNS-VND	0,00	0,02	0,14	-
GRASP	**2-opt**	**VND**	**VNS**	**VNS-VND**
2-opt	-	1,00	1,00	1,00
VND	0,00	-	0,00	1,00
VNS	0,00	1,00	-	1,00
VNS-VND	0,00	0,00	0,00	-

Tabela 5: Comparação dos p-valores entre os algoritmos para a instância 20.1.

A Tabela 5 apresenta os resultados para a instância simétrica 20.1. Dentre os meméticos, aquele com VNS-VND foi superior em relação aos demais com exceção do VNS, reforçando que uma pertubação de vizinhança como a do VNS pode ser chave para algumas instâncias menores. VNS e VND não foram conclusivamente diferentes entre si, e todos foram melhores que o 2-opt.

Nos resultados para o GRASP, O VND-VND foi superior aos outros, com um destaque para o VND que na instância 20.1 se mostrou melhor que o VNS, e ambas foram conclusivamente melhores que o 2-opt. VND é uma heurística com foco na descida, se a meta-heurística mais externa proporcionar uma solução em uma vizinhança com potencial, é esperado que o VND obtenha resultados melhores que o VNS, que possui um foco maior na exploração de vizinhanças.

O impacto do VNS ou do VND depende do tamanho da vizinhança e da hibridização em que se encontram. Dado os resultados da instância 20.1 e a análise feita, o GRASP conseguiu construir soluções com melhor potencial antes de submetê-las ao VND.

Memético	**2-opt**	**VND**	**VNS**	**VNS-VND**
2-opt	-	1,00	1,00	1,00
VND	0,00	-	0,59	1,00
VNS	0,00	0,41	-	1,00
VNS-VND	0,00	0,00	0,00	-
GRASP	**2-opt**	**VND**	**VNS**	**VNS-VND**
2-opt	-	1,00	1,00	1,00
VND	0,00	-	1,00	1,00
VNS	0,00	0,00	-	1,00
VNS-VND	0,00	0,00	0,00	-

Tabela 6: Comparação dos p-valores entre os algoritmos para a instância 20.2.

A Tabela 6 apresenta os resultados para a instância simétrica 20.2. Dentre os meméticos, aquele com VNS-VND foi superior em relação aos demais, VNS e VND não foram conclusivamente diferentes entre si, e todos foram melhores que o 2-opt.

Nos resultados para o GRASP, O VND-VND foi superior aos outros, com um destaque para o VNS que nessa instância se mostrou melhor que o VND, e ambas foram conclusivamente melhores que o 2-opt. VNS é uma heurística com foco na exploração. Dado que é esperado que o VND obtenha resultados melhores que o VNS ao explorar uma vizinhança com potencial, o resultado indica que para essa instância, a construção do GRASP não foi eficiente em proporcionar essa vizinhança com alto potencial de explotação

ao VND, possibilitando o VNS a obter resultados superiores.

Memético	**2-opt**	**VND**	**VNS**	**VNS-VND**
2-opt	-	1,00	1,00	1,00
VND	0,00	-	0,00	0,00
VNS	0,00	1,00	-	1,00
VNS-VND	0,00	1,00	0,00	-
GRASP	**2-opt**	**VND**	**VNS**	**VNS-VND**
2-opt	-	1,00	1,00	1,00
VND	0,00	-	1,00	1,00
VNS	0,00	0,00	-	1,00
VNS-VND	0,00	0,00	0,00	-

Tabela 7: Comparação dos p-valores entre os algoritmos para a instância 50.1.

Memético	**2-opt**	**VND**	**VNS**	**VNS-VND**
2-opt	-	1,00	1,00	1,00
VND	0,00	-	0,00	0,00
VNS	0,00	1,00	-	1,00
VNS-VND	0,00	1,00	0,00	-
GRASP	**2-opt**	**VND**	**VNS**	**VNS-VND**
2-opt	-	1,00	1,00	1,00
VND	0,00	-	0,97	1,00
VNS	0,00	0,03	-	1,00
VNS-VND	0,00	0,00	0,00	-

Tabela 8: Comparação dos p-valores entre os algoritmos para a instância 50.2.

As tabelas 7 e 8 apresentam os resultados para as instâncias simétricas 50.1 e 50.2 respectivamente. O algoritmo memético com VND e o GRASP com VNS-VND foram conclusivamente melhores que os demais. Tanto as variantes com VNS quanto as variantes com VND foram superiores que as variantes com 2-opt. Esses resultados indicam que para esse tamanho de instância, a etapa genética do algoritmo cumpre bem sua função de gerar soluções em boas vizinhanças, de forma que a condição de parada é mais proveitosamente consumida por um algoritmo com foco em descida como o VND do que por uma hibridização VNS-VND.

Memético	**2-opt**	**VND**	**VNS**	**VNS-VND**
2-opt	-	1,00	1,00	1,00
VND	0,00	-	0,00	1,00
VNS	0,00	1,00	-	1,00
VNS-VND	0,00	0,00	0,00	-
GRASP	**2-opt**	**VND**	**VNS**	**VNS-VND**
2-opt	-	1,00	1,00	1,00
VND	0,00	-	1,00	1,00
VNS	0,00	0,00	-	1,00
VNS-VND	0,00	0,00	0,00	-

Tabela 9: Comparação dos p-valores entre os algoritmos para a instância 100.1.

As duas tabelas apontam que a hibridização do memético com o VNS-VND funcionam melhor que a hibridização do memético com VNS, reforçando a importância da descida variável nessas instâncias. Enquanto a hibridização do GRASP com VNS funciona melhor que a hibridização do GRASP com VND para esses tamanhos de instância, mais um indicativo de que a fase de construção do GRASP não está gerando boas soluções para explotação e o VNS cumpre melhor o papel exploratório.

Memético	**2-opt**	**VND**	**VNS**	**VNS-VND**
2-opt	-	1,00	1,00	1,00
VND	0,00	-	0,00	1,00
VNS	0,00	1,00	-	1,00
VNS-VND	0,00	0,00	0,00	-
GRASP	**2-opt**	**VND**	**VNS**	**VNS-VND**
2-opt	-	1,00	1,00	1,00
VND	0,00	-	0,00	0,53
VNS	0,00	1,00	-	1,00
VNS-VND	0,00	0,47	0,00	-

Tabela 10: Comparação dos p-valores entre os algoritmos para a instância 100.2.

As tabelas 9 e 10 apresentam os resultados para as instâncias simétricas 100.1 e 100.2 respectivamente. Os resultados para os meméticos mostram o VNS-VND se destacando em qualidade e tem o VND sendo superior ao VNS. Todos com resultados melhores

que o 2-opt.

O GRASP com VND e o GRASP com VNS-VND foram melhores que os demais GRASPs, mas não foram conclusivamente um melhor que o outro, essa superiodidade do GRASP com VND sobre o VNS se repete nas instâncias maiores, indicando que a sinergia entre GRASP e VND é melhora com o aumento da instância, dado que em instâncias menores, o GRASP com VNS obteve melhores resultados. Ambas se saíram melhores que o 2-opt.

Memético	**2-opt**	**VND**	**VNS**	**VNS-VND**
2-opt	-	1,00	1,00	1,00
VND	0,00	-	0,00	1,00
VNS	0,00	1,00	-	1,00
VNS-VND	0,00	0,00	0,00	-
GRASP	**2-opt**	**VND**	**VNS**	**VNS-VND**
2-opt	-	1,00	1,00	1,00
VND	0,00	-	0,00	1,00
VNS	0,00	1,00	-	1,00
VNS-VND	0,00	0,00	0,00	-

Tabela 11: Comparação dos p-valores entre os algoritmos para a instância 200.1.

Memético	**2-opt**	**VND**	**VNS**	**VNS-VND**
2-opt	-	1,00	1,00	1,00
VND	0,00	-	0,00	1,00
VNS	0,00	1,00	-	1,00
VNS-VND	0,00	0,00	0,00	-
GRASP	**2-opt**	**VND**	**VNS**	**VNS-VND**
2-opt	-	1,00	1,00	1,00
VND	0,00	-	0,00	1,00
VNS	0,00	1,00	-	1,00
VNS-VND	0,00	0,00	0,00	-

Tabela 12: Comparação dos p-valores entre os algoritmos para a instância 200.2.

As tabelas 11, 12, apresentam os resultados para as instâncias simétrica 200.1 e 200.2 respectivamente. A variante do memético e do GRASP com VNS-VND foram

conclusivamente melhores que os demais. As variantes com VNS e as variantes com VND foram superiores às variantes com 2-opt. As variantes com VND se mostraram superiores às variantes com VNS, mais uma vez indicando uma relação entre a efetividade do VND com o tamanho da instância.

Diante das instâncias simétricas apresentandas, podemos concluir que os algoritmos GRASP com VNS-VND e Memético com VNS-VND demonstraram consistentemente o melhor desempenho na maioria dos experimentos. Isso se deve à sinergia entre o VNS e o VND, proporcionando um equilíbrio eficaz entre exploração e explotação de soluções.

Por outro lado as hibridizações com 2-opt foram piores em todas instâncias. Isso ocorre porque o 2-opt utiliza apenas uma vizinhança, limitando sua capacidade de escapar do ótimo local.

Nos experimentos com algoritmos meméticos, observou-se um desempenho consistente nas instâncias de tamanho pequeno, onde as combinações de memético com VNS e memético com VND não apresentaram diferenças estatisticamente significativas. Isso é atribuído ao tamanho pequeno dessas instâncias, que oferecem um espaço de busca limitado. No entanto, nas instâncias maiores, o memético com VND se destacou de forma conclusiva em relação ao memético com VNS. Isso sugere que, à medida que o tamanho da instância aumenta, a capacidade do VND em intensificar as melhores soluções se torna mais valiosa do que a busca por diversificação oferecida pelo VNS, resultando em melhorias significativas no desempenho.

Quanto aos algoritmos GRASP, foi observada uma tendência. Em instâncias com até 100 vértices, o GRASP com VNS geralmente superou o GRASP com VND, exceto em casos específicos. Isso pode ser explicado pelo fato de que o GRASP, pode gerar soluções construtivas próximas a um ótimo local dessas instâncias, e o VNS ajuda a escapar desses ótimos locais por meio de sua estratégia de diversificação. Por outro lado, em instâncias maiores, como 100.2, 200.1 e 200.2, o GRASP com VND demonstrou um desempenho superior. Aumentando o tamanho da instância, o GRASP pode gerar soluções menos próximas dos ótimos locais, tornando a capacidade de intensificação do VND mais eficaz em melhorar as soluções encontradas. Esse padrão é semelhante ao observado nas combinações memético com VNS e memético com VND, ressaltando a importância da estratégia de intensificação em instâncias maiores e mais complexas.

4.2.2 Resultados para instâncias assimétricas

A Tabela 13, apresenta os resultados para a instância assimétrica 10.1. Entre os meméticos, o hibridizado com VNS-VND foi superior em relação aos demais com exceção do VNS. Isso ressalta a importância de incorporar uma perturbação de vizinhança, como a do VNS, em instâncias menores. Além disso, todos foram melhores que o 2-opt, que

Memético	**2-opt**	**VND**	**VNS**	**VNS-VND**
2-opt	-	1,00	1,00	1,00
VND	0,00	-	1,00	1,00
VNS	0,00	0,00	-	0,94
VNS-VND	0,00	0,00	0,06	-
GRASP	**2-opt**	**VND**	**VNS**	**VNS-VND**
2-opt	-	0,50	1,00	1,00
VND	0,50	-	1,00	1,00
VNS	0,00	0,00	-	1,00
VNS-VND	0,00	0,00	0,00	-

Tabela 13: Comparação dos p-valores entre os algoritmos para a instância 10.1.

usa apenas uma vizinhança.

Entre os GRASPs, o VND e o 2-opt foram estatisticamente iguais, provavelmente por serem focados na fase de explotação, as soluções construídas pelo GRASP não foi capaz de construir uma solução com uma boa área para explotação para a instância 10.1. O VNS foi superior ao VND e ao 2-opt por conseguir escapar das áreas construídas pelo GRASP e o VNS-VND foi melhor que o VNS por acrescentar ao VNS uma busca local mais variável.

Memético	**2-opt**	**VND**	**VNS**	**VNS-VND**
2-opt	-	1,00	1,00	1,00
VND	0,00	-	0,50	0,50
VNS	0,00	0,50	-	0,50
VNS-VND	0,00	0,50	0,50	-

Tabela 14: Comparação dos p-valores entre os algoritmos para a instância 10.2.

A Tabela 14 apresenta os resultados para a instância assimétrica 10.2. Para essa instância, os algoritmos meméticos VNS, VND e VNS-VND não foram conclusivamente melhor entre si, em instâncias pequenas o espaço de busca é pequeno e pode acontecer de encontrarem a mesma solução. Todos eles foram conclusivamente saíram melhor que o 2-opt.

Para os algoritmos GRASP aplicados, como resultado da primeira fase do teste de Krsukal-Wallis, foi aceita a hipótese nula (H_0) de que as distribuições de todos os grupos são semelhantes.

Memético	**2-opt**	**VND**	**VNS**	**VNS-VND**
2-opt	-	1,00	1,00	1,00
VND	0,00	-	1,00	1,00
VNS	0,00	0,00	-	0,43
VNS-VND	0,00	0,00	0,57	-
GRASP	**2-opt**	**VND**	**VNS**	**VNS-VND**
2-opt	-	1,00	1,00	1,00
VND	0,00	-	1,00	1,00
VNS	0,00	0,00	-	1,00
VNS-VND	0,00	0,00	0,00	-

Tabela 15: Comparação dos p-valores entre os algoritmos para a instância 20.1.

A Tabela 15, apresenta os resultados para a instância assimétrica 20.1. Em relação aos meméticos, o memético com VNS-VND foi superior em relação aos demais com exceção do VNS, e todos foram melhores que o 2-opt. Seguindo a mesma tendência da outra instância menor, 10.1.

Os resultados dos GRASPs apontam o VNS-VND como o mais efetivo. O VNS foi superior ao VND, que por sua vez foi superior ao 2-opt. Como também foi visto nas instâncias simétricas, existe uma sinergia do GRASP com o VNS que funciona para algumas instâncias nessa faixa de tamanho.

Memético	**2-opt**	**VND**	**VNS**	**VNS-VND**
2-opt	-	1,00	1,00	1,00
VND	0,00	-	0,99	1,00
VNS	0,00	0,01	-	1,00
VNS-VND	0,00	0,00	0,00	-
GRASP	**2-opt**	**VND**	**VNS**	**VNS-VND**
2-opt	-	0,00	1,00	1,00
VND	1,00	-	1,00	1,00
VNS	0,00	0,00	-	1,00
VNS-VND	0,00	0,00	0,00	-

Tabela 16: Comparação dos p-valores entre os algoritmos para a instância 20.2.

A Tabela 16, apresenta os resultados para a instância assimétrica 20.2. Os algorit-

mos meméticos e GRASPs obtiveram resultados semelhantes, com as hibridizações com o VNS-VND superando as demais, e o VNS superando o VND. A hibridização do memético com VNS para instâncias desse porte funcionou melhor com instâncias assimétricas, visto que para as instâncias assimétricas 20.1 e 20.2 o VNS superou o VND.

Memético	**2-opt**	**VND**	**VNS**	**VNS-VND**
2-opt	-	1,00	1,00	1,00
VND	0,00	-	0,00	0,52
VNS	0,00	1,00	-	1,00
VNS-VND	0,00	0,48	0,00	-
GRASP	**2-opt**	**VND**	**VNS**	**VNS-VND**
2-opt	-	1,00	1,00	1,00
VND	0,00	-	0,11	1,00
VNS	0,00	0,89	-	1,00
VNS-VND	0,00	0,00	0,00	-

Tabela 17: Comparação dos p-valores entre os algoritmos para a instância 50.1.

Memético	**2-opt**	**VND**	**VNS**	**VNS-VND**
2-opt	-	1,00	1,00	1,00
VND	0,00	-	0,00	0,51
VNS	0,00	1,00	-	1,00
VNS-VND	0,00	0,49	0,00	-
GRASP	**2-opt**	**VND**	**VNS**	**VNS-VND**
2-opt	-	1,00	1,00	1,00
VND	0,00	-	1,00	1,00
VNS	0,00	0,00	-	1,00
VNS-VND	0,00	0,00	0,00	-

Tabela 18: Comparação dos p-valores entre os algoritmos para a instância 50.2.

As tabelas 17 e 18 apresentam os resultados para as instâncias assimétricas 50.1 e 50.2 respectivamente. Os resultados dos algoritmos meméticos indicam que o VNS-VND e o VND não foram diferentes entre si, mas foram superiores aos demais. Isso indica uma melhora na eficiência do memético com VND com o aumento do tamanho da instância.

Entre os GRASPs, o VNS-VND foi superior aos outros. A instância 50.1 não

apresentou diferença entre o VNS e o VND, enquanto a instância 50.2 manteve a tendência das instâncias menores de sinergia do GRASP com VNS superando o GRASP com VND.

Memético	**2-opt**	**VND**	**VNS**	**VNS-VND**
2-opt	-	1,00	1,00	1,00
VND	0,00	-	0,00	0,52
VNS	0,00	1,00	-	1,00
VNS-VND	0,00	0,48	0,00	-
GRASP	**2-opt**	**VND**	**VNS**	**VNS-VND**
2-opt	-	1,00	1,00	1,00
VND	0,00	-	0,00	1,00
VNS	0,00	1,00	-	1,00
VNS-VND	0,00	0,00	0,00	-

Tabela 19: Comparação dos p-valores entre os algoritmos para a instância 100.1.

Memético	**2-opt**	**VND**	**VNS**	**VNS-VND**
2-opt	-	1,00	1,00	1,00
VND	0,00	-	0,00	0,07
VNS	0,00	1,00	-	1,00
VNS-VND	0,00	0,93	0,00	-
GRASP	**2-opt**	**VND**	**VNS**	**VNS-VND**
2-opt	-	1,00	1,00	1,00
VND	0,00	-	0,00	1,00
VNS	0,00	1,00	-	1,00
VNS-VND	0,00	0,00	0,00	-

Tabela 20: Comparação dos p-valores entre os algoritmos para a instância 100.2.

As tabelas 19 e 20 apresentam os resultados para as instâncias assimétricas 100.1 e 100.2 respectivamente. O algoritmo memético com VNS-VND e o GRASP com VNS-VND foram conclusivamente melhores que os demais, com exeção das hibridizações com VND. Assim como nas instâncias simétricas, conforme o tamanho da instância assimétrica aumenta, melhor funciona o VND em relação ao VNS. Em espaços maiores, a explotação pode ser mais impactante do que a exploração.

As tabelas 21 e 22 apresentam os resultados para as instâncias assimétricas 200.1,

Memético	2-opt	VND	VNS	VNS-VND
2-opt	-	1,00	1,00	1,00
VND	0,00	-	0,00	1,00
VNS	0,00	1,00	-	1,00
VNS-VND	0,00	0,00	0,00	-
GRASP	**2-opt**	**VND**	**VNS**	**VNS-VND**
2-opt	-	1,00	1,00	1,00
VND	0,00	-	0,00	1,00
VNS	0,00	1,00	-	1,00
VNS-VND	0,00	0,00	0,00	-

Tabela 21: Comparação dos p-valores entre os algoritmos para a instância 200.1.

Memético	2-opt	VND	VNS	VNS-VND
2-opt	-	1,00	1,00	1,00
VND	0,00	-	0,00	1,00
VNS	0,00	1,00	-	1,00
VNS-VND	0,00	0,00	0,00	-
GRASP	**2-opt**	**VND**	**VNS**	**VNS-VND**
2-opt	-	1,00	1,00	1,00
VND	0,00	-	0,00	1,00
VNS	0,00	1,00	-	1,00
VNS-VND	0,00	0,00	0,00	-

Tabela 22: Comparação dos p-valores entre os algoritmos para a instância 200.2.

200.2, respectivamente. O algoritmo memético com VNS-VND e o GRASP com VNS-VND foram conclusivamente melhores que os demais. As hibridizações com VND também foram superiores às hibridizações com VNS. Isso indica que em instâncias maiores, a busca local com descida mais intensa tende a gerar resultados melhores que aqueles mais focados em variar área de busca.

Nesse grupo de instâncias assimétricas, mais uma vez as combinações de memético com VNS-VND e GRASP com VNS-VND obtiveram os melhores resultados. O GRASP com VNS apresentou um funcionamento melhor nas instâncias menores, enquando as combinações com o VND melhoraram a medida que os tamanhos das instâncias aumentaram.

De modo geral, dadas as instâncias testadas, não foi observada uma grande diferença entre os resultados das instâncias simétricas e assimétricas. Pode-se concluir que em instâncias menores, é importante a construção de uma boa solução para que seja aplicada uma busca local mais simples como o 2-opt, caso contrário, a busca local precisa aplicar estratégias para fugir de ótimos locais e explorar mais vizinhanças, como o VNS. Com o aumento das instâncias, estratégias de busca local mais variável como o VND tendem a ter um impacto maior que a construção ou a exploração de vizinhanças. Por fim a combinação VNS-VND busca o melhor de ambas, conseguindo uma exploração eficaz e uma explotação intensa, funcionando bem para diversos tamanhos de instância, seja hibridizado com o memético ou hibridizado com o GRASP.

4.3 Comparação dos melhores algoritmos entre si

Através dos testes de Mann-Whitney, foi realizada uma avaliação da performance de cada instância entre os dois algoritmos de destaque da seção anterior, o memético com VNS-VND e o GRASP com VNS-VND.

Instância	**Resultado do teste de Mann-Whitney**
10.1	Memético VNS-VND x GRASP VNS-VND com p-valor 0,50
10.2	Memético VNS-VND x GRASP VNS-VND com 0,50
20.1	Memético VNS-VND melhor com p-valor 0,00
20.2	GRASP VNS-VND x Memético VNS-VND com p-valor 0,08
50.1	Memético VNS-VND melhor com p-valor 0,00
50.2	Memético VNS-VND melhor com p-valor 0,00
100.1	Memético VNS-VND melhor com p-valor 0,00
100.2	GRASP VNS-VND melhor com p-valor 0,01
200.1	Memético VNS-VND x GRASP VNS-VND 0,35
200.2	Memético VNS-VND melhor com p-valor 0,00

Tabela 23: Resultados dos testes de Mann-Whitney para instâncias simétricas.

As tabelas 23 e 24 apresentam a hipótese alternativa de que algum dos algoritmos foi superior, seguida do p-valor. A hipótese nula é que não há diferenças significativa nos resultados que apontem superioridade de um dos algoritmos. Interpretamos o p-valor inferior a 0,05 como evidência estatisticamente significativa de que a hipótese alterativa é verdadeira. p-valor no intervalo [0,05; 0,95] indicam que a hipótese nula foi aceita.

Considerando o memético com VNS-VND como A e o GRASP com VNS-VND como B, o teste de Mann-Whitney calculou o p-valor para a hipótese de A superior a B

Instância	Resultado do teste de Mann-Whitney
10.1	Memético VNS-VND x GRASP VNS-VND p-valor 0,50
10.2	Memético VNS-VND x GRASP VNS-VND p-valor 0,50
20.1	Memético VNS-VND melhor com p-valor 0,00
20.2	Memético VNS-VND melhor com p-valor 0,00
50.1	Memético VNS-VND melhor com p-valor 0,00
50.2	Memético VNS-VND melhor com p-valor 0,00
100.1	GRASP VNS-VND melhor com p-valor 0,00
100.2	Memético VNS-VND melhor com p-valor 0,00
200.1	GRASP VNS-VND melhor com p-valor 0,00
200.2	Memético VNS-VND melhor com p-valor 0,00

Tabela 24: Resultados dos testes de Mann-Whitney para instâncias assimétricas.

e para B superior a A. As tabelas desta seção foram construídas considerando a hipótese dentre estas que apresentou o menor p-valor.

As instâncias menores, com 10 nós, apresentaram um p-valor de 0,50, indicando que não houve superioridade por parte de nenhum dos algoritmos. Isso se deve ao fato de que em instâncias mais simples, as meta-heurísticas mais avançadas tendem convergir para a mesma solução e encontrar a solução exata. Nas instâncias com 20 nós, houve uma superioridade do Memético com VNS-VND, com exceção apenas da instância simétrica 20.2 onde não houve uma evidência significativa que rejeitasse a hipótese nula, pelo mesmo motivo de convergirem para a mesma solução.

As instâncias de 50 nós mostram melhor performance do memético com VNS-VND, apontando que a hibridização com algoritmo genético funciona melhor que a hibridização com GRASP, e que a estratégia evolutiva pode ser melhor que a construção GRASP, na função de explorar áreas de solução, para esse tamanho de instância.

Para as instâncias maiores que 100, não houve uma clara vantagem entre os algoritmos. O memético foi superior nas instâncias simétricas 100.1, 200.2 e nas assimétricas 100.2 e 200.2. O GRASP foi superior na instância simétrica 100.2 e nas assimétricas 100.1 e 200.1. A instância simétrica 200.1 aceitou a hipótese nula.

Esse equilíbrio com ambos os algoritmos mostrando-se equivalentes em desempenho sugere que em certos contextos, a escolha entre os algoritmos pode não ser tão impactante quanto se possa pensar inicialmente. Em instâncias maiores, o memético perde um pouco de suas características evolutivas, visto que o critério geral de parada pode ser atingido antes de se gerar uma nova população. A diferença entre os dois algoritmos fica por conta da capacidade de exploração de uma geração genética (seleção, *crossover* e mutação) e

da construção gulosa com características aleatórias do GRASP.

Finalizando, uma reflexão mais profunda dos dados sugere que o algoritmo memético tende a ser mais eficaz em instâncias de até 50 nós. Contudo, à medida que as instâncias aumentam em tamanho, essa vantagem aparente se torna menos pronunciada, resultando em uma disputa mais equilibrada entre os dois algoritmos.

Esta análise, portanto, destaca que ambos os algoritmos apresentam suas forças em diferentes contextos, reiterando a importância de entender apropriadamente as nuances de cada um para aplicação no PCTCP.

4.4 Comparação das médias de tempo entre algoritmos

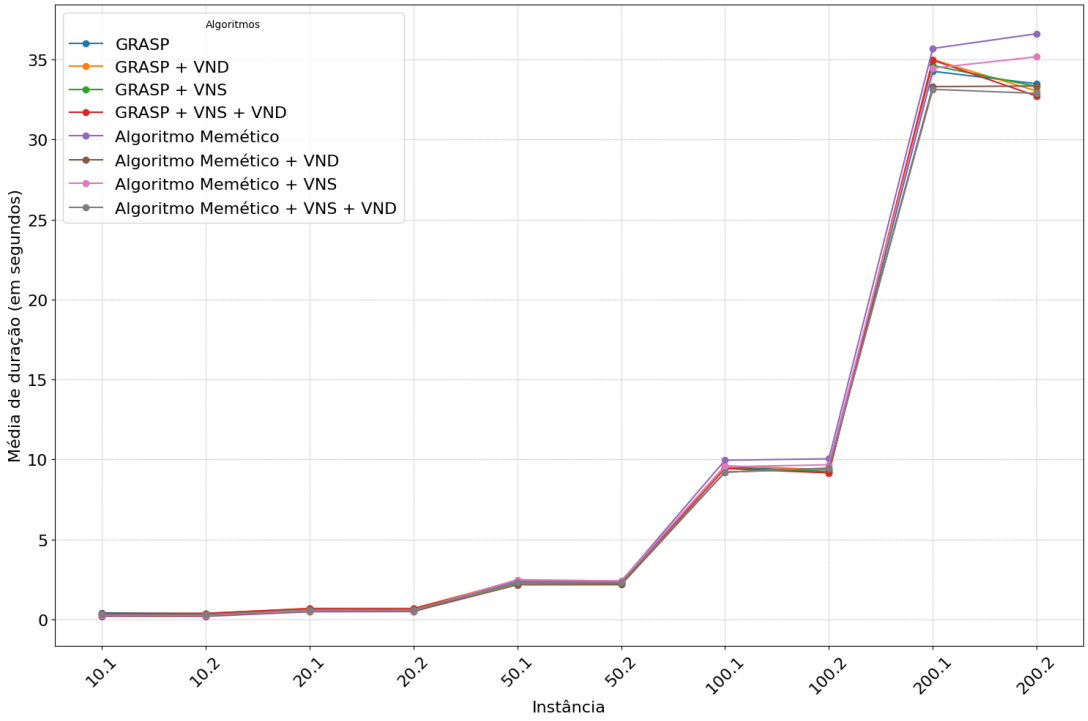

Figura 6: Média de tempo de execução dos algoritmos para instâncias simétricas.

Nas figuras 6 e 7 temos o gráfico exibindo o tempo de execução para cada instância e para cada algoritmo. O eixo x representa a instância e o eixo y representa a média de duração das execuções do algoritmo em segundos.

Analisando as médias de tempo para cada instância e para cada algoritmo, observamos que naturalmente existe um aumento de tempo diretamente proporcional ao número de nós na instância. Podemos também concluir que o fato da instância ser simétrica ou assimétrica não teve grande interferência no tempo de execução. Os algoritmos também

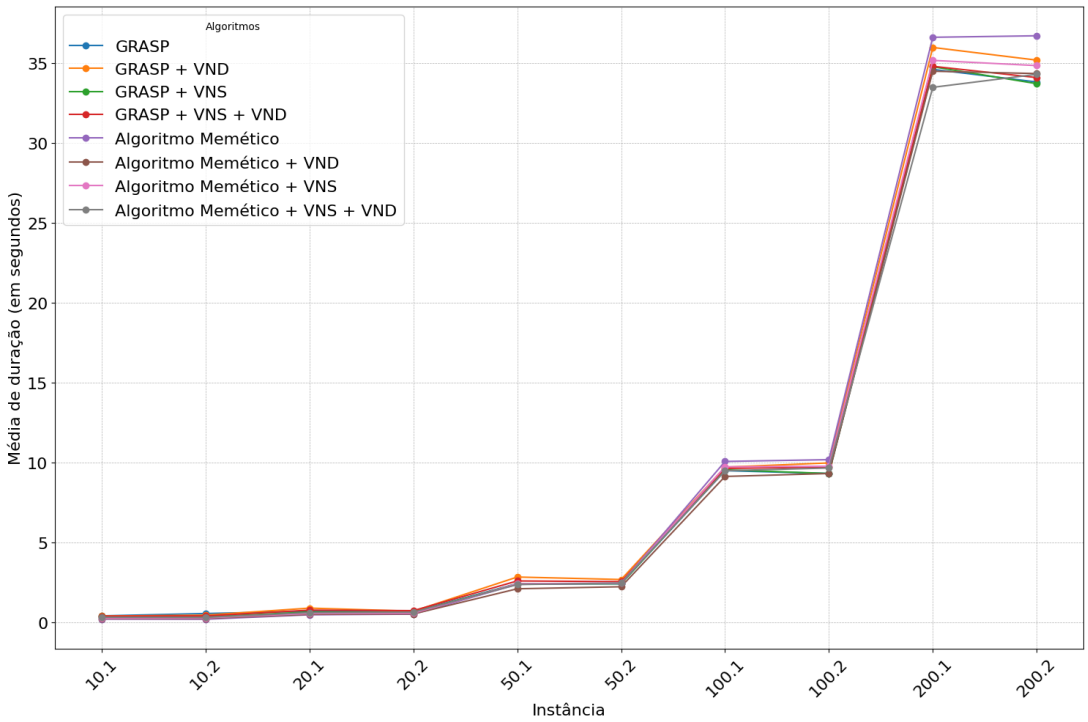

Figura 7: Média de tempo de execução dos algoritmos para instâncias assimétricas.

não apresentaram uma diferença de tempo significativa entre si. Um indicativo de que utilizar como critério de parada geral a quantidade total de cálculo da função objetivo pode emparelhar o tempo de execução diferentes algoritmos.

Capítulo 5

Conclusão

Este trabalho propôs uma investigação sobre uma variante do tradicional Problema do Caixeiro Viajante, acrescentando o desafio da coleta de prêmios. Através da hibridização, baseada em algoritmos bem estabelecidos na literatura, almejou-se não apenas desenvolver soluções mais robustas para o PCVCP, mas também avaliar o desempenho de diferentes técnicas de otimização quando combinadas.

Ao longo do estudo, foram desenvolvidas oito estratégias de solução, divididas entre os algoritmos memético e GRASP, ambas integradas com diferentes técnicas de busca local: 2-opt, VNS, VND e a combinação VNS-VND.

Destaca-se, dentre os resultados obtidos, a superioridade da combinação VNS-VND hibridizadas tanto com o algoritmo memético hibridizado quanto com o GRASP, inspirado na literatura. Esse resultado enfatiza o potencial dos algoritmos meméticos, quando adequadamente combinados com estratégias de busca local, para alcançar soluções de destaque em problemas de otimização.

A combinação da natureza evolutiva dos algoritmos meméticos com a técnica de busca local VNS-VND mostrou uma exploração balanceada do espaço de soluções, enfatizando a interação harmoniosa entre exploração e exploitação. O contexto deste trabalho reforça a ideia de que sempre há margem para aprimoramento e que a combinação inovadora de técnicas pode levar a descobertas significativas.

Este estudo serve como um trabalho preliminar e sinaliza várias direções promissoras para investigações futuras. Uma revisão experimental do estado da arte em algoritmos para o PCVCP poderia lançar luz sobre a performance relativa das soluções propostas neste trabalho, em um contexto mais amplo. Adicionalmente, a integração de técnicas multi-agente na hibridização poderia oferecer uma perspectiva mais colaborativa e adaptativa na busca por soluções. Outras possibilidades seriam estender o PCVCP para multi-objetivo ou investigar o uso de aprendizado de máquina para melhorar o desempenho das meta-heurísticas através da seleção de algoritmos e parâmetros.

Referências Bibliográficas

[1] Philip Adewole, Adio Akinwale, and Kehinde Otubamowo. A genetic algorithm for solving travelling salesman problem. *International Journal of Advanced Computer Sciences and Applications*, 2, 01 2011.

[2] Baruch Awerbuch, Yossi Azar, Avrim Blum, and Santosh S. Vempala. New approximation guarantees for minimum-weight k-trees and prize-collecting salesmen. *SIAM J. Comput.*, 28:254–262, 1999.

[3] Egon Balas. The prize collecting traveling salesman problem. *Networks*, 19(6):621–636, 1989.

[4] Egon Balas. The prize collecting traveling salesman problem and its applications. In *The traveling salesman problem and its variations*, pages 663–695. Springer, 2007.

[5] Egon Balas and G Martin. Roll-a-round: Software package for scheduling the rounds of a rolling mill. *Copyright Balas and Martin Associates*, 104, 1985.

[6] Reuven Bar-Yehuda, Guy Even, and Shimon Moni Shahar. On approximating a geometric prize-collecting traveling salesman problem with time windows. *Journal of Algorithms*, 55(1):76–92, 2005.

[7] Stefan Bleuler, Marco Laumanns, Lothar Thiele, and Eckart Zitzler. Pisa—a platform and programming language independent interface for search algorithms. In *Evolutionary Multi-Criterion Optimization: Second International Conference, EMO 2003, Faro, Portugal, April 8–11, 2003. Proceedings 2*, pages 494–508. Springer, 2003.

[8] Christian Blum, Jakob Puchinger, Günther R. Raidl, and Andrea Roli. Hybrid metaheuristics in combinatorial optimization: A survey. *Applied Soft Computing*, 11(6):4135–4151, 2011.

[9] Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM computing surveys (CSUR)*, 35(3):268–308, 2003.

[10] Guillermo Cabrera-Guerrero, Silvana Roncagliolo, Juan Riquelme, Claudio Cubillos, and Ricardo Soto. An hybrid particle swarm optimization - simulated annealing algorithm for the probabilistic traveling salesman problem. *Studies in Informatics and Control*, 21, 03 2012.

[11] T-H Hubert Chan, Haotian Jiang, and Shaofeng H-C Jiang. A unified ptas for prize collecting tsp and steiner tree problem in doubling metrics. *ACM Transactions on Algorithms (TALG)*, 16(2):1–23, 2020.

[12] Chetan Chauhan, Ravindra Gupta, and Kshitij Pathak. Survey of methods of solving tsp along with its implementation using dynamic programming approach. *International journal of computer applications*, 52(4), 2012.

[13] Antonio Augusto Chaves, Fabrício Lacerda Biajoli, Otávio Massashi Mine, and Marcone Jamilson Freitas Souza. Metaheurísticas híbridas para resolução do problema do caixeiro viajante com coleta de prêmios. *Production*, 17:263–272, 2007.

[14] William Jay Conover. *Practical Nonparametric Statistics*. Wiley Series in Probability and Statistics. Wiley, 1999.

[15] Mauro Dell'Amico, Francesco Maffioli, and Anna Sciomachen. A lagrangian heuristic for the prize collectingtravelling salesman problem. *Annals of Operations Research*, 81(0):289–306, 1998.

[16] Mauro Dell'Amico, Francesco Maffioli, and Peter Värbrand. On prize-collecting tours and the asymmetric travelling salesman problem. *International Transactions in Operational Research*, 2(3):297–308, 1995.

[17] Yong Deng, Yang Liu, and Deyun Zhou. An improved genetic algorithm with initial population strategy for symmetric TSP. *Mathematical Problems in Engineering*, 2015:1–6, 10 2015.

[18] Marco Dorigo and Luca Maria Gambardella. Ant colonies for the travelling salesman problem. *Biosystems*, 43(2):73–81, 1997.

[19] Thomas Feo and Mauricio Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 03 1995.

[20] Islame Felipe da Costa Fernandes. *Hybridizing metaheuristics for multi-and many-objective problems in a multi-agent architecture*. Tese (doutorado em ciência da computação), Universidade Federal do Rio Grande do Norte, Natal, 2022.

[21] Matteo Fischetti and Paolo Toth. An additive approach for the optimal solution of the prize collecting traveling salesman problem. volume 231, pages 319–343. North-Holland Amsterdam, 1988.

[22] Michael R. Garey and David S. Johnson. Computers and intractability a guide to the theory of np-completeness. *Networks*, 1979.

[23] Michel Gendreau, Alain Hertz, and Gilbert Laporte. New insertion and postoptimization procedures for the traveling salesman problem. *Operations Research*, 40(6):1086–1094, 1992.

[24] Edward Kh Gimadi and Oxana Tsidulko. Asymptotically optimal algorithms for the prize-collecting traveling salesman problem on random inputs. In *International Conference on Learning and Intelligent Optimization*, pages 201–207. Springer, 2019.

[25] Fred Glover. Future paths for integer programming and links to artificial intelligence. *Computers Operations Research*, 13(5):533–549, 1986. Applications of Integer Programming.

[26] David E Goldberg, Bradley Korb, and Kalyanmoy Deb. Messy genetic algorithms: Motivation, analysis, and first results. *Complex systems*, 3(5):493–530, 1989.

[27] Bruce L Golden, Larry Levy, and Rakesh Vohra. The orienteering problem. *Naval Research Logistics (NRL)*, 34(3):307–318, 1987.

[28] Bruce L Golden, Qiwen Wang, and Li Liu. A multifaceted heuristic for the orienteering problem. *Naval Research Logistics (NRL)*, 35(3):359–366, 1988.

[29] Leonardo M Gomes, Viviane B Diniz, and Carlos A Martinhon. An hybrid grasp+ vnd metaheuristic for the prize-collecting traveling salesman problem. *XXXII Simpósio Brasileiro de Pesquisa Operacional*, pages 1657–1665, 2000.

[30] Carlos Groba, Antonio Sartal, and Xosé H. Vázquez. Solving the dynamic traveling salesman problem using a genetic algorithm with trajectory prediction: An application to fish aggregating devices. *Computers and Operations Research*, 56:22–32, 2015.

[31] Martin Grötschel and Olaf Holland. Solution of large-scale symmetric travelling salesman problems. *Mathematical Programming*, 51(1-3):141–202, 1991.

[32] Alain Hertz and Michel Mittaz. A variable neighborhood descent algorithm for the undirected capacitated arc routing problem. *Transportation Science*, 35:425–434, 11 2001.

[33] Kazuma Honda, Yuichi Nagata, and Isao Ono. A parallel genetic algorithm with edge assembly crossover for 100,000-city scale TSPs. In *2013 IEEE Congress on Evolutionary Computation*, pages 1278–1285, 2013.

[34] Abid Hussain, Yousaf Shad Muhammad, and Muhammad Nauman Sajid. A simulated study of genetic algorithm with a new crossover operator using traveling salesman problem. *Punjab University Journal of Mathematics*, 51(5), 2020.

[35] Fozia Hanif Khan, Nasiruddin Khan, Syed Inayatullah, and Shaikh Tajuddin Nizami. Solving TSP problem by using genetic algorithm. *International Journal of Basic & Applied Sciences*, 9(10):79–88, 2009.

[36] Natalio Krasnogor and Jim Smith. A memetic algorithm with self-adaptive local search: TSP as a case study. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)*, 05 2000.

[37] Gilbert Laporte. The traveling salesman problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59(2):231–247, 1992.

[38] Gilbert Laporte and Silvano Martello. The selective travelling salesman problem. *Discrete applied mathematics*, 26(2-3):193–207, 1990.

[39] Eugene L Lawler, Jan Karel Lenstra, AHG Rinnooy Kan, and David Bernard Shmoys. *The traveling salesman problem: a guided tour of combinatorial optimization.*, volume 37. JSTOR, 1986.

[40] Jan Karel Lenstra and AHG Rinnooy Kan. Some simple applications of the travelling salesman problem. *Journal of the Operational Research Society*, 26(4):717–733, 1975.

[41] Dong Li, Huaitao Shi, Jianchang Liu, Shubin Tan, Chi Li, and Yu Xie. Research on improved particle-swarm-optimization algorithm based on ant-colony-optimization algorithm. In *2017 29th Chinese Control And Decision Conference (CCDC)*, pages 853–858, 2017.

[42] Shen Lin. Computer solutions of the traveling salesman problem. *Bell System Technical Journal*, 44(10):2245–2269, 1965.

[43] Shen Lin and Brian W Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, 21(2):498–516, 1973.

[44] Junjun Liu and Wenzheng Li. Greedy permuting method for genetic algorithm on traveling salesman problem. In *2018 8th International Conference on Electronics Information and Emergency Communication (ICEIEC)*, pages 47–51, 2018.

[45] Kin-Ming Lo, Wei-Ying Yi, Pak-Kan Wong, Kwong-Sak Leung, Yee Leung, and Sui-Tung Mak. A genetic algorithm with new local operators for multiple traveling salesman problems. *International Journal of Computational Intelligence Systems*, 11(1):692–705, 2018.

[46] Leo Lopez, Michael W Carter, and Michel Gendreau. The hot strip mill production scheduling problem: A tabu search approach. *European Journal of Operational Research*, 106(2-3):317–335, 1998.

[47] Thiago Soares Marques, Sidemar Fideles Cezario, Elizabeth Ferreira Gouvêa Goldbarg, Marco César Goldbarg, and Sílvia Maria Diniz Monteiro Maia. Quota traveling salesman with passengers and collection time. In *2019 8th Brazilian Conference on Intelligent Systems (BRACIS)*, pages 299–304. IEEE, 2019.

[48] Olivier Martin, Steve Otto, and Edward Felten. Large-step markov chains for the traveling salesman problem. *Complex Systems*, 5, 08 1997.

[49] Rajesh Matai, Surya Singh, and M.L. Mittal. *Traveling Salesman Problem: an Overview of Applications, Formulations, and Solution Approaches*. 11 2010.

[50] Valdir A Melo and Carlos A Martinhon. Metaheurísticas híbridas para o problema do caixeiro viajante com coleta de prêmios. In *Simpósio Brasileiro de Pesquisa Operacional (SBPO)*, volume 36, pages 1295–1306, 2004.

[51] Clair E Miller, Albert W Tucker, and Richard A Zemlin. Integer programming formulation of traveling salesman problems. *Journal of the ACM (JACM)*, 7(4):326–329, 1960.

[52] Nenad Mladenović and Pierre Hansen. Variable neighborhood search. *Computers & operations research*, 24(11):1097–1100, 1997.

[53] Pablo Moscato, Carlos Cotta, Alexandre Mendes, et al. *Memetic algorithms*. Springer, 2002.

[54] Pawel B. Myszkowski, Maciej Laszczyk, and Kamil Dziadek. Non-dominated sorting tournament genetic algorithm for multi-objective travelling salesman problem. In Maria Ganzha, Leszek A. Maciaszek, and Marcin Paprzycki, editors, *Proceedings of the 2019 Federated Conference on Computer Science and Information Systems, FedCSIS 2019, Leipzig, Germany, September 1-4, 2019*, volume 18 of *Annals of Computer Science and Information Systems*, pages 67–76, 2019.

[55] Una-May O'Reilly. *An Analysis of Genetic Programming.* PhD thesis, Ottawa-Carleton Institute for Computer Science, Carleton University, Ottawa, Ontario, Canada, 1995.

[56] Odivaney Pedro, Rodney Saldanha, and Ricardo Camargo. A tabu search approach for the prize collecting traveling salesman problem. *Electronic Notes in Discrete Mathematics*, 41:261–268, 2013.

[57] Joseph F Pekny and Donald L Miller. An exact parallel algorithm for the resource constrained traveling salesman problem with application to scheduling with an aggregate deadline. In *Proceedings of the 1990 ACM annual conference on cooperation*, pages 208–214, 1990.

[58] Petrică C. Pop, Ovidiu Cosma, Cosmin Sabo, and Corina Pop Sitar. A comprehensive survey on the generalized traveling salesman problem. *European Journal of Operational Research*, 2023.

[59] Roneeta Purkayastha, Tanmay Chakraborty, Anirban Saha, and Debarka Mukhopadhyay. *Study and Analysis of Various Heuristic Algorithms for Solving Travelling Salesman Problem—A Survey*, pages 61–70. 04 2020.

[60] Günther R Raidl. A unified view on hybrid metaheuristics. In *International workshop on hybrid metaheuristics*, pages 1–12. Springer, 2006.

[61] Sudip Kumar Sahana et al. An improved modular hybrid ant colony approach for solving traveling salesman problem. *GSTF Journal on Computing (JoC)*, 1(2), 2014.

[62] Alexander Schrijver. On the history of combinatorial optimization (till 1960). In K. Aardal, G.L. Nemhauser, and R. Weismantel, editors, *Discrete Optimization*, volume 12 of *Handbooks in Operations Research and Management Science*, pages 1–68. Elsevier, 2005.

[63] Bruno C.H. Silva, Islame F.C. Fernandes, Marco C. Goldbarg, and Elizabeth F.G. Goldbarg. Quota travelling salesman problem with passengers, incomplete ride and collection time optimization by ant-based algorithms. *Computers Operations Research*, 120:104950, 2020.

[64] Petr Stodola, Karel Michenka, Jan Nohel, and Marian Rybansky. Hybrid algorithm based on ant colony optimization and simulated annealing applied to the dynamic traveling salesman problem. *Entropy*, 22:884, 08 2020.

[65] Chutian Sun. A study of solving traveling salesman problem with genetic algorithm. In *2020 9th International Conference on Industrial Technology and Management (ICITM)*, pages 307–311, 2020.

[66] El-Ghazali Talbi. A taxonomy of hybrid metaheuristics. *Journal of Heuristics*, 8:541–564, 01 2002.

[67] Kang-Ping Wang, Lan Huang, Chun-Guang Zhou, and Wei Pang. Particle swarm optimization for traveling salesman problem. In *Proceedings of the 2003 International Conference on Machine Learning and Cybernetics (IEEE Cat. No.03EX693)*, volume 3, pages 1583–1585 Vol.3, 2003.

[68] Zicheng Wang, Xiutang Geng, and Zehui Shao. An effective simulated annealing algorithm for solving the traveling salesman problem. *Journal of Computational and Theoretical Nanoscience*, 6:1680–1686, 07 2009.

[69] Pawel Winter. Steiner problem in networks: A survey. *Networks*, 17(2):129–167, 1987.

[70] Jinhui Yang, Xiaohu Shi, Maurizio Marchese, and Yanchun Liang. An ant colony optimization method for generalized TSP problem. *Progress in Natural Science*, 18(11):1417–1422, 2008.