



UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

BUSCA LOCAL RÁPIDA EM VIZINHANÇA GRANDE PARA O PROBLEMA DO
LADRÃO VIAJANTE

Caio Philipe Vargas de Souza

Orientadores

Dra. Adriana Cesário de Faria Alvim

Dr. Pedro Nuno de Souza Moura

RIO DE JANEIRO, RJ - BRASIL
MARÇO DE 2024

Busca Local Rápida em Vizinhança Grande para o Problema do Ladrão Viajante

Caio Philipe Vargas de Souza

DISSERTAÇÃO APRESENTADA COMO REQUISITO PARCIAL PARA OBTENÇÃO
DO TÍTULO DE MESTRE PELO PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁ-
TICA DA UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO (UNIRIO).
APROVADA PELA COMISSÃO EXAMINADORA ABAIXO ASSINADA.

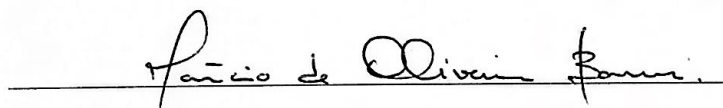
Aprovada por:



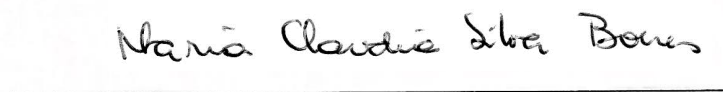
Dra. Adriana Cesário de Faria Alvim — UNIRIO



Dr. Pedro Nuno de Souza Moura — UNIRIO



Dr. Márcio de Oliveira Barros — UNIRIO



Dra. Maria Claudia Silva Boeres — UFES

RIO DE JANEIRO, RJ - BRASIL

MARÇO DE 2024

Catálogo informatizada pelo(a) autor(a)

S719 Souza, Caio Philipe Vargas de
Busca Local Rápida em Vizinhaça Grande para o Problema
do Ladrão Viajante / Caio Philipe Vargas de Souza. -- Rio
de Janeiro, 2024.
73 f

Orientadora: Adriana Cesário de Faria Alvim.
Coorientador: Pedro Nuno de Souza Moura.
Dissertação (Mestrado) - Universidade Federal do Estado
do Rio de Janeiro, Programa de Pós-Graduação em Informática,
2024.

1. Otimização Combinatória. 2. Meta-heurísticas. 3.
Problema do Ladrão Viajante. I. Alvim, Adriana Cesário de
Faria, orient. II. Moura, Pedro Nuno de Souza, coorient.
III. Título.

Souza, Caio Philipe Vargas de. **Busca Local Rápida em Vizinhança Grande para o Problema do Ladrão Viajante**. UNIRIO, 2024. 74 páginas. Dissertação de Mestrado. Departamento de Informática Aplicada, UNIRIO.

RESUMO

Existe uma lacuna crescente entre a pesquisa acadêmica e os problemas reais da indústria envolvidos na aplicação de meta-heurísticas. Pesquisadores, muitas vezes, focam esforços em resolver instâncias cada vez maiores de problemas já muito estudados e fundamentalmente iguais a quando propostos décadas atrás, enquanto problemas reais crescem em complexidade envolvendo combinação e interdependência de dois ou mais desses problemas tradicionais.

O Problema do Ladrão Viajante se encaixa nesse contexto ao combinar dois problemas tradicionais da área de otimização combinatória na busca de capturar essa complexidade existente em problemas reais.

Este trabalho propõe uma Busca Local Rápida baseada na meta-heurística Busca em Vizinhança Grande a fim de solucionar, de forma heurística, o Problema do Ladrão Viajante, que descarta soluções vizinhas de baixo potencial de melhoria e evita o custoso cálculo da função objetivo. Adicionalmente, são apresentados resultados computacionais obtidos sobre milhares de instâncias de teste da literatura comprovando a robustez e eficácia da heurística proposta.

Palavras-chave: Problema do Ladrão Viajante, Problema do Caixeiro Viajante, Problema da Mochila, Heurísticas, Meta-heurísticas, Busca Local Rápida, Busca em Vizinhança Grande, Otimização Combinatória.

Souza, Caio Philipe Vargas de. **Fast Large Neighborhood Search for the Travelling Thief Problem**. UNIRIO, 2024. 74 pages. Masters Dissertation. Departamento de Informática Aplicada, UNIRIO.

ABSTRACT

There is a growing gap between academic research and real world problems found in the industry revolving around the application of metaheuristics. Researchers often focus their efforts in solving larger instances of problems that are already well understood and fundamentally similar to those proposed decades prior, while real world problems increase in complexity involving the combination and interdependence of two or more of these tradicional problems.

The Travelling Thief Problem fits this context by combining two tradicional problems from the combinatorial optimization area aiming to capture the complexity present in real world problems.

This work proposes a Fast Local Search based on the Large Neighborhood Search to solve, in a heuristic way, the Travelling Thief Problem, discarding neighbor solutions with low potential of improvement and avoid the fitness function's costly calculation. Additionally, this works presents computational results obtained from thousands of test instances from the literature demonstrating the proposed heuristic's accuracy and robustness.

Keyword: Travelling Thief Problem, Travelling Salesman Problem, Knapsack Problem, Heuristics, Metaheuristics, Fast Local Search, Large Neighborhood Search, Combinatorial Optimization.

Sumário

1	Introdução	2
1.1	Objetivos	2
1.2	Organização da Dissertação	3
2	O Problema do Ladrão Viajante	4
2.1	Problema do Caixeiro Viajante	4
2.2	Problema da Mochila	7
2.3	O Problema do Ladrão Viajante	8
3	Conceitos Fundamentais	12
3.1	Heurística de Lin-Kernighan para o PCV	12
3.2	Triangulação de Delaunay	13
3.3	Busca Local Rápida	13
3.4	Busca em Vizinhança Grande	15
4	Trabalhos Relacionados	16
4.1	Trabalho de Bonyadi et al., de 2014	16
4.2	Trabalho de Polyakovskiy et al., de 2014	18
4.3	Trabalho de Mei, Li e Yao, de 2014	20

4.4	Trabalho de Faulkner et al., de 2015	21
4.5	Trabalhos de Yafrani e Ahiod, de 2016 e 2018	22
4.6	Trabalho de Wagner, de 2016	24
4.7	Outros Trabalhos da Literatura	24
5	Busca Local Rápida em Vizinhaça Grande para o Problema do Ladrão Viajante	26
5.1	Construção Gulosa de Solução para o PLV	26
5.1.1	Rota Inicial	27
5.1.2	Escolha dos Itens da Mochila para uma Dada Rota	27
5.1.3	Solução Gulosa para o PLV	29
5.2	Busca Local para o PLV	30
5.2.1	Definição de Relações de Vizinhaças para o PLV	30
5.2.2	Lista de Candidatos	31
5.2.3	Definição de Relação de Subvizinhaças para o PLV	32
5.2.4	Busca Local Rápida	36
5.3	Conjunto de Soluções Iniciais	37
5.4	Busca em Vizinhaça Grande para o PLV	39
5.4.1	Procedimento de Destruição	39
5.4.2	Procedimento de Reconstrução	41
5.4.3	Reativação de Subvizinhaças	42
5.4.4	Rota em Vizinhaça Grande	43
5.5	Busca Local Rápida em Vizinhaça Grande para o PLV	44
6	Experimentos Computacionais	47
6.1	Instâncias do Problema PLV	47

6.2	Valores dos Parâmetros	49
6.3	Ambiente Computacional	50
6.4	Comparação dos Resultados	50
6.4.1	Comparação com os resultados de Wagner et al., 2018	51
6.4.2	Comparação com as competições GECCO de 2017 e 2023	56
6.4.3	Comparação com o Trabalho de Yafrani e Ahiod, de 2018	61
6.4.4	Comparação com o Trabalho de Wuijts e Thierens, de 2019	63
6.4.5	Comparação com o Trabalho de Chagas e Wagner, de 2022	65
6.5	Considerações Finais	67
7	Conclusões	68
7.1	Contribuições	68
7.2	Trabalhos Futuros	69
	Referências Bibliográficas	71

Lista de Figuras

2.1	Exemplo de matriz de distâncias de uma instância euclidiana do PCV. Foi aplicada a função <i>topo</i> às distâncias euclidianas a fim de se obterem somente valores inteiros.	6
2.2	Exemplo de instância e solução ótima para o Problema do Caixeiro Viajante.	7
2.3	Exemplo de instância e solução do Problema do Ladrão Viajante.	11
3.1	Exemplo de Diagrama de Voronoi e Triangulação de Delaunay. Os círculos azuis representam pontos no plano cartesiano. A Diagrama de Voronoi é representado pelas linhas claras, enquanto a Triangulação de Delaunay é representado pelas linhas escuras.	14
6.1	Mapa de calor comparando a heurística BLRVG-PLV com a heurística S5 utilizando as instâncias de Polyakovskiy et al.[1] e a métrica Diferença Percentual Relativa. Cada quadrado representa uma instância do PLV. Quanto mais azul o quadrado, maior a diferença positiva entre a solução encontrada pelo BLRVG-PLV para a instância. Quadrados em tons de vermelho representam instâncias para as quais a BLRVG-PLV obteve soluções piores que a heurística S5. Soluções com valor da função objetivo similares, ou seja, com DPR próximo a 0%, são representadas pela cor cinza.	54
6.2	Comparação da quantidade de melhores soluções agrupadas por instância do TSPLIB entre as heurísticas BLRVG-PLV e heurística S5. O eixo <i>x</i> indica uma instância do TSPLIB em ordem não decrescente do número de cidades, enquanto o eixo <i>y</i> indica a quantidade de instâncias do PLV geradas a partir da instância do TSPLIB às quais a heurística BLRVG-PLV obteve melhores soluções em relação à heurística S5.	55

6.3	Mapa de calor utilizando a métrica DPR comparando os resultados da BLRVG-PLV com os resultados de JC obtidos na competição GECCO 2017 para as instâncias de Polyakovskiy et al.[1].	58
6.4	Mapa de calor utilizando a métrica DPR comparando os resultados da BLRVG-PLV com os resultados de MY obtidos na competição GECCO 2017 para as instâncias de Polyakovskiy et al.[1].	59
6.5	Comparação da média do valor de DPR obtido pela BLRVG-PLV quando comparada aos resultados submetidos por MY à competição GECCO 2017. O eixo x indica a fração da capacidade da mochila, o eixo “y” indica a média do valor de DPR.	60
6.6	Mapa de calor comparando a heurística BLRVG-PLV com a heurística CS2SA* utilizando a métrica DPR, em relação às 20 instâncias do TSPLIB que foram adotadas.	62
6.7	Mapa de calor comparando a heurística BLRVG-PLV com a heurística MEA2P na categoria “Vários itens por cidade” utilizando a métrica DPR.	64
6.8	Mapa de calor comparando a heurística BLRVG-PLV com a heurística MEA2P na categoria “Um item por cidade” utilizando a métrica DPR.	65
6.9	Mapa de calor comparando a heurística BLRVG-PLV com a heurística WSM utilizando a métrica DPR.	67

Lista de Tabelas

2.1	Exemplo de instância do Problema do Caixeiro Viajante.	6
2.2	Exemplo de instância do Problema da Mochila com capacidade W igual a 5.	8
6.1	Comparação dos resultados da heurística BLRVG-PLV com o <i>benchmark</i> de Wagner.	53
6.2	Comparação dos resultados da heurística BLRVG-PLV com as soluções obtidas pelos competidores da competição GECCO 2017.	57
6.3	Comparação dos resultados da heurística BLRVG-PLV com as heurísticas CS2SA* e CS2SA-R.	61
6.4	Comparação dos resultados obtidos pela heurística BLRVG-PLV com os obtidos pela heurística MEA2P.	64
6.5	Comparação dos resultados obtidos pela heurística BLRVG-PLV com os obtidos pela heurística WSM.	66

1. Introdução

De acordo com Michalewicz[2], existe uma lacuna crescente entre a pesquisa acadêmica e os problemas reais da indústria envolvidos na aplicação de meta-heurísticas. Pesquisadores, muitas vezes, focam esforços em resolver instâncias cada vez maiores de problemas já muito estudados e fundamentalmente iguais a quando propostos décadas atrás, enquanto problemas reais crescem em complexidade envolvendo combinação e interdependência de dois ou mais desses problemas tradicionais.

O Problema do Ladrão Viajante (PLV) [3] se encaixa nesse contexto ao combinar dois problemas tradicionais da área de otimização combinatória na busca de capturar essa complexidade existente em problemas reais.

Este trabalho propõe uma Busca Local Rápida baseada na meta-heurística Busca em Vizinhança Grande a fim de solucionar, de forma heurística, o PLV, que descarta soluções vizinhas de baixo potencial de melhoria e evita o custoso cálculo da função objetivo.

1.1 Objetivos

Os objetivos da pesquisa são:

- (i) Propor uma heurística baseada na Busca Local Rápida [4] e na meta-heurística Busca em Vizinhança Grande [5] para o Problema do Ladrão Viajante;
- (ii) Propor, executar e analisar resultados de estudo experimental utilizando a heurística proposta, com instâncias referência da literatura. Comparar os resultados encontrados pela heurística desenvolvida com os resultados encontrados na literatura. Os fatores considerados no estudo comparativo são: a eficácia (qualidade das soluções) e a robustez (menor sensibilidade às características das instâncias).

1.2 Organização da Dissertação

Esta dissertação está dividida em sete capítulos: o Capítulo 1 contém a motivação e os objetivos da pesquisa; no Capítulo 2 é apresentada a definição do Problema do Ladrão Viajante, em conjunto com seus problemas componentes; no Capítulo 3 são apresentados os conceitos fundamentais que serão utilizados no restante do trabalho; o Capítulo 4 faz uma apresenta os trabalhos relacionados da literatura para resolver o Problema do Ladrão Viajante; no Capítulo 5 é apresentada em detalhes a heurística proposta para solucionar o Problema do Ladrão Viajante; no Capítulo 6 são abordados os experimentos computacionais realizados e a comparação dos resultados obtidos por esses experimentos com os da literatura. Por fim, o Capítulo 7 contém as conclusões extraídas dos experimentos, contribuições da pesquisa e propostas de trabalhos futuros.

2. O Problema do Ladrão Viajante

O Problema do Ladrão Viajante (PLV) combina dois problemas tradicionais da área de Otimização Combinatória: o Problema do Caixeiro Viajante (PCV) e o Problema da Mochila (PM) [3].

2.1 Problema do Caixeiro Viajante

O Problema do Caixeiro Viajante (em inglês, *Travelling Salesman Problem*) é um dos problemas $\mathcal{NP}$ -Difíceis mais conhecidos e estudados na literatura de Otimização Combinatória [6]. Nesse problema, dispõe-se de um conjunto de cidades para o qual um caixeiro deve visitar todas as cidades, sem repetição e retornando à cidade inicial, na menor distância possível [7].

Formalmente, dado um conjunto de n cidades $C = \{1, \dots, n\}$ e uma matriz de distâncias $D = \{d_{ij} \geq 0; \forall i, j \in C\}$, o Problema do Caixeiro Viajante busca encontrar uma rota de menor distância entre as cidades em C , representada por uma conjunto de variáveis binárias x_{ij} , em que:

$$x_{ij} = \begin{cases} 1, & \text{o caixeiro percorre o caminho entre a cidade } i \text{ e } j \\ 0, & \text{caso contrário} \end{cases} \quad (2.1)$$

A formulação do PCV pode ser definida como um problema de programação inteira [7]:

$$\text{minimizar: } \sum_{i=1}^n \sum_{j=1}^n d_{ij} x_{ij} \quad (2.2)$$

$$\text{sujeito a: } x_{ii} = 0 \quad i = 1, \dots, n \quad (2.3)$$

$$\sum_{i=1}^n x_{ij} = 1 \quad j = 1, \dots, n \quad (2.4)$$

$$\sum_{j=1}^n x_{ij} = 1 \quad i = 1, \dots, n \quad (2.5)$$

$$\sum_{j < i=k}^n x_{ij} + \sum_{i > j=k}^n x_{ij} = 2 \quad k = 1, \dots, n \quad (2.6)$$

$$x_{ij} \in \{0, 1\} \quad i = 1, \dots, n; j = 1, \dots, n \quad (2.7)$$

A função objetivo, dada pela Equação 2.2, corresponde à minimização do caminho percorrido pelo caixeiro. A Equação 2.3 garante que não seja possível percorrer o caminho ligando uma cidade à ela mesma. As Equações 2.4 e 2.5 garantem que o caixeiro saia e entre em todas as cidades exatamente uma vez. A Equação 2.6 garante que há uma única rota percorrendo todas as cidades. Por fim, a Equação 2.7 garante a integralidade das variáveis x_{ij} .

Representando a solução do PCV por uma lista de permutação $x = (x_1, \dots, x_n)$, em que $x_i \in C$, $1 \leq i \leq n$ é a i -ésima cidade visitada. A função objetivo do PCV pode ser definido como:

$$\text{minimizar: } \sum_{i=1}^{n-1} d_{x_i x_{i+1}} + d_{x_n 1} \quad (2.8)$$

A definição do Problema do Caixeiro Viajante dada pela Equação 2.8 será a definição utilizada no restante deste trabalho.

Dado dois pontos $a = (a_1, a_2)$ e $b = (b_1, b_2)$ no plano cartesiano, a distância euclidiana $d(a, b)$ entre esses pontos é um número real não negativo, definida pela Equação 2.9.

$$d(a, b) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2} \quad (2.9)$$

Instâncias do PCV podem disponibilizar a matriz de distâncias indiretamente. Uma forma comum é posicionar as cidades em um plano cartesiano e utilizar a distância

euclidiana entre os pares de cidades para compor essa matriz. A Tabela 2.1 ilustra uma instância do problema construída dessa maneira. Na tabela, cada linha corresponde a uma determinada cidade. A coluna “Índice” indica o índice da cidade e as colunas “x” e “y” indicam suas coordenadas no plano cartesiano.

Tabela 2.1: Exemplo de instância do Problema do Caixeiro Viajante.

Índice	x	y
1	5	5
2	4	4
3	1	7
4	7	6
5	6	3

Para esse exemplo, no cálculo da matriz de distâncias ilustrada na Figura 2.1, considerou-se a aplicação da função *topo* (menor inteiro maior ou igual ao valor) às distâncias euclidianas, a fim de se obterem somente valores inteiros. Além disso, a distância de uma cidade a ela própria é sempre zero nesse tipo de instância, por isso foi omitida.

$$D = \begin{bmatrix} - & 2 & 5 & 3 & 3 \\ 2 & - & 5 & 4 & 3 \\ 5 & 5 & - & 7 & 7 \\ 3 & 4 & 7 & - & 4 \\ 3 & 3 & 7 & 4 & - \end{bmatrix}$$

Figura 2.1: Exemplo de matriz de distâncias de uma instância euclidiana do PCV. Foi aplicada a função *topo* às distâncias euclidianas a fim de se obterem somente valores inteiros.

É importante notar que a matriz da Figura 2.1 é simétrica. Essa característica vale para qualquer instância construída dessa forma uma vez que, para a distância euclidiana, vale a propriedade de simetria: dadas duas cidades a e b, $d(a, b) = d(b, a)$. Variantes do PCV que respeitem essa propriedade são denominadas de Problema do Caixeiro Viajante Simétrico [8].

A Figura 2.2a ilustra graficamente esse exemplo. Na figura, foi escolhida a primeira cidade da instância como cidade inicial da rota. Essa cidade é marcada com uma estrela e a distância dessa cidade às demais é ilustrada por linhas tracejadas. A Figura 2.2b mostra uma solução ótima cuja distância total é igual a 20. É possível notar que há duas opções de rota partindo da primeira cidade que resultam na mesma distância total, a saber: as rotas representadas pelas soluções $x' = (1, 4, 5, 2, 3)$ e $x'' = (1, 3, 2, 5, 4)$. Esse exemplo ilustra que, para instâncias do tipo euclidianas, percorrer a rota em qualquer sentido resulta no mesmo valor da função objetivo representada pela Equação 2.8.

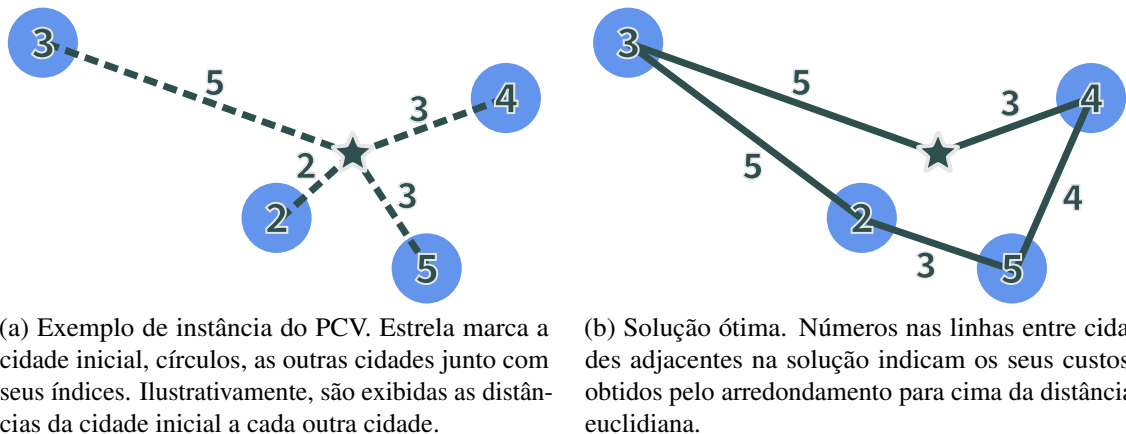


Figura 2.2: Exemplo de instância e solução ótima para o Problema do Caixeiro Viajante.

2.2 Problema da Mochila

O Problema da Mochila (em inglês, *Knapsack Problem*) é uma classe de problemas $\mathcal{NP}$ -Difíceis [9] que consiste em selecionar dentre um conjunto de itens quais devem ser adicionados a uma mochila, de forma a obter o maior valor agregado dos itens, observando o limite de capacidade da mochila. Esse problema possui inúmeras variantes [10], porém, de interesse para este trabalho, é o Problema Binário da Mochila, em que um item é adicionado em sua totalidade ou não à mochila. Neste texto, utiliza-se Problema da Mochila (ou PM) como sinônimo do Problema Binário da Mochila.

Formalmente, dada uma mochila de capacidade $W > 0$, um conjunto $B = \{1, \dots, m\}$ de m itens, uma lista $p = (p_1, \dots, p_m)$ de valores positivos dos itens (*profits*) e uma lista $w = (w_1, \dots, w_m)$ de pesos positivos dos itens (*weights*), o Problema da Mochila busca encontrar um subconjunto de itens em B cujo valor agregado seja máximo, respeitando a capacidade W da mochila. Uma solução desse problema é representada por uma lista de variáveis binárias $z = (z_1, \dots, z_m)$, em que:

$$z_i = \begin{cases} 1, & \text{caso o item } i \text{ seja adicionado à mochila} \\ 0, & \text{caso contrário} \end{cases} \quad (2.10)$$

A formulação do PM pode ser definida como um problema de programação inteira [9]:

$$\text{maximizar: } \sum_{i=1}^m p_i z_i \quad (2.11)$$

$$\text{sujeito a: } \sum_{i=1}^m w_i z_i \leq W \quad (2.12)$$

$$\sum_{i=1}^m w_i > W \quad (2.13)$$

$$w_i \leq W \quad i = 1, \dots, m \quad (2.14)$$

$$z_i \in \{0, 1\} \quad i = 1, \dots, m \quad (2.15)$$

A função objetivo, dada pela Equação 2.11, corresponde à maximização dos valores dos itens adicionados à mochila, e a Equação 2.12 corresponde à restrição de capacidade.

A Equação 2.13 garante que o peso total dos itens de uma instância seja maior que a capacidade da mochila, caso contrário a solução ótima é trivialmente obtida pela adição de todos os itens na mochila. A Equação 2.14 garante que cada item individualmente caiba na mochila, caso contrário o item pode ser trivialmente excluído da solução. A Equação 2.15 corresponde à restrição de integralidade das variáveis z_i .

A Tabela 2.2 ilustra uma instância do PM com quatro itens e capacidade W igual a 5. A coluna “Índice” indica o índice do item na instância, as colunas “p” e “w” indicam, respectivamente, o valor e o peso do item correspondente.

Tabela 2.2: Exemplo de instância do Problema da Mochila com capacidade W igual a 5.

Índice	p	w
1	2	1
2	5	2
3	3	4
4	4	2

Uma busca exaustiva nessa instância encontra a solução $z = (1, 1, 0, 1)$, representando a escolha dos itens $\{1, 2, 4\}$, com valor total igual a 11 e peso total igual a 5. Essa solução utiliza, portanto, toda a capacidade da mochila.

2.3 O Problema do Ladrão Viajante

O Problema do Ladrão Viajante une características do PCV com o PM. Assim como no PCV, há um conjunto de cidades que o ladrão deve percorrer, sem repetição e retornando

à cidade inicial, fixada pela instância. Em cada cidade, com exceção da inicial, há um conjunto de itens, com seus respectivos valores e pesos, que podem ser “roubados”. Cada um desses itens aparece em exatamente uma cidade. O ladrão possui uma mochila de capacidade limitada e deve pagar um aluguel por seu uso proporcional ao tempo utilizado para percorrer o trajeto. A velocidade que o ladrão se locomove entre as cidades é reduzida conforme o peso dos itens carregados na mochila aumenta [3].

Formalmente, dado um conjunto de n cidades $C = \{1, \dots, n\}$, uma matriz de distâncias $D = \{d_{ij} \geq 0\}$ entre as cidades i e j em que $i, j \in \{1, \dots, n\}$, um conjunto $B = \{1, \dots, m\}$ de m itens, uma lista $p = (p_1, \dots, p_m)$ de valores positivos dos itens, uma lista $w = (w_1, \dots, w_m)$ de pesos positivos dos itens, uma lista de conjuntos de itens por cidade $A = (a_1, \dots, a_n)$ tal que $a_1 = \emptyset$ e $b \in B$ pertence exatamente a um conjunto $a_c \in A$ se, e somente se, o item b está disponível para ser roubado na cidade $c \in C$, um valor $W > 0$ de capacidade da mochila, um valor de aluguel $R \geq 0$ da mochila por unidade de tempo e valores de velocidade máxima e mínima $v^+ > v^- > 0$ de movimentação do ladrão, a solução do Problema do Ladrão Viajante pode ser definida em duas partes: uma rota a ser percorrida e um conjunto de itens a serem roubados.

Adotando-se a definição de solução do PCV, descrita na Seção 2.1, a parte referente à rota da solução do PLV é definida como uma permutação das cidades em C representada por uma lista $x = (x_1, \dots, x_n)$, em que $x_i \in C$ corresponde à i -ésima cidade visitada. Por convenção, a primeira cidade da rota da solução é fixada como a primeira cidade da instância, isto é, $x_1 = 1$. A parte da solução referente à configuração da mochila segue a mesma definição estabelecida para o PM, descrita na Seção 2.2. Assim, uma configuração de mochila é representada por uma lista de variáveis binárias $z = (z_1, \dots, z_m)$, com $z_i = 1$ se, e somente se, o item i é adicionado à mochila e 0, caso contrário.

O PLV é definido formalmente a seguir adaptando a formulação proposta por Mei, Li e

Yao[11]:

$$\text{maximizar: } f(x, z) = \sum_{i=1}^m p_i z_i - R \left(\sum_{i=1}^{n-1} \frac{d_{x_i x_{i+1}}}{v_i} + \frac{d_{x_n x_1}}{v_n} \right) \quad (2.16)$$

$$\text{sujeito a: } \sum_{i=1}^m w_i z_i \leq W \quad (2.17)$$

$$\sum_{i=1}^m w_i > W \quad (2.18)$$

$$w_i \leq W \quad k = 1, \dots, n \quad (2.19)$$

$$v_k = v^+ - \frac{v^+ - v^-}{W} q_k \quad k = 1, \dots, n \quad (2.20)$$

$$q_k = \sum_{i=1}^k \sum_{j \in \alpha_{x_i}} w_j z_j \quad k = 1, \dots, n \quad (2.21)$$

$$z_i \in \{0, 1\} \quad i = 1, \dots, m \quad (2.22)$$

$$x_1 = 1 \quad (2.23)$$

A Equação 2.16 descreve a função objetivo como a maximização do valor total dos itens roubados subtraído do aluguel a ser pago pela mochila no tempo total utilizado para percorrer a rota. De forma semelhante ao PM, a Equação 2.17 garante que a capacidade da mochila não seja ultrapassada, a Equação 2.18 garante que o peso total dos itens da instância seja maior que a capacidade da mochila e a Equação 2.19 garante que cada item individualmente caiba na mochila. A Equação 2.20 define a velocidade v_k do ladrão saindo da k -ésima cidade na rota como uma interpolação linear entre os parâmetros v^+ e v^- , proporcionalmente ao peso acumulado q_k pelo ladrão saindo da k -ésima cidade da rota definido na Equação 2.21. A Equação 2.22 corresponde à restrição de integralidade das variáveis z_i . Finalmente, a Equação 2.23 fixa a cidade inicial conforme descrito na definição do problema.

O Problema do Ladrão Viajante pode ser reduzido ao PCV em instâncias sem itens [11] ou ao PM em instâncias cujo valor do aluguel da mochila for 0, uma vez que o ladrão pode se movimentar livremente sem considerar as distâncias entre as cidades. Sendo assim, o Problema do Ladrão Viajante contém os problemas PCV e PM como casos particulares, consistindo então em um problema $\mathcal{NP}$ -difícil [11].

A Figura 2.3a ilustra uma instância do PLV ao combinar os exemplos mostrados nas Seções 2.1 e 2.2. Neste exemplo, a cidade i contém somente o item $i - 1$ disponível, ou seja, $m = n - 1$, $\alpha_1 = \emptyset$ e $\alpha_i = \{i - 1\}$. Nessa figura, para cada cidade, o semicírculo superior possui raio proporcional ao valor do item que está localizado nela, enquanto o

inferior possui raio proporcional ao peso do item. Por simplicidade, é ilustrado um exemplo com exatamente um item por cidade. Entretanto, note que a formulação do problema não possui essa limitação.

Considerando os valores de $v^+ = 1$, $v^- = 0,1$, $R = 0,25$ e $W = 5$, a Figura 2.3b ilustra a solução ótima formada por $x = (1, 4, 3, 2, 5)$ e $z = (0, 1, 0, 1)$ com valor da função objetivo igual a 0,69.

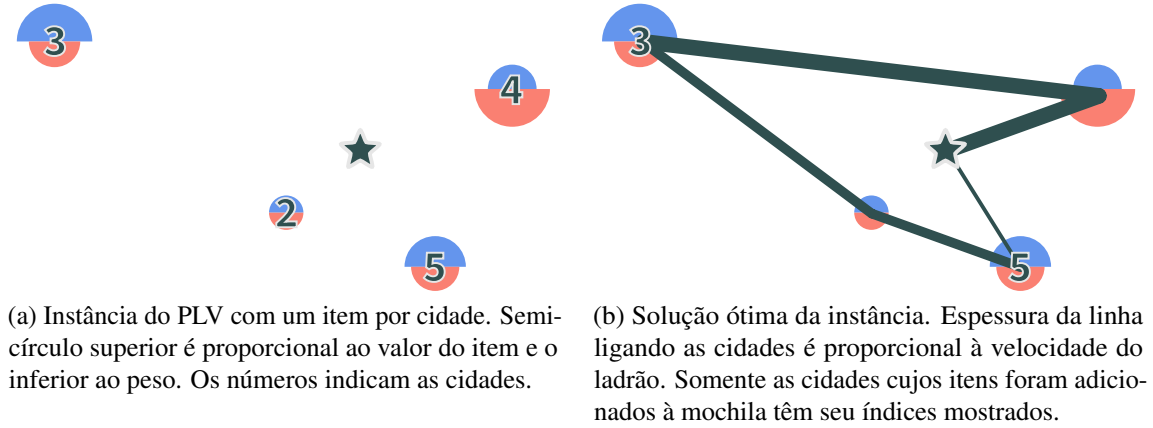


Figura 2.3: Exemplo de instância e solução do Problema do Ladrão Viajante.

Essa instância serve para ilustrar uma diferença importante entre o Problema do Ladrão Viajante e os seus subproblemas componentes: tanto a solução ótima do subproblema do Caixeiro Viajante quanto do subproblema da Mochila não necessariamente aparecem como componentes da solução ótima de uma instância do PLV. Também é importante notar que, diferentemente do que acontece no componente do PCV utilizando o plano cartesiano conforme definido na Seção 2.1, o sentido tomado pela rota de uma solução do PLV é relevante. No subproblema do Caixeiro Viajante definido em um espaço métrico, tal como o plano cartesiano, percorrer uma solução (rota) em qualquer sentido resulta no mesmo valor da função objetivo. No PLV, por sua vez, devido ao acúmulo de peso dos itens coletados na mochila que influencia a velocidade do ladrão, o sentido é relevante. A Figura 2.3b ilustra esse ponto por meio da espessura da linha ligando as cidades. A rota começa pela linha de maior espessura, consequentemente de maior velocidade, e termina com a linha de menor espessura, visto que o ladrão já acumula alguns itens na sua mochila, o que diminui sua velocidade.

3. Conceitos Fundamentais

Este capítulo apresenta os conceitos fundamentais utilizados pelos principais trabalhos da literatura sobre o Problema do Ladrão Viajante e utilizados por este trabalho.

3.1 Heurística de Lin-Kernighan para o PCV

A Heurística de Lin-Kernighan [8] é uma heurística que pode ser aplicada ao Problema Simétrico do Caixeiro Viajante e se baseia na escolha de um número variável k de cidades a serem trocadas com outras cidades de uma rota, de forma que a nova rota possua menor distância total. Segundo os autores, valores k para esse número de cidades iguais a 2 ou 3 geram sucesso moderado em encontrar boas rotas, entretanto, é difícil especificar *a priori* qual o melhor valor de k . O esforço computacional aumenta rapidamente conforme o valor de k aumenta e é difícil saber com antecedência o melhor valor de k que leve a um bom balanço entre tempo de execução e qualidade da solução. Para se contornar a dificuldade em se fixar um valor de k *a priori*, a heurística de Lin-Kernighan incrementa o valor k de cidades a serem trocadas quando não é mais possível encontrar melhores soluções com a quantidade corrente de k de trocas.

A forma como a Heurística de Lin-Kernighan escapa do mínimo local é permitindo que trocas individuais gerem soluções piores que a corrente, desde que todo o conjunto de k trocas gere uma rota de menor distância, o que é realizado da seguinte forma: a heurística considera, individualmente, as trocas de maior ganho seguidas das de menor perda. Caso, após o total de k trocas, não se obtenha uma rota melhor que a corrente, um procedimento de *backtracking* é utilizado para escolher alternativas às trocas realizadas anteriormente. A heurística termina quando um determinado número de alternativas são consideradas em todos os *backtrackings* sem encontrar melhorias. Os autores da Heurística de Lin-Kernighan propõem o uso de cinco alternativas para os dois primeiros níveis de *backtracking*, enquanto níveis posteriores consideram somente uma troca.

Uma popular extensão da Heurística de Lin-Kernighan é a Heurística Chained Lin-Kernighan que propõe um número variável de alternativas utilizadas no *backtracking* para cada nível [12]. Essa heurística utiliza também perturbações aleatórias na solução para o caso em que todo *backtracking* seja realizado sem encontrar melhoria.

3.2 Triangulação de Delaunay

Dado um conjunto de n cidades $C = \{1, \dots, n\}$ e uma matriz de distâncias $D = \{d_{ij} \geq 0; \forall i, j \in C\}$ em que d_{ij} é a distância da cidade i à cidade j , a Região de Voronoi de um ponto $i \in C$ é definida como $VR(i) = \{p \in \mathbb{R}^2 \mid d_{pi} \leq d_{pj}, \forall j = \{1, \dots, n\}\}$, em outras palavras, $VR(i)$ é o conjunto de todos os pontos que estão tão próximos de i quanto de quaisquer outra cidade em C [13]. O conjunto de todas as Regiões de Voronoi é chamado de Diagrama de Voronoi.

Dado um Diagrama de Voronoi, a Triangulação de Delaunay de um conjunto de cidades C de tamanho n , definido como $T(C) = (t_1, \dots, t_n)$, é tal que $j \in t_i$ se, e somente se, $VR(i) \cap VR(j) \neq \emptyset$ [13].

A Figura 3.1 ilustra os conceitos de Diagrama de Voronoi e Triangulação de Delaunay para um conjunto C de $n = 51$ cidades. Os círculos azuis representam as cidades do conjunto C posicionadas no plano cartesiano. Para todos os pares a e b de cidades distintas em C , as linhas claras da Figura 3.1 representam os pontos em que vale a desigualdade $VR(a) \cap VR(b) \neq \emptyset$, ou seja, pontos cuja distância à cidade a e à cidade b é a mesma e não há cidade mais próxima. Dada a Triangulação de Delaunay $T(C) = (t_1, \dots, t_n)$ as linhas escuras da Figura 3.1 unem cidades a e b onde $a \in t_b$ ou $b \in t_a$, formando triângulos.

Krasnogor et al.[14] demonstraram empiricamente que cidades adjacentes em uma rota ótima do PLV têm grande probabilidade de estarem presentes em um mesmo triângulo da Triangulação de Delaunay da instância.

3.3 Busca Local Rápida

Alsheddy et al.[4] propõem, em conjunto com uma Busca Local Guiada, uma Busca Local Rápida como uma maneira de reduzir o tamanho de uma vizinhança e melhorar a eficiência de uma busca local. A Busca Local Rápida é particularmente útil em instâncias grandes de um problema, especialmente quando realizar repetitivas buscas exaustivas em toda a vizinhança é computacionalmente intratável.

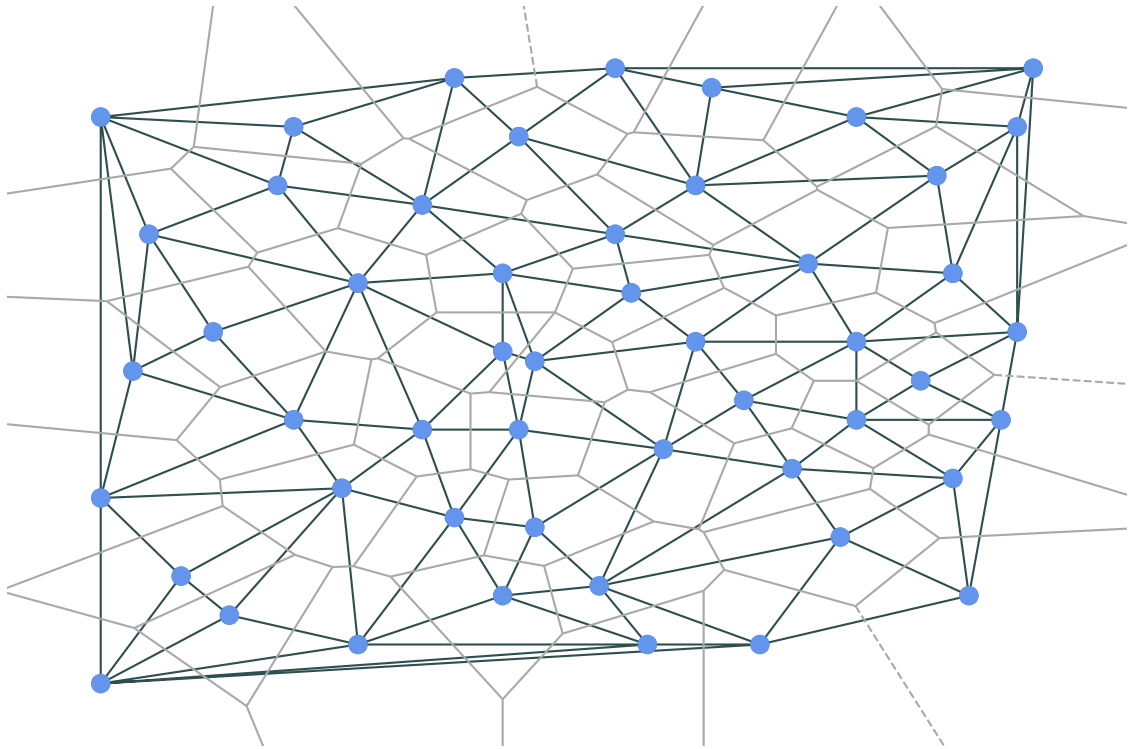


Figura 3.1: Exemplo de Diagrama de Voronoi e Triangulação de Delaunay. Os círculos azuis representam pontos no plano cartesiano. A Diagrama de Voronoi é representado pelas linhas claras, enquanto a Triangulação de Delaunay é representado pelas linhas escuras.

Na Busca Local Rápida, a vizinhança escolhida para uma busca local é dividida em subvizinhanças menores e um booleano de “ativação” é vinculado a cada uma dessas subvizinhanças. Somente as subvizinhanças cujo booleano de “ativação” seja verdadeiro são consideradas para a busca. Nesse caso, a subvizinhança é denominada “ativa”.

Durante a busca, cada subvizinhança ativa é considerada individualmente, em ordem arbitrária, e a busca não reinicia após encontrar um vizinho na subvizinhança corrente melhor do que a melhor solução até então. Ao invés disso, prossegue para a próxima subvizinhança na ordem escolhida. Essa ordem pode ser fixa ou adaptável à fase em que se encontra a heurística.

Inicialmente, todas as subvizinhanças estão ativas, se uma subvizinhança examinada não possuir um vizinho melhor do que a melhor solução até então, ela se torna “inativa” e não mais será considerada na busca. Caso contrário, a subvizinhança permanece ativa, a melhor solução até então é atualizada e outras subvizinhanças podem ser “reativadas” para serem consideradas novamente na busca. A reativação é aplicada a subvizinhanças nas quais se espera que haja vizinhos de melhor valor de função objetivo.

Conforme a solução melhora, se torna difícil encontrar vizinhos melhores e cada vez menos subvizinhanças se mantêm ativas até que eventualmente todas as subvizinhanças se

tornam desativadas. Nesse caso, a solução encontrada é um *mínimo local aproximado* [4].

A inativação e reativação de subvizinhanças promovem movimentos que melhoram partes específicas de uma solução e examinam menos vizinhanças com baixo potencial de melhoria.

A Busca Local Rápida, em geral, não encontra soluções tão boas quanto uma busca local tradicional, entretanto, ao reduzir o tamanho da vizinhança, por meio do uso de subvizinhanças inativas, espera-se que a velocidade da busca compense essa perda de qualidade quando aplicada em conjunto com uma heurística.

3.4 Busca em Vizinhança Grande

A maioria das meta-heurísticas define uma vizinhança explicitamente por uma operação de movimentação na solução. Na meta-heurística Busca em Vizinhança Grande, em inglês *Large Neighborhood Search* - LNS, a vizinhança é definida implicitamente por procedimentos de destruição e reconstrução de uma solução [5].

A Busca em Vizinhança Grande foi proposta por Shaw[15] para o Problema de Roteamento de Veículos que, assim como o PCV, envolve encontrar uma ou mais rotas de menor valor de uma função objetivo que atenda determinadas restrições. A heurística proposta é baseada no processo de contínuo relaxamento (destruição), em que certas cidades são removidas da rota, e reotimização (reconstrução), em que as cidades removidas são reinseridas à rota em posições que melhorem o valor da função objetivo. O processo de destruição envolve escolher cidades que resultem em boas oportunidades para o método de reconstrução gerar boas soluções. O critério utilizado Shaw[15] para a destruição é o de proximidade entre cidades. Deve-se também descartar cidades que, ao serem removidas, tenham alta probabilidade de serem reinseridas na mesma posição pelo método de reinserção. Com o intuito de obter uma heurística eficiente, a Busca em Vizinhança Grande almeja remover o menor número possível de cidades que resultem em uma melhor solução quando essas são readicionadas à rota. Sendo assim, a heurística inicia fazendo a remoção de 1 cidade e, conforme a heurística prossegue sem melhorias, aumenta o número de cidades a serem removidas e readicionadas. Já no processo de reconstrução, é importante decidir se será feita uma reinserção ótima, isto é, uma reinserção das cidades removidas em uma localidade da rota que resulte no maior decremento da função objetivo, o que pode ser lento e gerar soluções parecidas com as já encontradas, ou permitir que a reinserção não seja ótima, gerando maior diversificação [5].

4. Trabalhos Relacionados

Este capítulo apresenta os principais trabalhos, suas abordagens e contribuições para a solução do Problema do Ladrão Viajante. Cada seção apresenta um trabalho relevante da literatura, descrevendo a abordagem aplicada e os resultados obtidos.

4.1 Trabalho de Bonyadi et al., de 2014

Uma das primeiras tentativas de resolver o PLV foi em 2014, por meio da heurística Baseada em Densidade (*Density-based* ou DH) e da heurística CoSolver [16].

A heurística DH ignora a interdependência entre os subproblemas do PLV e resolve cada um desses subproblemas de forma separada. Inicialmente, é gerada uma rota para o subproblema do Caixeiro Viajante utilizando a heurística CHAINED-LIN-KERNIGHAN [12], descrita na Seção 3.1, em seguida, é feita uma escolha gulosa de configuração de mochila para essa rota obtida.

Mais especificamente, dada uma rota $x = (x_1, \dots, x_n)$ de tamanho n , o conjunto B de itens de tamanho m , a matriz de distância D entre as cidades da instância, o valor dos itens $p = (p_1, \dots, p_m)$, o peso dos itens $w = (w_1, \dots, w_m)$, um valor $W > 0$ de capacidade da mochila, um valor de aluguel $R \geq 0$ da mochila por unidade de tempo, os valores de velocidade máxima e mínima $v^+ > v^- > 0$ de movimentação do ladrão e a lista de cidades $A' = (a'_1, \dots, a'_m)$ em que cada item se encontra, a heurística DH define uma estimativa do benefício $g = (g_1, \dots, g_m)$ da inclusão de um item à mochila dada a rota x pela Equação 4.1. A Equação 4.2 é equivalente ao tempo gasto pelo ladrão para percorrer a rota x a partir da cidade onde se encontra o item i até o final da rota e retornar à primeira cidade da instância. A Equação 4.3 é equivalente à distância percorrida pelo ladrão até o final da rota x a partir da cidade onde se encontra o item i e retornando à primeira cidade

da instância.

$$g_i = p_i - R \times t'_i \quad (4.1)$$

$$\text{onde: } t'_i = \frac{d'_i}{v^+ - w_i \frac{v^+ - v^-}{W}} \quad (4.2)$$

$$d'_i = \sum_{k=x_{a'_i}}^{n-1} d_{x_k x_{k+1}} + d_{x_n x_1} \quad (4.3)$$

A heurística DH cria uma configuração de mochila por meio da inserção dos itens a uma mochila, inicialmente vazia, pela ordem não decrescente de g_i . O item é inserido somente se o peso total da mochila, após sua inserção, for menor ou igual à capacidade máxima W da mochila, e se a inserção desse item não diminuir o valor da função objetivo do PLV, detalhada na Equação 2.16.

Já a heurística CoSolver decompõe o PLV em dois subproblemas, denominados Problema do Caixeiro Viajante com Mochila e Problema da Mochila em Rota; resolve os dois subproblemas de forma independente; e, por fim, compõe uma solução para o PLV a partir das soluções desses dois subproblemas.

O Problema do Caixeiro Viajante com Mochila (*Traveling Salesman with Knapsack Problem* ou TSKP) é equivalente a encontrar uma rota para o PLV dada uma configuração de mochila fixa como parte da instância e o Problema da Mochila em Rota (*Knapsack on the Route Problem* ou KRP) é equivalente a encontrar uma configuração de mochila para o PLV dada uma rota fixa como parte da instância.

Dada uma instância do PLV, a heurística CoSolver inicialmente resolve o subproblema da Mochila dessa instância utilizando a abordagem de Programação Dinâmica (ver [17]), desconsiderando as cidades. A partir da solução do subproblema da Mochila, são gerados e resolvidos sucessivos KRPs e TSKPs. O TSKP é resolvido utilizando um algoritmo exato e o KRP utiliza uma heurística proposta pelos autores.

Com o objetivo de comparar as heurísticas DH e CoSolver, Bonyadi et al.[16] geraram 45 instâncias do PLV com o número de cidades variando de 3 a 33 e o número de itens variando de 10 a 146. A solução ótima dessas instâncias foi obtida por meio de uma busca exaustiva. A qualidade das heurísticas DH e CoSolver foi avaliada por meio de uma métrica que calcula a proximidade do valor da solução obtida por cada heurística com o valor da solução ótima.

Os resultados da heurística CoSolver foram melhores para boa parte das instâncias

consideradas pelos autores se comparada à heurística DH. Em especial, para instâncias com menos de 10 cidades e instâncias em que a razão entre o número de itens e a quantidade de cidades é maior que 10, a heurística CoSolver obteve soluções muito próximas das soluções ótimas. Considerando a qualidade superior da heurística CoSolver se comparada à heurística DH, os autores concluem que considerar a interdependência entre subproblemas é vantajoso para resolver problemas com múltiplos componentes, como é o caso do PLV.

4.2 Trabalho de Polyakovskiy et al., de 2014

Polyakovskiy et al.[1] propuseram, juntamente com um conjunto de instâncias do PLV, a aplicação de três heurísticas para a solução do PLV: Heurística Simples (*Simple Heuristic* ou SH), Busca Local Aleatória (*Random Local Search* ou RLS) e um Algoritmo Evolutivo (*Evolutionary Algorithm* ou EA).

A heurística SH é uma heurística gulosa muito similar à heurística DH descrita na Seção 4.1. Além da estimativa de benefício definida pela Equação 4.1, a heurística SH utiliza uma estratégia para eliminar itens não vantajosos, em que, caso um item não vantajoso seja adicionado à mochila para uma dada rota, o valor da função objetivo é decrescido se comparado ao valor da função objetivo para a mesma rota e mochila antes da inclusão do item não vantajoso.

Dados os mesmos valores utilizados pela Equação 4.1, o tempo t' gasto pelo ladrão para percorrer toda a rota x com a mochila vazia, assim como o tempo gasto pelo ladrão para percorrer a rota x a partir da cidade onde se encontra o item i até o final da rota e retornar à primeira cidade da instância dado pela Equação 4.2, caso a Equação 4.4 possua um valor negativo para determinado item i e rota x , o item i é considerado não vantajoso e nunca deve ser adicionado à mochila para a rota x , independente da configuração da mochila. Isso permite à heurística SH ignorar o item i enquanto a rota da solução corrente for x .

$$u_i = R \times t' + (p_i - R \times t'_i) \quad (4.4)$$

As heurísticas RLS e EA utilizam uma vizinhança denominada *Bit Flip* em que, dado um item em uma configuração de mochila, é utilizada a operação de *flip*: caso o item esteja na mochila, ele é removido, e, caso não esteja, é então adicionado.

A heurística RLS é uma busca local aleatória em que a melhor solução encontrada até então é inicializada a partir de uma rota x obtida pela heurística CHAINED-LIN-KERNIGHAN e uma configuração de mochila vazia z . É então escolhido iterativamente

um item aleatório i da instância para aplicar a vizinhança *Bit-Flip*, criando uma nova configuração de mochila z' . Caso o valor da função objetivo do PLV, definida pela Equação 2.16, seja maior para a rota x e essa nova mochila z' , a melhor solução encontrada até então é atualizada com a nova mochila z' .

A heurística EA é um algoritmo evolutivo que gera soluções iniciais da mesma forma que a heurística RLS. Para a criação de uma configuração de mochila candidata z' , é aplicado o movimento do *Bit Flip* a cada um dos itens da mochila com probabilidade de $1/m$, onde m é a quantidade de itens da instância. Caso o valor da função objetivo do PLV seja maior para a rota x e a nova mochila z' , a melhor solução encontrada até o momento é atualizada com a nova mochila z' .

Para testar as heurísticas propostas, os autores propuseram também um conjunto de 9720 instâncias que viria a se tornar o conjunto padrão de instâncias para comparação de heurísticas para o PLV.

As heurísticas SH, RLS e EA foram executadas para todas as 9720 instâncias propostas. A heurística SH é uma heurística gulosa e não possui componente aleatório, sendo assim, foi executada somente uma vez. Por sua vez, as heurísticas RLS e EA possuem componente aleatório e, portanto, foram executadas 30 vezes cujo critério de parada adotado correspondeu a um tempo máximo de 10 minutos ou a 10.000 iterações sem melhoria.

Em geral, a heurística EA obteve melhores resultados para instâncias pequenas do PLV devido à sua melhor capacidade de escapar de mínimos locais se comparada à heurística RLS [1]. Porém, para diversas instâncias maiores, a heurística RLS encontra melhores soluções se comparada à heurística EA devido ao fato de a heurística EA ter maior probabilidade de gerar configurações de mochila que invalidam a restrição de capacidade do PLV, o que impede que sejam gerados novos vizinhos e que uma nova solução melhor que a encontrada até então seja descoberta. Para algumas das maiores instâncias do conjunto das 9720 propostas, a heurística SH obteve as melhores soluções, visto que as heurísticas RLS e EA iniciam a partir de uma configuração de mochila vazia e não conseguem gerar uma solução melhor dentro do critério de parada estabelecido. Os autores concluem, por meio desses experimentos, que encontrar uma boa solução inicial é um passo importante para heurísticas aplicadas ao PLV.

4.3 Trabalho de Mei, Li e Yao, de 2014

A heurística MATLS é um Algoritmo Memético com Busca Local em Dois Estágios (em inglês *Memetic Algorithm with the Two-stage Local Search* - MATLS), proposta por Mei, Li e Yao[11], que divide a busca local em dois estágios de maneira similar à heurística CoSolver explicada na Seção 4.1. O primeiro estágio de MATLS busca melhorar a rota de uma solução do PLV, enquanto o segundo estágio de MATLS melhora a configuração da mochila dada a rota obtida no estágio anterior.

Com o objetivo de diminuir a complexidade da avaliação da função objetivo do PLV, a heurística MATLS aproxima a função objetivo no primeiro estágio como o equivalente a minimizar a distância total da rota da solução. No segundo estágio, a heurística MATLS utiliza três aproximações do ganho na função objetivo ao se adicionar ou remover um item i de uma configuração de mochila dada certa rota. A primeira aproximação, denominada “aproximação para mochila vazia”, é igual ao valor do ganho na função objetivo caso o item i seja inserido em uma mochila vazia. A segunda aproximação, denominada “aproximação de pior caso”, considera o ganho caso o item i seja o último item a ser adicionado à mochila quando percorrendo uma dada rota, ou seja, considera que a capacidade total dos itens da mochila diferentes de i foi atingida em cidade anterior à do item i . A última aproximação, denominada “ganho esperado”, considera que o peso dos itens da mochila está homogeneamente distribuído entre todas as cidades da rota, o que implica em uma aceleração constante do ladrão durante o percurso e, por conseguinte, facilita o cálculo aproximado da função objetivo.

Baseada nas três aproximações de ganho de item, é criada uma heurística para inserção dos itens em uma mochila vazia dada uma rota. Os itens são inicialmente ordenados pela sua aproximação de ganho para mochila vazia, em seguida os itens ordenados são considerados individualmente para inserção. Primeiro é checada a aproximação de pior caso, se o valor dessa aproximação for positivo, então é garantido que o item melhora a função objetivo e o item é adicionado à mochila. Caso o ganho aproximado do pior caso não seja positivo, é checado se ganho esperado é positivo, nesse caso o item é adicionado à mochila, visto que é provável que ele gere um aumento na função objetivo.

A heurística MATLS cria uma população inicial com 10 indivíduos contendo rotas obtidas pela heurística CHAINED-LIN-KERNIGHAN e 20 indivíduos com uma rota criada a partir de da “Árvore Geradora Mínima”. Para cada indivíduo da solução inicial, é gerada uma configuração de mochila utilizando uma heurística gulosa similar à heurística SH descrita na Seção 4.2. Em cada geração do MATLS, dois pais são selecionados para

crossover da rota, gerando a rota do descendente. Uma configuração de mochila gulosa é gerada para o descendente e uma busca local é aplicada ao descendente utilizando as aproximações do ganho de adição do item à mochila. Por fim, o descendente é adicionado à população no lugar do indivíduo de menor valor da função objetivo.

O autores da heurística MATLS utilizaram 18 instâncias com mais de 10.000 cidades e 100.000 itens dentre as instâncias propostas por Polyakovskiy et al.[1] para comparar seus resultados com as heurísticas RLS e EA descritas na Seção 4.2. O critério de parada foi de 10 minutos, igual aos utilizados pelas heurísticas RLS e EA. A heurística MATLS obteve uma melhor solução para todas as instâncias escolhidas em comparação às heurísticas RLS e EA. Os autores da heurística MATLS concluem então que é importante considerar estratégias de redução de complexidade no projeto de heurísticas para o PLV.

4.4 Trabalho de Faulkner et al., de 2015

Faulkner et al.[18] propuseram uma heurística gulosa, chamada PACKITERATIVE, de criação de uma configuração de mochila a partir de uma rota. Dado um conjunto B de itens de tamanho m , o valor dos itens $p = (p_1, \dots, p_m)$, o peso dos itens $w = (w_1, \dots, w_m)$, a distância d'_i percorrida pelo ladrão a partir da cidade em que o item i se encontra até o final da rota x e retornando à primeira cidade, dada pela Equação 4.3 e um parâmetro livre α , a heurística PACKITERATIVE utiliza uma pontuação dada pela Equação 4.5 para escolher, de forma gulosa, quais itens adicionar a uma configuração de mochila dada uma rota.

$$g_i = \frac{p_i^\alpha}{w_i^\alpha d_i'}, i \in B \quad (4.5)$$

Em conjunto com uma busca local utilizando a vizinhança *Bit Flip*, descrita na Seção 4.2, foi também criada uma busca local com a vizinhança *Insertion*, em que uma cidade é removida da rota e adicionada em outro ponto da rota.

A partir da combinação das heurísticas CHAINED-LIN-KERNIGHAN, PACKITERATIVE, busca local com a vizinhança *Bit Flip*, busca local com a vizinhança *Insertion* e a heurística EA descrita na Seção 4.2, os autores criaram onze heurísticas:

- Heurística S1: executa a CHAINED-LIN-KERNIGHAN e depois PACKITERATIVE;
- Heurística S2: executa a Heurística S1 e depois uma busca local com a vizinhança *Bit Flip*;
- Heurística S3: executa a Heurística S1 e depois a heurística EA;

- Heurística S4: executa a Heurística S1 e depois uma busca local com a vizinhança *Insertion*;
- Heurística S5: repete a heurística S1 até que o critério de parada seja atingido;
- Heurística C1: executa a Heurística S1 e depois repete a execução alternada de uma busca local na vizinhança *Bit Flip* e uma busca local na vizinhança INSERTION;
- Heurística C2: executa a Heurística S1 e depois repete a execução alternada de uma busca local na vizinhança *Bit Flip*, a heurística EA e uma busca local na vizinhança INSERTION;
- Heurística C3: gera rotas com a heurística CHAINED-LIN-KERNIGHAN por 10% do tempo utilizado como critério de parada, depois aplica a heurística PACKITERATIVE a essas rotas, escolhe a melhor solução dentre as obtidas dessa forma e repete alternadamente a aplicação de uma busca local na vizinhança *Bit Flip* e uma busca local na vizinhança INSERTION;
- Heurística C4: gera rotas com a heurística CHAINED-LIN-KERNIGHAN por 10% do tempo utilizado como critério de parada, depois aplica a heurística PACKITERATIVE a essas rotas, escolhe a melhor solução dentre as obtidas dessa forma e repete alternadamente a aplicação de uma busca local na vizinhança *Bit Flip*, a heurística EA e uma busca local na vizinhança INSERTION;
- Heurística C5: repete a heurística C1 até que o critério de parada seja atingido;
- Heurística C6: repete a heurística C2 até que o critério de parada seja atingido.

A heurística S5 obteve a melhor performance se comparada às outras heurísticas propostas pelos autores e se comparada às heurísticas SH, RLS e EA descritas na Seção 4.2, quando executada por 10 minutos em 72 das instâncias de Polyakovskiy et al.[1].

Adicionalmente, uma investigação feita para o valor de α na Equação 4.5 demonstrou que expoentes maiores para os valores de p_i e w_i , se comparados ao expoente de d'_i , resultam em configurações de mochila com maior valor da função objetivo do PLV em conjunto com uma rota. Sendo assim, o expoente de d'_i é omitido na Equação 4.5, o que é equivalente a ser igual a 1. Foram testados valores para α no intervalo $[1, 6]$ com incremento de 0,25. Não foi possível chegar a um resultado conclusivo de qual valor de α é melhor para todos os tipos de instância.

4.5 Trabalhos de Yafrani e Ahiod, de 2016 e 2018

A heurística CS2SA para o PLV é uma extensão da heurística CoSolver, descrita na Seção 4.1, que decompõe o PLV em dois subproblemas, denominados Problema

do Caixeiro Viajante com Mochila e Problema da Mochila em Rota, e resolve os dois subproblemas de forma independente, realizando certa comunicação [19].

A solução inicial da heurística CS2SA é obtida de forma similar à heurística MATLS descrita na Seção 4.3. Em seguida, é feita uma busca local na rota, com a configuração de mochila fixa, utilizando a vizinhança *2-opt* em que são removidas duas arestas da rota e adicionadas outras duas, completando a rota. A vizinhança *2-opt* equivale a inverter uma subsequência da lista de permutação de cidades e corresponde a uma das vizinhanças clássicas para o PCV [20]. É utilizada a Triangulação de Delaunay, explicada na Seção 3.2, como lista de candidatos da vizinhança da busca local na rota. Por fim, é utilizada a meta-heurística *Simulated Annealing* para fazer uma busca na configuração da mochila com a rota fixa escolhendo itens aleatórios e aplicando a vizinhança *Bit Flip*. A heurística CS2SA repete a busca local na rota e o *Simulated Annealing* na configuração da mochila enquanto não conseguir encontrar solução melhor do que a melhor solução encontrada até então, ou até que o critério de parada seja atingido.

Há um erro no cálculo da função objetivo da heurística CS2SA em que as distâncias entre as cidades são arredondadas para o inteiro mais próximo, ao invés de ser aplicada a função “topo”, arredondando para o menor inteiro maior ou igual à distância entre as cidades [21], o que impede que os resultados do artigo sejam utilizados. Entretanto, a heurística CS2SA foi revisitada na forma de duas variantes: CS2SA* e CS2SA-R [22]. As heurísticas CS2SA* e CS2SA-R são similares à heurística CS2SA, porém, com foco na melhoria da implementação por meio de diversas técnicas de pré-processamento para aumentar a velocidade do cálculo da função objetivo do PLV. Outras mudanças foram a adição de uma busca local na configuração da mochila utilizando a vizinhança *Bit Flip* e a modificação da busca local com a vizinhança *2-opt* para terminar quando a melhoria obtida entre soluções sucessivas for menor que um determinado valor parametrizado. A heurística CS2SA-R utiliza a heurística CS2SA* por meio de sucessivos *restarts*. Foram realizados procedimentos de ajuste fino de parâmetros para cada heurística.

Na comparação dos resultados, Yafrani e Ahiod[22] utilizaram três tipos de instâncias com 20 diferentes quantidade de cidades, totalizando 60 instâncias das propostas por Polyakovskiy et al.[1]. Foram escolhidas as heurísticas MATLS e S5 para comparação. As heurísticas CS2SA*, CS2SA-R e S5 obtiveram melhores soluções que a heurística MATLS para quase todas as instâncias escolhidas, em particular para instâncias com mais de 1.000 itens. A heurística CS2SA* tem melhores resultados para instâncias grandes, enquanto a heurística CS2SA-R tem melhores resultados para instâncias pequenas.

4.6 Trabalho de Wagner, de 2016

Wagner[23] propõe heurísticas baseadas em um algoritmo de otimização da Colônia de Formigas, chamado em inglês de *MAX-MIN ant system* - MMAS, para construir rotas de uma instância do PLV, em que formigas constroem a rota escolhendo a próxima cidade baseada em uma probabilidade proporcional ao feromônio entre os pares de cidades. Dada uma rota construída pelo MMAS, é utilizada a heurística *PACKITERATIVE*, descrita na Seção 4.4, para construir uma configuração de mochila. A partir de uma solução criada dessa forma, é aplicada, opcionalmente, uma busca local utilizando as vizinhanças *Bit Flip*, *Insertion* ou a heurística EA, etapa denominada pelos autores de *boosting*.

Foram utilizadas duas configurações de MMAS: MMASls3 e MMASls4, gerando assim as heurísticas M3 e M4. Às heurísticas M3 e M4 é possível adicionar o *boosting*, gerando as heurísticas M3B e M4B.

As heurísticas M3, M4, M3B e M4B foram executadas 30 vezes durante 10 minutos em 108 das 9720 instâncias propostas por Polyakovskiy et al.[1]. Esses experimentos mostraram que as heurísticas baseadas no MMAS escolhem rotas mais longas se comparadas às heurísticas S1-5, C1-6 e MATLS, porém de melhor valor da função objetivo. MMAS obteve melhores soluções para instâncias com até 250 e 2.000 itens. O autor conclui que isso ocorre, pois as heurísticas baseadas no MMAS focam mais em gerar boas rotas para o PLV do que em gerar boas rotas especificamente para o subproblema do Caixeiro Viajante.

4.7 Outros Trabalhos da Literatura

Wuijts e Thierens[21] estudaram as características do *Fitness Landscape* de instâncias pequenas do PLV com buscas locais utilizando as vizinhanças *2-opt*, *Insertion*, *Bit Flip* e uma vizinhança denominada *Exchange*, onde, dados dois itens aleatórios de uma mochila, um pertencente à solução e outro não, suas posições são trocadas, removendo o que está na solução e adicionando o que não está. A partir dessa análise, Wuijts e Thierens[21] propõem um Algoritmo Memético denominado MEA2P que utiliza o *crossover Edge-Assembly* para uma rota da solução e o *crossover EAX* para a configuração da mochila.

Chagas e Wagner[24] estudaram uma variante do PLV com duas funções objetivos e utilizam uma heurística baseada no Método de Soma Ponderada, em inglês *Weighted-Sum Method* - WSM, para transformar as duas funções objetivos em somente uma. Assim como os outros trabalhos da área, Chagas e Wagner[24] utilizaram a heurística CHAINED-LINKERNIGHAN para gerar uma rota inicial. A configuração de mochila gerada a partir da

rota inicial utiliza um heurística gulosa aleatória proposta pelos autores. Como trabalho adicional, os autores adaptaram sua heurística para a variante tradicional do PLV, que é a variante abordada neste trabalho.

5. Busca Local Rápida em Vizinhaça Grande para o Problema do Ladrão Viajante

Propõe-se, neste trabalho, uma Busca Local Rápida em Vizinhaça Grande aplicada ao Problema do Ladrão Viajante (BLRVG-PLV). A heurística proposta é a combinação de uma Busca Local Rápida [4], usada para minimizar o custoso cálculo da função objetivo do PLV em subvizinhanças de baixo potencial de melhoria, com uma Busca em Vizinhaça Grande [15], usada para gerar perturbações com o intuito de escapar de mínimos locais.

A BLRVG-PLV parte de uma solução inicial gulosa e faz uma Busca Local Rápida em que todas as subvizinhanças estão ativas. Após atingir um mínimo local aproximado, é aplicado um processo de destruição e reconstrução da rota da solução e reativadas somente as subvizinhanças modificadas nesse processo, limitando o número de subvizinhanças ativas. Essa reativação seletiva permite a Busca Local Rápida focar em regiões de mudança, afastando a nova solução da anterior e criando maiores chances de escapar do mínimo local, além de evitar que seja feita a busca em subvizinhanças sem modificações limitando o custoso cálculo da função objetivo.

A heurística Busca Local Rápida em Vizinhaça Grande para o Problema do Ladrão Viajante (BLRVG-PLV), apresentada na Seção 5.5, possui as seguintes etapas: construção gulosa de uma solução (Seção 5.1), Busca Local Rápida (Seção 5.2), conjunto de soluções iniciais (Seção 5.3) e Busca em Vizinhaça Grande (Seção 5.4).

5.1 Construção Gulosa de Solução para o PLV

A construção gulosa de uma solução para o PLV foi dividida em duas fases: a primeira cria uma rota inicial sem itens, e a segunda fase, para uma dada rota, escolhe de forma gulosa os itens da mochila, criando, assim, uma configuração para a mochila. A seguir é apresentada uma descrição detalhada de cada uma das duas fases, seguida do algoritmo

final que utilizando as duas fases.

5.1.1 Rota Inicial

O primeiro passo da heurística proposta para construir uma solução inicial para o PLV envolve construir uma rota entre as cidades da instância desconsiderando os itens a serem adicionados à mochila. Esse passo equivale a encontrar uma solução para o subproblema do Caixeiro Viajante. Para tal, foi utilizado o algoritmo CHAINED-LIN-KERNIGHAN [12] que é uma adaptação da heurística de Lin-Kernighan [8], descritos na Seção 3.1, proposta para o PCV.

A heurística CHAINED-LIN-KERNIGHAN recebe como entrada um conjunto de cidades C , uma matriz de distâncias D entre as cidades em C e um parâmetro r referente à semente do gerador de números pseudo-aleatórios. Devido ao fato de que essa heurística possui componentes aleatórios na sua implementação, é possível obter diferentes rotas para uma mesma instância, variando o parâmetro r .

5.1.2 Escolha dos Itens da Mochila para uma Dada Rota

Uma vez obtida a rota inicial, é necessária uma configuração inicial para a mochila. Criou-se, portanto, um algoritmo guloso que, dada uma rota, são adicionados itens, ordenados pela estimativa do benefício da inclusão do item na solução corrente enquanto houver espaço na mochila.

Considerando-se uma heurística construtiva para uma solução inicial do PM, um critério guloso de seleção de itens, comumente adotado na literatura, é a razão entre o valor de um item e seu peso [10]. Estendendo-se esse conceito para o PLV, é possível adaptar esse critério guloso ao se incluir no denominador da razão a distância da cidade em que esse item é selecionado até o final da rota. Uma forma de modelar essa pontuação foi introduzida por Faulkner et al.[18] e foi definida na Equação 4.5.

O Algoritmo 1 descreve o cálculo da pontuação dos itens de uma mochila dada uma rota x de tamanho n , a matriz de distâncias D , o conjunto de itens B de tamanho m , o valor dos itens $p = (p_1, \dots, p_m)$, o peso dos itens $w = (w_1, \dots, w_m)$, a lista de conjuntos de disponibilidade dos itens $A = (a_1, \dots, a_n)$ e o parâmetro δ como expoente do cálculo da pontuação definida na Equação 4.5.

Na linha 1, é inicializada a distância até o final da rota com a distância da última cidade da rota até a cidade inicial. Na linha 2 é iniciado um laço que percorre as cidades da

Algoritmo 1: PONTUAÇÃO-GULOSA(x, D, B, p, w, A, δ)

```
1  $q \leftarrow d_{x_n 1}$ 
2 para  $i = n, n-1, \dots, 2$  faça
3   para cada  $j \in a_{x_i}$  faça
4      $g_j \leftarrow \frac{p_j^\delta}{w_j^\delta q}$ 
5   fim
6    $q \leftarrow q + d_{x_{i-1} x_i}$ 
7 fim
8 retorna  $(g_1, \dots, g_m)$ 
```

rota do final para o início, ignorando a primeira cidade da rota que não possui itens, de forma a se obter a distância acumulada até a cidade corrente x_i , por meio da linha 6. Nas linhas 3-5, para cada item disponível na cidade x_i , é calculada a sua pontuação conforme a Equação 4.5. Finalmente, a lista de pontuações calculadas para os itens é retornada na linha 8.

O Algoritmo ESCOLHA-GULOSA-ITENS, detalhado no Algoritmo 2, utiliza a pontuação obtida no Algoritmo 1, para, seguindo um critério guloso, adicionar em uma mochila vazia tantos itens quantos couberem, respeitando a sua capacidade e priorizando aqueles de maior pontuação. Adicionalmente, são testados valores para o parâmetro δ de forma a encontrar uma configuração de mochila que gere maior valor para a função objetivo $f(\cdot, \cdot)$.

O Algoritmo 2 recebe como entrada uma rota x de tamanho n , a matriz de distâncias D entre as cidades que se encontram os itens, o conjunto B de itens de tamanho m , a função objetivo $f(\cdot, \cdot)$ dada pela Equação 2.16, o valor dos itens $p = (p_1, \dots, p_m)$, o peso dos itens $w = (w_1, \dots, w_m)$, a lista de conjuntos de disponibilidade dos itens $A = (a_1, \dots, a_n)$, a capacidade W da mochila e o parâmetro referente ao conjunto Δ de expoentes a serem testados. Como retorno, produz uma configuração de mochila z de maior valor objetivo $f(x, z)$ dentre as pontuações calculadas, tomando-se x e um $\delta \in \Delta$.

Na linha 1, inicializa-se a melhor configuração de mochila como uma mochila vazia. Na linha 4, é criada uma lista de itens em ordem não crescente pela pontuação calculada na linha 3. Nas linhas 5-12, os itens são adicionados em uma nova mochila vazia, respeitando sua capacidade e priorizando os itens de maior pontuação. Assim, nas linhas 13-15, caso a mochila criada nas linhas 6-12 gere um valor para a função objetivo maior que o valor da melhor mochila encontrada até então, atualiza-se a melhor mochila. Ao fim, na linha 17, é retornada a configuração da melhor mochila encontrada.

Algoritmo 2: ESCOLHA-GULOSA-ITENS($x, D, B, f(\cdot, \cdot), p, w, A, W, \Delta$)

```
1  $z = (z_1, \dots, z_m) \leftarrow (0, \dots, 0)$ 
2 para cada  $\delta \in \Delta$  faça
3    $g \leftarrow \text{PONTUAÇÃO-GULOSA}(x, D, B, p, w, A, \delta)$ 
4   Cria  $b' = (b'_1, \dots, b'_m)$  tal que  $g_{b'_1} \geq \dots \geq g_{b'_m}$ 
5    $w' \leftarrow 0$ 
6    $z' = (z'_1, \dots, z'_m) \leftarrow (0, \dots, 0)$ 
7   para  $i = 1, \dots, m$  faça
8     se  $w' + w_{b'_i} \leq W$  então
9        $w' \leftarrow w' + w_{b'_i}$ 
10       $z'_{b'_i} \leftarrow 1$ 
11    fim
12  fim
13  se  $f(x, z') > f(x, z)$  então
14     $z \leftarrow z'$ 
15  fim
16 fim
17 retorna  $z$ 
```

5.1.3 Solução Gulosa para o PLV

Esta seção descreve um algoritmo de construção de uma solução inicial para o PLV utilizando os conceitos das seções anteriores de construção de rota sem itens e de inserção dos itens em uma mochila para dada rota seguindo um critério guloso.

Como a direção de uma rota no PLV é importante para a solução do problema, visto que o peso dos itens selecionados nas cidades percorridas por essa rota influenciam na velocidade do ladrão, são então comparadas as mochilas geradas por uma rota e pelo inverso dessa mesma rota. O Algoritmo 3 descreve esse procedimento. As entradas e parâmetros são os mesmos do Algoritmo 2 e é retornada uma solução gulosa para o PLV, composta por uma rota e uma configuração de mochila.

Nesse algoritmo, na linha 1, cria-se uma configuração de mochila para a rota informada utilizando-se o Algoritmo 2. Na linha 2, inverte-se a rota mantendo a cidade inicial fixa e, na linha 3, é gerada uma configuração de mochila para essa nova rota invertida novamente utilizando-se o Algoritmo 2. Ao final, são comparados os valores da função objetivo para a solução original e a solução considerando a rota invertida, retornando-se aquela que possuir o maior valor (linhas 4-8).

Algoritmo 3: SOLUÇÃO-GULOSA-PLV($x, D, B, f(\cdot, \cdot), p, w, A, W, \Delta$)

```
1  $z \leftarrow \text{ESCOLHA-GULOSA-ITENS}(x, D, B, f(\cdot, \cdot), p, w, A, W, \Delta)$ 
2  $x' \leftarrow (1, x_n, x_{n-1}, \dots, x_2)$ 
3  $z' \leftarrow \text{ESCOLHA-GULOSA-ITENS}(x', D, B, f(\cdot, \cdot), p, w, A, W, \Delta)$ 
4 se  $f(x, z) > f(x', z')$  então
5   | retorna  $x, z$ 
6 senão
7   | retorna  $x', z'$ 
8 fim
```

5.2 Busca Local para o PLV

Com uma solução inicial criada a partir do algoritmo guloso, apresentado na Seção 5.1.3, o próximo passo é aplicar uma busca local.

5.2.1 Definição de Relações de Vizinhanças para o PLV

A seguir, a definição de três relações de vizinhança que serão aplicadas na busca local proposta:

- *Bit Flip* ou Inversão de Bit: adiciona um item à mochila caso o item não esteja nela, ou remove caso esteja. Equivalente a “inverter” um componente da parte da solução referente à configuração da mochila;
- *2-opt*: remove duas arestas da rota e adiciona outras duas completando a rota. Equivale a inverter uma subsequência da lista de permutação de cidades e corresponde a uma das vizinhanças clássicas para o PCV [20]; e
- *2h-opt*: a vizinhança *2h-opt* é uma particularidade da vizinhança *3-opt*, considerando apenas cinco arestas em vez de seis, em que o “h” no seu nome se refere a *half* (“metade” em inglês). Equivale a mover, na lista de permutação de cidades, uma cidade de uma dada posição a uma alguma outra posição [20].

O Algoritmo 4 cria uma nova configuração de mochila conforme a vizinhança *Bit Flip*. Dada uma configuração de mochila z de tamanho m e um item $1 \leq i \leq m$, retorna uma configuração de mochila com o item i “invertido”.

Os Algoritmos 5 e 6 têm como entrada uma rota x de tamanho n e posições i e j nessa rota tal que $1 < i < j \leq n$. O Algoritmo 5 produz uma nova rota aplicando a vizinhança *2-opt* que inverte a subsequência de cidades entre os índices i e j , ambos inclusos. O Algoritmo 6 produz uma nova rota aplicando a vizinhança *2h-opt* que move a cidade de

Algoritmo 4: BIT-FLIP(z, i)

```
1 se  $z_i = 1$  então
2   | flip  $\leftarrow 0$ 
3 senão
4   | flip  $\leftarrow 1$ 
5 fim
6 retorna ( $z_1, \dots, z_{i-1}, \text{flip}, z_{i+1}, \dots, z_m$ )
```

índice j para a posição i da rota e desloca todas as cidades, a partir da posição i , para frente.

Algoritmo 5: 2-OPT(x, i, j)

```
1 retorna ( $x_1, \dots, x_{i-1}, x_j, x_{j-1}, \dots, x_{i+1}, x_i, x_{j+1}, x_{j+2}, \dots, x_n$ )
```

Algoritmo 6: 2H-OPT(x, i, j)

```
1 retorna ( $x_1, \dots, x_{i-1}, x_j, x_i, x_{i+1}, \dots, x_{j-1}, x_{j+1}, x_{j+2}, \dots, x_n$ )
```

5.2.2 Lista de Candidatos

Dadas uma rota x e duas posições nessa rota, i e $j > i$, as vizinhanças *2-opt* e *2h-opt*, detalhadas na Seção 5.2.1, possuem como características em comum o fato de ambas gerarem uma nova rota em que as cidades nas posições de 1 até $i - 1$ permanecem no mesmo lugar e a cidade na posição j é posicionada em i . Devido a essa característica, é possível aplicar um filtro eliminando valores de posições j com baixa probabilidade de melhorar a função objetivo para determinados x e i .

Conforme detalhado nos Capítulos 3 e 4, há diversas estratégias para a redução do tamanho da vizinhanças do subproblema do Caixeiro Viajante no intuito de evitar o cálculo da função objetivo de um vizinho com baixo potencial de melhoria. No PLV, é comum a utilização da Triangulação de Delaunay, explicada na Seção 3.2, para criar uma lista de candidatos para a vizinhança das rotas [11, 19, 25, 26].

Este trabalho propõe uma *Lista de Candidatos de Adjacência* para cada cidade da instância como uma extensão do conceito de adjacência da Triangulação de Delaunay, adaptado do Conjunto de Candidatos de Delaunay [13].

Foi utilizado o Algoritmo TRIANGULAÇÃO-DELAUNAY que recebe como argumentos um conjunto de cidades $C = \{1, \dots, n\}$ de tamanho n e uma matriz de distâncias D entre essas cidades. Retorna uma lista $T = (t_1, \dots, t_n)$ em que cada elemento $t_i \subseteq C$, $1 \leq i \leq n$, é um subconjunto de cidades que compartilham triangulação com a cidade i .

As Listas de Candidatos de Adjacência $T^* = (t_1^*, \dots, t_n^*)$ estendem a Triangulação de Delaunay $T = (t_1, \dots, t_n)$, de tal forma que cada elemento t_i^* é um subconjunto de cidades em C em que $j \in t_i^*$ se $j \in t_i$ ou se $j \in t_k, \forall k \in t_i$. Na prática, isso equivale a estender a adjacência definida pela Triangulação de Delaunay às cidades “adjacentes às adjacentes”.

O Algoritmo 7 recebe como entrada a Triangulação de Delaunay $T = (t_1, \dots, t_n)$ de um conjunto de cidade C de tamanho n e retorna uma lista T^* em que cada elemento $t_i^* \subseteq C, 1 \leq i \leq n$ é uma Lista de Candidatos de Adjacência a i conforme definido nessa seção.

Algoritmo 7: LISTA-CANDIDATOS-ADJACÊNCIA(T)

```

1 para  $i = 1, \dots, n$  faça
2    $t_i^* \leftarrow t_i$ 
3   para cada  $j \in t_i$  faça
4      $t_i^* = t_i^* \cup t_j$ 
5   fim
6 fim
7  $T^* \leftarrow (t_1^*, \dots, t_n^*)$ 
8 retorna  $T^*$ 

```

Nesse algoritmo, para cada cidade i , na linha 2, é inicializado o seu Conjunto de Candidatos de Adjacência como o resultado da Triangulação de Delaunay para essa cidade i . Nas linhas 3-5, é estendido tal Conjunto de Candidatos de Adjacência com o resultado da Triangulação de Delaunay para as cidades adjacentes a i . Por fim, os Conjuntos de Candidatos de Adjacência de todas as cidades são retornados na linha 8.

5.2.3 Definição de Relação de Subvizinhanças para o PLV

Inspirado na Busca Local Guiada Rápida [4], foi introduzido o conceito de subvizinhança à busca local proposta. As vizinhanças *2-opt* e *2h-opt*, descritas na Sessão 5.2.1, são divididas em subvizinhanças identificadas pela posição $i > 1$ de uma cidade na rota. Em cada uma dessas subvizinhanças, busca-se uma cidade na posição $j > i$ para substituir a cidade de posição i que gere uma rota de maior valor da função objetivo em conjunto com uma configuração de mochila. Nessa busca são consideradas somente cidades que estejam na Lista de Candidatos de Adjacência da cidade anterior à cidade de posição i na rota, eliminando assim o cálculo da função objetivo de vizinhos com baixa probabilidade de melhoria.

O Algoritmo 8 descreve essa operação. Esse algoritmo recebe, como entrada, uma rota x de tamanho n , uma subvizinhança identificada pela posição i de uma cidade na rota e

sua vizinhança $V(\cdot, \cdot, \cdot)$ com as mesmas entradas dos Algoritmos 5 e 6, uma configuração de mochila z , as Listas de Candidatos de Adjacência $T^* = (t_1^*, \dots, t_n^*)$ e a função objetivo $f(\cdot, \cdot)$. Retorna a posição j no qual o resultado da função $V(x, i, j)$ é uma rota vizinha de maior função objetivo $f(\cdot, \cdot)$ na subvizinhança e configuração de mochila informadas.

Algoritmo 8: MELHOR-VIZINHO-SUBVIZINHANÇA($x, i, V(\cdot, \cdot, \cdot), z, T^*, f(\cdot, \cdot)$)

```

1   $j \leftarrow i$ 
2  para cada  $c \in t_{x_{i-1}}^*$  faça
3       $j' \leftarrow$  posição de  $c$  em  $x$ 
4      se  $j' > i$  então
5           $x' \leftarrow V(x, i, j')$ 
6          se  $f(x', z) > f(x, z)$  então
7               $j \leftarrow j'$ 
8               $x \leftarrow x'$ 
9          fim
10 fim
11 fim
12 retorna  $j$ 

```

Nesse algoritmo, na linha 1, é inicializado o identificador da posição do melhor vizinho com o valor de i . Como a busca considera somente cidades que estão após a posição de i na rota (garantido na linha 4), isso equivale a considerar a rota sem modificação como a melhor da subvizinhança. Na linha 2, é inicializado um laço percorrendo somente as cidades que estão na Lista de Candidatos de Adjacência da cidade na posição imediatamente anterior a i . Na linha 5, é criado um vizinho conforme a vizinhança $V(\cdot, \cdot, \cdot)$ informada. Nas linhas 6-9, é atualizada a posição identificadora do melhor vizinho encontrado caso a função objetivo do movimento associado ao novo vizinho seja maior. Finalmente, na linha 12, é retornado o identificador do melhor vizinho encontrado.

Uma subvizinhança pode estar “ativa” ou “inativa”, refletindo na busca, onde são consideradas somente as subvizinhanças ativas, ignorando as que estão inativas. Essa característica é representada por um conjunto de subvizinhanças ativas H para cada vizinhança, onde $c \in H$ se, e somente se, a subvizinhança da cidade c está ativa para busca nessa vizinhança. Uma subvizinhança pode ser “ativada” ou “reativada” ao se adicionar a cidade referente a essa subvizinhança em um conjunto de subvizinhanças ativas.

Durante o procedimento de criação de soluções iniciais, que será explicado posteriormente no Algoritmo 12, o conjunto de subvizinhanças ativas H é inicializado com todas as subvizinhanças de uma vizinhança. Também será explicado posteriormente no Algoritmo 15 a criação do conjunto de subvizinhanças ativas contendo somente as

subvizinhanças envolvidas no processo de destruição e reconstrução da rota conforme a meta-heurística Busca em Vizinhaça Grande.

Para cada subvizinhança, busca-se uma rota que maximize a função objetivo utilizando o Algoritmo 8. Se encontrada uma rota diferente da atual, são reativadas as subvizinhanças identificadas pela cidade na posição da rota que gerou a melhoria e a da posição seguinte. As subvizinhanças são consideradas pela busca na ordem em que aparecem na rota. Após a busca em um subvizinhança, independente se encontrado ou não uma melhor rota vizinha, é seguida a busca para a próxima subvizinhança.

O Algoritmo 9 detalha esse procedimento. Como entrada, recebe uma rota x de tamanho n , uma vizinhança $V(\cdot, \cdot, \cdot)$, o conjunto de subvizinhanças ativas H para consideração na busca, uma configuração de mochila z , as Listas de Candidatos de Adjacência T^* e a função objetivo $f(\cdot, \cdot)$. Retorna uma rota x onde todas as subvizinhanças ativas H possuem o melhor vizinho, juntamente com o conjunto de subvizinhanças reativadas durante a busca.

Algoritmo 9: BUSCA-SUBVIZINHANÇAS-ROTA($x, V(\cdot, \cdot, \cdot), H, z, T^*, f(\cdot, \cdot)$)

```

1  $H' \leftarrow \emptyset$ 
2 para  $i = 2, \dots, n-1$  faça
3   se  $x_i \in H$  então
4      $j \leftarrow \text{MELHOR-VIZINHO-SUBVIZINHANÇA}(x, i, V(\cdot, \cdot, \cdot), z, T^*, f(\cdot, \cdot))$ 
5     se  $j > i$  então
6        $H' \leftarrow H' \cup \{x_j, x_{j+1}\}$ 
7        $x \leftarrow V(x, i, j)$ 
8     fim
9   fim
10 fim
11 retorna  $x, H'$ 

```

O algoritmo inicia o conjunto de subvizinhanças reativadas na linha 1. O laço na linha 2 percorre as subvizinhanças na ordem em que aparecem na rota, ignorando a primeira cidade da rota que é fixa e a última que não tem candidatos para substituição. Na linha 3, é checada se a subvizinhança da cidade na posição i está ativa, caso contrário a busca não é feita para essa subvizinhança. Na linha 4, é feita a busca pelo melhor vizinho da subvizinhança, se este for maior que o atual (linha 5) são reativadas as subvizinhanças referentes à melhor cidade encontrada e à cidade posterior na linha 6. Na linha 7 é atualizada a rota com o melhor vizinho encontrada na busca na subvizinhança da rota. Por fim, na linha 11 é retornada a rota x onde todas as subvizinhanças ativas H possuem o melhor vizinho, juntamente com o conjunto de subvizinhanças H' reativadas durante a busca.

Uma busca também é feita sobre as subvizinhanças da configuração da mochila utilizando a vizinhança *Bit Flip*. De forma semelhante ao Algoritmo 9, é escolhido um conjunto de subvizinhanças “ativas” a serem aplicadas ao operador de *Bit Flip*. Independente de encontrar ou não uma melhor configuração de mochila, o próximo item na ordem de busca das subvizinhanças é considerado para busca. Também é mantido um conjunto de subvizinhanças reativadas da rota com as cidades onde se encontram os itens que sofreram modificação durante a busca.

O Algoritmo 10 detalha a busca nas subvizinhanças da mochila. Esse algoritmo recebe, como entrada, uma rota x de tamanho n , uma configuração de mochila z de tamanho m , a lista de conjuntos de disponibilidade dos itens por cidade $A = (a_1, \dots, a_n)$, um conjunto de itens ativos para busca Q , o peso dos itens $w = (w_1, \dots, w_m)$, e a função objetivo $f(\cdot, \cdot)$. Retorna a configuração de mochila z onde todas as subvizinhanças ativas Q possuem o melhor vizinho, juntamente com o conjunto de subvizinhanças da rota reativadas durante a busca.

Algoritmo 10: BUSCA-SUBVIZINHANÇAS-MOCHILA($x, z, A, Q, f(\cdot, \cdot)$)

```

1   $H' \leftarrow \emptyset$ 
2  para  $i = 1, \dots, n$  faça
3      para cada  $j \in a_{x_i}$  faça
4          se  $j \in Q$  então
5               $z' \leftarrow \text{Bit-Flip}(z, j)$ 
6               $w' \leftarrow 0$ 
7              para  $k = 1, \dots, m$  faça
8                  se  $z_k = 1$  então
9                       $w' \leftarrow w' + w_k$ 
10                 fim
11             fim
12             se  $w' \leq W \wedge f(x, z') > f(x, z)$  então
13                  $H' \leftarrow H' \cup \{x_i\}$ 
14                  $z \leftarrow z'$ 
15             fim
16         fim
17     fim
18 fim
19 retorna  $z, H'$ 

```

O algoritmo inicia o conjunto de subvizinhanças reativadas na linha 1. O laço na linha 2 percorre todas as cidades na ordem que aparecem na rota. Na linha 3, para cada cidade i , o laço percorre os itens dessa cidade i . Na linha 4, é checado se a subvizinhança referente ao item está ativa, se for o caso é criada a configuração de mochila vizinha na

linha 5. Nas linhas 6-10 é calculado o peso da configuração da mochila vizinha z' . Caso o peso da configuração da mochila vizinha seja menor ou igual que a capacidade da mochila e o valor da função objetivo da configuração de mochila vizinha seja maior que a atual, a cidade onde esse item se encontra é adicionada à lista de subvizinhanças reativadas da rota na linha 13 e essa configuração da mochila é atualizada na linha 14. Por fim, na linha 19 é retornada a configuração de mochila z onde todas as subvizinhanças ativas Q possuem o melhor vizinho, juntamente com o conjunto de subvizinhanças da rota reativadas H' durante a busca.

5.2.4 Busca Local Rápida

O Algoritmo 11 descreve uma Busca Local Rápida utilizando as buscas em subvizinhanças na rota e na configuração da mochila, definidas na Seção 5.2.3, conforme descrito a seguir. Como entrada, recebe uma rota x de tamanho n , uma configuração de mochila z de tamanho m , o conjunto de itens da instância B , a lista de conjuntos de disponibilidade dos itens por cidade $A = (a_1, \dots, a_n)$, o conjunto de subvizinhanças ativas H para busca na rota, as Listas de Candidatos de Adjacência T^* e a função objetivo $f(\cdot, \cdot)$. Retorna um mínimo local composto pela rota x e configuração z para as subvizinhanças ativas H .

Algoritmo 11: BUSCA-LOCAL-RÁPIDA($x, z, B, A, H, T^*, w, f(\cdot, \cdot)$)

```

1   $H^0 \leftarrow H$ 
2   $H^h \leftarrow H$ 
3  faça
4       $Q^+ \leftarrow \{i \mid i \in B, z_i = 1\}$ 
5       $Q^- \leftarrow \{i \mid i \in B, z_i = 0\}$ 
6       $z, H' \leftarrow \text{BUSCA-SUBVIZINHANÇAS-MOCHILA}(x, z, A, Q^+, f(\cdot, \cdot))$ 
7       $z, H'' \leftarrow \text{BUSCA-SUBVIZINHANÇAS-MOCHILA}(x, z, A, Q^-, f(\cdot, \cdot))$ 
8       $H^0 \leftarrow H^0 \cup H' \cup H''$ 
9       $H^h \leftarrow H^h \cup H' \cup H''$ 
10      $x, H^{h'} \leftarrow \text{BUSCA-SUBVIZINHANÇAS-ROTA}(x, 2\text{-OPT}, H^0, z, T^*, f(\cdot, \cdot))$ 
11      $H^0 \leftarrow \emptyset$ 
12      $H^h \leftarrow H^h \cup H^{h'}$ 
13      $x, H^{o'} \leftarrow \text{BUSCA-SUBVIZINHANÇAS-ROTA}(x, 2H\text{-OPT}, H^h, z, T^*, f(\cdot, \cdot))$ 
14      $H^h \leftarrow \emptyset$ 
15      $H^0 \leftarrow H^0 \cup H^{o'}$ 
16 enquanto  $(H' \cup H'' \cup H^{o'} \cup H^{h'}) \neq \emptyset$ 
17 retorna  $x, z$ 

```

Nas linhas 1 e 2 são inicializados os conjuntos de subvizinhanças ativas, das vizi-

nhanças 2-OPT e 2H-OPT, respectivamente, com o conjunto H informado na entrada do algoritmo. Nas linhas 4 e 5, os itens da configuração da mochila são divididos em dois conjuntos de subvizinhanças ativas. Um conjunto contém somente os itens inclusos na configuração corrente z da mochila e outro conjunto contém os itens não inclusos nessa configuração z . Nas linhas 6 e 7, é feita a busca nas subvizinhanças ativas da configuração da mochila, o que também gera conjuntos de reativação das subvizinhanças da rota H' e H'' para as cidades em que se encontram os itens modificados. É importante notar que a ordem das buscas nas subvizinhanças da mochila foi deliberadamente escolhida para primeiro remover os itens da configuração da mochila e depois considerar a adição. Isso permite que primeiro seja “liberado” o máximo de capacidade da mochila, dando chance a itens pesados de serem eventualmente adicionados. Nas linhas 8 e 9 são adicionadas aos conjuntos de subvizinhanças ativas das vizinhanças 2-OPT e 2H-OPT as cidades reativadas nas buscas nas subvizinhanças da mochila nas linhas 6 e 7. Na linha 10 é feita a busca nas subvizinhanças da vizinhança 2-OPT da rota, considerando ativas somente as subvizinhanças do conjunto H^0 . Após a busca, são removidas todas as cidades do conjunto H^0 na linha 11. As cidades reativadas retornadas na linha 10 são adicionadas ao conjunto de subvizinhanças ativas H^h referente à vizinhança 2H-OPT na linha 12. Esse procedimento é repetido nas linhas 13-15, trocando as vizinhanças 2-OPT e 2H-OPT uma com a outra. Diferente do proposto por Alsheddy et al.[4], todas as subvizinhanças são feitas inativas após uma busca nas linhas 11 e 14, mesmo se a busca em alguma subvizinhança for bem sucedida. Isso é possível pois nas linhas 12 e 15 essas subvizinhanças são reativas com os resultados das buscas nas outras vizinhanças. Quando uma das buscas nas subvizinhanças nas linhas 6, 7, 10 e 13 encontra um vizinho de melhor valor da função objetivo $f(\cdot, \cdot)$, é retornado, junto com esse vizinho, um conjunto não vazio de subvizinhanças reativadas para busca na rota. Sendo assim, o laço termina na linha 16 quando todos esses conjuntos de subvizinhanças reativadas estiverem vazios, indicando que não houve melhoria na iteração.

5.3 Conjunto de Soluções Iniciais

No intuito de encontrar uma boa solução inicial antes de aplicar a Busca em Vizinhança Grande, a heurística proposta gera um conjunto de soluções iniciais para uma instância a partir de diferentes rotas iniciais e configuração gulosa da mochila, posteriormente aplicando a Busca Local Rápida com todas as vizinhanças ativas.

O Algoritmo 12 explica esse procedimento. Como entrada, recebe o conjunto de cidades C de tamanho n , a matriz de distâncias D , o conjunto de itens B de tamanho m , a fun-

ção objetivo $f(\cdot, \cdot)$, o valor dos itens $p = (p_1, \dots, p_m)$, o peso dos itens $w = (w_1, \dots, w_m)$, a lista de conjuntos de disponibilidade dos itens $A = (a_1, \dots, a_n)$, a capacidade W da mochila, o parâmetro referente ao conjunto Δ de expoentes a serem testados no algoritmo guloso de configuração da mochila e o parâmetro r referente à semente do gerador de números pseudo-aleatórios.

Esse algoritmo recebe também um parâmetro $\mu > 1$ que controla a quantidade de soluções a serem geradas. São geradas soluções até que sejam geradas μ soluções seguidas que não possuam valor da função objetivo maior que o melhor encontrado até então. Como retorno, produz um conjunto S de soluções para o PLV.

Algoritmo 12: SOLUÇÕES-INICIAIS($C, D, B, f(\cdot, \cdot), p, w, A, W, T^*, \Delta, \mu, r$)

```

1   $S \leftarrow \emptyset$ 
2   $f^* \leftarrow -\infty$ 
3   $\mu' \leftarrow 0$ 
4  enquanto  $\mu' < \mu$  faça
5       $x \leftarrow \text{CHAINED-LIN-KERNIGHAN}(C, D, r)$ 
6       $x, z \leftarrow \text{SOLUÇÃO-GULOSA-PLV}(x, D, B, f(\cdot, \cdot), p, w, A, W, \Delta)$ 
7       $H \leftarrow C \setminus \{1\}$ 
8       $x, z \leftarrow \text{BUSCA-LOCAL-RÁPIDA}(x, z, B, A, H, T^*, w, f(\cdot, \cdot))$ 
9       $S \leftarrow S \cup \{x, z\}$ 
10     se  $f(x, z) > f^*$  então
11          $f^* \leftarrow f(x, z)$ 
12          $\mu' \leftarrow 0$ 
13     senão
14          $\mu' \leftarrow \mu' + 1$ 
15     fim
16 fim
17 retorna  $S$ 

```

Nas linhas 1-3 são inicializados o retorno do algoritmo, o maior valor da função objetivo encontrada até então e um contador de soluções geradas sem melhoria no valor da função objetivo. Na linha 5 é criada uma rota para a instância utilizando o Algoritmo CHAINED-LIN-KERNIGHAN, detalhado na Seção 3.1, considerando somente o componente do PLV referente ao Problema do Caixeiro Viajante. Na linha 6 é criada uma configuração de mochila utilizando a heurística gulosa da Seção 5.1.2. Com essa solução gulosa, é feita a Busca Local Rápida na linha 8 considerando todas as subvizinhanças ativas, com exceção da primeira cidade da instância que é fixada na rota da solução. Na linha 9 é adicionada a solução encontrada ao conjunto S de retorno do algoritmo. Caso essa solução possua um valor da função objetivo maior que o melhor valor encontrado até então, o que é checado

na linha 10, o valor da melhor função objetivo encontrada é atualizado na linha 11 e o contador de soluções geradas sem melhorias é zerado na linha 12, caso contrário o contador de soluções geradas sem melhoria é incrementado na linha 14. Por fim, na linha 17, é retornado o conjunto de soluções geradas.

Como o melhor valor da função objetivo é inicializado com $-\infty$ na linha 2 e a função $f(\cdot, \cdot)$ sempre retorna um valor finito, a linha 12 é executada pelo menos uma vez, o que garante que o algoritmo produza um conjunto possuindo, no mínimo, $\mu + 1$ soluções.

O Algoritmo CHAINED-LIN-KERNIGHAN invocado na linha 5 possui um componente aleatório, ou seja, diferentes chamadas a esse algoritmo podem gerar diferentes rotas para um mesmo conjunto de entradas. Sendo assim, as soluções geradas por cada iteração do laço podem ser diferentes, o que pode adicionar soluções diferentes ao conjunto de soluções iniciais retornado pelo algoritmo na linha 9.

5.4 Busca em Vizinhança Grande para o PLV

Para escapar do mínimo local, é proposta uma heurística baseada na meta-heurística Busca em Vizinhança Grande (BVG) [15] em que são feitas sucessivas destruições e reconstruções da rota em pontos definidos a partir de uma cidade escolhida aleatoriamente e aplicada novamente a Busca Local Rápida considerando ativas somente as subvizinhanças das cidades envolvidas nesses pontos de destruição e reconstrução.

5.4.1 Procedimento de Destruição

Definindo-se uma sub-rota de x como uma permutação de um subconjunto de x , a heurística proposta utiliza a Triangulação de Delaunay, descrita na Seção 3.2 para escolher pontos de destruição de uma rota, gerando sub-rotas que, posteriormente, serão unificadas em uma nova rota.

Dado um conjunto de cidades C de uma instância, uma cidade $c \in C$ em uma rota x e suas cidades adjacentes t_c segundo a Triangulação de Delaunay, o método de destruição proposto cria um conjunto de sub-rotas Γ onde cada cidade em $t_c \cup \{1\}$ possui exatamente uma sub-rota $\gamma \in \Gamma$ começando com essa cidade e não há sub-rotas em Γ que não comecem com uma cidade em $t_c \cup \{1\}$.

Adicionalmente, as sub-rotas do conjunto Γ definido dessa forma não possuem cidades em comum, com exceção da primeira cidade da instância que deve aparecer no começo de

exatamente uma sub-rota e no final de exatamente outra. Também é necessário que a união de todas sub-rotas em Γ seja igual a C , em outras palavras, que não haja cidades em C que não estejam em uma sub-rota de Γ .

A cidade final de cada sub-rota em Γ é escolhida como sendo a cidade anterior a uma cidade em t_c na rota x . Dessa forma, as cidades iniciais e finais de cada sub-rota se encontram na Lista de Candidatos de Adjacência de c conforme definido na Seção 5.2.2. Isso equivale a dizer que quaisquer cidades iniciais e finais de duas sub-rotas em Γ são boas candidatas a serem adjacentes em uma rota da solução, com exceção da primeira cidade da instância.

O Algoritmo 13 descreve a destruição de uma rota. Dada uma rota x de tamanho n , a Triangulação de Delaunay $T = (t_1, \dots, t_n)$ da instância e o parâmetro r referente à semente do gerador de números pseudo-aleatórios, retorna um conjunto Γ de sub-rotas de x conforme definido nessa seção.

Algoritmo 13: DESTRUIÇÃO-ROTA(x, T, r)

```

1  $c \leftarrow$  escolha uma cidade aleatória em  $x$ 
2  $\Gamma \leftarrow \emptyset$ 
3  $i \leftarrow 1$ 
4 para  $j = 2, \dots, n$  faça
5   se  $x_j \in t_c$  então
6      $\Gamma \leftarrow \Gamma \cup \{(x_i, x_{i+1}, \dots, x_{j-2}, x_{j-1})\}$ 
7      $i \leftarrow j$ 
8   fim
9 fim
10  $\Gamma \leftarrow \Gamma \cup \{(x_i, x_{i+1}, \dots, x_n, 1)\}$ 
11 retorna  $\Gamma$ 

```

Na linha 1 é escolhida uma cidade aleatória na rota cujo conjunto de adjacência da Triangulação de Delaunay guiará o procedimento de destruição. Na linha 3, o conjunto de sub-rotas Γ é inicialmente feito vazio. Na linha 3, é inicializada uma variável indicando o índice de início de uma sub-rota como a primeira cidade da rota. O laço na linha 4 percorre a rota a partir da segunda cidade até o final, o que garante que a primeira cidade da rota não esteja no final de uma sub-rota criada. Na linha 5 é checado se uma dessas cidades está no conjunto de adjacência da Triangulação de Delaunay da cidade escolhida na linha 3, se positivo na linha 6 é adiciona uma sub-rota ao conjunto de sub-rotas Γ , partindo da cidade na posição i até a cidade anterior à cidade da posição j . Na linha 7, é atualizado o início de uma nova sub-rota. Na linha 10, é adicionada uma última sub-rota ao conjunto Γ , composta das cidades restantes na rota x , finalizando com a cidade na posição 1, o que marca o

retorno do caixeiro à cidade inicial. Finalmente, na linha 11, o conjunto de sub-rotas Γ é retornado.

É importante notar que, na primeira execução da linha 6 no laço, o valor de i é igual a 1 e o valor de x_1 é igual a 1, conforme definição da solução na Seção 2.3. Isso, associado à sub-rotas adicionada ao conjunto Γ na linha 10, garante que o conjunto Γ possui exatamente uma sub-rotas começando na primeira cidade e exatamente outra sub-rotas terminando na primeira cidade.

5.4.2 Procedimento de Reconstrução

Como a destruição da rota gera sub-rotas cujas cidades iniciais e finais pertencem à Lista de Candidatos de Adjacência umas das outras (com exceção das rotas que começam ou terminam com a cidade inicial), o que é equivalente a dizer que essas cidades são boas candidatas a serem adjacentes em uma solução, foi adotado um critério aleatório de reconstrução da rota.

Uma rota parcial $x^p = (x_1^p, \dots, x_{|x^p|}^p)$ de um conjunto de cidades C é definida como uma permutação de um subconjunto de C onde $x_1^p = 1$. Uma rota parcial pode ser estendida por uma sub-rotas, onde os elementos dessa sub-rotas que não existem na rota parcial são adicionados ao final da rota parcial na ordem em que aparecem na sub-rotas. Opcionalmente, é possível indicar o final de uma rota parcial com a cidade 1 em qualquer posição diferente da primeira. Nesse caso não pode haver cidades após essa posição e a rota parcial não pode ser estendida. Caso a rota seja finalizada dessa forma, o conjunto formado pelas cidades da rota parcial deve ser igual a C , formando uma rota completa caso a última cidade seja removida.

Sendo assim, dada uma rota parcial x^p , a reconstrução de uma rota envolve escolher aleatoriamente uma sub-rotas de $\gamma \in \Gamma$ que não contenha a cidade inicial para estender essa rota parcial x^p . Essa extensão pode ser feita com a sub-rotas como existe ou invertida, escolhendo-se aleatoriamente. Para garantir uma rota parcial válida, essa rota parcial é iniciada com a sub-rotas que inicia com a primeira cidade da instância e finalizada com a sub-rotas que termina com a primeira cidade da instância. Por fim, uma rota válida é trivialmente criada a partir dessa rota parcial através da exclusão da última cidade que é sempre a primeira da instância e é repetida no início e final de uma sub-rotas finalizada.

A reconstrução da rota é descrita no Algoritmo 14. Recebe como entrada o conjunto Γ de sub-rotas criado pelo Algoritmo 13 e o parâmetro r referente à semente do gerador de números pseudo-aleatórios. Como resultado, produz uma rota x reconstruída a partir das

sub-rotas em Γ .

Algoritmo 14: RECONSTRUÇÃO-ROTA(Γ, r)

```

1  $\gamma^- \leftarrow$  remove de  $\Gamma$  a sub-rota que começa com a cidade 1
2  $\gamma^+ \leftarrow$  remove de  $\Gamma$  a sub-rota que termina na cidade 1
3  $x^p \leftarrow (\gamma_1^-, \dots, \gamma_{|\gamma^-|}^-)$ 
4  $\Gamma' \leftarrow \Gamma$  em ordem aleatória
5 para cada  $\gamma \in \Gamma'$  faça
6   se uma chance aleatória de 50% então
7      $\gamma \leftarrow (\gamma_{|\gamma|}, \gamma_{|\gamma|-1}, \dots, \gamma_1)$ 
8   fim
9    $x^p \leftarrow (x_1^p, \dots, x_{|x^p|}^p, \gamma_1, \dots, \gamma_{|\gamma|})$ 
10 fim
11  $x^p \leftarrow (x_1^p, \dots, x_{|x^p|}^p, \gamma_1^+, \dots, \gamma_{|\gamma^+|}^+)$ 
12 se uma chance aleatória de 50% então
13    $x \leftarrow (1, x_2^p, x_3^p, \dots, x_{|x^p|-1}^p)$ 
14 senão
15    $x \leftarrow (1, x_{|x^p|-1}^p, x_{|x^p|-2}^p, \dots, x_2^p)$ 
16 fim
17 retorna  $x$ 

```

Esse algoritmo trata especialmente as sub-rotas que contêm a primeira cidade da instância para garantir que a rota parcial resultante seja válida, isto é, que a rota parcial comece e termine na primeira cidade da instância. Para isso, nas linhas 1 e 2, são removidas do conjunto Γ as sub-rotas que começam e terminam com a primeira cidade da instância. Na linha 3, é inicializada uma rota parcial x^p com a sub-rota que inicia com a primeira cidade da instância. Na linha 5, é feito um laço sobre as sub-rotas seguindo a ordem aleatória definida na linha 4. Cada sub-rota é então invertida com uma chance de 50% nas linhas 6-8. Na linha 9, é estendida a rota parcial com essa sub-rota. Na linha 11, é estendida a rota parcial com a sub-rota que termina na primeira cidade da instância. Por fim, nas linhas 12-16, é criada uma rota a partir da rota parcial com 50% de chance de inversão da ordem.

5.4.3 Reativação de Subvizinhanças

Com o intuito de agilizar e guiar a busca local para diferentes soluções, foi optado por reativar somente as subvizinhanças das cidades iniciais e finais das sub-rotas geradas na destruição. Essa reativação seletiva permite guiar a Busca Local Rápida para novas soluções uma vez que para cada execução da Busca Local Rápida são consideradas

diferentes subvizinhanças na busca. O Algoritmo 15 detalha esse procedimento. Como entrada recebe o conjunto de sub-rotas criadas pelo Algoritmo 13 e retorna um conjunto H de cidades ativas para busca nas vizinhanças da rota, excluindo a primeira cidade da instância que deve sempre permanecer na primeira posição da rota conforme definição da solução na Seção 2.3.

Algoritmo 15: SUBVIZINHANÇAS-REATIVADAS(Γ)

```

1  $H \leftarrow \bigcup_{\gamma \in \Gamma} \{\gamma_1, \gamma_{|\gamma|}\}$ 
2  $H \leftarrow H \setminus \{1\}$ 
3 retorna  $H$ 

```

5.4.4 Rota em Vizinhança Grande

Na intenção de controlar o grau de diversificação da heurística proposta, foi optado por fazer sucessivas aplicações dos algoritmos de destruição e reconstrução em uma rota. O Algoritmo 16 detalha esse procedimento. Recebe como entrada uma rota x de tamanho n , o conjunto C de cidades da instância, a Triangulação de Delaunay T da instância, a quantidade λ' de vezes a ser aplicada a destruição e reconstrução à rota e o parâmetro r referente à semente do gerador de números pseudo-aleatórios. Como retorno produz uma rota x destruída e reconstruída λ' vezes e o conjunto H de cidades ativas para busca nas vizinhanças da rota.

Algoritmo 16: ROTA-EM-VIZINHANÇA-GRANDE(x, C, T, λ', r)

```

1  $H \leftarrow \emptyset$ 
2 repita  $\lambda'$  vezes
3    $\Gamma \leftarrow \text{DESTRUIÇÃO-ROTA}(x, T, r)$ 
4    $x \leftarrow \text{RECONSTRUÇÃO-ROTA}(\Gamma, r)$ 
5    $H' \leftarrow \text{SUBVIZINHANÇAS-REATIVADAS}(\Gamma)$ 
6    $H \leftarrow H \cup H'$ 
7 fim
8 retorna  $x, H$ 

```

Nesse algoritmo, é inicializado o conjunto de cidades ativas para busca local como um conjunto vazio na linha 1. O laço nas linhas 2-6 repete a destruição e reconstrução da rota λ' vezes. Nas linhas 3 e 4, é realizado o processo de destruição e reconstrução da rota, na linha 6 são adicionadas ao conjunto de cidades ativas para busca local as cidades referentes às subvizinhanças reativadas. Por fim, na linha 8, é retornada a rota x destruída e reconstruída λ' vezes junto com o conjunto de cidades ativas H para busca local nas subvizinhanças da rota x .

5.5 Busca Local Rápida em Vizinhança Grande para o PLV

Finalmente, é possível definir a Busca Local Rápida em Vizinhança Grande aplicada ao Problema do Ladrão Viajante (BLRVG-PLV) utilizando os conceitos apresentados anteriormente neste capítulo.

A BLRVG-PLV gera um conjunto inicial de soluções a partir de uma rota gerada pelo algoritmo CHAINED-LIN-KERNIGHAN para o subproblema do Caixeiro Viajante, uma configuração gulosa de mochila para essa rota, seguido de uma Busca Local Rápida considerando todas as subvizinhanças ativas. Essas soluções são ordenadas priorizando as soluções de melhor valor da função objetivo $f(\cdot, \cdot)$. A cada uma dessas soluções, em ordem, são aplicadas sucessivas destruições e reconstruções da rota, seguido de uma escolha gulosa da mochila e uma Busca Local Rápida considerando ativas somente as subvizinhanças reativadas durante a destruição e reconstrução da rota.

Esse procedimento é repetido até que seja executado um número de iterações sem melhorias para a solução corrente controlado por um parâmetro ψ . Após atingir esse número de iterações sem melhorias, a próxima solução do conjunto de soluções iniciais é considerada pelo algoritmo.

A proposta original da Busca em Vizinhança Grande aumenta a quantidade de cidades removidas e reinseridas da rota conforme o número de iterações sem melhoria da heurística [15]. Devido à reconstrução proposta ser aleatória e sua qualidade depender das sub-rotas iniciarem e terminarem em cidades próximas, somente é possível alcançar sub-rotas com essa característica realizando a reconstrução utilizando sub-rotas destruídas a partir de uma única cidade. Sendo assim, foi optado por realizar uma quantidade de destruições e reconstruções proporcional ao número de iterações sem melhorias da solução corrente, ao invés de modificar a quantidade de cidades envolvidas na destruição e reconstrução. O valor mínimo dessa quantidade é 1 e o máximo é um parâmetro λ da heurística. Esse procedimento de aumentar a quantidade de destruições e reconstruções conforme a heurística possui iterações sem melhorias equivale a fazer pequenas modificações em novas soluções encontradas e gradualmente aumentar o número de modificações conforme a heurística prossegue sem encontrar uma melhoria, no intuito de promover a diversificação das soluções exploradas.

O Algoritmo 17 descreve esse procedimento. Recebe como entrada o conjunto de cidades C de tamanho n , a matriz de distâncias D , o conjunto de itens B de tamanho m , a função objetivo $f(\cdot, \cdot)$, o valor dos itens $p = (p_1, \dots, p_m)$, o peso dos itens $w = (w_1, \dots, w_m)$, a lista de conjuntos de disponibilidade dos itens $A = (a_1, \dots, a_n)$ e a

capacidade W da mochila. Recebe também os parâmetros Δ de expoentes a serem testados no algoritmo guloso de configuração da mochila, a quantidade de soluções iniciais μ geradas sem melhoria, a quantidade de iterações ψ sem melhoria da solução corrente, o número máximo λ de destruições e reconstruções da solução corrente e a semente r do gerador de números pseudo-aleatórios.

Algoritmo 17: BLRVG-PLV($C, D, B, f(\cdot, \cdot), p, w, A, W, \Delta, \mu, \psi, \lambda, r$)

```

1   $T \leftarrow \text{TRIANGULAÇÃO-DELAUNAY}(C)$ 
2   $T^* \leftarrow \text{LISTA-CANDIDATOS-ADJACÊNCIA}(T)$ 
3   $x \leftarrow (1, \dots, n)$ 
4   $z \leftarrow (0, \dots, 0)$ 
5  enquanto o critério de parada não for atingido faça
6       $S \leftarrow \text{SOLUÇÕES-INICIAIS}(C, D, B, f(\cdot, \cdot), p, w, A, W, T^*, \Delta, \mu, r)$ 
7       $S' = (s'_1, \dots, s'_{|S'|}) \leftarrow S$  em ordem não crescente do valor de  $f(\cdot, \cdot)$ 
8      para  $i = 1, \dots, |S'|$  faça
9           $\{x', z'\} \leftarrow s'_i$ 
10          $\psi' \leftarrow 0$ 
11         enquanto  $\psi' < \psi$  faça
12              $\psi' \leftarrow \psi' + 1$ 
13              $\lambda' \leftarrow \left\lceil \lambda \frac{\psi'}{\psi} \right\rceil$ 
14              $x'', H \leftarrow \text{ROTA-EM-VIZINHANÇA-GRANDE}(x', C, T, \lambda', r)$ 
15              $x'', z'' \leftarrow \text{SOLUÇÃO-GULOSA-PLV}(x'', D, B, f(\cdot, \cdot), p, w, A, W, \Delta)$ 
16              $x'', z'' \leftarrow \text{BUSCA-LOCAL-RÁPIDA}(x'', z'', B, A, H, T^*, w, f(\cdot, \cdot))$ 
17             se  $f(x'', z'') > f(x', z')$  então
18                  $x' \leftarrow x''$ 
19                  $z' \leftarrow z''$ 
20                  $\psi' \leftarrow 0$ 
21             fim
22         fim
23         se  $f(x', z') > f(x, z)$  então
24              $x \leftarrow x'$ 
25              $z \leftarrow z'$ 
26         fim
27     fim
28 retorna  $x, z$ 

```

Nas linhas 1 e 2, são obtidas a *Triangulação de Delaunay* e os *Conjuntos de Candidatos de Adjacência*, respectivamente. Nas linhas 3 e 4, são inicializados os componentes de uma solução com uma rota onde todas as cidades aparecem em sequência (permutação

identidade) e uma configuração de mochila vazia. O algoritmo é executado até que um critério de parada seja atingido. Primeiramente, são geradas soluções iniciais e ordenadas em ordem não decrescente da função objetivo $f(\cdot, \cdot)$ nas linhas 6 e 7. Em seguida, itera-se sobre essas soluções iniciais nas linhas 8-26. A solução corrente é separada em seus componentes rota e configuração da mochila na linha 9.

Na linha 10, o contador de iterações sem melhorias ψ' é inicializado. Enquanto esse contador ψ' for menor que o parâmetro ψ do algoritmo, o contador ψ' é incrementado na linha 12 e são repetidas a destruição e reconstrução da rota na linha 14, a busca gulosa da configuração da mochila na linha 15 e a Busca Local Rápida na linha 16 considerando somente as subvizinhanças reativadas na linha 14. Caso a nova solução gerada seja melhor que a corrente, os componentes da solução corrente são atualizados nas linhas 18 e 19, e o contador de iterações sem melhorias ψ' é zerado na linha 20.

Após o algoritmo executar ψ iterações sem melhoria, é atualizada a melhor solução até o momento nas linhas 23 a 25. O laço mais externo termina quando um critério de parada é atingido e a melhor solução encontrada é retornada na linha 29.

6. Experimentos Computacionais

Neste capítulo descrevem-se os experimentos computacionais realizados aplicando-se a heurística Busca Local Rápida em Vizinhaça Grande aplicada ao Problema do Ladrão Viajante (BLRVG-PLV) em um conjunto de instâncias de teste disponíveis na literatura. Os resultados obtidos pela heurística BLRVG-PLV são comparados com os trabalhos mais relevantes da literatura.

6.1 Instâncias do Problema PLV

O principal conjunto de instâncias utilizado pela literatura do PLV, doravante denominadas “instâncias do PLV”, foi proposto por Polyakovskiy et al.[1], utilizando como base 81 das instâncias do conjunto de instâncias TSPLIB [13] para o PCV que utilizam a distância euclidiana como métrica de distância, conforme definido na Seção 2.1.

As instâncias escolhidas do TSPLIB variam de 51 a 85.900 cidades. Para cada uma das instâncias do TSPLIB, os autores geraram os seguintes três tipos de instâncias para o Problema da Mochila (PM) utilizando as recomendações de Martello, Pisinger e Toth[27]:

- Não correlacionados, em inglês *uncorrelated* (UNC): os valores e pesos dos itens são escolhidos aleatoriamente de forma independente no intervalo $[1, 10^3]$, o que gera valores e pesos não correlacionados. Para as instâncias do PM com pouca ou nenhuma correlação entre os valores e pesos são mais fáceis de se encontrarem boas soluções.
- Não correlacionados com pesos semelhantes, em inglês *uncorrelated with similar weights* (USW): os pesos dos itens são escolhidos aleatoriamente no restrito intervalo $[10^3, 10^3 + 10]$, gerando pesos semelhantes, enquanto os valores dos itens são escolhidos aleatoriamente no intervalo maior $[1, 10^3]$, gerando pouca correlação entre os valores e pesos. Para instâncias do PM com itens de peso semelhantes são

mais difíceis de se encontrar boas soluções.

- Itens limitados com valores e pesos fortemente correlacionados, em inglês *Bounded Strongly Correlated* (BSC): os pesos dos itens são escolhidos aleatoriamente no intervalo $[1, 10^3]$, já o valor do item é definido como o seu peso adicionado a 100. Cada um desses itens pode ser adicionado a uma solução mais de uma vez dado um certo limite. Isso é equivalente a um Problema Binário da Mochila no qual diferentes itens podem ter o mesmo valor e peso. Esse é o tipo de instância do PM para o qual é mais difícil de se encontrarem boas soluções.

Para cada combinação de instância do PCV e tipo de PM, são escolhidas quatro quantidades de itens por cidade correspondentes a $\{1, 3, 5, 10\}$. Cada cidade da instância gerada dessa forma possui exatamente a mesma quantidade de itens, exceto a primeira cidade que nunca contém itens. Além disso, para cada combinação de instância do PCV, tipo de PM e quantidade de itens por cidade, são escolhidas 10 capacidades de mochila de valores $1/11, 2/11, \dots, 10/11$ do peso total dos itens da instância.

Considerando as 81 instâncias do PCV contidas no TSPLIB, os três tipos de PM, as quatro quantidades de itens por cidade e as 10 capacidades da mochila, o total de instâncias do PLV propostas por Polyakovskiy et al.[1] é igual a $81 \times 3 \times 4 \times 10 = 9.720$.

Todas as instâncias do PLV possuem a velocidade máxima $v^+ = 1,0$ e a velocidade mínima $v^- = 0,1$. O valor R do aluguel por unidade de tempo de uma instância do PLV é definido como o valor total dos itens adicionados à mochila pela solução ótima do componente referente ao PM da instância, dividido pelo tempo total para percorrer uma rota quase ótima obtida pela heurística Chained Lin-Kernighan [12], referente ao subproblema do Caixeiro Viajante da instância.

Esta dissertação nomeia as instâncias do PLV utilizando suas quatro características separadas pelo caractere *underscore*. O primeiro valor indica qual instância do TSPLIB gerou a instância do PLV. A instância do TSPLIB possui um conjunto de letras identificadoras seguido do número de cidades. O segundo valor é o número de itens por cidade. O terceiro valor é o tipo do Problema da Mochila, “bsc”, “usw” ou “unc”. O último valor é o numerador $1, 2, \dots, 10$ da fração que representa a capacidade da mochila. Como exemplo a instância do PLV “eil51_3_unc_05” foi gerada a partir da instância “eil51”, contém 51 cidades, 3 itens por cidade com exceção da primeira, totalizando $(51 - 1) \times 3 = 150$ itens, possui uma mochila com itens do tipo *Uncorrelated* (*unc*) e a capacidade da mochila é igual a $5/11$ do peso total dos itens.

6.2 Valores dos Parâmetros

A implementação desenvolvida nessa dissertação utiliza os seguintes valores para os parâmetros da heurística BLRVG-PLV:

- Expoentes a serem testados na escolha gulosa dos itens da mochila, conforme descrito na Seção 5.1.2: $\Delta = \{1, \dots, 10\}$. Como a criação de uma solução gulosa para o PLV utiliza o conjunto de valores Δ uma vez para uma rota na ordem que é dada e outra para a rota na ordem inversa, é feito um total de 20 testes de expoentes para gerar uma configuração gulosa de mochila dada uma rota;
- Quantidade de soluções iniciais geradas sem melhoria: $\mu = 5$. Como a geração de soluções iniciais, descrita na Seção 5.3, gera soluções até que sejam geradas μ soluções seguidas que não possuam valor da função objetivo maior que o melhor encontrado até então, e o valor da melhor função objetivo encontrada até então é inicializado com $-\infty$, é garantido uma geração de no mínimo seis soluções;
- Quantidade de iterações sem melhoria na solução corrente para parar a aplicação da heurística proposta: $\psi = 5.000$;
- Quantidade máxima de destruições e reconstruções proporcional ao número de iterações sem melhoria na heurística BLRVG-PLV: $\lambda = 5$. A quantidade de destruições e reconstruções λ' aplicada no Algoritmo 17 varia de 1 a $\lambda = 5$, proporcionalmente ao número de iterações sem melhorias da solução corrente. Como a quantidade máxima de iterações sem melhoria na solução corrente ψ é igual a 5.000, o valor de λ' é incrementado de 1 a cada 1.000 iterações sem melhoria da solução corrente do Algoritmo 17;
- Critério de parada de 10 minutos de execução. Apesar do tempo de execução não ser um bom critério de parada por não garantir a reprodutibilidade dos resultados em diferentes ambientes computacionais [28], esse é um critério de parada muito comumente utilizado pela literatura do PLV [11, 1, 18, 19, 29] e também por competições relacionadas ao PLV [30, 31]. Sendo assim, esse foi o critério de parada escolhido para os experimentos computacionais realizados;
- Sementes da sequência pseudoaleatória igual a $r \in \{1, \dots, 120\}$ para cada uma das 120 instâncias do PLV geradas a partir de uma mesma instância do TSPLIB, conforme explicado na Seção 6.1. Devido à variação das soluções do conjunto de soluções iniciais geradas pelo Algoritmo 12 dependerem somente da rota retornada pela heurística Chained Lin-Kernighan, foi optado por variar a semente para instâncias do PLV geradas a partir de uma mesma instância do TSPLIB. Caso não houvesse essa variação de semente, as mesmas rotas seriam utilizadas nas soluções

iniciais de instâncias do PLV geradas a partir de uma mesma instância do TSPLIB, o que poderia enviesar os experimentos. Para o caso de múltiplas execuções de uma mesma instância do PLV, é adicionado 120 à semente utilizada na execução anterior para gerar uma nova semente.

6.3 Ambiente Computacional

A heurística proposta foi implementada utilizando a linguagem de programação Rust [32]. Rust é uma linguagem de programação desenvolvida pela Mozilla Research, que possui alto desempenho com foco em segurança e garante ausência de erros de acesso de memória. O compilador utilizado foi o rustc na versão 1.75.0-nightly. Os experimentos foram realizados em um computador com processador Intel Core i7-11370H 3,30 GHz e 16GB de memória DDR4 3,20 GHz rodando o sistema operacional Linux Mint.

O computador utilizado para executar os experimentos possui 8 núcleos de CPU e a implementação da heurística proposta não utiliza paralelismo. Dessa forma, optou-se por executar 7 instâncias do problema concorrentemente, cada uma em um núcleo de CPU, deixando um núcleo livre.

Cada uma das 9.720 instâncias do PLV foi executada 5 vezes por 10 minutos utilizando diferentes sementes para iniciar o gerador de números pseudoaleatórios conforme explicado na Seção 6.2. Sendo assim, o tempo total de processamento foi de $(9.720 \times 5 \times 10 \text{ minutos}) / 7 = 69.428,57 \text{ minutos} \approx 48,21 \text{ dias}$.

6.4 Comparação dos Resultados

O objetivo principal da Busca Local Rápida em Vizinhança Grande para o PLV (BLRVG-PLV) é ser uma heurística eficaz e robusta para o Problema do Ladrão Viajante. Conforme definido por Barr et al.[33], uma heurística eficaz encontra soluções de melhor qualidade que outras heurísticas, já uma heurística robusta é uma heurística menos sensível às características das instâncias e ajustes de parâmetros. Nesta seção, os resultados obtidos pela BLRVG-PLV e os resultados obtidos pela literatura são comparados e apresentados conforme as diferentes características das instâncias para demonstrar a eficácia e robustez da BLRVG-PLV.

No intuito de demonstrar robustez, a heurística BLRVG-PLV foi executada cinco vezes para todas as 9.720 instâncias do conjunto de instâncias proposto por Polyakovskiy et al.[1]

no ambiente computacional descrito na Seção 6.3 utilizando sempre os mesmos valores de parâmetros descritos na Seção 6.2, com exceção do parâmetro “semente”. A menos que explicitamente dito o contrário, o valor reportado e o valor usado para comparação da função objetivo dos resultados da BLRVG-PLV, por instância, será a mediana dos valores da função objetivo obtidos dentre as cinco execuções da BLRVG-PLV para cada instância.

Devido a não haver padronização de como diferentes trabalhos relatam seus valores de solução, este trabalho compara todos esses valores por meio do arredondamento para o inteiro mais próximo, principalmente no auxílio à comparação com soluções reportadas pela literatura na abordagem a instâncias pequenas, cujos valores são muito próximos.

Trabalhos sobre o PLV tipicamente comparam soluções indicando se uma solução é melhor, pior ou igual a uma outra. Com o intuito de quantificar essa comparação entre soluções, alguns trabalhos adotam a métrica Diferença Percentual Relativa (DPR) [16, 24] que também será adotada por este trabalho. Dadas duas heurísticas a e b e os valores de função objetivo f_a e f_b obtidos por essas heurísticas para uma instância, a Diferença Percentual Relativa entre as soluções fornecidas pelas heurísticas a e b , $DPR(f_a, f_b)$, é descrita pela Equação 6.1 e expressa, em percentual, o quão melhor a solução obtida pela heurística a é em relação à solução fornecida pela heurística b .

$$DPR(f_a, f_b) = \frac{f_a - f_b}{|f_b|} \times 100\% \quad (6.1)$$

6.4.1 Comparação com os resultados de Wagner et al., 2018

Wagner et al.[34], dentre outras contribuições, criou um conjunto de valores da função objetivo das instâncias do PLV, que funciona como um *benchmark*, a partir da execução de 21 heurísticas para o PLV. Todas as 9.720 instâncias propostas por Polyakovskiy et al.[1] foram executadas por 10 minutos pelas seguintes heurísticas:

- SH, RLS, EA [16];
- DH [1];
- MATLS [11];
- S1, S2, S3, S4, S5, C1, C2, C3, C4, C5, C6 [18];
- CS2SA [19];
- M3, M4, M3B, M4B [23].

O ambiente computacional utilizado por Wagner et al.[34] correspondeu a um processador Intel Xeon E5430 2,66 GHz que é inferior ao processador adotado nos experimentos

50com a BLRVG-PLV, detalhado na Seção 6.3, o que certamente tende a beneficiar a BLRVG- PLV na comparação dos resultados.

Conforme detalhado na Seção 4.5, a heurística CS2SA possui um erro no cálculo da função objetivo, logo não é possível utilizar essa heurística na comparação dos resultados. O conjunto de valores da função objetivo, com exceção dos valores da heurística CS2SA, é doravante chamado de “*benchmark* de Wagner”.

A Tabela 6.1 mostra a comparação da heurística BLRVG-PLV proposta por este trabalho com as heurísticas do *benchmark* de Wagner. A coluna “Heur.” indica com qual das heurísticas do *benchmark* de Wagner é feita a comparação. A coluna “Total” indica a quantidade total de instâncias reportadas pelo *benchmark* de Wagner para a heurística. Algumas heurísticas apresentam erro na execução ou atingem a condição de parada sem produzir resultado para algumas instâncias, sendo assim, a coluna “Total” ocasionalmente pode conter um valor menor que 9.720. A coluna “Pior” indica para quantas instâncias a heurística BLRVG-PLV obteve uma solução pior que a da heurística comparada. A coluna “Pior%” indica a porcentagem da coluna “Total” de soluções piores obtidas pela heurística BLRVG-PLV em relação à heurística comparada. A coluna “Igual” indica para quantas instâncias a heurística BLRVG-PLV obteve uma solução igual à da heurística comparada. A coluna “Igual%” indica a porcentagem da coluna “Total” de soluções iguais obtidas pela heurística BLRVG-PLV em relação à heurística comparada. A coluna “MouI” indica para quantas instâncias a heurística BLRVG-PLV obteve uma solução melhor ou igual à da heurística comparada. A coluna “MouI%” indica a porcentagem da coluna “Total” de soluções melhores ou iguais obtidas pela heurística BLRVG-PLV em relação à heurística comparada. A coluna “Melhor” indica para quantas instâncias a heurística BLRVG-PLV obteve uma solução melhor que a da heurística comparada. Finalmente, a coluna “Melhor%” indica a porcentagem da coluna “Total” de soluções melhores obtidas pela heurística BLRVG-PLV em relação à heurística comparada.

Wagner et al.[34] concluiu que a heurística S5 encontra, em média, soluções melhores dentre as heurísticas estudadas, principalmente para instâncias grandes. A comparação da heurística BLRVG-PLV com o *benchmark* de Wagner por meio da Tabela 6.1 corrobora essa conclusão, visto que a pior performance da BLRVG-PLV foi quando comparada com a heurística S5, em que a heurística BLRVG-PLV encontrou 14,78% do “Total” de soluções piores que a heurística S5, que é o maior valor da tabela para a coluna “Pior%”.

Para cada uma das 9.720 instâncias do PLV foi calculada a Diferença Percentual Relativa entre os resultados obtidos pela heurística BLRVG-PLV e os resultados obtidos pela heurística S5 contidos no *benchmark* de Wagner. Para as instâncias que foi possível

Tabela 6.1: Comparação dos resultados da heurística BLRVG-PLV com o *benchmark* de Wagner.

Heur.	Total	Pior	Pior%	Igual	Igual%	MouI	MouI%	Melhor	Melhor%
SH	9.720	2	0,02	0	0,00	9.718	99,98	9.718	99,98
RLS	9.720	2	0,02	19	0,20	9.718	99,98	9.699	99,78
EA	9.720	6	0,06	18	0,19	9.714	99,94	9.696	99,75
DH	9.720	2	0,02	0	0,00	9.718	99,98	9.718	99,98
MATLS	9.516	159	1,67	2	0,02	9.357	98,33	9.355	98,31
S1	9.720	71	0,73	7	0,07	9.649	99,27	9.642	99,20
S2	9.720	89	0,92	25	0,26	9.631	99,08	9.606	98,83
S3	9.720	72	0,74	17	0,17	9.648	99,26	9.631	99,08
S4	9.720	63	0,65	22	0,23	9.657	99,35	9.635	99,13
S5	9.720	880	9,05	36	0,37	8.840	90,95	8.804	90,58
C1	9.720	100	1,03	22	0,23	9.620	98,97	9.598	98,74
C2	9.720	105	1,08	30	0,31	9.615	98,92	9.585	98,61
C3	9.720	412	4,24	82	0,84	9.308	95,76	9.226	94,92
C4	9.720	488	5,02	89	0,92	9.232	94,98	9.143	94,06
C5	9.720	90	0,93	117	1,20	9.630	99,07	9.513	97,87
C6	9.720	96	0,99	134	1,38	9.624	99,01	9.490	97,63
M3	8.999	122	1,36	75	0,83	8.877	98,64	8.802	97,81
M4	9.000	90	1,00	84	0,93	8.910	99,00	8.826	98,07
M3B	8.378	135	1,61	300	3,58	8.243	98,39	7.943	94,81
M4B	8.404	107	1,27	310	3,69	8.297	98,73	7.987	95,04

gerar resultados a tempo, os valores de DPR calculados dessa forma variam no intervalo $[-17, 12\%, 730, 48\%]$. A Figura 6.1 ilustra um mapa de calor com os valores DPR calculados dessa forma.

O mapa de calor da Figura 6.1 é organizado em 81 linhas, representando as instâncias do TSPLIB em ordem não decrescente do número de cidades da instância. Cada coluna da Figura 6.1 é uma instância do PLV gerada a partir da instância do TSPLIB da linha correspondente. As instâncias do PLV de cada linha são divididas pelos três tipos de mochila: BSC, USW e UNC. Dentro de cada divisão por tipo de mochila, as instâncias são divididas pelas quantidades de itens por cidade: 1, 3, 5 e 10. Dentro de cada divisão por quantidades de itens por cidade, as instâncias são posicionadas em ordem não decrescente da capacidade da mochila. Instâncias com valor de DPR positivo são denotadas na cor azul, que é mais escura conforme o DPR aumenta, já instâncias com o valor de DPR negativo são denotadas na cor vermelha, que é mais escura conforme o DPR diminui.

Na intenção de melhor representar os valores mais comuns de DPR na Figura 6.1, das 9.720 instâncias consideradas, as 54 (0,56%) instâncias com valor de DPR inferior a -2% são representadas por -2% e as 1.429 (14,70%) instâncias com valor DPR superior a 10% são representadas pelo valor 10% .

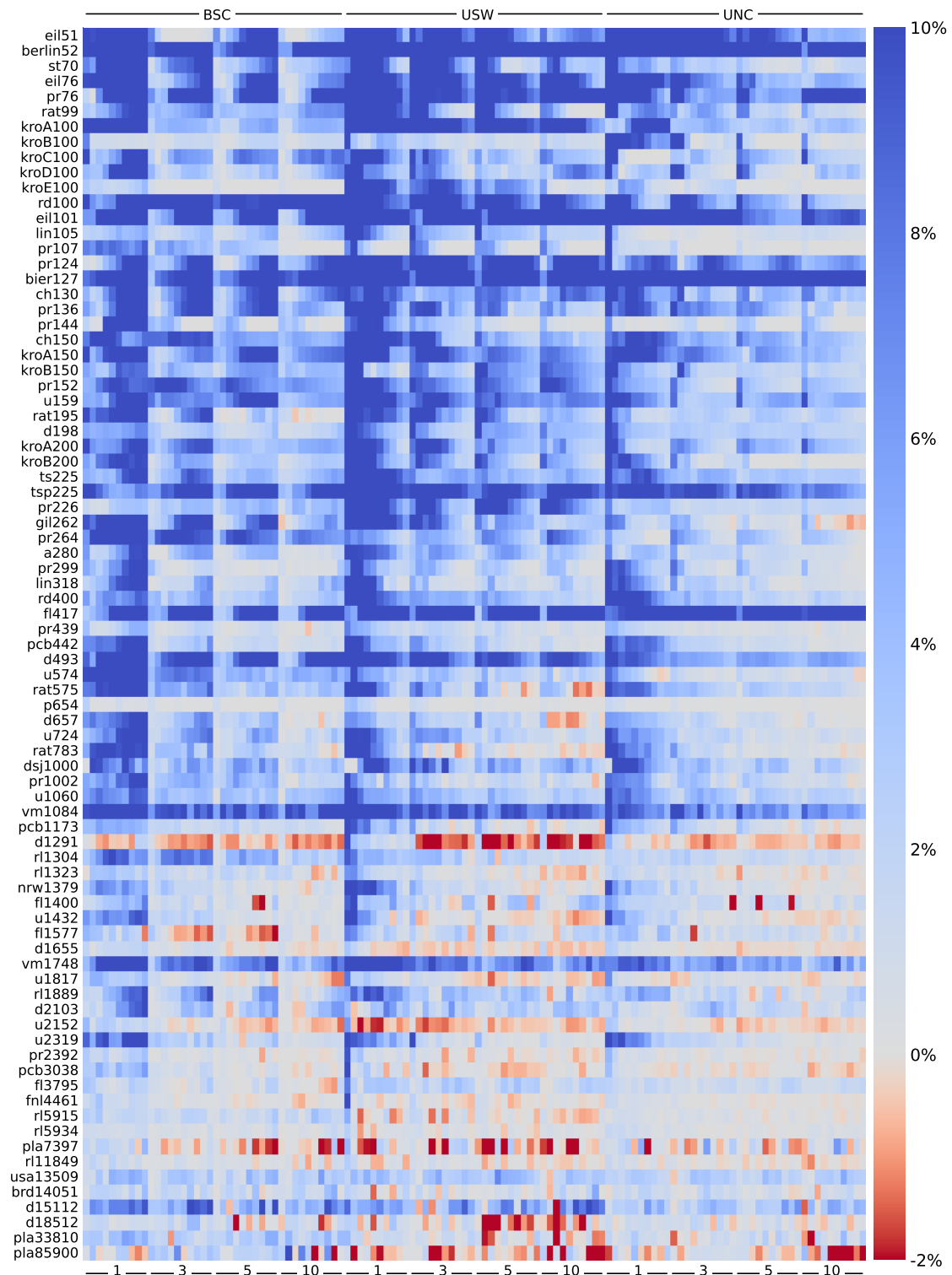
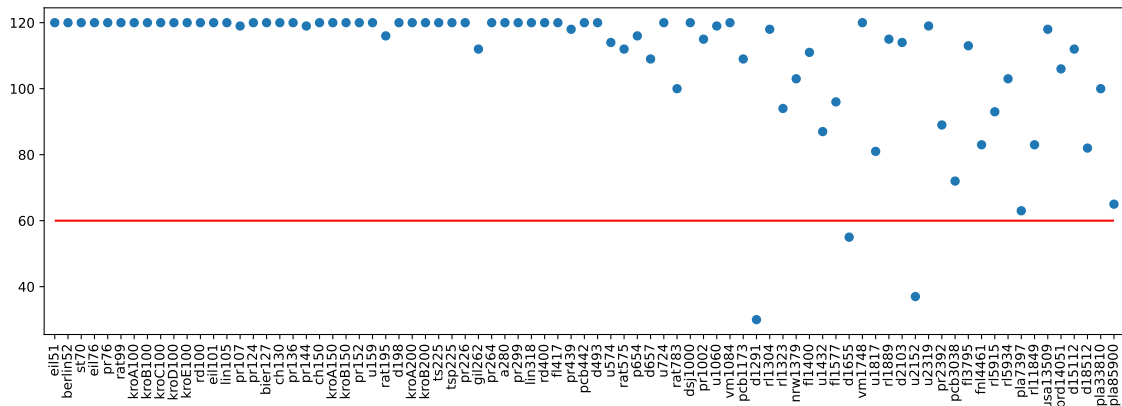


Figura 6.1: Mapa de calor comparando a heurística BLRVG-PLV com a heurística S5 utilizando as instâncias de Polyakovskiy et al.[1] e a métrica Diferença Percentual Relativa. Cada quadrado representa uma instância do PLV. Quanto mais azul o quadrado, maior a diferença positiva entre a solução encontrada pelo BLRVG-PLV para a instância. Quadrados em tons de vermelho representam instâncias para as quais a BLRVG-PLV obteve soluções piores que a heurística S5. Soluções com valor da função objetivo similares, ou seja, com DPR próximo a 0%, são representadas pela cor cinza.

É possível observar por meio da Figura 6.1 que a heurística BLRVG-PLV encontra melhores soluções se comparada com a heurística S5 para instâncias com menor número de cidades. Das 6.360 instâncias com até 1.173 cidades, a BLRVG-PLV encontrou uma solução melhor em 6.242 (98,14%) delas. Já para as 3.360 instâncias com mais de 1.173 cidades, a BLRVG-PLV encontrou 2.562 (76,25%) soluções melhores. Têm-se 54 (0,55%) soluções obtidas pela BLRVG-PLV possuindo valor de DPR inferior a -2% , todas com 1.291 cidades ou mais, já 1.492 (15,35%) das soluções obtidas pela BLRVG-PLV obtiveram um valor de DPR maior que 10%.

A Figura 6.2 compara as heurísticas BLRVG-PLV e S5 por instância do TSPLIB. O eixo x indica, em ordem não decrescente do número de cidades, qual instância do TSPLIB deu origem às instâncias do PLV. Para cada instância do TSPLIB, existem 120 instâncias do PLV conforme detalhado na Seção 6.1. O eixo y indica a quantidade dessas instâncias para as quais a BLRVG-PLV encontrou melhores soluções que a heurística S5. Pontos acima da linha horizontal vermelha indicam que a heurística BLRVG-PLV encontrou maior quantidade de soluções melhores quando comparada à heurística S5, à medida que pontos abaixo da linha horizontal vermelha indicam que a heurística S5 encontrou maior quantidade de soluções melhores.



A análise da comparação da BLRVG-PLV com as heurísticas do *benchmark* de Wagner é favorável à eficácia e à robustez da BLRVG-PLV. Para todas as heurísticas comparadas, a BLRVG-PLV conseguiu encontrar melhores soluções para mais de 90% das instâncias. A melhor heurística dentre as consideradas por Wagner et al.[34] foi a heurística S5 em relação à qual a BLRVG-PLV encontra soluções melhores em 90,58% das instâncias. Na comparação das instâncias do PLV agrupadas por instância do TSPLIB, a BLRVG-PLV encontra melhores soluções em todos os casos exceto três, demonstrando sua robustez. A simplicidade da heurística S5, envolvendo somente uma heurística gulosa para gerar a configuração da mochila se mostra competitiva para instâncias com maior quantidade de cidades, entretanto a busca local rápida implementada pela BLRVG-PLV permite explorar melhor o espaço de soluções de cada instância encontrando melhores soluções no geral, especialmente para soluções com um menor número de cidades e, consequentemente, itens.

6.4.2 Comparação com as competições GECCO de 2017 e 2023

A conferência *The Genetic and Evolutionary Computation Conference - GECCO* [35] organiza competições desde o ano 2014 para o Problema do Ladrão Viajante, tanto na sua formulação original como em variantes. Nessa competição, os participantes se dividem em equipes e enviam as soluções obtidas, assim como as implementações associadas, no intuito de comparar o número de melhores soluções encontradas. Em particular, cabe destacar as duas últimas competições para a formulação original do PLV, que é a versão abordada por este trabalho, organizadas em 2017 [30] e 2023 [31].

Na competição GECCO 2017 [30], os participantes necessitavam somente enviar as soluções obtidas para as instâncias de Polyakovskiy et al.[1], sem a obrigação de apresentar informações adicionais. Dessa forma, não é possível saber qual tipo de abordagem foi adotada, tampouco qual o critério de parada escolhido nem qual foi o ambiente computacional adotado. Nessa edição, houve a participação de quatro equipes competidoras. Quatro competidores participaram da GECCO 2017: Jonatas Chagas, Johannes Karder, Mohamed El Yafrani e Wim Vancroonenburg. A Tabela 6.2 mostra a comparação da heurística BLRVG-PLV proposta por este trabalho com os resultados da competição GECCO 2017. A coluna “Comp.” indica a sigla do nome e sobrenome do competidor cujos resultados são comparados, a coluna “Total” indica a quantidade de soluções submetidas pelo competidor, as demais colunas devem ser interpretadas da mesma forma que a Tabela 6.1.

É possível observar na Tabela 6.2 que os competidores JK e WV enviaram soluções para uma fração pequena do total de 9.720 instâncias do PLV e que a heurística BLRVG-

Tabela 6.2: Comparação dos resultados da heurística BLRVG-PLV com as soluções obtidas pelos competidores da competição GECCO 2017.

Comp.	Total	Pior	Pior%	Igual	Igual%	MouI	MouI%	Melhor	Melhor%
JC	9.720	485	4,99	865	8,90	9.235	95,01	8.370	86,11
JK	60	0	0,00	17	28,33	60	100,00	43	71,67
MY	9.716	1.108	11,40	326	3,36	8.608	88,60	8.282	85,24
WV	9	0	0,00	0	0,00	9	100,00	9	100,00

PLV obteve uma solução melhor ou igual para todas essas instâncias. É possível observar também que os competidores JC e MY enviaram soluções para quase todas as instâncias do PLV disponíveis e que a heurística BLRVG-PLV encontra soluções melhores em mais de 85% dos casos.

As Figuras 6.3 e 6.4, que devem ser interpretadas de forma similar à Figura 6.1, ilustram mapas de calor comparando os resultados obtidos pela heurística BLRVG-PLV com os enviados à competição GECCO 2017 pelos competidores JC e MY, respectivamente, utilizando a metrica DPR.

Das 9.720 instâncias consideradas, as 24 (0,25%) instâncias que obtiveram valor DPR inferior a -2% são representadas por -2% e as 4.646 (47,8%) instâncias que obtiveram valor DPR superior a 10% são representadas por 10% na Figura 6.3. Já na Figura 6.4 das 9.720 instâncias consideradas, as 283 (2,91%) instâncias que obtiveram valor DPR inferior a -2% são representadas por -2% e as 569 (5,85%) instâncias que obtiveram valor DPR superior a 10% são representadas por 10% na Figura 6.3.

A Figura 6.3 mostra que a BLRVG-PLV obtém soluções de DPR iguais ou superiores a 10% se comparadas às submetidas por JC para uma vasta seleção de instâncias dentre as de maior número de cidades. A BLRVG-PLV encontra uma solução melhor que a submetida por JC para todas as instâncias com 442 cidades ou mais, com exceção das instâncias em que o valor do aluguel por unidade de tempo é $R = 0$, que podem ser observadas em cinza ou vermelho na linha “dsj1000” da Figura 6.3. Observando a função objetivo dada pela Equação 2.16, é possível notar que, dado o valor de $R = 0$, não há custo em percorrer uma rota, eliminando o impacto da rota da solução no valor da função objetivo. Nesse caso, a instância do PLV é equivalente a uma instância do Problema da Mochila, tornando menos interessantes comparações utilizando instâncias com essa característica. Para instâncias com menos de 442 cidades a BLRVG-PLV encontra com frequência soluções melhores ou iguais que as submetidas por JC.

Na Figura 6.4, comparando a heurística BLRVG-PLV com os resultados submetidos por MY, é apresentado um resultado mais homogêneo com relação ao número de cidades

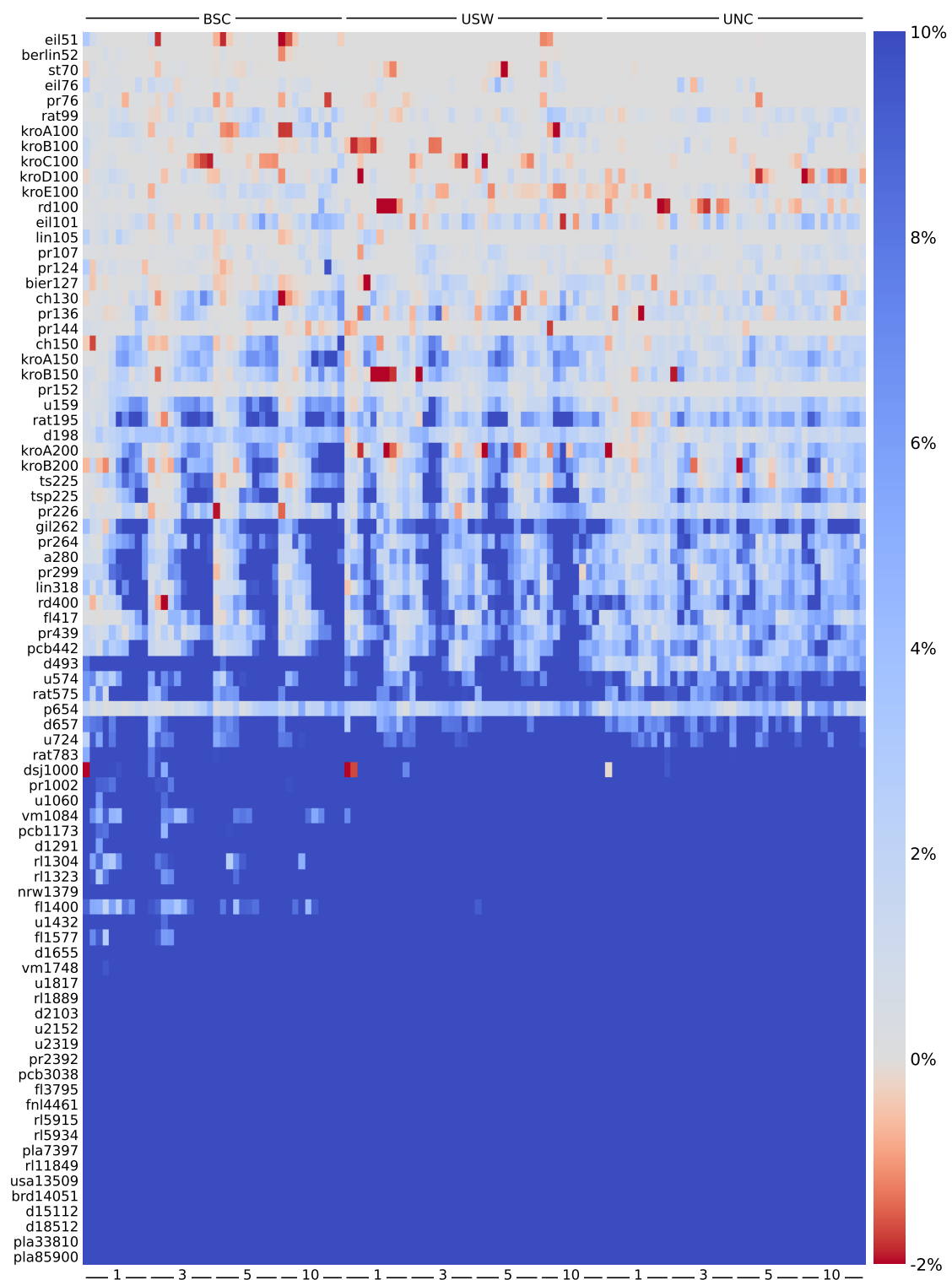


Figura 6.3: Mapa de calor utilizando a métrica DPR comparando os resultados da BLRVG-PLV com os resultados de JC obtidos na competição GECCO 2017 para as instâncias de Polyakovskiy et al.[1].

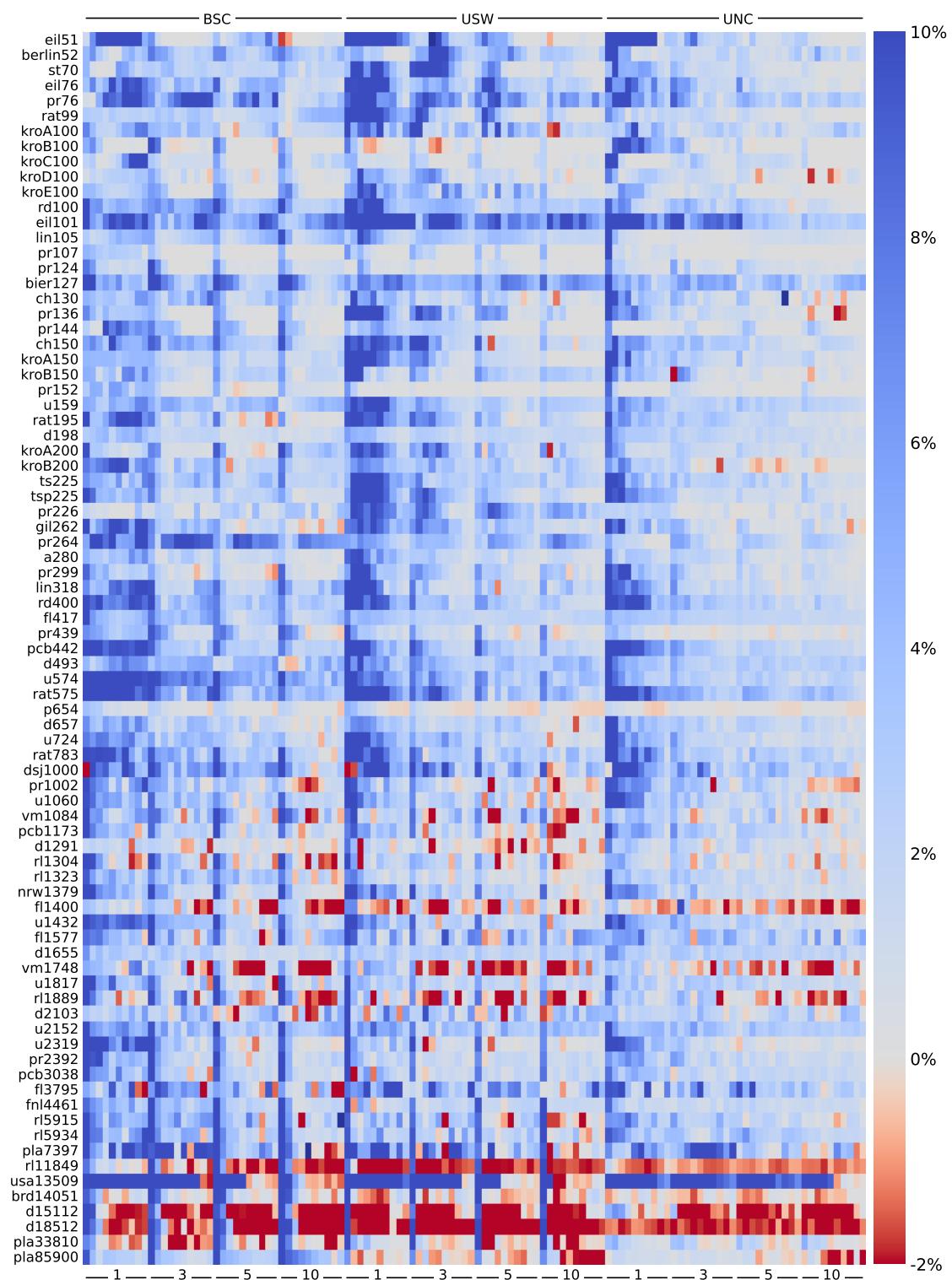


Figura 6.4: Mapa de calor utilizando a métrica DPR comparando os resultados da BLRVG-PLV com os resultados de MY obtidos na competição GECCO 2017 para as instâncias de Polyakovskiy et al.[1].

da instância, com valores favoráveis à BLRVG-PLV para instâncias com menor número de cidades. Entretanto, é possível notar valores de DPR favoráveis à BLRVG-PLV para instâncias com mochilas de menor capacidade. A Figura 6.5 compara as heurísticas por capacidade da mochila da instância. O eixo “x” indica a fração da quantidade total de itens que a capacidade da mochila possui, enquanto o eixo “y” indica a média do valor de DPR obtida na comparação da BLRVG-PLV com os resultados submetidos por MY para as instâncias com a capacidade de mochila igual à fração do eixo “x”.

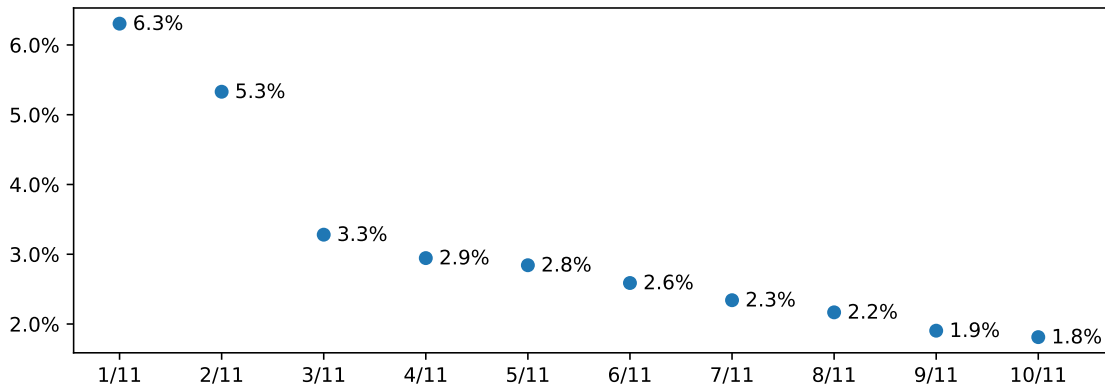


Figura 6.5: Comparação da média do valor de DPR obtido pela BLRVG-PLV quando comparada aos resultados submetidos por MY à competição GECCO 2017. O eixo x indica a fração da capacidade da mochila, o eixo “y” indica a média do valor de DPR.

A BLRVG-PLV divide os itens da mochila em subvizinhanças e primeiro considera a busca local em subvizinhanças nas quais o item se encontra na mochila utilizando a vizinhança *Bit-Flip* conforme detalhado na Seção 5.2.1. Em outras palavras, a BLRVG-PLV prioriza remover itens da mochila ao invés de adicioná-los. Para mochilas com capacidades pequenas, isso pode contribuir na obtenção de melhores soluções se comparadas a outras heurísticas sem essa característica, uma vez que a remoção de itens pode liberar espaço para a adição de itens pesados porém de grande valor, o que é uma possível explicação para a comparação favorável à BLRVG-PLV para instâncias de menor capacidade de mochila.

A competição GECCO 2023 [31] utilizou regras diferentes da competição de 2017. Na GECCO 2023, os competidores enviaram soluções para nove instâncias predeterminadas, assim como a implementação de uma heurística para ser executada em nove outras instâncias não reveladas previamente, totalizando 18 instâncias. Dessa maneira, os competidores foram julgados pelo número de melhores soluções, dentre todos os competidores, encontradas para as 18 instâncias. Uma versão preliminar da BLRVG-PLV competiu na GECCO 2023, obtendo o primeiro lugar dentre cinco equipes participantes¹.

¹Página da competição GECCO 2023: <<https://sites.google.com/view/ttp-gecco2023/home>>

6.4.3 Comparação com o Trabalho de Yafrani e Ahiod, de 2018

Yafrani e Ahiod[22] propuseram as heurísticas CS2SA* e CS2SA-R para o PLV, conforme descrito na Seção 4.5, e testaram as heurísticas em 60 das 9.720 instâncias de Polyakovskiy et al.[1]. Foram selecionadas instâncias do PLV derivadas das seguintes instâncias do TSPLIB: eil76, kroA100, ch130, u159, a280, u574, u724, dsj1000, rl1304, fl1577, d2103, pcb3038, fnl4461, pla7397, rl11849, usa13509, brd14051, d15112, d18512 e pla33810. Para cada uma dessas instâncias do TSPLIB, os autores selecionaram três categorias de instância do PLV:

- Categoria “1_bsc_01”: instância contém um item por cidade, o tipo do subproblema da Mochila é BSC e a capacidade da mochila é 1/11 do peso total dos itens da instância;
- Categoria “5_usw_05”: instância contém 5 itens por cidade, o tipo do subproblema da Mochila é USW e a capacidade da mochila é 5/11 do peso total dos itens da instância;
- Categoria “10_unc_10”: instância contém 10 itens por cidade, o tipo do subproblema da Mochila é UNC e a capacidade da mochila é 10/11 do peso total dos itens da instância.

Os autores implementaram as heurísticas CS2SA* e CS2SA-R em Java, executaram cada instância escolhida 10 vezes utilizando 10 minutos de execução como critério de parada e reportam a média obtida dessas 10 execuções. O ambiente computacional não foi especificado.

Nesse sentido, a Tabela 6.3 mostra a comparação da heurística BLRVG-PLV proposta por este trabalho com as heurísticas CS2SA* e CS2SA-R. As colunas devem ser interpretadas da mesma forma que as da Tabela 6.1.

Tabela 6.3: Comparação dos resultados da heurística BLRVG-PLV com as heurísticas CS2SA* e CS2SA-R.

Heur.	Total	Pior	Pior%	Igual	Igual%	MouI	MouI%	Melhor	Melhor%
CS2SA*	60	15	25,00	0	0,00	45	75,00	45	75,00
CS2SA-R	60	5	8,33	0	0,00	55	91,67	55	91,67

É possível notar pela Tabela 6.3 que, dentre as duas heurísticas comparadas, a heurística CS2SA* possui um melhor desempenho quando comparada à BLRVGB-PLV.

A Figura 6.6 ilustra um mapa de calor que expressa a comparação dos resultados obtidos pela heurística BLRVG-PLV com os resultados da heurística CS2SA*. Para cada uma das 60 instâncias utilizadas por Yafrani e Ahiod[22], foi calculada a Diferença Percentual

Relativa entre as heurísticas BLRVG-PLV e S5 que varia no intervalo $[-4,38\%, 27,00\%]$. 5 instâncias possuem valor DPR inferior a -2% e 14 instâncias possuem valor de DPR superior a 10% e suas cores são representadas utilizando os valores de DPR -2% e 10% , respectivamente. O mapa de calor da Figura 6.6 é organizado em 20 colunas, representando as instâncias do TSPLIB que foram adotadas, em ordem não decrescente do número de cidades da instância. As linhas da Figura 6.6 são instâncias do PLV geradas a partir da instância do TSPLIB da coluna, divididas nas categorias “1_bsc_01”, “5_usw_05” e “10_unc_10”. Instâncias com valor de DPR positivo são denotadas na cor azul, que é mais escura conforme o DPR aumenta, já instâncias com o valor de DPR negativo são denotadas na cor vermelha, que é mais escura conforme o DPR diminui. Soluções com valor da função objetivo similares, ou seja, com DPR próximo a 0% , são representadas pela cor cinza. O valor do DPR é mostrado para cada instância sem considerar os limites de $[-2\%, 10\%]$.

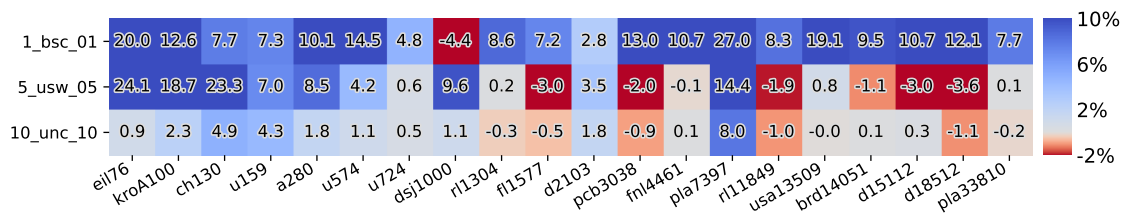


Figura 6.6: Mapa de calor comparando a heurística BLRVG-PLV com a heurística CS2SA* utilizando a métrica DPR, em relação às 20 instâncias do TSPLIB que foram adotadas.

A Figura 6.6 permite concluir que a BLRVG-PLV encontra melhores soluções para todas as instâncias da categoria “1_bsc_01” com exceção da instância gerada a partir da instância “dsj_1000” do TSPLIB que possui valor de aluguel por unidade de tempo $R = 0$ e que, conforme discutido na Seção 6.4.2, não é considerada por este trabalho uma instância interessante para comparação.

As instâncias da categoria “5_usw_05” apresentaram maior diferença entre os resultados encontrados. Para instâncias dessa categoria com muitas cidades, a BLRVG-PLV encontrou soluções até $3,62\%$ piores que a CS2SA* comparando-se por meio da métrica DPR. A BLRVG-PLV encontrou um total de 13 soluções melhores para essa categoria de instâncias e 7 soluções piores.

As soluções encontradas para a categoria “10_unc_10” foram muito semelhantes entre as heurísticas comparadas. Em 14 das 20 instâncias desse tipo, a BLRVG-PLV encontrou uma solução melhor, já em 6 das 20 instâncias a BLRVG-PLV encontrou uma solução pior. Independente da solução encontrada, os valores de DPR foram muito próximos de 0% , indicando soluções de valores muito próximos. Uma exceção é a instância originada do

“pla7397” que obteve um valor de DPR igual a 7,98%.

Semelhante à comparação com a heurística S5 descrita na Seção 6.4.1, a heurística BLRVG-PLV encontrou melhores soluções para todas as instâncias com 1.000 ou menos cidades, com exceção da instância “dsj_1000_1_bsc_01” cujo aluguel por unidade de tempo é $R = 0$, chegando a valores de DPR superiores a 20% para algumas instâncias. Outro destaque são as instâncias da coluna “pla7397” para as quais foram encontradas soluções significativamente melhores.

Em resumo, a heurística BLRVG-PLV possui uma comparação favorável com as heurísticas CS2SA* e CS2SA-R. Na comparação detalhada com a heurística CS2SA*, a BLRVG-PLV obteve melhores resultados principalmente para instâncias com 1.000 cidades ou menos e na categoria “1_bsc_01”. Diversas soluções encontradas pela BLRVGB-PLV possuem DPR maior que 10% quando comparadas à CS2SA*. Mesmo em casos em que BLRVG-PLV não conseguiu encontrar soluções melhores que a CS2SA*, a solução fornecida é, no máximo, 3,6% pior para instâncias de interesse.

6.4.4 Comparação com o Trabalho de Wuijts e Thierens, de 2019

Wuijts e Thierens[21] fizeram uma análise do *fitness landscape* para instâncias pequenas do PLV e propuseram a heurística MEA2P, conforme descrito na Seção 4.7. A MEA2P foi testada em 510 instâncias de Polyakovskiy et al.[1] divididas em dois subconjuntos:

- Vários itens por cidade: os autores utilizaram todas as 120 instâncias do PLV originadas das instâncias “eil51”, “berlin52” e “pr76” do TSPLIB, totalizando 360 instâncias;
- Um item por cidade: os autores utilizaram 30 instâncias do PLV com um único item por cidade originadas das instâncias “kroA100”, “bier127”, “ch130”, “rat195” e “a280” do TSPLIB, totalizando 150 instâncias.

A heurística MEA2P foi executada 10 vezes utilizando como critério de parada 2.500 iterações da busca local. Conforme descrito na Seção 6.2, a BLRVG-PLV utiliza como critério de parada 10 minutos de execução. Os autores de MEA2P citam, para comparação, o tempo de execução de 9 segundos para as instâncias geradas a partir do “eil51” que é a instância com o menor número de cidades considerada. Já para a instância com o maior número de cidades considerada, que são aquelas geradas a partir da instância “a280” do TSPLIB, os autores reportam um tempo de execução de até 23 minutos. A heurística MEA2P foi implementada utilizando a linguagem C# e executada em um processador Intel Core i7-2600K 3,40 GHz, que é ligeiramente superior ao utilizado para executar a

heurística BLRVG-PLV, conforme detalhado na Seção 6.3.

A Tabela 6.4 mostra a comparação da heurística BLRVG-PLV proposta por este trabalho com a heurística MEA2P. As colunas devem ser interpretadas da mesma forma que as da Tabela 6.1.

Tabela 6.4: Comparação dos resultados obtidos pela heurística BLRVG-PLV com os obtidos pela heurística MEA2P.

Heur.	Total	Pior	Pior%	Igual	Igual%	MouI	MouI%	Melhor	Melhor%
MEA2P	510	77	15,10	253	49,61	433	84,90	180	35,29

Ao se observar a coluna “Igual%” da Tabela 6.4, é possível notar que quase metade das soluções obtidas pela BLRVG-PLV têm o mesmo valor de função objetivo que as soluções obtidas por MEA2P. É possível observar também que a BLRVG-PLV encontra soluções melhores que a MEA2P com frequência maior que o dobro de vezes que encontra soluções piores.

A Figura 6.7 ilustra o mapa de calor de comparação das instâncias da categoria “Vários itens por cidade” entre as heurísticas BLRVG-PLV e MEA2P utilizando a métrica DPR que varia no intervalo $[-2,9\%, 0,8\%]$. Suas cores e valores devem ser interpretadas de forma semelhante às da Figura 6.6. O eixo “y” representa a instância do TSPLIB e a quantidade de itens por cidade, separados por *underscore*. O eixo “x” representa o tipo de mochila da instância e a capacidade da mochila. A Figura 6.7 permite notar que, para algumas instâncias do tipo “bsc” com 51 cidades e baixa capacidade da mochila, a BLRVG-PLV obtém soluções até 2% piores que a MEA2P, entretanto para grande parte das instâncias a BLRVG-PLV obtém soluções iguais ou melhores.

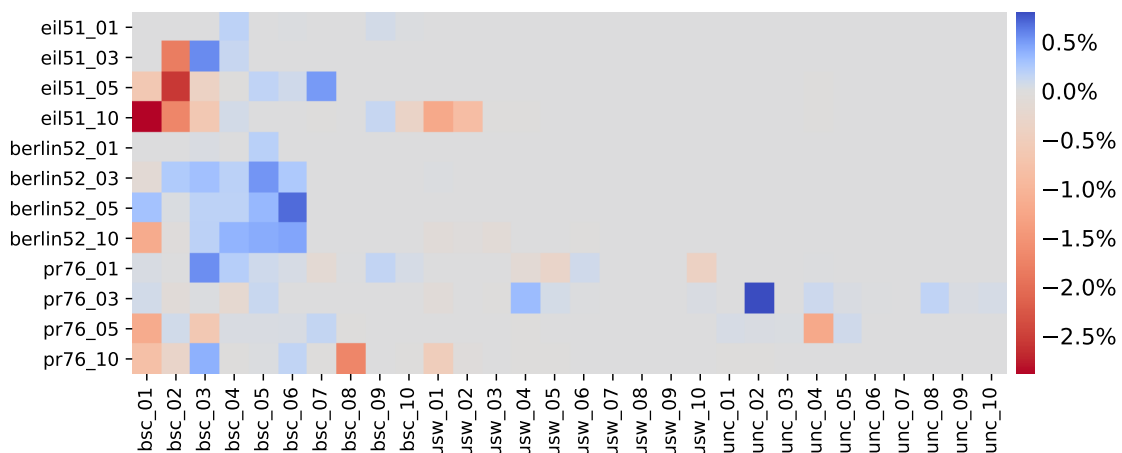


Figura 6.7: Mapa de calor comparando a heurística BLRVG-PLV com a heurística MEA2P na categoria “Vários itens por cidade” utilizando a métrica DPR.

Por sua vez, a Figura 6.8 ilustra o mapa de calor de comparação das instâncias da

categoria “Um item por cidade” entre as heurísticas BLRVG-PLV e MEA2P utilizando a métrica DPR que varia no intervalo $[-1,9\%, 2,0\%]$. Suas cores e valores devem ser interpretados tal qual conduzido para a Figura 6.6. O eixo “y” representa a instância do TSPLIB e o tipo de mochila da instância, separados por *underscore*. O eixo “x” representa a capacidade da mochila. A Figura 6.8 apresenta resultados mais homogêneos na comparação das heurísticas. O valor de DPR das piores soluções encontradas pela BLRVG-PLV chega a $-1,9\%$. Entretanto, cabe ressaltar que o tempo de execução do MEA2P para essas heurísticas excede os 10 minutos.

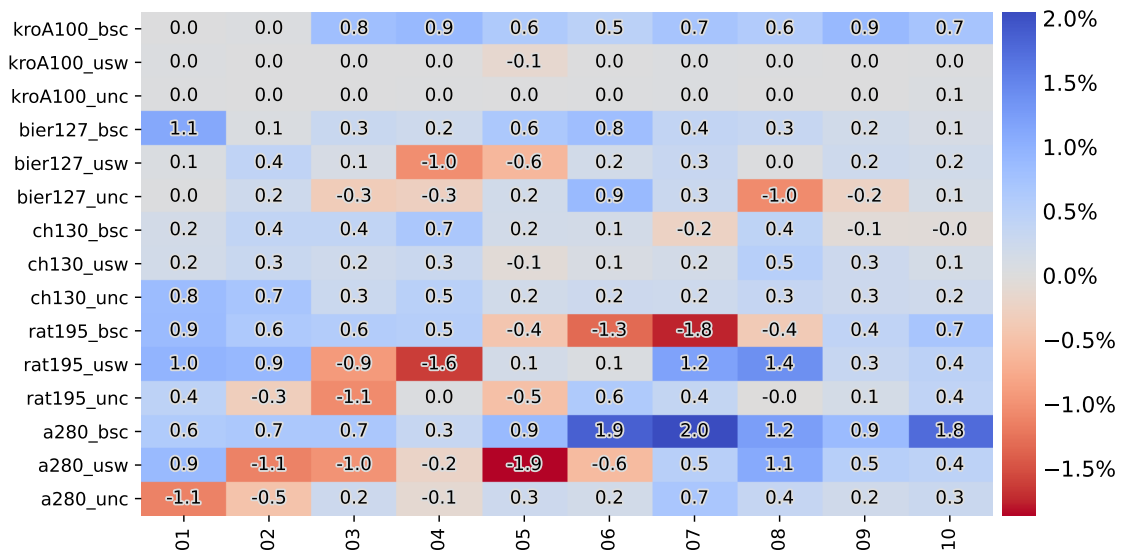


Figura 6.8: Mapa de calor comparando a heurística BLRVG-PLV com a heurística MEA2P na categoria “Um item por cidade” utilizando a métrica DPR.

Wuijts e Thierens[21] decidiram focar seus esforços em instâncias menores do PLV e reportam o tempo de execução de até 23 minutos para instâncias com 280 cidades, o que pode tornar inviável sua execução para instâncias maiores. A BLRVG-PLV almeja ser uma heurística robusta, ou seja, uma heurística que encontre boas soluções para quaisquer características da instância. O fato de a BLRVG-PLV encontrar muitas soluções melhores ou iguais a uma heurística que foca em instâncias pequenas quando comparadas utilizando as mesmas instâncias pequenas demonstra a qualidade da BLRVG-PLV.

6.4.5 Comparação com o Trabalho de Chagas e Wagner, de 2022

Chagas e Wagner[24] desenvolveram a heurística WSM para outra variante do PLV e adaptaram sua heurística para também reportar soluções para a variante original do PLV que é a abordada por este trabalho. Os autores da WSM executaram a heurística 30 vezes por instância, com o critério de parada de 10 minutos para cada execução. Todos os valores de função objetivo das soluções foram reportados e foi escolhida a mediana das 6.430

execuções para representar a função objetivo de cada instância. Foram utilizadas 960 instâncias do PLV com as seguintes características:

- Oito instâncias do TSPLIB: “eil51”, “pr152”, “a280”, “dsj1000”, “fnl4461”, “usa13509”, “pla33810” e “pla85900”;
- Todas as quatro variações de quantidade de itens por cidade: 1, 3, 5 e 10;
- Todos as três variações do tipo da mochila: BSC, USW e UNC;
- Todas as 10 variações de capacidade da mochila: {1/11, 2/11, ..., 10/11}.

A Tabela 6.5 mostra a comparação da heurística BLRVG-PLV proposta por este trabalho com a heurística WSM. As colunas devem ser interpretadas da mesma forma que as da Tabela 6.1. É possível observar que a BLRVBG-PLV encontra soluções melhores que WSM para 95,21% das instâncias aplicadas.

Tabela 6.5: Comparação dos resultados obtidos pela heurística BLRVG-PLV com os obtidos pela heurística WSM.

Heur.	Total	Pior	Pior%	Igual	Igual%	MouI	MouI%	Melhor	Melhor%
WSM	960	38	3,96	8	0,83	922	96,04	914	95,21

Apesar das heurísticas BLRVG-PLV e WSM terem sido projetadas para variantes diferentes do PLV, ainda é possível obter conclusões por meio da comparação entre as duas. O mapa de calor da Figura 6.9 apresenta a comparação das heurísticas BLRVG-PLV e WSM utilizando a métrica DPR. As cores e valores devem ser interpretados de forma semelhante aos da Figura 6.6. O eixo “y” representa a instância do TSPLIB e a quantidade de itens por cidade, separados por *underscore*. O eixo “x” representa o tipo de mochila da instância e a capacidade da mochila.

Na Figura 6.9, é possível notar que a BLRVG-PLV obtém resultados melhores que a WSM, com exceção de algumas instâncias derivadas da instância “pla85900” do TSPLIB. Instâncias derivadas da “pla85900” são muito particulares dentre o conjunto de instâncias propostas por Polyakovskiy et al.[1] por conterem uma quantidade de cidades e, consequentemente itens, muito elevada se comparadas ao restante das instâncias. Instâncias derivadas de “pla85900” possuem 85.900 cidades que é mais de 250% maior que o segundo maior conjunto de instâncias com maior número de cidades, “pla33810”.

Conforme explicado no Capítulo 5, a BLRVG-PLV gera um conjunto inicial de soluções utilizando a heurística Chained Lin-Kernighan para criar uma rota, uma heurística gulosa para gerar a configuração inicial da mochila a partir da rota e, em seguida, aplica uma busca local rápida na solução com todas as suas subvizinhanças ativas. Para as instâncias do PLV derivadas da instância “pla33810” a heurística Chained Lin-Kernighan demora em

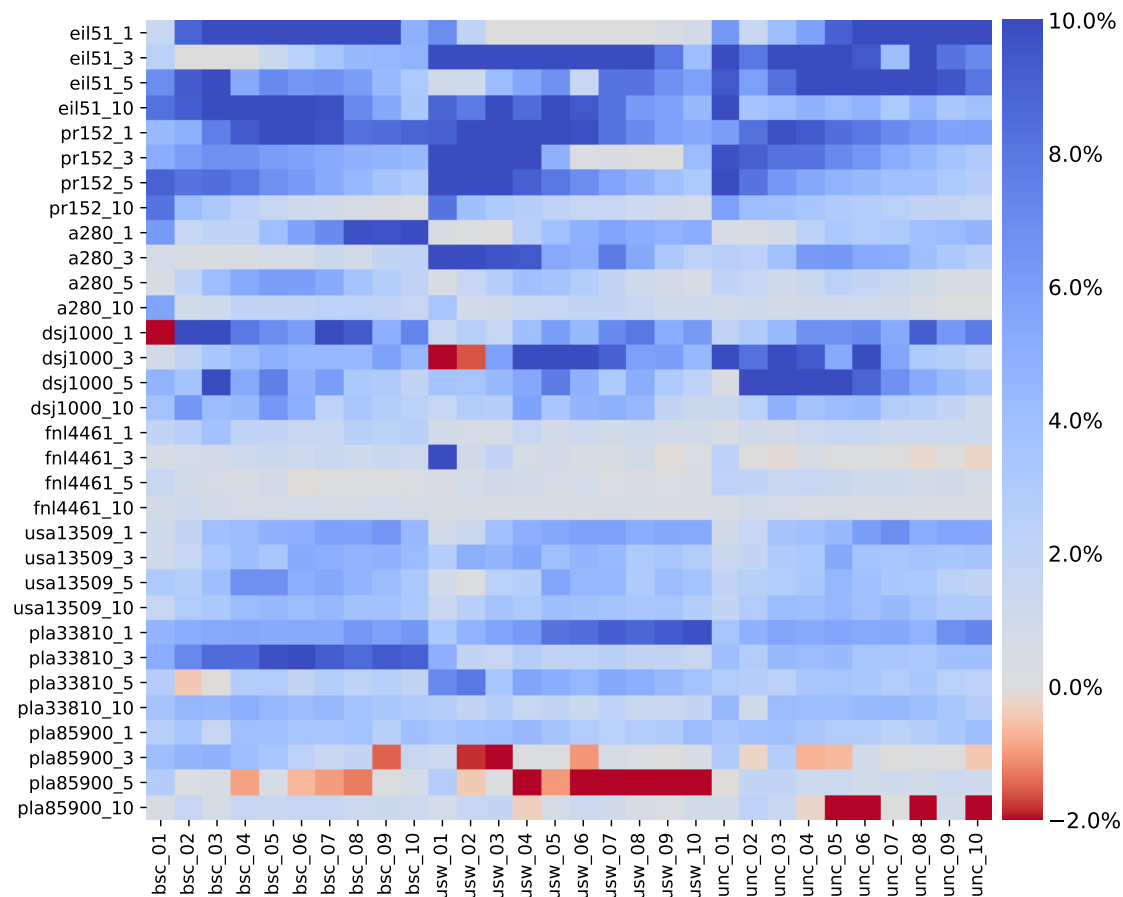


Figura 6.9: Mapa de calor comparando a heurística BLRVG-PLV com a heurística WSM utilizando a métrica DPR.

torno de 1 minuto para gerar uma rota, a heurística gulosa da mochila demora um tempo desprezível, enquanto os 9 minutos restantes são dedicados à busca local que não consegue terminar dentro do critério de parada de 10 minutos estabelecidos e gera uma solução que não é um mínimo local aproximado. A heurística BLRVG-PLV pode se beneficiar de uma melhor heurística gulosa de geração inicial da mochila para os casos extremos, em que a BLRVG-PLV não consegue executar todos seus componentes dentro do critério de parada estabelecido.

6.5 Considerações Finais

Este capítulo apresentou os resultados dos experimentos computacionais realizados com o intuito de demonstrar a eficácia e robustez da heurística proposta Busca Local Rápida em Vizinhaça Grande para o o PLV. O próximo capítulo contém as contribuições e ideias de melhorias futuras.

7. Conclusões

O principal objetivo deste trabalho foi apresentar uma heurística eficaz e robusta para o Problema do Ladrão Viajante. A heurística BLRVG-PLV proposta combina estratégias consagradas da literatura como a heurística Lin–Kernighan e a vizinhança de Delaunay (ambas para o Problema do Caixeiro Viajante), uma heurística gulosa para a configuração da mochila dada uma rota, uma busca local rápida em três vizinhanças divididas em subvizinhanças que podem ser ignoradas caso necessário, uma lista restrita de candidatos usada na vizinhança e a meta-heurística Busca em Vizinhança Grande.

7.1 Contribuições

Barr et al.[33] descreve um conjunto de propriedades que uma heurística deve ter para que, caso alguma delas esteja presente, o trabalho se caracterize como uma contribuição à literatura. Conforme averiguado nos experimentos computacionais e comparação dos resultados com a literatura realizados neste trabalho, a heurística BLRVG-PLV contribui com duas das propriedades definidas por Barr et al.[33]: eficácia, que corresponde à qualidade de uma heurística encontrar melhores soluções do que outras abordagens propostas na literatura, e robustez, que corresponde à qualidade de uma heurística que não é tão sensível às características particulares das instâncias.

A heurística BLRVG-PLV proposta foi implementada e executada 5 vezes, por um tempo máximo de 10 minutos por execução, para cada uma das 9.720 instâncias de Polyakovskiy et al.[1] comumente adotadas na literatura, e comparada com todos os trabalhos relevantes da literatura inclusive os mais recentes, como também com aqueles mais recentes e com competições envolvendo o PLV. A BLRVG-PLV obteve resultados favoráveis em todas as comparações realizadas, inclusive obtendo o primeiro lugar em uma competição organizada pela conferência GECCO em 2023, com uma versão preliminar da implementação.

Os trabalhos que utilizam todas as 9.720 instâncias do PLV são formados pelos trabalhos reportados no *benchmark* de Wagner [34] em conjunto com as soluções enviadas pelos competidores Mohamed El Yafrani e Wim Vancroonenburg para a competição GECCO 2017 [30]. A BLRVG-PLV encontra soluções melhores em no mínimo 85% das instâncias quando comparada individualmente aos trabalhos que utilizam todas as instâncias conforme é possível observar nas Tabelas 6.1 e 6.2. Ainda que os trabalhos que utilizam todas as instâncias do PLV sejam agrupados em um único conjunto e seja tomada a melhor solução para cada instância nesse conjunto, a BLRVG-PLV continua obtendo um resultado favorável na comparação. Nesse caso, a BLRVG-PLV encontra a melhor solução em 6.177 (63.55%) das instâncias e encontra soluções pelo menos 5% melhores utilizando a métrica Diferença Percentual Relativa para 370 (3,81%) das instâncias. A BLRVG-PLV possui comparação favorável também em relação à heurística MEA2P que foca exclusivamente em instâncias pequenas, com menos de 280 cidades, conforme detalhado na Seção 6.4.4, encontrando soluções melhores ou iguais em quase 85% das 510 instâncias que foram utilizadas na comparação.

7.2 Trabalhos Futuros

A BLRVG-PLV encontra boas soluções para o PLV para todos os tipos de instâncias, porém a heurística se mostrou particularmente eficaz para instâncias menores 1, 173 cidades ou menos do PLV. É necessária uma melhor investigação para constatar se as estratégias propostas pela BLRVG-PLV possuem esse diferencial quando comparadas com outros trabalhos para essas instâncias menores ou se há melhorias a serem feitas na BLRVG-PLV para trazer a qualidade já boa das soluções de instâncias maiores ao patamar das soluções encontradas para instâncias menores. Algumas propostas de trabalhos futuros envolvem o estudo e aplicação de uma alternativa à heurística gulosa proposta para geração inicial da configuração da mochila, assim como a aplicação de lista de candidatos e desativação de subvizinhanças à vizinhança da mochila, que atualmente são adotadas somente nas vizinhanças da rota.

Excluindo-se a semente do gerador de números pseudoaleatórios, a BLRVG-PLV possui quatro parâmetros que controlam seu comportamento. A escolha dos valores, desses parâmetros foi feita empiricamente durante a implementação por meio de testes preliminares, isolados e com limitado número de instâncias. Acredita-se que a BLRVG-PLV, por meio de um estudo experimental rigoroso de ajuste fino dos valores dos parâmetros, pode se beneficiar obtendo, eventualmente, soluções de qualidade ainda melhores.

Uma outra possibilidade de trabalho futuro é estender a aplicação do método proposto a algum outro problema. Por exemplo, existem uma variação do Problema do Ladrão Viajante utilizando múltiplas funções objetivos e outra em que o ladrão não é obrigado a visitar todas as cidades. Por meio de ajustes das vizinhanças da busca local é possível adaptar a BLRVG-PLV e verificar o seu desempenho para essas variações do PLV.

Assim, conclui-se que as principais contribuições desta dissertação de mestrado foram incrementar o conjunto de melhores soluções conhecidas para instâncias do Problema do Ladrão Viajante e propor uma heurística robusta e eficaz para o Problema do Ladrão Viajante.

Referências Bibliográficas

- 1 POLYAKOVSKIY, S. et al. A comprehensive benchmark set and heuristics for the traveling thief problem. In: *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*. Vancouver BC Canada: ACM, 2014. p. 477–484.
- 2 MICHALEWICZ, Z. Quo Vadis, Evolutionary Computation?: On a Growing Gap between Theory and Practice. In: LIU, J. et al. (Ed.). *Advances in Computational Intelligence*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. v. 7311, p. 98–121.
- 3 BONYADI, M. R.; MICHALEWICZ, Z.; BARONE, L. The travelling thief problem: The first step in the transition from theoretical problems to realistic problems. In: *2013 IEEE Congress on Evolutionary Computation*. Cancun, Mexico: IEEE, 2013. p. 1037–1044.
- 4 ALSHEDDY, A. et al. Guided Local Search. In: MARTÍ, R.; PANOS, P.; RESENDE, M. G. (Ed.). *Handbook of Heuristics*. Cham: Springer International Publishing, 2016. p. 1–37.
- 5 PISINGER, D.; ROPKE, S. Large Neighborhood Search. In: GENDREAU, M.; POTVIN, J.-Y. (Ed.). *Handbook of Metaheuristics*. Boston, MA: Springer US, 2010. v. 146, p. 399–419.
- 6 GAREY, M. R.; JOHNSON, D. S. *Computers and Intractability: A Guide to the Theory of NP-completeness*. New York: W. H. Freeman, 1979. (A Series of Books in the Mathematical Sciences).
- 7 DANTZIG, G.; FULKERSON, R.; JOHNSON, S. Solution of a Large-Scale Traveling-Salesman Problem. *Journal of the Operations Research Society of America*, v. 2, n. 4, p. 393–410, nov. 1954.
- 8 LIN, S.; KERNIGHAN, B. W. An Effective Heuristic Algorithm for the Traveling-Salesman Problem. *Operations Research*, v. 21, n. 2, p. 498–516, abr. 1973.
- 9 MARTELLO, S.; TOTH, P. Algorithms for Knapsack Problems. In: *North-Holland Mathematics Studies*. [S.l.]: Elsevier, 1987. v. 132, p. 213–257.
- 10 MARTELLO, S.; TOTH, P. *Knapsack Problems: Algorithms and Computer Implementations*. Chichester: J. Wiley, 1990. (Wiley-Interscience Series in Discrete Mathematics and Optimization).

- 11 MEI, Y.; LI, X.; YAO, X. Improving Efficiency of Heuristics for the Large Scale Traveling Thief Problem. In: DICK, G. et al. (Ed.). *Simulated Evolution and Learning*. Cham: Springer International Publishing, 2014. v. 8886, p. 631–643.
- 12 APPELEGATE, D.; COOK, W.; ROHE, A. Chained Lin-Kernighan for Large Traveling Salesman Problems. *INFORMS Journal on Computing*, v. 15, n. 1, p. 82–92, fev. 2003.
- 13 REINELT, G. Fast Heuristics for Large Geometric Traveling Salesman Problems. *ORSA Journal on Computing*, v. 4, n. 2, p. 206–217, maio 1992.
- 14 KRASNOGOR, N. et al. A new hybrid heuristic for large geometric traveling salesman problems based on the delaunay triangulation. jul. 1999.
- 15 SHAW, P. Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems. In: GOOS, G. et al. (Ed.). *Principles and Practice of Constraint Programming — CP98*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998. v. 1520, p. 417–431.
- 16 BONYADI, M. R. et al. Socially inspired algorithms for the travelling thief problem. In: *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*. Vancouver BC Canada: ACM, 2014. p. 421–428.
- 17 CORMEN, T. H. (Ed.). *Introduction to Algorithms*. 3rd ed. ed. Cambridge, Mass: MIT Press, 2009.
- 18 FAULKNER, H. et al. Approximate Approaches to the Traveling Thief Problem. In: *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. Madrid Spain: ACM, 2015. p. 385–392.
- 19 YAFRANI, M. E.; AHIOD, B. Population-based vs. Single-solution Heuristics for the Travelling Thief Problem. In: *Proceedings of the Genetic and Evolutionary Computation Conference 2016*. Denver Colorado USA: ACM, 2016. p. 317–324.
- 20 BENTLEY, J. J. Fast Algorithms for Geometric Traveling Salesman Problems. *ORSA Journal on Computing*, v. 4, n. 4, p. 387–411, nov. 1992.
- 21 WUIJTS, R. H.; THIERENS, D. Investigation of the traveling thief problem. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. Prague Czech Republic: ACM, 2019. p. 329–337.
- 22 YAFRANI, M. E.; AHIOD, B. Efficiently solving the Traveling Thief Problem using hill climbing and simulated annealing. *Information Sciences*, v. 432, p. 231–244, mar. 2018.
- 23 WAGNER, M. Stealing Items More Efficiently with Ants: A Swarm Intelligence Approach to the Travelling Thief Problem. In: DORIGO, M. et al. (Ed.). *Swarm Intelligence*. Cham: Springer International Publishing, 2016. v. 9882, p. 273–281.
- 24 CHAGAS, J. B.; WAGNER, M. A weighted-sum method for solving the bi-objective traveling thief problem. *Computers & Operations Research*, v. 138, p. 105560, fev. 2022.
- 25 NAMAZI, M. et al. A Cooperative Coordination Solver for Travelling Thief Problems. arXiv, 2019.

- 26 ZHANG, Z. et al. Solving the Traveling Thief Problem Based on Item Selection Weight and Reverse-Order Allocation. *IEEE Access*, v. 9, p. 54056–54066, 2021.
- 27 MARTELLO, S.; PISINGER, D.; TOTH, P. Dynamic Programming and Strong Bounds for the 0-1 Knapsack Problem. *Management Science*, v. 45, n. 3, p. 414–424, mar. 1999.
- 28 JOHNSON, D. A theoretician’s guide to the experimental analysis of algorithms. dez. 2001.
- 29 YAFRANI, M. E.; AHIOD, B. Cosolver2B: An efficient local search heuristic for the Travelling Thief Problem. In: *2015 IEEE/ACS 12th International Conference of Computer Systems and Applications (AICCSA)*. Marrakech, Morocco: IEEE, 2015. p. 1–5.
- 30 TTP 2017 Competition - Optimisation of Problems with Multiple Interdependent Components. 2017. <https://cs.adelaide.edu.au/~optlog/TTP2017Comp/>.
- 31 TTP - GECCO 2023. 2023. <https://sites.google.com/view/ttp-gecco2023/home>.
- 32 KLABNIK, S.; NICHOLS, C. *The Rust Programming Language*. 2nd edition. ed. San Francisco: No Starch Press, 2023.
- 33 BARR, R. S. et al. Designing and reporting on computational experiments with heuristic methods. *Journal of Heuristics*, v. 1, n. 1, p. 9–32, set. 1995.
- 34 WAGNER, M. et al. A case study of algorithm selection for the traveling thief problem. *Journal of Heuristics*, v. 24, n. 3, p. 295–320, jun. 2018.
- 35 GECCO 2024 | HomePage. 2024. <http://gecco-2024.sigevo.org/HomePage>.

