Universidade Federal do Rio Grande do Norte
Centro de Ciências Exatas e da Terra
Departmento de Informática e Matemática Aplicada
Programa de Pós-Graduação em Sistemas e Computação
Doutorado Acadêmico em Ciência da Computação

Problema do Caixeiro Viajante Alugador com Passageiros

Gustavo de Araujo Sabry

Natal-RN
Junho de 2020

Gustavo de Araujo Sabry

Problema do Caixeiro Viajante Alugador com Passageiros

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Sistemas e Computação do Departamento de Informática e Matemática Aplicada da Universidade Federal do Rio Grande do Norte como requisito para a obtenção do grau de Doutor em Ciência da Computação.

Linha de Pesquisa: *Algoritmos Experimentais*

Orientador

Prof. Dr. Marco Cesar Goldbarg

PPgSC – Programa de Pós-Graduação em Sistemas e Computação
DIMAp – Departamento de Informática e Matemática Aplicada
CCET – Centro de Ciências Exatas e da Terra
UFRN – Universidade Federal do Rio Grande do Norte

Natal-RN

Junho de 2020

Universidade Federal do Rio Grande do Norte - UFRN
Sistema de Bibliotecas - SISBI
Catalogação de Publicação na Fonte. UFRN - Biblioteca Central Zila Mamede

Sabry, Gustavo de Araujo.
Problema do caixeiro viajante alugador com passageiros / Gustavo de Araujo Sabry. - 2020.
137 f.: il.

Tese (doutorado) - Universidade Federal do Rio Grande do Norte, Centro de Ciências Exatas e da Terra, Programa de Pós-Graduação em Sistemas e Computação.
Orientador: Prof. Dr. Marco Cesar Goldbarg.
Coorientadora: Profa. Dra. Elizabeth Ferreira Gouvêa Goldbarg.

1. Problema do caixeiro viajante alugador com passageiros - Tese. 2. Problema do caixeiro viajante - Tese. 3. Otimização combinatória - Tese. I. Goldbarg, Marco Cesar. II. Goldbarg, Elizabeth Ferreira Gouvêa. III. Título.

RN/UF/BCZM CDU 004.4:519.16

Elaborado por Ana Cristina Cavalcanti Tinoco - CRB-15/262

GUSTAVO DE ARAÚJO SABRY

"Problema do Caixeiro Viajante Alugador com Passageiros"

Esta Tese foi julgada adequada para a obtenção do título de doutor em Ciência da Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Sistemas e Computação do Departamento de Informática e Matemática Aplicada da Universidade Federal do Rio Grande do Norte.

Dr. MARCO CÉSAR GOLDBARG (UFRN)
Presidente – Orientador

Profa. Dra. ANNE MAGALY DE PAULA CANUTO
Coordenadora do PPgSC

Banca Examinadora

Dra. ELIZABETH FERREIRA GOUVEA GOLDBARG (UFRN)
Examinadora Interna

Dr. MATHEUS DA SILVA MENEZES (UFERSA)
Examinador Externo

Dr. PAULO HENRIQUE ASCONAVIETA DA SILVA (IFRS)
Examinador Externo

Dra. THATIANA CUNHA NAVARRO DE SOUZA (UFERSA)
Examinadora Externa

Discente: **GUSTAVO DE ARAÚJO SABRY**

Junho, 2020

Dedico este trabalho à minha eterna afilhada Camila Nascimento Câmara de Lemos (*in memoriam*), mais conhecida como Ascam, a filha da Madoam.

29/03/1999 ⋆ – 12/11/2019 †

Agradecimentos

Gostaria de agradecer, primeiramente, à minha família. Em especial a meu pai (José Sabry), minhas mães (Haydé Sabry e Maria do Amparo), meus irmãos (Diego Sabry e Rayner Viana), minha noiva (Samara Dias) e meus gatos. Vocês participaram diretamente da minha rotina nesses quatro anos, me deram apoio incondicional e incentivaram a buscar sempre o melhor. Sem vocês, eu não sei se chegaria aonde estou hoje.

Agradecimento especial aos professores Marco Cesar Goldbarg e Elizabeth Ferreira Gouvêa Goldbarg que além de me orientar academicamente de forma excelente, também me ensinaram lições valiosas na pesquisa científica e trilharam o caminho para que eu atingisse o objetivo do trabalho adequadamente.

Deixo minha profunda gratidão ao professor Matheus da Silva Menezes, por ter acompanhado meu trabalho desde o início e contribuído bastante com as modelagens presentes no trabalho, com idéias/estratégias que poderiam ser aplicados ao tema estudado e, principalmente, com a motivação que me deu para finalizar este trabalho.

Agradeço aos meus amigos, por entenderem a minha jornada e me apoiarem. Acho que agora poderemos voltar a se ver. Espero que nossas amizades continuem as mesmas.

Não posso deixar de agradecer à todos os colegas do *LAE* (Laboratório de Algoritmos Experimentais), pois o grupo me mostrou que a troca de conhecimentos e discussões acerca de nossos temas é essencial para um melhor desenvolvimento de nossos trabalhos. Agradeço, em especial, à José Filho, Thiago Soares, Bruno Honorato e Yuri Kelvin.

Aproveito para fazer agradecimentos extra-especiais à minha afilhada Camila Nascimento por, mesmo tão jovem, ter me ensinado tanto sobre a vida e à Dona Estelita, por ter me criado e me ensinado tanto sobre amor, perseverança e honestidade. Obrigado por me vigiarem e me protegerem dia após dias.

Agradeço também ao *NPAD* (Núcleo de Processamento de Alto Desempenho) por dar total apoio a uma parte dos experimentos computacionais apresentados neste trabalho.

Finalmente, agradeço a todos que de uma forma direta ou indireta contribuíram para o sucesso dessa empreitada. Fica registrado o meu muito obrigado a todos.

"É preciso sair da ilha para ver a ilha. Não nos vemos se não saímos de nós."

(José Saramago)

Problema do Caixeiro Viajante Alugador com Passageiros

Autor: Gustavo de Araujo Sabry
Orientador: Prof. Dr. Marco Cesar Goldbarg

Resumo

Este trabalho apresenta uma nova variante do Problema do Caixeiro Viajante ainda não descrita na literatura, denominada de Problema do Caixeiro Viajante Alugador com Passageiros. Neste problema são disponibilizados um conjunto de cidades, um conjunto de veículos e um conjunto de passageiros em potencial. O *tour* do caixeiro pode ser realizado utilizando diferentes automóveis, ou seja, o problema engloba o processo de aluguel de veículos. O modelo proposto também inclui elementos relacionados ao compartilhamento dos assentos do veículos utilizado, ou seja, nas cidades podem haver pessoas interessadas em viajar para um determinado destino e dispostas a dividir os custos com o caixeiro enquanto estão embarcadas no veículo. O objetivo do problema é determinar, em um grafo, o ciclo Hamiltoniano de menor custo considerando as trocas de veículos e os embarques de passageiros durante o percurso. O problema é composto por várias decisões interligadas: a sequência das cidades visitadas, a ordem dos carros utilizados, as cidades onde os automóveis devem ser alugados/devolvidos, bem como o esquema de embarque dos passageiros. A definição do problema proposto neste trabalho envolve a combinação de dois conceitos importantes que estão sendo amplamente utilizados no setor de transportes: o aluguel e o compartilhamento de veículos. São propostas três formulações de programação inteira mista. Estas formulações são linearizadas utilizando técnicas diferentes, resultando em seis modelos lineares. Estes modelos são implementados em um *solver* e validados. Além disso, também são apresentadas três heurísticas ingênuas e três meta-heurísticas para solucionar o problema. Experimentos computacionais comparativos e testes de desempenho são realizados sobre uma amostra de 90 instâncias. Os resultados obtidos são comparados e as conclusões são reportadas.

Palavras-chave: Problema do Caixeiro Viajante Alugador com Passageiros, Problema do Caixeiro Viajante, Otimização Combinatória.

The Traveling Car Renter with Passengers

Author: Gustavo de Araujo Sabry
Advisor: Prof. Dr. Marco Cesar Goldbarg

Abstract

This paper presents a new variant of the Traveling Salesman Problem not yet described in the literature, called the Traveling Car Renter with Passengers. This problem provides a set of cities, a set of vehicles and a set of potential passengers. The salesman's tour can be done using different vehicles, i.e., the problem encompasses the process of car rental. The proposed model also includes elements related to the sharing of the seats of the used vehicle, i.e., in the cities there may be people interested in traveling to a certain destination and willing to share the costs with the salesman while they are aboard in the vehicle. The objective of the problem is to determine, in a graph, the Hamiltonian cycle with the lowest cost considering the vehicles' exchanges and the shipments along the tour. The problem is made up of several interlinked decisions: the sequence of visited cities, the order of used cars, the cities where the cars must be rented and/or delivered and the passengers' boarding scheme. The definition of the proposed problem involves the combination of two important concepts that are currently being widely used in the field of transportation: car rental and ride-sharing. Three formulations of mixed integer programming are proposed. These formulations are linearized using different techniques, resulting in six linearized models. These models are implemented in a solver and validated. In addition, three naive heuristics and three metaheuristics are presented to solve the problem. Comparative computational experiments and performance tests are performed on a set of 90 instances. The results obtained are compared and the conclusions are reported.

Keywords: Traveling Car Renter with Passengers, Traveling Salesman Problem, Combinatorial Optimization.

Lista de figuras

1 *PCV-P* – Exemplo de instância . . . . . p. 28

2 *PCV-P* – Construção da solução - Parte 1 . . . . . p. 28

3 *PCV-P* – Construção da solução - Parte 2 . . . . . p. 29

4 *CaRS* – Exemplo de instância . . . . . p. 34

5 *CaRS* – Construção da solução - Parte 1 . . . . . p. 34

6 *CaRS* – Construção da solução - Parte 2 . . . . . p. 35

7 *CaRSP* – Informações referentes aos carros . . . . . p. 42

8 *CaRSP* – Construção da Solução - Parte 1 . . . . . p. 43

9 *CaRSP* – Construção da Solução - Parte 2 . . . . . p. 44

10 *CaRSP* – Representação da solução . . . . . p. 60

11 *PAP* – Inserção de cidade inicial como cidade final . . . . . p. 60

12 Busca Local – *LKH* Passageiros . . . . . p. 77

13 Busca Local – 2-*swap* Custos & Devoluções . . . . . p. 84

14 *CaRSP* – *Path Relinking* . . . . . p. 85

15 *CaRSP* – *Path Relinking* – Reparação de Carros Inviáveis . . . . . p. 86

16 Algoritmo de Colônia de Formigas – Informação Heurística de Passageiros p. 90

17 Algoritmo de Colônia de Formigas – Insere Trecho deElite . . . . . p. 91

Lista de tabelas

1 *CaRSP* – Informações referentes aos passageiros . . . . . p. 43

2 Fatores multiplicativos dos elementos das matrizes de custo e de retorno p. 93

3 Parâmetros utilizados pelo *irace* para o ajuste de parâmetros . . . . . p. 95

4 Parâmetros selecionados pelo *irace* para os algoritmos propostos . . . . p. 95

5 Resumo dos resultados obtidos pelos algoritmos exatos . . . . . p. 97

6 Resumo dos resultados obtidos pelos algoritmos heurísticos . . . . . p. 99

7 Resultados obtidos para o *PAP* . . . . . p. 111

8 Resultados obtidos pelo $DFJ1$, $MTZ1$ e $GG1$ . . . . . p. 114

9 Resultados obtidos pelo $DFJ2$, $MTZ2$ e $GG2$ . . . . . p. 115

10 Resultados obtidos pelas heurísticas ingênuas . . . . . p. 118

11 Resultados obtidos pelas meta-heurísticas . . . . . p. 120

12 Dados sobre os procedimentos utilizados no Algoritmo Memético . . . . p. 124

13 Dados sobre os procedimentos utilizados no *GRASP* . . . . . p. 125

14 Dados sobre os procedimentos utilizados no *VNS* . . . . . p. 125

15 Dados sobre os procedimentos utilizados no Algoritmo de Colônia de Formigas . . . . . p. 125

16 *p-valores* para instâncias Euclidianas em relação às soluções . . . . . p. 126

17 *p-valores* para instâncias Euclidianas em relação ao tempo de execução p. 126

18 *p-valores* para instâncias não-Euclidianas em relação às soluções . . . . p. 126

19 *p-valores* para instâncias não-Euclidianas em relação ao tempo de execuçãop. 127

20 *p-valores* para instâncias simétricas em relação às soluções . . . . . p. 127

21 *p-valores* para instâncias simétricas em relação ao tempo de execução . p. 127

22 *p-valores* para instâncias assimétricas em relação às soluções . . . . . p. 127

23 *p-valores* para instâncias assimétricas em relação ao tempo de execução p. 128

Lista de abreviaturas e siglas

PCV – Problema do Caixeiro Viajante

CaRS – Traveling Car Renter Problem (Problema do Caixeiro Alugador)

PCV-P – Problema do Caixeiro Viajante com Passageiros

ABLA – Associação Brasileira das Locadoras de Automóveis

CaRSP – Traveling Car Renter with Passengers

PAP – Problema de Atribuição de Passageiros

GRASP – Greed Randomized Search Procedure

PCV-MCa – Problema do Caixeiro Viajante com Múltiplas Caronas

PCVPQ – Problema do Caixeiro Viajante com Passageiros e Quota

PCV-PL – Problema do Caixeiro Viajante com Passageiros e Lotação

VNS – Variable Neighborhood Search

PCVP-BoTc – Problema do Caixeiro Viajante com Coleta Opcional de Bônus, Tempo de Coleta e Passageiros

VND – Variable Neighborhood Descent

AC – Algoritmos Científicos

pCaRS – Prize Collecting Traveling Car Renter Problem

ALSP – Adaptive Local Search Procedure

IALSP – Iterated Adaptive Local Search Procedure

DFJ – Dantzig-Fulkerson-Johnson

MTZ – Miller-Tucker-Zemlin

GG – Gavish e Graves

LRC – Lista Restrita de Candidatos

ACS – Ant Colony System

Lista de símbolos

G – Grafo ponderado

N – Conjunto de vértices

M – Conjunto de arestas

n – Cardinalidade do conjunto de vértices, N

m – Cardinalidade do conjunto de arestas, M

L – Conjunto de passageiros em potencial

k – Limite de passageiros do veículo

d_{ij} – Custo operacional de se deslocar do vértice i para o vértice j

org(l) – Cidade de origem do passageiro l

dst(l) – Cidade de destino do passageiro l

bud(l) – Limite financeiro do passageiro l

f_{ij} – Variável binária que indica se o veículo atravessa a aresta (i, j) de i para j

v_{lij} – Variável binária que indica se passageiro l está embarcado no veículo durante a travessia da aresta (i, j) de i para j

u_i – Ordem em que o vértice i é visitado no *tour*

C – Conjunto de veículos disponíveis

d_{ij}^c – Custo operacional do carro c de se deslocar do vértice i para o vértice j

γ_{ij}^c – Taxa de retorno do carro c do vértice i para o vértice j

f_{ij}^c – Variável binária que indica se carro c atravessa a aresta (i, j) de i para j

ω_{ij}^c – Variável binária que indica se carro c é alugado no vértice j e devolvido no vértice i

y_i^c – Variável binária que indica se carro c é alugado no vértice i

z_i^c – Variável binária que indica se carro c é devolvido no vértice i

k^c – Limite de passageiros do carro c

v^c_{lij} – Variável binária que indica se passageiro l está no carro c atravessando a aresta (i, j) de i para j

h^c_i – Variável binária que indica se o caixeiro chega na cidade i usando o carro c

b^c_i – Variável binária que indica se o caixeiro deixa a cidade i usando o carro c

$u_{i,j}$ – Inteiros não negativos arbitrários

α^c_{ij} – Variável real cujo valor é inversamente proporcional ao número de passageiros no carro c atravessando a aresta (i,j) de i para j

g^c_{lij} – Variável real fruto da linearização da multiplicação entre α^c_{ij} e v^c_{lij}

H – Conjunto de assentos de um veículo

x^c_h – Variável binária que indica se o assento h do carro c está ocupado ou não

x^{ch}_{ij} – Variável binária que indica se o assento h do carro c está ocupado enquanto atravessa a aresta (i, j) de i para j

q_{li} – Variável real não-negativa que indica o custo, para o passageiro l, de visitar a cidade i no percurso entre sua origem e seu destino

L^* – Lista de pessoas aptas a embarcar após a etapa de pré-processamento

c_i – Custo de usar o carro $sol.Carros[i]$ para viajar de $sol.Cidades[i]$ para $sol.Cidades[i+1]$

$k^{sol.Carros[i]}$ – Limite de passageiros do carro $sol.Carros[i]$

iorg(l) – Índice da cidade de origem do passageiro l em *sol.Cidades*

idst(l) – Índice da cidade de destino do passageiro l em *sol.Cidades*

e_l – Variável binária que indica se o passageiro l embarcou

g_{ih} – Variável binária que indica se o assento h está ocupado na i-ésima cidade visitada de *sol.Cidades*

sol_c – Lista com $|C|$ soluções geradas pela heurística *NAIVE2*

α – *GRASP* : fator que define o quão aleatório ou guloso será o procedimento

Δ – Diferença simétrica entre duas soluções

p^k_{ij} – Probabilidade da formiga k, que está no vértice i, de escolher j como seu próximo vértice

τ_{ij} – Quantidade de feromônio associado à aresta (i, j)

η_{ij} – Valor heurístico que representa a atratividade da formiga visitar o vértice j após visitar o vértice i

α – Determina a influência do feromônio

β – Determina a influência da informação heurística

N_i^k – Conjunto de vértices que ainda não foram visitados pela formiga k

ρ – Taxa de evaporação do feromônio

$\Delta\tau_{ij}^k$ – Quantidade de feromônio depositada pela formiga k na trilha entre i e j

Q – Constante utilizada no depósito de feromônio

L_k – Comprimento total do *tour* da k-ésima formiga

S_k – *Tour* construído pela formiga k

p_{ij}^{ck} – Probabilidade da formiga k, que está no vértice i, de escolher j como seu próximo vértice utilizando o carro c

τ_{ij}^c – Quantidade de feromônio associado à aresta (i, j) utilizando o carro c

η_{ij}^c – Valor heurístico que representa a atratividade da formiga visitar o vértice j após visitar o vértice i utilizando o carro c

σ_{ij} – Valor heurístico que representa a demanda de passageiros indo de i para j

γ – Determina a influência da informação heurística dos passageiros

δ_{ij} – Quantidade de passageiros cuja origem e destino são, respectivamente, as cidades i e j

T^{bs} – Melhor solução corrente

C^{bs} – Custo da melhor solução corrente

$\Delta\tau_{ij}^{bs}$ – Corresponde ao inverso do custo da melhor solução corrente

λ – número real distribuído uniformemente entre $[1, 1.5]$

Lista de algoritmos

1 *PAP* – Etapa de Pré-Processamento . . . . . p. 61

2 *PAP* – Heurística . . . . . p. 64

3 Algoritmo Memético para o *CaRSP* . . . . . p. 67

4 Geração da população inicial . . . . . p. 68

5 *CaRSP* – Reprodução . . . . . p. 70

6 Mutação – Remove Carro . . . . . p. 71

7 Mutação – Adiciona Carro . . . . . p. 72

8 Mutação – Troca Interna de Carros . . . . . p. 73

9 Mutação – Troca Externa de Carros . . . . . p. 74

10 Mutação – Altera Aluguel/Devolução . . . . . p. 75

11 *GRASP* Genérico . . . . . p. 79

12 *VNS* Genérico . . . . . p. 80

13 Algoritmo *GRASP* para o *CaRSP* . . . . . p. 81

14 Busca Local – Avalia Carros . . . . . p. 83

15 Algoritmo de Colônia de Formigas Genérico . . . . . p. 86

16 Algoritmo de Colônia de Formigas para o *CaRSP* . . . . . p. 88

Sumário

1 Introdução p. 20

1.1 Objetivos . . . . . p. 22

1.2 Metodologia da Pesquisa . . . . . p. 23

1.3 Organização do Trabalho . . . . . p. 24

1.4 Contribuições da Pesquisa . . . . . p. 25

2 Trabalhos Relacionados p. 27

2.1 Problema do Caixeiro Viajante com Passageiros . . . . . p. 27

2.1.1 Descrição do Problema . . . . . p. 27

2.1.2 Variações do *PCV-P* . . . . . p. 29

2.1.3 Estado da Arte . . . . . p. 30

2.1.4 Modelagem Matemática . . . . . p. 31

2.2 Problema do Caixeiro Viajante Alugador . . . . . p. 33

2.2.1 Descrição do Problema . . . . . p. 33

2.2.2 Variações do *CaRS* . . . . . p. 35

2.2.3 Estado da Arte . . . . . p. 36

2.2.4 Modelagem Matemática . . . . . p. 37

2.2.5 Correções das Modelagens . . . . . p. 39

3 Problema do Caixeiro Viajante Alugador com Passageiros p. 41

3.1 Descrição do Problema . . . . . p. 41

3.1.1 Exemplo de Solução . . . . . p. 42

3.2 Formulações Matemáticas . . . . . p. 44
3.2.1 Modelo baseado em *DFJ* . . . . . p. 44
3.2.2 Modelo baseado em *MTZ* . . . . . p. 47
3.2.3 Modelo baseado em *GG* . . . . . p. 50
3.3 Linearização . . . . . p. 52
3.3.1 Eliminação das Divisões (1º método) . . . . . p. 52
3.3.2 Eliminação das Divisões (2º método) . . . . . p. 54
3.3.3 Eliminação das Multiplicações . . . . . p. 56
3.4 Modelos Linearizados . . . . . p. 58

4 Algoritmos Propostos p. 59
4.1 Representação da Solução . . . . . p. 59
4.2 Problema de Atribuição de Passageiros . . . . . p. 60
4.2.1 Formulação Exata . . . . . p. 60
4.2.2 Heurística . . . . . p. 63
4.3 Heurísticas Ingênuas . . . . . p. 64
4.3.1 *NAIVE1* . . . . . p. 65
4.3.2 *NAIVE2* . . . . . p. 65
4.3.3 *NAIVE3* . . . . . p. 66
4.4 Algoritmo Memético . . . . . p. 66
4.4.1 Algoritmo Memético Proposto . . . . . p. 66
4.4.1.1 População Inicial . . . . . p. 68
4.4.1.2 Reprodução . . . . . p. 69
4.4.1.3 Mutação . . . . . p. 70
4.4.1.3.1 Remove Carro . . . . . p. 70
4.4.1.3.2 Adiciona Carro . . . . . p. 71
4.4.1.3.3 Troca Interna de Carros . . . . . p. 73

4.4.1.3.4 Troca Externa de Carros . . . . . p. 74
4.4.1.3.5 Altera Aluguel/Devolução . . . . . p. 74
4.4.1.4 Busca Local . . . . . p. 76
4.4.1.4.1 Passageiros & Carros . . . . . p. 76
4.4.1.4.2 Passageiros & Cidades . . . . . p. 76
4.4.1.4.3 *LKH* Passageiros . . . . . p. 76
4.4.1.4.4 2-*opt* . . . . . p. 77
4.4.1.4.5 Altera Cidades sem Demandas . . . . . p. 77
4.4.1.4.6 Altera Passageiros . . . . . p. 78
4.5 *GRASP + VNS + Path Relinking* . . . . . p. 78
4.5.1 *GRASP* . . . . . p. 78
4.5.2 *Variable Neighborhood Search* (*VNS*) . . . . . p. 79
4.5.3 *Path Relinking* . . . . . p. 80
4.5.4 *GRASP + VNS + Path Relinking* Proposto . . . . . p. 81
4.5.4.1 Procedimento *geraSolucao()* . . . . . p. 81
4.5.4.2 Procedimento *VNS()* . . . . . p. 82
4.5.4.2.1 Troca Aleatória de Cidades . . . . . p. 82
4.5.4.2.2 Avalia Carros . . . . . p. 83
4.5.4.2.3 2-*swap* Custos & Devoluções . . . . . p. 84
4.5.4.3 Procedimento *PathRelinking()* . . . . . p. 85
4.6 Algoritmo de Colônia de Formigas . . . . . p. 86
4.6.1 Algoritmo de Colônia de Formigas Proposto . . . . . p. 88
4.6.1.1 Procedimento *iniciaFeromonio()* . . . . . p. 89
4.6.1.2 Procedimento *geraFormigas()* . . . . . p. 89
4.6.1.3 Procedimento *buscaLocal()* . . . . . p. 90
4.6.1.4 Procedimento *atualizaFeromonio()* . . . . . p. 91

5 Experimentos Computacionais p. 92

5.1 Banco de Instâncias . . . . . p. 92

5.2 Metodologia dos Experimentos . . . . . p. 94

5.3 Comparação entre os *PAPs* exato e heurístico . . . . . p. 96

5.4 Algoritmos Exatos . . . . . p. 96

5.5 Algoritmos Heurísticos . . . . . p. 98

5.6 Análise Comparativa de Resultados . . . . . p. 99

5.7 Considerações sobre os Resultados . . . . . p. 100

6 Considerações finais p. 102

6.1 Trabalhos Futuros . . . . . p. 104

Referências p. 105

Apêndice I p. 111

Apêndice II p. 114

Apêndice III p. 118

Apêndice IV p. 124

Apêndice V p. 126

Apêndice VI p. 129

Anexo I p. 131

Anexo II p. 133

Anexo III p. 134

1 Introdução

O Problema do Caixeiro Viajante (PCV) é um dos mais tradicionais e conhecidos problemas de programação matemática (APPLEGATE *et al.*, 2007) . É considerado um problema clássico na área de otimização combinatória e consiste na busca de uma rota, para um caixeiro, em que seja percorrida a menor distância possível, começando em uma cidade qualquer, visitando cada uma das cidades uma única vez e regressando à cidade inicial. Ou seja, dado um grafo ponderado $G = (N, M)$, no qual N representa o conjuto de vértices e M o conjunto de arestas, o objetivo do PCV é encontrar um ciclo Hamiltoniano de custo mínimo. É classificado como um problema NP-Difícil (GAREY; JOHNSON, 1979) e possui inúmeras aplicações práticas em diversas áreas, como sistemas de manufatura, transportes, escalonamento de tarefas, problemas de coletas e entregas de produtos, roteiros de veículos, localizações de facilidades, roteamentos, redes de telecomunicações, dentre outros (GOLDBARG; LUNA, 2005).

Este trabalho foca, especificamente, em duas variantes do PCV recentemente relatadas na literatura, o Problema do Caixeiro Alugador ou *Traveling Car Renter Problem* ($CaRS$) (GOLDBARG; ASCONAVIETA; GOLDBARG, 2012) e o Problema do Caixeiro Viajante com Passageiros (PCV-P) (CALHEIROS, 2017). A definição do problema proposto neste trabalho envolve a combinação de dois conceitos importantes que estão sendo amplamente utilizados no setor de transportes: o aluguel e o compartilhamento de veículos.

Segundo Menezes (2014), o $CaRS$ é uma generalização do PCV onde o cliente pode utilizar carros alugados para visitar um determinado conjunto de cidades, minimizando o custo relacionado ao aluguel de carros e de percurso. Existem várias opções de veículos de empresas diferentes, disponíveis em cada cidade. Esta multiplicidade de opções abre uma variedade de possibilidades de escolha para o motorista sobre os carros mais atraentes para viajar diferentes trechos do percurso pretendido.

Esta variante possui aplicação na área do turismo, onde os clientes costumam começar e terminar sua turnê na mesma cidade. De acordo com a Associação Brasileira das Locadoras de Automóveis (*ABLA*) (2019), o mercado de locação de veículos está em crescente expansão. Entre 2016 e 2018, a quantidade de locadoras de veículos no Brasil cresceu 17,71%, o número de empregos no setor subiu 7,89% e o faturamento bruto aumentou 10,87%, atingindo um total de 15,3 bilhões de reais. De acordo a empresa *Mobility*, no comparativo entre 2019 e 2020, o índice de locação de veículos cresceu 64% na América Latina e 100% no Brasil (BRASILTURIS, 2020). Isto se deve ao crescimento do segmento de terceirização de frotas e uma mudança cultural por parte dos brasileiros em relação ao uso de seu veículo.

A prática de *ridesharing*, atualmente em ampla expansão no mundo (AGATZ *et al.* 2012), também designado *carpooling*, é uma dentre as mais populares estratégias para solução de problemas referentes ao compartilhamento do uso de um veículo entre vários passageiros, tal como das despesas envolvidas (AMEY; ATTANUCCI; MISHALANI, 2011).

O *PCV-P* é uma variante do *PCV* com aspectos de *ridesharing*, ou seja, em que há o compartilhamento dos assentos do veículos ao longo do percurso. Nesta versão é possível embarcar passageiros no veículo do caixeiro e, assim, ratear as despesas de rota entre os ocupantes do carro. É uma versão do *PCV* em que um cliente pretende viajar utilizando um único veículo para visitar um determinado conjunto de cidades, de forma que nos vértices existem passageiros aptos a embarcar para outra cidade do roteiro. Uma vez a bordo, o passageiro só pode deixar o veículo em seu destino. A capacidade do veículo e o limite financeiro dos passageiros devem ser respeitados. Este problema busca minimizar o custo do *tour* através do compartilhamento de despesas da viagem com eventuais caronas.

Esta versão possui aplicação na otimização de sistemas de compartilhamento de viagens, tal como *ridesharing* e *carpool*. De acordo com a Agência Brasil (2018), o setor de transporte contribui com um quarto das emissões globais de gases de efeito estufa e é a área em que as irradiações de carbono mais crescem. Entre os modais que mais contribuem com as emissões de dióxido de carbono, os carros leves lideram com 45% do volume emitido, o que pode provocar sérios efeitos negativos à saúde. As viagens com um único ocupante, que em 2019 corresponderam à aproximadamente 76,4% da população que dirige para o trabalho (CITYLAB, 2019), combinadas com o alto número de veículos nas estradas, aumentam o congestionamento, a emissão de gases, o consumo de combustível e o estresse entre as pessoas (ZHANG; ZHANG, 2018). Isto pode ser amenizado pelo

uso eficiente de veículos através da prática de *ridesharing*, que está sendo proposta neste trabalho.

Iniciativas como o aluguel e o compartilhamento de veículos tem o potencial de aumentar a taxa de ocupação dos automóveis em estradas ou cidades e minimizar custos a partir da divisão de despesas. Além disso, também podem reduzir a quantidade de veículos transitando e, consequentemente, os impactos ambientais causados pela poluição, os engarrafamentos e áreas ocupadas por carros estacionados.

Neste trabalho é apresentado o Problema do Caixeiro Viajante Alugador com Passageiros ou *Traveling Car Renter with Passengers* (*CaRSP*), uma nova variante do *PCV* que une as características e restrições do *CaRS* e do *PCV-P*, ambos classificados como problemas NP-Difíceis (MENEZES, 2014; CALHEIROS, 2015), incorporando aspectos referentes ao aluguel e ao compartilhamento de veículos.

O *CaRSP* pode ser descrito como uma variação do *CaRS* onde o caixeiro vai reduzir suas despesas combinando o uso de diferentes carros alugados e otimizando a ocupação dos assentos dos veículos. O caixeiro pode trocar de veículo sempre que julgar necessário, contanto que o novo veículo comporte todos os passageiros que já estavam a bordo. Cada passageiro em potencial possui uma cidade de origem, uma cidade de destino e uma restrição orçamentária para o custo de sua viagem. Uma vez a bordo, o passageiro só pode deixar o veículo em seu destino. Este problema permite que cada cidade tenha mais de um passageiro. Os carros disponíveis podem ter diferentes capacidades. O objetivo do caixeiro continua sendo viajar com o menor custo possível. Como uma herança dos problemas dos quais é derivado, as seguintes restrições devem ser respeitadas: a capacidade dos veículos, a taxa de retorno de cada carro trocado e todos passageiros a bordo devem ser entregues em seus destinos considerando seus limites financeiros. O problema é composto por várias decisões interligadas: a sequência das cidades visitadas, a ordem dos carros utilizados, as cidades onde os automóveis devem ser alugados/devolvidos, bem como o esquema de embarques e desembarques dos passageiros ao longo do percurso.

1.1 Objetivos

O principal objetivo da pesquisa é apresentar e examinar um novo problema de roteamento de veículos não descrito na literatura, denonimado Problema do Caixeiro Viajante Alugador com Passageiros (*CaRSP*). O problema é definido através de formulações matemáticas e, visando oferecer ao estado da arte algoritmos competitivos, o trabalho

desenvolve um estudo algorítmico para a solução do novo problema.

Este estudo visa definir o problema, elaborar formulações matemáticas, criar um banco de instâncias, implementar heurísticas ingênuas e meta-heurísticas, assim como analisar o desempenho computacional dos algoritmos propostos.

Com o objetivo de obter as soluções iniciais para o problema, será criado um banco de instâncias adaptado do *CaRS* e, além disso, as formulações matemáticas serão devidamente linearizadas e implementadas em um *solver*, definindo os primeiros limites inferiores para o problema.

Como parte da pesquisa, serão desenvolvidos algoritmos heurísticos adaptados ao problema proposto, bem como será feita a análise estatística de resultados e desempenho computacional entre os mesmos. Também serão implementados três modelos matemáticos, três heurísticas ingênuas e três meta-heurísticas para solucionar o problema.

Para atingir estes objetivos, foi definida uma sequência de atividades específicas: (1) Estudar e analisar o estado da arte de problemas relacionados; (2) Definir o problema e propor formulações matemáticas; (3) Linearizar os modelos propostos; (4) Desenvolver um banco de instâncias; (5) Analisar o espaço de soluções do problema; (6) Implementar as formulações matemáticas linearizadas em um *solver*; (7) Implementar heuríticas ingênuas; (8) Implementar meta-heurísticas; (9) Realizar experimentos computacionais com os algoritmos apresentados, efetuando análise dos resultados e testes de desempenho.

1.2 Metodologia da Pesquisa

A primeira etapa da pesquisa foi relacionada ao estudo e definição de formulações para os problemas *CaRS* e *PCV-P*, servindo como base fundamental para a investigação contida neste trabalho. Para ambos os problemas, foram implementados e analisados os modelos propostos na literatura, permitindo encontrar pequenas inconsistências e propor correções, conforme é apresentado na seção 2.2.5 e no Anexo II, que são incorporadas aos modelos propostos para o *CaRSP*.

A etapa seguinte englobou a criação de um banco de instâncias para o problema. Nesta etapa, utilizou-se como base o banco de instâncias do *CaRS* (GOLDBARG; ASCONAVIETA; GOLDBARG, 2012) adicionando características como: a capacidade de cada veículo, a origem, o destino e a restrição financeira de cada passageiro. Foram desenvolvidas 90 instâncias para o *CaRSP*, divididas em Euclidianas, não-Euclidianas, simétricas e assi-

métricas. Estas instâncias possuem de 4 a 200 cidades, de 2 a 5 carros e de 10 a 727 passageiros em potencial.

Em seguida, foram formuladas três modelagens matemáticas não-lineares para o *CaRSP* e propostas técnicas de linearização. Estes modelos foram implementados em um *solver* e aplicados na resolução de 54 instâncias. Esta etapa foi fundamental para analisar o desempenho das formulações propostas e definir limites inferiores.

Para proporcionar uma melhor análise do problema, também foram implementados três algoritmos que utilizam diferentes abordagens ingênuas para solucionar o problema proposto. Estas estratégias se baseiam na ideia de tentar resolver separadamente os subproblemas que compõem o *CaRSP*. A partir do tempo de execução e da qualidade das soluções obtidas é possível analisar se as tomadas de decisão estão interligadas. Estes procedimentos foram aplicados na resolução de 90 instâncias.

Além disso, foram implementados três algoritmos meta-heurísticos na busca por soluções do problema, efetuando as devidas adaptações em termos de estruturas de representação das soluções, operadores de busca local e a definição de novas estratégias adaptadas ao problema, de forma a conseguir algoritmos competitivos na busca de soluções para o *CaRSP*. Estas técnica foram aplicadas na resolução de 90 instâncias.

Com os resultados obtidos nos experimentos, através da análise de desempenho dos algoritmos propostos e por meio de comparações estatísticas entre as soluções encontradas pelos diversos algoritmos propostos, é possível validar as estratégias mais adaptadas ao problema e observar a dificuldade em solucioná-lo.

1.3 Organização do Trabalho

Este trabalho está dividido em 6 capítulos, de forma que o Capítulo 2 apresenta detalhes mais aprofundados sobre os problemas dos quais o *CaRSP* é derivado. A definição do Problema do Caixeiro Viajante Alugador com Passageiros, tal como suas formulações matemáticas e linearizações são mostradas no Capítulo 3. Também são propostas uma formulação exata e uma heurística para o Problema de Atribuição de Passageiros, tal como implementações de três heurísticas ingênuas e três meta-heurísticas adaptadas ao problema, conforme visto no Capítulo 4. A metodologia utilizada na etapa experimental, tal como a geração do banco de instâncias, além de todos os resultados exatos e heurísticos obtidos através dos experimentos computacionais são apresentados no Capítulo 5, possiblitando análise comparativa de desempenho tanto em função do tempo de exe-

cução quanto em função da qualidade das soluções obtidas pelos algoritmos utilizados. Finalmente, apresentamos as conclusões e considerações acerca do trabalho realizado no Capítulo 6.

1.4 Contribuições da Pesquisa

As principais contribuições da pesquisa são resumidas nos tópicos a seguir:

- Apresenta a definição do *CaRSP*, um novo problema na literatura;
- Cria um grupo de 90 instâncias para o problema, derivado do *CaRSlib*;
- Desenvolve os primeiros modelos matemáticos não-lineares;
- Apresenta técnicas para linearizar as modelagens do problema;
- Valida todas as formulações matemáticas apresentadas;
- Detecta e propõe correções à inconsistências encontradas em formulações do *CaRS* relatadas na literatura;
- Reescreve de maneira inteligível a modelagem referente ao embarque de passageiros;
- Detecta e propõe correções à inconsistências encontradas em formulações para o Problema de Atribuição de Passageiros (*PAP*) relatadas na literatura;
- Implementa três técnicas ingênunas para solucionar o *CaRSP*;
- Desenvolve três abordagens meta-heurísticas para o problema proposto;
- Realiza experimentos computacionais considerando os diversos algoritmos propostos, contribuindo para a definição de algoritmos competitivos para o *CaRSP*;
- Apresenta um artigo intitulado "Problema do Caixeiro Viajante Alugador com Passageiros: Uma Abordagem Algorítmica" publicado no congresso *XVIII CLAIO, the Latin-Iberoamerican Conference on Operations Research*;
- Apresenta um artigo intitulado "Uma análise sobre o Problema do Caixeiro Viajante Alugador com Passageiros e seus subproblemas" publicado no periódico *Brazilian Journal of Development*;

- Apresenta um artigo intitulado "*Evolutionary algorithms for the Traveling Car Renter with Passengers*" aceito para publicação no congresso *2020 IEEE Congress on Evolutionary Computation*;
- Apresenta um artigo intitulado "*Models and Linearizations for the Traveling Car Renter with Passengers*" aceito para publicação no periódico *Optimization Letters*;
- Apresenta um artigo intitulado "*Memetic algorithms for the Traveling Salesman with Passengers*" submetido ao congresso *9th Brazilian Conference on Intelligent Systems*.

2 Trabalhos Relacionados

Neste capítulo serão apresentados os problemas dos quais o *CaRSP* se deriva, são eles: o Problema do Caixeiro Viajante com Passageiros (*PCV-P*) e o Problema do Caixeiro Viajante Alugador (*CaRS*), descritos, respectivamente, nas seções 2.1 e 2.2. Este capítulo tem por finalidade fazer uma revisão dos trabalhos relacionados ao Problema do Caixeiro Viajante Alugador com Passageiros, servindo como fundamentação teórica para o desenvolvimento desta pesquisa.

2.1 Problema do Caixeiro Viajante com Passageiros

Nesta seção são apresentados detalhes sobre o *PCV-P*, tais como sua descrição e exemplos (seção 2.1.1), variações do problema (seção 2.1.2), estado da arte (seção 2.1.3) e modelagem matemática (seção 2.1.4).

2.1.1 Descrição do Problema

De maneira geral, o *PCV-P* é uma variante do Problema do Caixeiro Viajante (*PCV*) que foi descrita de forma pioneira por Calheiros (2015). Considera-se um grafo $G = (N, M)$, onde N é o conjunto de nós (cidades), $|N| = n$, e M é um conjunto de arestas (estradas), $|M| = m$. O conjunto de pessoas precisando de caronas, ou seja, passageiros em potencial, é indicado por L. Cada elemento $l \in L$, possui uma origem, um destino e um limite financeiro. Neste problema, o caixeiro deve realizar seu *tour* analisando a demanda de passageiros ao longo das cidades. A origem de um passageiro l corresponde ao vértice em que este deve ser embarcado. Analogamente, o destino de l corresponde ao vértice em que o mesmo deve ser desembarcado. Todos os passageiros devem possuir destinos diferentes de suas origens. Cada trecho da viagem é dividida igualitariamente entre os passageiros ocupantes do carro e o caixeiro. O custo total da viagem de cada passageiro deve respeitar sua restrição orçamentária. Os passageiros que embarcarem no veículo

devem, obrigatoriamente, desembarcar em seus destinos. O veículo utilizado durante o percurso possui uma capacidade k de passageiros que deve ser respeitada. O *tour* começa e termina no vértice 1, que é considerado como uma base para o caixeiro. O objetivo do problema é encontrar o circuito de custo mínimo, representado pelo somatório do custo de cada aresta trafegada dividido pelo número de passageiros a bordo naquele trecho, respeitando as restrições supracitadas. Segundo Calheiros (2015), o subproblema de atribuir passageiros a um caminho pré-definido de maneira ótima é NP-Difícil.

Para um melhor entendimento do problema, um caso específico do *PCV-P* possuindo 4 cidades é mostrado a seguir. A Figura 1 ilustra a capacidade do veículos, a matriz de custos e todas as informações referente aos passageiros, tais como suas origens, destinos e limitações financeiras. As Figuras 2 e 3 ilustram a construção de uma solução para o problema. É possível observar, sequencialmente, as arestas atravessadas, a quantidade de passageiros a bordo em cada trecho e o custo de cada travessia. Quando ocorre um desembarque, as informações do respectivo usuário são riscadas da lista de passageiros e o custo total de sua viagem deve respeitar o seu limite financeiro.

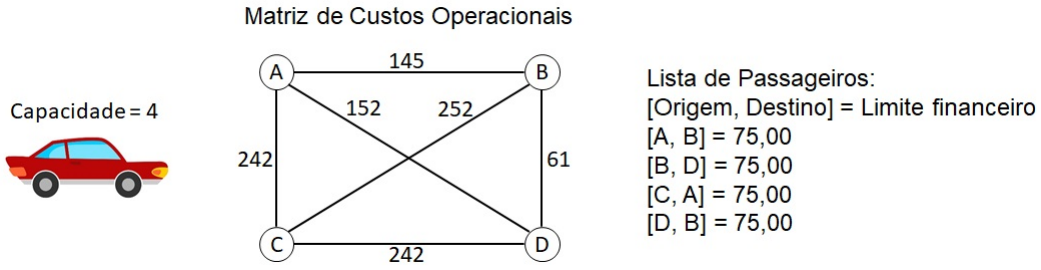

Figura 1: *PCV-P* – Exemplo de instância

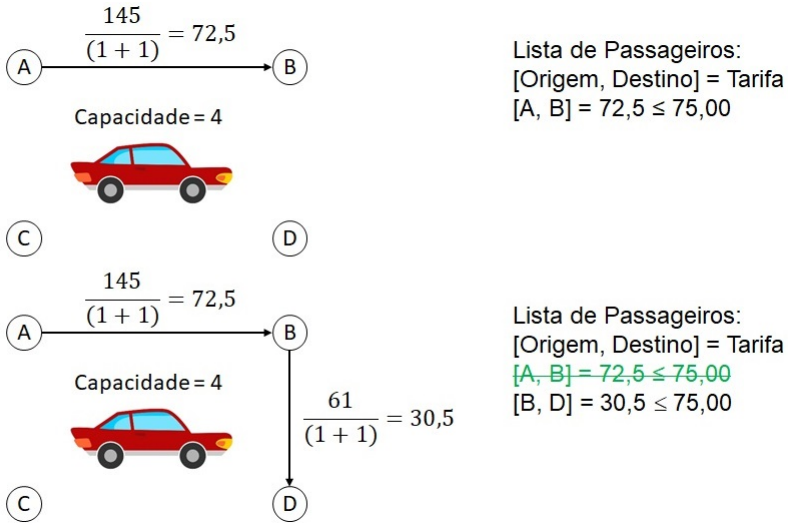

Figura 2: *PCV-P* – Construção da solução - Parte 1

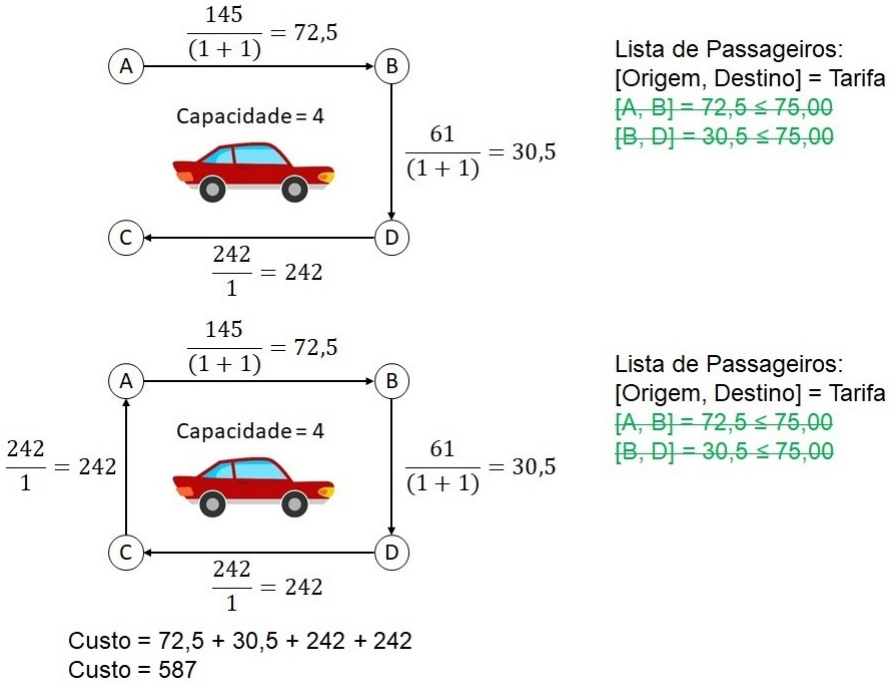

Figura 3: *PCV-P* – Construção da solução - Parte 2

2.1.2 Variações do *PCV-P*

O *PCV-P* admite a possibilidade de algumas situações específicas, podendo ser classificada de acordo com:

1. **Disponibilidade de passageiros (singular × plural)**: Na prática, não existe garantia de que haverão passageiros em todas cidades e muito menos a quantidade. O caso onde se há apenas um passageiro por cidade é denominado singular. Em qualquer outro caso, o problema é classificado como plural.

2. **Limite financeiro (único × múltiplo)**: Em geral, as pessoas pertencem à diferentes classes sociais e, consequentemente, possuem condições financeiras distintas. O caso onde todos os passageiros possuem a mesma resrição orçamentária é denominado de único. Para qualquer outro caso, o problema é definido como múltiplo.

3. **Simetria da distância entre as cidades (simétrico × assimétrico)**: Nos casos em que $d_{ij} = d_{ji}, \forall i, j \in N$, o problema é denominado simétrico. Em caso contrário, é definido como assimétrico.

4. **Existência de ligações no grafo de conexão que modela o problema (completo × incompleto)**: Quando todos os pares de cidades se conectam a partir de

uma aresta, o problema será denominado completo. O problema será considerado incompleto, caso contrário.

5. **Cidades visitadas (total × parcial)**: Casos onde todas as cidades são visitadas uma vez são classificados como total. Já em casos onde é possível que hajam subrotas, são denominados como parcial.

2.1.3 Estado da Arte

A prática de *ridesharing*, também designado *carpooling*, é uma dentre as mais populares estratégias para solução de problemas referentes ao compartilhamento do uso de um veículo entre vários passageiros, tal como das despesas envolvidas (AMEY; ATTANUCCI; MISHALANI, 2011).

Como dito anteriormente, no *PCV-P* o motorista compartilha o uso de um veículo entre vários passageiros, assim como as despesas envolvidas. De acordo com Agatz *et al.* (2012), o uso eficiente dos assentos vazios de um carro a partir da prática do *ridesharing* pode representar uma importante oportunidade de aumentar a taxa de ocupação e pode melhorar, substancialmente, a eficiência dos sistemas de transportes urbanos, reduzindo potencialmente congestionamento de tráfego, consumo de combustível e poluição.

O *PCV-P* foi proposto inicialmente por Calheiros (2015), apresentando o problema e suas definições, tal como uma formulação matemática que não foi validada. As soluções inicias propostas foram a partir do uso das meta-heurísticas Algoritmo Genético e Algoritmo Memético.

Calheiros (2017) apresentou um modelo matemático com função objetivo e restrição com características quadráticas, propôs uma técnica de linearização e validou a partir de experimentos computacionais. Além disso, abordou o problema a partir das meta-heurísticas Algoritmo Genético, Algorimo Memético, Algoritmo *GRASP* (*Greey Randomized Search Procedure*) e Algoritmo de Colônia de Formigas.

Araújo (2016) apresenta em seu trabalho o Problema do Caixeiro Viajante com Múltiplas Caronas (*PCV-MCa*), que difere do *PCV-P* pelo fato de que as cidades possuem mais de um passageiro. O trabalho propõe um modelo matemático com características não-lineares, porém não há experimentos que validem a formulação proposta. Foram desenvolvidas as meta-heurísticas Algoritmo Genético, Algoritmo Memético e Algoritmo de Colônia de Formigas.

Já Silva (2017), aborda o Problema do Caixeiro Viajante com Passageiros e Quota (*PCVPQ*), que inclui no tradicional problema de roteamento com coleta de bônus, o compartilhamento do veículo com passageiros para rateio de eventuais despesas. Além disso, o total de bônus coletado deve atender à uma quota mínima pré-estabelecida. O trabalho propõe e valida uma formulação matemática. Também desenvolveu uma abordagem ingênua e as meta-heurísticas *GRASP*, Algoritmo Genético e Algoritmo Memético.

Em Bastos (2017), é definido o Problema do Caixeiro Viajante com Passageiros e Lotação (*PCV-PL*), onde o caixeiro compartilha os custos da viagem com os passageiros e, além disso, pode se valer, também, de descontos aplicados à utilização de faixas de trânsito especiais que insentam o pagamento de pedágio à veículos que estejam utilizando sua capacidade máxima. O trabalho apresenta e valida formulações matemáticas para o problema. Também são desenvolvida as meta-heurísticas *Simulated Annealing*, Algoritmo *VNS* (*Variable Neighborhood Search*), Algoritmo Colônia de Abelhas e Algoritmo Genético.

Filho (2019) apresenta o Problema do Caixeiro Viajante com Coleta Opcional de Bônus, Tempo de Coleta e Passageiros (*PCVP-BoTc*), uma variante de roteamento com coleta seletiva que mescla elementos de *ridesharing*. O objetivo é otimizar as receitas do motorista que, seletivamente, define quais tarefas de entrega ou coleta serão executadas em sua rota. O efeito econômico da coleta é modelado através de um bônus. O trabalho apresenta e valida quatro formulações matemáticas para o problema. Além disso, também são desenvolvidas as meta-heurísticas *GRASP* com *VND* (*Variable Neighborhood Descent*) e um Algoritmo de Colônia de Formigas. Um experimento computacional baseado em 48 instâncias é descrito.

2.1.4 Modelagem Matemática

Esta modelagem foi proposta por Calheiros (2017) e incorpora algumas restrições relativas ao embarque de passageiros à formulação desenvolvida por Miller-Tucker-Zemlin (1960) para o *PCV*. O problema é formulado em (2.1)–(2.11). Os parâmetros e variáveis são definidos da seguinte forma.

Parâmetros

L: conjunto de índices dos passageiros;

org(l): cidade de origem do passageiro l, $l \in L$;

dst(l): cidade de destino do passageiro l, $l \in L$;

bud(l): limite financeiro do passageiro l, $l \in L$;

k: capacidade do carro;

d_{ij}: custo operacional de se deslocar do vértice i para o vértice j, $(i,j) \in M$.

Variáveis

f_{ij}: variável binária que indica se o veículo atravessa a aresta (i, j) de i para j $(f_{ij} = 1)$ ou não $(f_{ij} = 0)$;

v_{lij}: variável binária que indica se passageiro l está embarcado no veículo durante a travessia da aresta (i, j) de i para j $(v_{lij} = 1)$ ou não $(v_{lij} = 0)$;

u_i: ordem em que o vértice i é visitado no *tour*.

$$\min \sum_{i,j \in N} \frac{d_{ij} f_{ij}}{1 + \sum_{l \in L} v_{lij}} \tag{2.1}$$

sujeito a

$$\sum_{j \in N} f_{ij} = 1, \qquad \forall i \in N \tag{2.2}$$

$$\sum_{i \in N} f_{ij} = 1, \qquad \forall j \in N \tag{2.3}$$

$$u_i - u_j + 1 \leq n\,(1 - f_{ij})\,, \qquad \forall i, j \in N \backslash \{1\} \tag{2.4}$$

$$\sum_{l \in L} v_{lij} \leq k f_{ij}, \qquad \forall i, j \in N \tag{2.5}$$

$$\sum_{i \in N} v_{lir} + \sum_{i \in N} v_{l1i} = 0, \qquad \forall l \in L | r = org(l), r \neq 1 \tag{2.6}$$

$$\sum_{i \in N} v_{lsi} + \sum_{i \in N} v_{li1} = 0, \qquad \forall l \in L | s = dst(l), s \neq 1 \tag{2.7}$$

$$\sum_{j \in N} v_{lij} + \sum_{j \in N} v_{lji} = 0, \qquad \forall l \in L, i \in N | i \neq org(l), i \neq dst(l) \tag{2.8}$$

$$\sum_{i,j \in N} \frac{d_{ij} f_{ij}}{1 + \sum_{l \in L} v_{lij}} \leq bud(l) \qquad \forall l \in L \tag{2.9}$$

$$f_{ij}, v_{lij} \in \{0, 1\}, \qquad \forall i, j \in N, \forall l \in L \tag{2.10}$$

$$u_i \in \mathbb{N}, \qquad i \in N \backslash \{1\} \tag{2.11}$$

A função objetivo (2.1) soma o custo das arestas trafegadas e divide pelo número de pessoas que estão no carro naquele trecho, incluindo o caixeiro. As restrições (2.2) e (2.3) garantem que cada vértice deverão ter somente duas arestas adjacentes, uma de entrada e uma de saída. A restrição (2.4) é a contribuição de Miller-Tucker-Zemlin (1960)

no modelo tradicional do *PCV*, por meio da eliminação de *subtours*. Ou seja, as restrições (2.2)–(2.4) asseguram um ciclo Hamiltoniano para o motorista, representado pela variável f_{ij}. A restrição (2.5) certifica que os passageiros irão percorrer o mesmo caminho que o caixeiro e que a capacidade do veículo não seja excedida. A restrição (2.6) inviabiliza o retorno de qualquer passageiro à sua origem e que passageiros cuja origem não seja a cidade inicial estejam a bordo partindo da cidade 1. A restrição (2.7) assegura que nenhum passageiro irá passar de seu destino e também que passageiros cujo destino não seja a cidade inicial estejam a bordo a caminho da cidade 1. As restrições (2.6) e (2.7) impedem o ciclo de passageiros durante o *tour*, ou seja, garante que no primeiro trecho só poderão estar embarcados passageiros cuja origem seja a cidade inical e que no último trecho só estarão a bordo passageiros cujo destino seja a cidade 1. Para garantir que as arestas ativas dos passageiros formem um caminho, é introduzida a restrição (2.8). Por fim, a restrição (2.9) incorpora os limites financeiros de cada passageiro. As equações (2.10) e (2.11) garantem a integridade e não-negatividade das variáveis de decisão.

2.2 Problema do Caixeiro Viajante Alugador

Nesta seção são apresentados detalhes sobre *CaRS*, tais como sua descrição e exemplos (seção 2.2.1), variações do problema (seção 2.2.2), estado da arte (seção 2.2.3), modelagem matemática (seção 2.2.4) e correções para modelos relatados na literatura (seção 2.2.5).

2.2.1 Descrição do Problema

O *CaRS* é uma variante do *PCV* e foi descrito de forma pioneira por Goldbarg, Asconavieta e Goldbarg (2012), sendo também objeto de estudo do trabalho realizado por Asconavieta (2011). Considera-se um grafo $G = (N, M)$, onde N é o conjunto de nós (cidades), $|N| = n$, e M é um conjunto de arestas (estradas), $|M| = m$. O caixeiro viajante, que é também o motorista, deve viajar com veículos alugados. Um conjunto de carros, C, está disponível para o caixeiro. Os veículos podem ser alugados e devolvidos em qualquer cidade. Custos operacionais específicos são associados a cada carro, incluindo consumo de combustível, pagamento de pedágio e custos referentes ao aluguel. Existe uma taxa extra a ser paga relacionada ao custo da empresa de levar um carro de onde foi entregue de volta à cidade onde foi alugado. Cada carro só pode ser alugado uma vez. O *tour* começa e termina no vértice 1, que é considerado como uma base para o caixeiro. O objetivo é encontrar o ciclo de custo mínimo em G considerando todos os aluguéis e devoluções de

carros durante o trajeto. Dado que o *PCV* é NP-Difícil (GAREY; JOHNSON, 1979), e é um caso especial do *CaRS* quando apenas um carro é usado para realizar o percurso, o problema *CaRS* também pode ser classificado como NP-Difícil (MENEZES, 2014).

Para um melhor entendimento do problema, um caso específico do *CaRS* possuindo 2 carros e 4 cidades é mostrado a seguir. A Figura 4 ilustra as matrizes de custos e as matrizes de retornos de ambos os veículos.

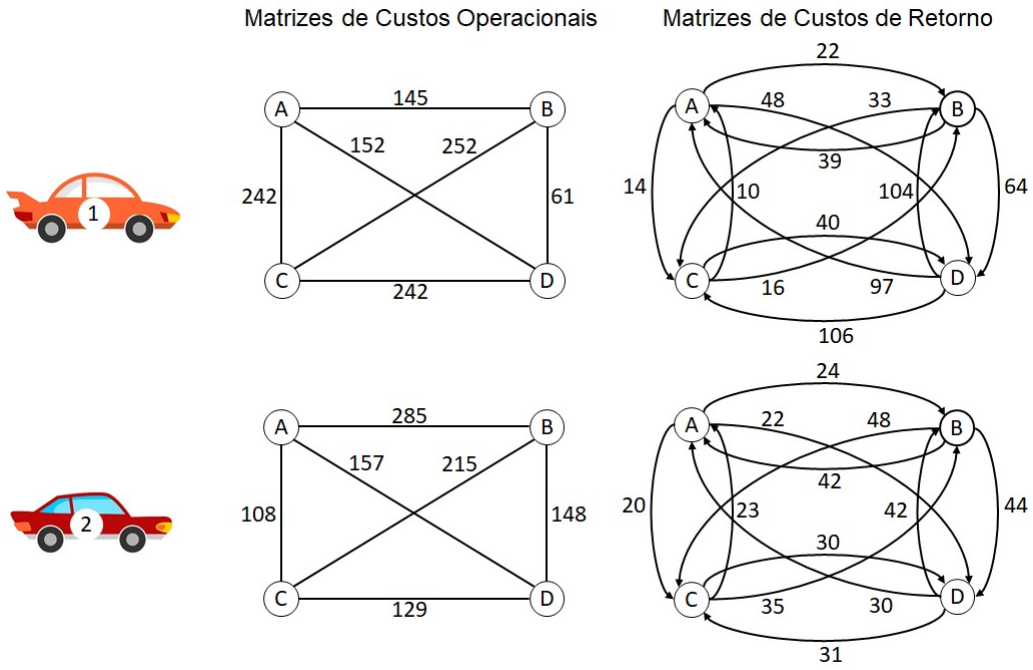

Figura 4: *CaRS* – Exemplo de instância

As Figuras 5 e 6 ilustram o passo a passo da construção de uma solução para o problema. É possível observar, sequencialmente, os veículos que estão sendo utilizados em cada aresta atravessada tal como o custo associado. E, também, são destacados os pontos onde há troca de veículos, ou seja, quando se deve pagar uma taxa para devolver o carro que estava sendo utilizado para a cidade aonde o mesmo foi alugado.

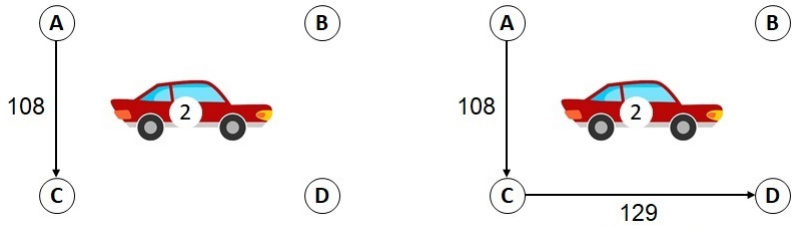

Figura 5: *CaRS* – Construção da solução - Parte 1

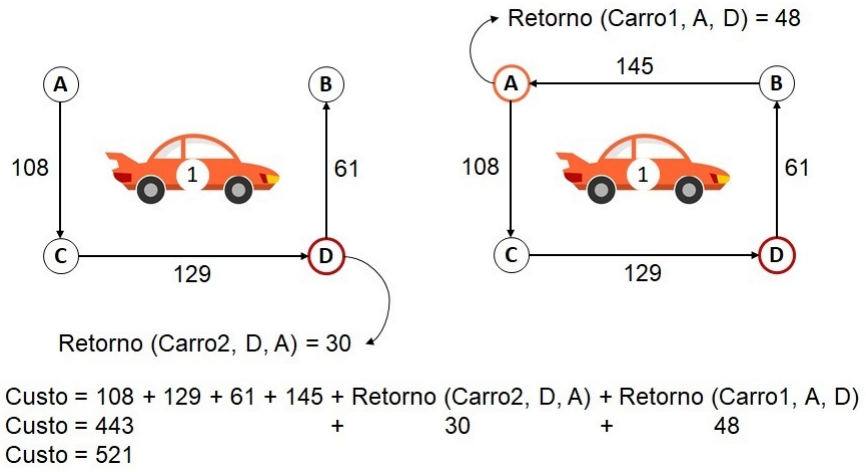

Figura 6: *CaRS* – Construção da solução - Parte 2

2.2.2 Variações do *CaRS*

De acordo com a definição do problema apresentada por Goldbarg, Asconavieta e Goldbarg (2012), o problema admite a possibilidade de algumas situações específicas, podendo ser classificada de acordo com:

1. **Disponibilidade de carros para aluguel (parcial × total)**: Em situações reais, não existe a garantia de que as empresas de locação de veículos possuam filiais em todas as cidades de um *tour*. Sendo assim, não se pode estabelecer de forma rigorosa, que qualquer carro pode ser alugado em qualquer cidade. O caso em que é possível alugar todos os carros em todas as cidades é denominado total. Em qualquer outro caso, o problema é definido como parcial.

2. **Alternativas de devolução do carro alugado (restrito × irrestrito)**: As empresas de locação podem não operar serviços de recepção de veículos em todas as cidades, não garantindo que qualquer carro alugado possa ser devolvido em qualquer cidade. No caso em que a recepção pode ser realizada em todas as cidades, o problema é denominado irrestrito. Caso contrário, é definido como restrito.

3. **Integridade de contrato (sem repetição × com repetição)**: Nos casos em que não seja permitido que o mesmo tipo de carro seja alugado mais de uma vez no *tour* do caixeiro, o problema é classificado como sem repetição. O problema é denominado com repetição em qualquer outro caso.

4. **Cálculo dos custos de devolução do carro alugado (livre × vinculado)**: Os custos de devolução dos carros podem ser constituídos por valores independentes da

topologia ou restrição da rede. Nesse caso, o problema é considerado livre. No caso em que estes custos são calculados levando em conta a rota empregada pelo carro para retornar à sua base, o problema é dito vinculado.

5. **Simetria da distância entre as cidades (simétrico $\times$ assimétrico)**: Nos casos em que $d_{ij}^c = d_{ji}^c$, $\forall i, j \in N, \forall c \in C$, o problema é denominado simétrico. Em caso contrário, é definido como assimétrico.

6. **Existência de ligações no grafo de conexão que modela o problema (completo $\times$ incompleto)**: Quando todos os pares de cidades se conectam a partir de uma aresta, o problema será denominado completo. O problema será tido como incompleto, caso contrário.

2.2.3 Estado da Arte

O *CaRS* foi proposto inicialmente por Asconavieta (2011), apresentando o problema, suas definições e um banco de instâncias. As soluções iniciais propostas foram a partir do uso das meta-heurísticas *GRASP*, Algoritmo *VND*, *GRASP* hibridizado com *VND*, Algoritmo Memético, Algoritmo de Colônia de Formigas e Algoritmo Transgenético. Também foi apresentada uma primeira solução exata para o problema a partir da implementação de um *backtracking*, técnica que implicitamente enumera todas as possíveis configurações utilizando permutações dos carros disponíveis.

Goldbarg, Asconavieta e Goldbarg (2012) apresentam o problema geral do *CaRS*, assim como suas variantes. Além disso, são apresentadas duas abordagens meta-heurísticas: Algoritmo Memético e *GRASP* hibridizado com *VND*. O trabalho relata um experimento computacional em um conjunto de 40 instâncias, divididas em Euclidianas e não-Euclidianas.

Já em Goldbarg *et al.* (2013), é proposta a primeira formulação matemática para o problema, um modelo de programação quadrática inteira. Este modelo é linearizado e validado a partir de experimentos computacionais. Além disso, o trabalho também apresenta a implementação da meta-heurística Algoritmo Transgenético.

Felipe (2014) propõe em seu trabalho uma nova abordagem meta-heurística denominada Algoritmos Científicos (AC). Além disso, apresenta sua aplicação ao *CaRS* e obtém resultados bastante promissores, superando os relatados na literatura, principalmente no que se refere ao tempo de processamento. Um estudo acerca de 20 instãncias não-Euclidianas é relatado.

Menezes (2014) apresenta dois modelos não-lineares para o *CaRS*, dos quais um é proposto pelo autor. As formulações são linearizadas a partir de duas técnicas diferentes, dando origem a quatro modelos linearizados. Todos os modelos são validados a partir de experimentos computacionais. Além disso, o trabalho também apresenta uma variante do *CaRS* denominada *pCaRS* (*Prize Collecting Traveling Car Renter Problem*), onde há um bônus associado à visita de cada cidade. O total de bônus coletado durante o percurso é denominado de satisfação e o problema pré-estabelece satisfações mínimas que devem ser atendidas. O objetivo do problema é selecionar um subconjunto de cidades a serem visitadas, visando a minimização do custo total de viagem, incluindo os aluguéis e devoluções dos veículos e que a satisfação mínima seja respeitada. Um banco de instâncias é proposto para o problema. Uma formulação matemática para o *pCaRS* é apresentada e validada. Além disso, são apresentadas as meta-heurísticas *GRASP* com *VNS* e *Path Relinking*, Algoritmo Memético e Algoritmo Transgenético para solucionar o *pCaRS*.

Temos em Rios, Goldbarg e Quesquén (2017) a proposta de um modelo de programação inteira mista para o problema. O trabalho relata a implementação de uma meta-heurística híbrida composta por Algoritmos Científicos e *ALSP* (*Adaptive Local Search Procedure*). Um estudo de caso com 21 instâncias é relatado.

O trabalho de Goldbarg *et al.* (2017) apresenta três formulações matemáticas para o *CaRS*, onde duas possuem funções objetivo quadráticas e uma possui restrições quadráticas. Os modelos são linearizados e implementados em diferentes *solvers*. Um experimento computacional com 50 instâncias é reportado.

Em Rios (2018), são apresentados procedimentos híbridos que combinam o uso de meta-heurísticas e métodos baseados em Programação Linear. São hibridizados: *AC*, *VND*, *ALSP* e uma variante do *ALSP* chamada *IALSP* (*Iterated Adaptive Local Search Procedure*). As seguintes técnicas são propostas para solucionar o *CaRS*: *AC+ALSP*, *AC+IALSP* e *AC+VND+IALSP*. O trabalho relata experimentos computacionais para 56 instâncias, divididas em Euclidianas e não-Euclidianas.

2.2.4 Modelagem Matemática

Este é um dos modelos propostos por Menezes (2014) e apresentado em Goldbarg *et al.* (2017) baseado na formulação de Dantzig-Fulkerson-Johnson (1954) para o *PCV*. O problema é formulado em (2.12)–(2.22). Os parâmetros e variáveis são definidos da seguinte forma.

Parâmetros

C: conjunto de índices dos carros disponíveis;

d_{ij}^c: custo operacional do carro c de se deslocar do vértice i para o vértice j, $(i,j) \in M$, $c \in C$;

γ_{ij}^c: taxa de retorno do carro c do vértice i para o vértice j, $(i,j) \in M$, $c \in C$.

Variáveis

f_{ij}^c: variável binária que indica se carro c atravessa a aresta (i, j) de i para j ($f_{ij}^c = 1$) ou não ($f_{ij}^c = 0$);

ω_{ij}^c: variável binária que indica se carro c é alugado no vértice j e devolvido no vértice i ($\omega_{ij}^c = 1$) ou não ($\omega_{ij}^c = 0$);

y_i^c: variável binária que indica se carro c é alugado no vértice i ($y_i^c = 1$) ou não ($y_i^c = 0$);

z_i^c: variável binária que indica se carro c é devolvido no vértice i ($z_i^c = 1$) ou não ($z_i^c = 0$);

u_i: ordem em que o vértice i é visitado no *tour*.

$$\min \sum_{c \in C} \sum_{i,j \in N} d_{ij}^c f_{ij}^c + \sum_{c \in C} \sum_{i,j \in N} \gamma_{ij}^c \omega_{ij}^c \tag{2.12}$$

sujeito a

$$\sum_{c \in C} \sum_{j \in N} f_{ij}^c = 1, \qquad \forall i \in N \tag{2.13}$$

$$\sum_{c \in C} \sum_{i \in N} f_{ij}^c = 1, \qquad \forall j \in N \tag{2.14}$$

$$2 \leq u_i \leq n, \qquad \forall i \in N \setminus \{1\} \tag{2.15}$$

$$u_i - u_j + 1 \leq (n-1)\left(1 - \sum_{c \in C} f_{ij}^c\right), \qquad \forall i, j \in N \setminus \{1\} \tag{2.16}$$

$$y_i^c = \left(\sum_{j \in N} f_{ij}^c\right)\left(\sum_{c' \in C, c' \neq c} \sum_{h \in N} f_{hi}^{c'}\right), \qquad \forall c \in C, \forall i \in N \tag{2.17}$$

$$z_i^c = \left(\sum_{j \in N} f_{ji}^c\right)\left(\sum_{c' \in C, c' \neq c} \sum_{h \in N} f_{ih}^{c'}\right), \qquad \forall c \in C, \forall i \in N \tag{2.18}$$

$$\omega_{ij}^c = y_j^c z_i^c, \qquad \forall c \in C, \forall i, j \in N, i \neq j \tag{2.19}$$

$$\sum_{c \in C} y_1^c = 1 \tag{2.20}$$

$$f_{ij}^c, \omega_{ij}^c, y_i^c, z_i^c \in \{0, 1\}, \qquad \forall i, j \in N, \forall c \in C \tag{2.21}$$

$$u_i \in \mathbb{N}, \qquad i \in N \setminus \{1\} \tag{2.22}$$

A função objetivo (2.12) soma o custo de percorrer as arestas do grafo usando carros diferentes com as taxas de retorno para devolver os carros alugados às suas origens. Restrições (2.13) e (2.14) asseguram que cada vértice é visitado apenas uma vez. Restrição (2.13) garante que apenas um carro deixa o vértice i percorrendo uma aresta. Restrição (2.14) garante que apenas um carro percorre uma aresta e chega ao vértice j. Restrições (2.15) e (2.16) previnem *subtours* e correspondem à formulação *DFJ* para o *PCV* (DANTZIG; FULKERSON; JOHNSON, 1954). Restrição (2.15) assegura que o vértice i é o u_i-ésimo vértice visitado. Já que o problema requer que o *tour* se inicie no vértice 1, este vértice foi removido do grupo de nós considerados nas restrições (2.15) e (2.16). Restrição (2.16) relaciona as variáveis f_{ij}^c e u_i. Restrição (2.17) afirma que se carro c é alugado na cidade i, carro c viaja de i para outra cidade j, e algum outro carro, c', é usado para chegar em i vindo de outra cidade h. Semelhante à equação (2.17), a restrição (2.18) relaciona as variáveis f_{ij}^c e z_i^c considerando a devolução do carro c à cidade i. Restrição (2.19) relaciona as variáveis ω_{ij}^c, y_j^c e z_i^c. Restrição (2.20) assegura que um carro é alugado na cidade 1. Restrição (2.21) define as variáveis binárias e a restrição (2.22) define variáveis u_i para o intervalo de inteiros não-negativos.

2.2.5 Correções das Modelagens

A partir da análise das equações (2.17) e (2.20) do modelo apresentado na seção 2.2.4, é possível perceber uma inconsistência. A restrição (2.20) obriga que um carro seja alugado na cidade 1, mas a restrição (2.17) impõe que para que um carro c seja alugado em uma cidade i, o caixeiro deve, obrigatoriamente, ter chegado em i utilizando um carro c', tal que $c \neq c'$. Ou seja, desta maneira o caixeiro não seria capaz de realizar um *tour* utilizando apenas um veículo. Analogamente, a restrição (2.18) também causa a mesma inconsistência no processo de devolução de um veículo. Para adequar o modelo, as restrições (2.17) e (2.18) devem ser substituídas pelas equações (2.23)–(2.26), descritas a seguir.

$$y_i^c = \left(\sum_{j \in N} f_{ij}^c\right)\left(\sum_{c' \in C, c' \neq c}\sum_{h \in N} f_{hi}^{c'}\right), \qquad \forall c \in C, \forall i \in N \backslash \{1\} \quad (2.23)$$

$$\sum_{i \in N} f_{1i}^c = y_1^c, \qquad \forall c \in C \quad (2.24)$$

$$z_i^c = \left(\sum_{j \in N} f_{ji}^c\right)\left(\sum_{c' \in C, c' \neq c}\sum_{h \in N} f_{ih}^{c'}\right), \qquad \forall c \in C, \forall i \in N \backslash \{1\} \quad (2.25)$$

$$\sum_{i \in N} f_{i1}^{c} = z_{1}^{c}, \qquad \forall c \in C \quad (2.26)$$

Restrição (2.23) afirma que se carro c é alugado na cidade i, $i \neq 1$, carro c viaja de i para outra cidade j, e algum outro carro, c', é usado para chegar em i vindo de outra cidade h. Restrição (2.24) assegura que o carro alugado na cidade 1 é utilizado para percorrer a primeira aresta. Semelhante à equação (2.23), a restrição (2.25) relaciona as variáveis f_{ij}^{c} e z_{i}^{c} considerando a devolução do carro c à cidade i, $i \neq 1$. Restrição (2.26) assegura que o carro utilizado na última aresta percorrida é devolvido na cidade 1. O modelo corrigido consiste nas equações (2.12)–(2.16) e (2.19)–(2.26).

Esta mesma inconsistência também aparece nos modelos propostos por Rios, Goldbarg e Quesquén (2017), Rios, Goldbarg e Goldbarg (2017) e Rios (2018) que, por sua vez, são apresentados no Anexo I juntamente com as correções propostas. Outro modelo apresentado por Goldbarg *et al.* (2017) possui uma pequena inconsistência na função objetivo e é corrigido no Anexo II.

3 Problema do Caixeiro Viajante Alugador com Passageiros

Este capítulo define o Problema do Caixeiro Viajante Alugador com Passageiros (*CaRSP*). A descrição e exemplos de solução são apresentados na seção 3.1. A seção 3.2 descreve três formulações matemáticas não-lineares para o problema. Já a seção 3.3 propõe técnicas para linearizar os modelos propostos. Por fim, os modelos linearizados são apresentados na seção 3.4.

3.1 Descrição do Problema

Considera-se um grafo $G = (N, M)$, onde N é o conjunto de nós (cidades), $|N| = n$, e M é um conjunto de arestas (estradas), $|M| = m$. O caixeiro viajante, que é também o motorista, deve viajar com veículos alugados. Um conjunto de carros, C, está disponível para o caixeiro. Os veículos podem ser alugados e devolvidos em qualquer cidade. Custos operacionais específicos são associados a cada carro, incluindo consumo de combustível, pagamento de pedágio e custos referentes ao aluguel. Existe uma taxa extra a ser paga relacionada ao custo da empresa de levar um carro de onde foi entregue de volta à cidade onde foi alugado. O caixeiro pode trocar de veículo sempre que julgar necessário, contanto que o novo veículo comporte todos os passageiros que já estavam a bordo. O conjunto de pessoas precisando de caronas, ou seja, passageiros em potencial, é indicado por L. Um pedido de carona é associado a cada passageiro em potencial e consiste nas cidades de origem, destino e na tarifa máxima que ele(a) concorda em pagar por uma viagem. Uma vez a bordo, o passageiro só pode deixar o veículo em seu destino. O motorista compartilha o custo para atravessar uma aresta de G em um veículo específico com os passageiros no carro, ou seja, o custo é dividido igualmente entre os ocupantes do carro naquele trecho. O caixeiro é responsável por pagar o valor integral referente à taxa de devolução dos carros utilizados no percurso. Cada veículo tem sua capacidade específica de passageiros e é uma restrição para o número de pessoas transportadas. Cada carro

só pode ser alugado uma vez. O *tour* começa e termina no vértice 1, que é considerado como uma base para o caixeiro. O objetivo é encontrar o ciclo Hamiltoniano de custo mínimo em G considerando o ponto de vista do motorista. Então, de acordo com o que é descrito nas seções 2.1.2 e 2.2.2, trata-se de um problema com as seguintes variações herdadas do *CaRS* e do *PCV-P*: plural, múltiplo, simétrico/assimétrico, completo, total, irrestrito e sem repetição. Dado que o *PCV* é NP-Difícil (GAREY; JOHNSON, 1979), um caso especial do *CaRSP* quando apenas um carro é utilizado no percurso e não existe embarque de passageiros, o problema *CaRSP* também é NP-Difícil.

3.1.1 Exemplo de Solução

Para um melhor entendimento do problema, um caso específico do *CaRSP* possuindo 2 carros, 4 cidades e 10 passageiros é mostrado a seguir. A Figuras 7 ilustra as capacidades, as matrizes de custos e as matrizes de retornos referentes aos carros 1 e 2, respectivamente. Já a Tabela 1 exibe todas as informações referente aos passageiros, tal como a origem, o destino e o seu limite financeiro.

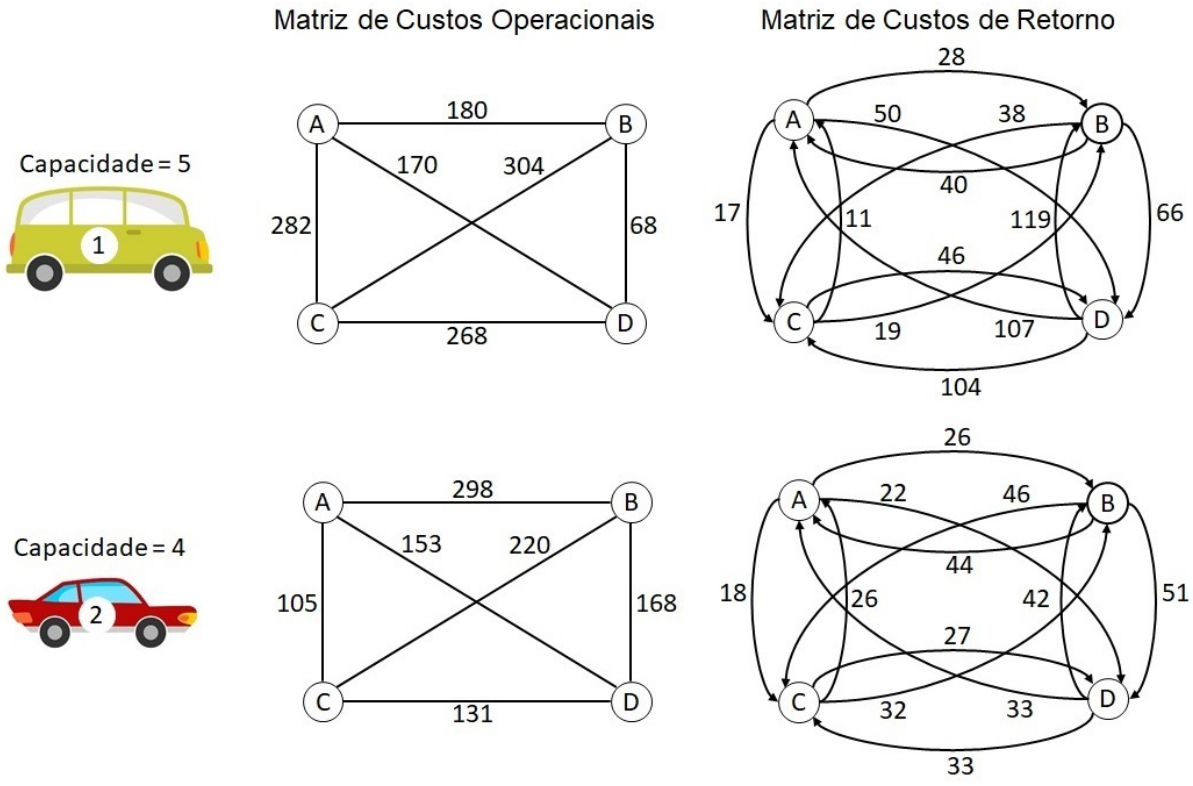

Figura 7: *CaRSP* – Informações referentes aos carros

Tabela 1: *CaRSP* – Informações referentes aos passageiros

Origem	Destino	Limite Financeiro	Origem	Destino	Limite Financeiro
A	D	57,92	B	A	128,98
A	C	42,78	B	C	87,26
B	A	126,44	B	A	127,74
B	D	31,01	B	A	127,88
B	C	75,37	C	D	59,74

As Figuras 8 e 9 ilustram a construção de uma solução para o problema. É possível observar, sequencialmente, os veículos que estão sendo utilizados em cada aresta atravessada, a quantidade de passageiros a bordo em cada trecho e o custo de cada travessia. Quando ocorre um desembarque, as informações do respectivo usuário são riscadas da lista de passageiros e o custo total de sua viagem deve respeitar o seu limite financeiro. Além disso, também, são destacados os pontos onde há troca de veículos, ou seja, quando se deve pagar uma taxa para devolver o carro que estava sendo utilizado para a cidade aonde o mesmo foi alugado.

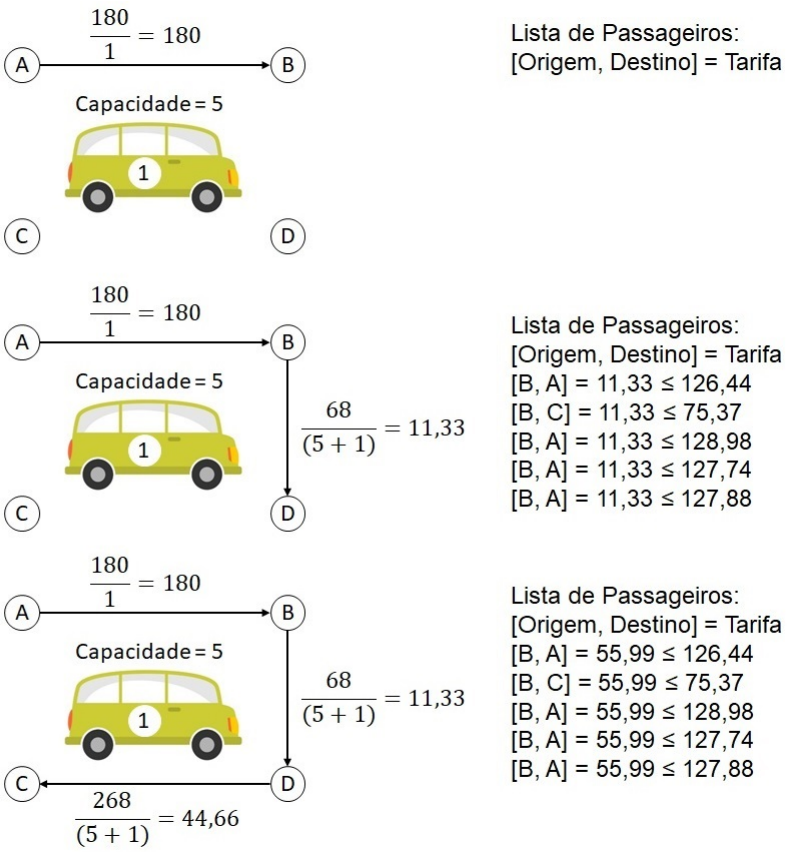

Figura 8: *CaRSP* – Construção da Solução - Parte 1

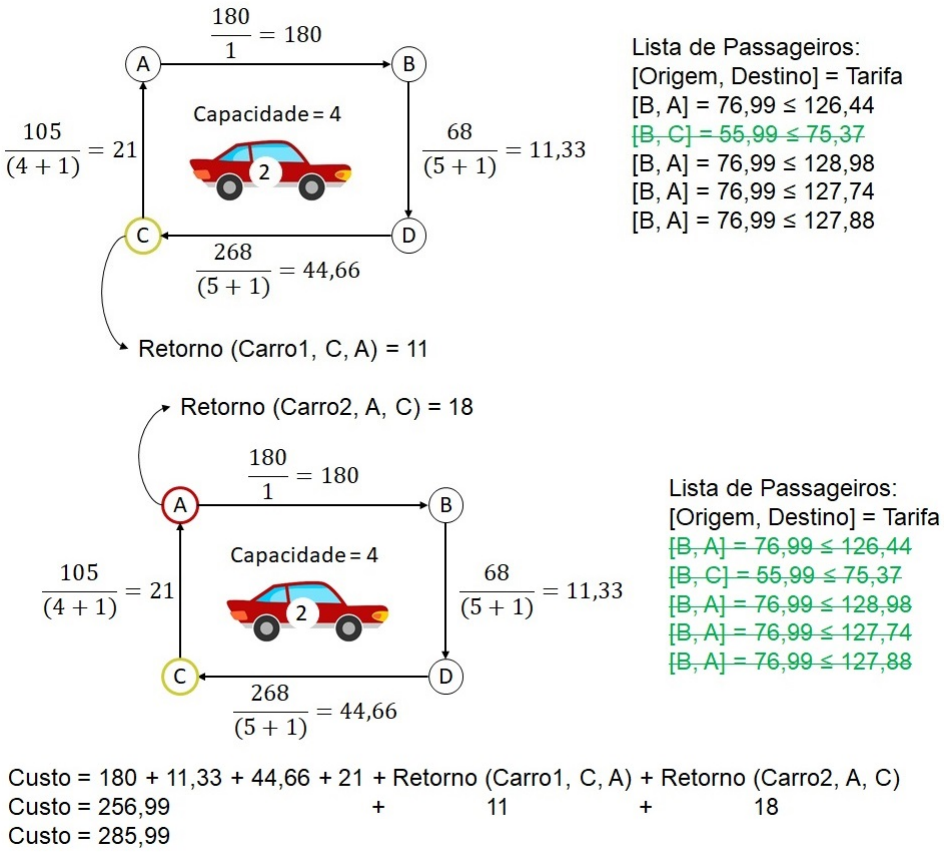

Figura 9: *CaRSP* – Construção da Solução - Parte 2

3.2 Formulações Matemáticas

Este trabalho apresenta três formulações matemáticas para o *CaRSP* baseadas em modelos de fluxo para o *PCV*. O primeiro modelo, apresentado na seção 3.2.1, baseia-se na formulação de Dantzig-Fulkerson-Johnson (*DFJ*) (DANTZIG; FULKERSON; JOHNSON, 1954). O segundo modelo, apresentado na seção 3.2.2, baseia-se na formulação de Miller-Tucker-Zemlin (*MTZ*) (MILLER; TUCKER; ZEMLIN, 1960). Já o terceiro modelo, apresentado na seção 3.2.3, baseia-se na formulação de Gavish e Graves (*GG*) (GAVISH; GRAVES, 1978).

3.2.1 Modelo baseado em *DFJ*

Este modelo é baseado em uma formulação proposta por Goldbarg *et al.* (2017), devidamente corrigida, conforme descrito na seção 2.2.5, com a adição de restrições relacionadas aos passageiros propostas por Calheiros (2017) reescritas de modo mais inteligível. A função objetivo também é modificada. O problema é formulado em (3.1)–(3.20). Os

parâmetros e variáveis são definidos da seguinte forma.

Parâmetros

L: conjunto de índices dos passageiros;

C: conjunto de índices dos carros disponíveis;

org(l): cidade de origem do passageiro l, $l \in L$;

dst(l): cidade de destino do passageiro l, $l \in L$;

bud(l): limite financeiro do passageiro l, $l \in L$;

k^c: capacidade do carro c, $c \in C$;

d_{ij}^c: custo operacional do carro c de se deslocar do vértice i para o vértice j, $(i,j) \in M$, $c \in C$;

γ_{ij}^c: taxa de retorno do carro c do vértice i para o vértice j, $(i,j) \in M$, $c \in C$.

Variáveis

f_{ij}^c: variável binária que indica se carro c atravessa a aresta (i, j) de i para j $(f_{ij}^c = 1)$ ou não $(f_{ij}^c = 0)$;

ω_{ij}^c: variável binária que indica se carro c é alugado no vértice j e devolvido no vértice i $(\omega_{ij}^c = 1)$ ou não $(\omega_{ij}^c = 0)$;

y_i^c: variável binária que indica se carro c é alugado no vértice i $(y_i^c = 1)$ ou não $(y_i^c = 0)$;

z_i^c: variável binária que indica se carro c é devolvido no vértice i $(z_i^c = 1)$ ou não $(z_i^c = 0)$;

v_{lij}^c: variável binária que indica se passageiro l está no carro c atravessando a aresta (i, j) de i para j $(v_{lij}^c = 1)$ ou não $(v_{lij}^c = 0)$;

u_i: ordem em que o vértice i é visitado no *tour*.

$$\min \sum_{c \in C} \sum_{i,j \in N} \frac{d_{ij}^c f_{ij}^c}{1 + \sum_{l \in L} v_{lij}^c} + \sum_{c \in C} \sum_{i,j \in N} \gamma_{ij}^c \omega_{ij}^c \tag{3.1}$$

sujeito a

$$\sum_{c \in C} \sum_{j \in N} f_{ij}^c = 1, \qquad \forall i \in N \tag{3.2}$$

$$\sum_{c \in C} \sum_{i \in N} f_{ij}^c = 1, \qquad \forall j \in N \tag{3.3}$$

$$2 \leq u_i \leq n, \qquad \forall i \in N \setminus \{1\} \tag{3.4}$$

$$u_i - u_j + 1 \leq (n-1)\left(1 - \sum_{c \in C} f_{ij}^c\right), \qquad \forall i, j \in N \setminus \{1\} \tag{3.5}$$

$$y_i^c = \left(\sum_{j \in N} f_{ij}^c\right)\left(\sum_{c' \in C, c' \neq c} \sum_{h \in N} f_{hi}^{c'}\right), \qquad \forall c \in C, \forall i \in N \setminus \{1\} \tag{3.6}$$

$$\sum_{i \in N} f_{1i}^c = y_1^c \qquad \forall c \in C \quad (3.7)$$

$$z_i^c = \left(\sum_{j \in N} f_{ji}^c \right) \left(\sum_{c' \in C, c' \neq c} \sum_{h \in N} f_{ih}^{c'} \right), \qquad \forall c \in C, \forall i \in N \backslash \{1\} \quad (3.8)$$

$$\sum_{i \in N} f_{i1}^c = z_1^c \qquad \forall c \in C \quad (3.9)$$

$$\omega_{ij}^c = y_j^c z_i^c, \qquad \forall c \in C, \forall i, j \in N, i \neq j \quad (3.10)$$

$$\sum_{c \in C} y_1^c = 1 \qquad (3.11)$$

$$\sum_{l \in L} v_{lij}^c \leq k^c f_{ij}^c, \qquad \forall c \in C, \forall i, j \in N \quad (3.12)$$

$$\sum_{c \in C} \sum_{i \in N} v_{li1}^c + \sum_{c \in C} \sum_{i \in N} v_{l1i}^c = 0, \qquad \forall l \in L | org(l) \neq 1,\ dst(l) \neq 1 \quad (3.13)$$

$$\sum_{c \in C} \sum_{i \in N} v_{lri}^c = \sum_{c \in C} \sum_{i \in N} v_{lis}^c, \qquad \forall l \in L | r = org(l),\ s = dst(l) \quad (3.14)$$

$$\sum_{c \in C} \sum_{i \in N} v_{lir}^c + \sum_{c \in C} \sum_{i \in N} v_{lsi}^c = 0, \qquad \forall l \in L | r = org(l),\ s = dst(l) \quad (3.15)$$

$$\sum_{c \in C} \sum_{j \in N} v_{lij}^c = \sum_{c \in C} \sum_{j \in N} v_{lji}^c, \qquad \forall i \in N, \forall l \in L | i \neq org(l), i \neq dst(l) \quad (3.16)$$

$$\sum_{c \in C} \sum_{i,j \in N} \frac{d_{ij}^c f_{ij}^c}{1 + \sum_{p \in L} v_{pij}^c} \leq bud(l), \qquad \forall l \in L \quad (3.17)$$

$$\sum_{i \in N} y_i^c \leq 1, \qquad \forall c \in C \quad (3.18)$$

$$f_{ij}^c, \omega_{ij}^c, y_i^c, z_i^c, v_{lij}^c \in \{0, 1\}, \qquad \forall i, j \in N, \forall c \in C, \forall l \in L \quad (3.19)$$

$$u_i \in \mathbb{N}, \qquad i \in N \backslash \{1\} \quad (3.20)$$

A função objetivo (3.1) tem duas partes. A primeira parte diz respeito ao custo, para o caixeiro, de percorrer as arestas do grafo usando carros diferentes. Ela soma o custo das arestas e divide pelo número de pessoas que estão no carro naquele trecho. Já a segunda parte diz respeito às taxas de retorno para devolver os carros alugados às suas origens. Restrições (3.2) e (3.3) asseguram que cada vértice é visitado apenas uma vez. Restrições (3.4) e (3.5) previnem *subtours* e correspondem à formulação *DFJ* para o *PCV* (DANTZIG; FULKERSON; JOHNSON, 1954). Restrição (3.4) assegura que o vértice i é o u_i-ésimo vértice visitado. Já que o problema requer que o *tour* se inicie no vértice 1, este vértice foi removido do grupo de nós considerados nas restrições (3.4) e (3.5). Restrição (3.5) relaciona as variáveis f_{ij}^c e u_i, indicando a sequência das arestas percorridas. Restrição

(3.6) afirma que se carro c é alugado na cidade i, considerando $i \neq 1$, carro c viaja de i para outra cidade j e, além disso, algum outro carro, c', é usado para chegar em i vindo de outra cidade h. A restrição (3.7) garante que o carro alugado na cidade 1 é utilizado para atravessar o primeiro arco. Semelhante à equação (3.6), a restrição (3.8) relaciona as variáveis f_{ij}^c e z_i^c considerando a devolução do carro c à cidade i, quando $i \neq 1$. A restrição (3.9) assegura que o carro usado para atravessar o último arco é devolvido na cidade 1. Restrição (3.10) relaciona as variáveis ω_{ij}^c, y_j^c e z_i^c, indicando quando um carro c é alugado na cidade j e devolvido na cidade i. Restrição (3.11) assegura que um carro é alugado na cidade 1. Restrição (3.12) garante que a capacidade do carro é respeitada. Restrição (3.13) assegura que nenhum passageiro cuja origem e destino são diferentes de cidade 1 viaje de/para esta cidade. Restrição (3.14) assegura que se um passageiro embarca no veículo, ele deve desembarcar apenas em seu destino. Restrição (3.15) previne passageiros de retornar às suas origens e de deixarem seus destinos. Restrição (3.16) garante que existe uma rota da origem até o destino de cada passageiro. Restrição (3.17) assegura que o limite financeiro de cada passageiro será respeitado. Restrição (3.18) garante que cada carro é alugado no máximo uma vez. Restrição (3.19) define as variáveis binárias e a restrição (3.20) define variáveis u_i para o intervalo de inteiros não-negativos.

3.2.2 Modelo baseado em *MTZ*

Este modelo é baseado em uma formulação proposta por Rios, Goldbarg e Quesquén (2017), devidamente corrigida, conforme descrito no Anexo I, com a adição de restrições relacionadas aos passageiros propostas por Calheiros (2017) reescritas de modo mais compreensível. O problema é formulado em (3.21)–(3.45). Os parâmetros e variáveis são os mesmos apresentados na seção 3.2.1 com o acréscimo das seguintes variáveis.

Variáveis

h_i^c: variável binária que indica se o caixeiro chega na cidade i usando o carro c ($h_i^c = 1$) ou não ($h_i^c = 0$);

b_i^c: variável binária que indica se o caixeiro deixa a cidade i usando o carro c ($b_i^c = 1$) ou não ($b_i^c = 0$).

$$\min \sum_{c \in C} \sum_{i,j \in N} \frac{d_{ij}^c f_{ij}^c}{1 + \sum_{l \in L} v_{lij}^c} + \sum_{c \in C} \sum_{i,j \in N} \gamma_{ij}^c \omega_{ij}^c \tag{3.21}$$

sujeito a

$$\sum_{c\in C}\sum_{j\in N} f_{ij}^c = 1, \qquad \forall i \in N \quad (3.22)$$

$$\sum_{c\in C}\sum_{i\in N} f_{ij}^c = 1, \qquad \forall j \in N \quad (3.23)$$

$$u_i - u_j + (n-1) * f_{ij}^c + (n-3) * f_{ji}^c \leq n-2,$$

$$\forall i,j = 2,...,n \mid i \neq j, \forall c \in C \quad (3.24)$$

$$h_j^c = \sum_{i\in N, i\neq j} f_{ij}^c, \qquad \forall j \in N, \forall c \in C \quad (3.25)$$

$$\sum_{c\in C} h_j^c = 1, \qquad \forall j \in N \quad (3.26)$$

$$b_i^c = \sum_{j\in N, i\neq j} f_{ij}^c, \qquad \forall i \in N, \forall c \in C \quad (3.27)$$

$$\sum_{c\in C} b_i^c = 1, \qquad \forall i \in N \quad (3.28)$$

$$z_i^c = h_i^c(1 - b_i^c), \qquad \forall i \in N\backslash\{1\}, \forall c \in C \quad (3.29)$$

$$z_1^c = h_1^c \qquad \forall c \in C \quad (3.30)$$

$$y_i^c = b_i^c(1 - h_i^c), \qquad \forall i \in N\backslash\{1\}, \forall c \in C \quad (3.31)$$

$$y_1^c = b_1^c \qquad \forall c \in C \quad (3.32)$$

$$\sum_{i\in N} z_i^c \leq 1, \qquad \forall c \in C \quad (3.33)$$

$$\sum_{i\in N} y_i^c \leq 1, \qquad \forall c \in C \quad (3.34)$$

$$\sum_{c\in C} z_1^c = 1 \quad (3.35)$$

$$\sum_{c\in C} y_1^c = 1 \quad (3.36)$$

$$\omega_{ij}^c = y_i^c z_j^c, \qquad \forall c \in C, \forall i,j \in N, i \neq j \quad (3.37)$$

$$\sum_{l\in L} v_{lij}^c \leq k^c f_{ij}^c, \qquad \forall c \in C, \forall i,j \in N \quad (3.38)$$

$$\sum_{c\in C}\sum_{i\in N} v_{li1}^c + \sum_{c\in C}\sum_{i\in N} v_{l1i}^c = 0, \qquad \forall l \in L | org(l) \neq 1,\ dst(l) \neq 1 \quad (3.39)$$

$$\sum_{c\in C}\sum_{i\in N} v_{lri}^c = \sum_{c\in C}\sum_{i\in N} v_{lis}^c, \qquad \forall l \in L | r = org(l),\ s = dst(l) \quad (3.40)$$

$$\sum_{c\in C}\sum_{i\in N} v^c_{lir} + \sum_{c\in C}\sum_{i\in N} v^c_{lsi} = 0, \qquad \forall l \in L | r = org(l),\ s = dst(l) \quad (3.41)$$

$$\sum_{c\in C}\sum_{j\in N} v^c_{lij} = \sum_{c\in C}\sum_{j\in N} v^c_{lji}, \qquad \forall i \in N, \forall l \in L | i \neq org(l), i \neq dst(l) \quad (3.42)$$

$$\sum_{c\in C}\sum_{i,j\in N} \frac{d^c_{ij} f^c_{ij}}{1 + \sum_{p\in L} v^c_{pij}} \leq bud(l), \qquad \forall l \in L \quad (3.43)$$

$$f^c_{ij}, \omega^c_{ij}, y^c_i, z^c_i, v^c_{lij} \in \{0,1\}, \qquad \forall i,j \in N, \forall c \in C, \forall l \in L \quad (3.44)$$

$$u_i \in \mathbb{N}, \qquad i \in N \backslash \{1\} \quad (3.45)$$

A função objetivo (3.21) tem duas partes. A primeira parte diz respeito ao custo, para o caixeiro, de percorrer as arestas do grafo usando carros diferentes. Ela soma o custo das arestas e divide pelo número de pessoas que estão no carro naquele trecho. A segunda parte da função objetivo diz respeito às taxas de retorno para devolver os carros alugados às suas origens. Restrições (3.22) e (3.23) asseguram que cada vértice é visitado apenas uma vez. Restrição (3.22) garante que apenas um carro deixa o vértice i percorrendo uma aresta. Restrição (3.23) garante que apenas um carro percorre uma aresta e chega no vértice j. Restrição (3.24) previne a formação de *subtours* e corresponde à formulação do MTZ para o PCV (MILLER; TUCKER; ZEMLIN, 1960). Restrições (3.25) e (3.26) asseguram que cada cidade é visitada apenas uma vez e por exatamente um carro. Restrições (3.27) e (3.28) garantem que cada cidade seja deixada apenas uma vez e por exatamente um carro. Restrição (3.29) assegura que cada carro alugado seja devolvido, isto é, z^c_i se torna ativo quando o caixeiro chega na cidade i em um carro diferente, $i \neq 1$. Restrição (3.30) garante que o carro usado para atravessar o último arco é devolvido na cidade 1. Restrição (3.31) garante que se um carro c é alugado na cidade i, o caixeiro chegou na cidade i em um carro diferente, $i \neq 1$. Restrição (3.32) assegura que o carro alugado na cidade 1 é utilizado para atravessar o primeiro arco. Restrições (3.33) e (3.34) asseguram que cada carro é alugado/devolvido no máximo uma vez. Restrições (3.35) e (3.36) garantem que apenas um carro é alugado e devolvido na cidade 1. Restrição (3.37) assegura que a taxa de retorno é paga quando um carro é alugado na cidade i e devolvido na cidade j, $i \neq j$. Restrição (3.38) garante que a capacidade do carro é respeitada. Restrição (3.39) assegura que nenhum passageiro cuja origem e destino são diferentes de cidade 1 viaje de/para esta cidade. Restrição (3.40) assegura que se um passageiro embarca no veículo, ele deve desembarcar apenas em seu destino. Restrição (3.41) previne passageiros de retornar às suas origens e de deixarem seus destinos. Restrição (3.42) garante que existe uma rota da origem até o destino de cada passageiro. Restrição (3.43) assegura que o limite financeiro de cada passageiro será

respeitado. Restrição (3.44) define as variáveis binárias e a restrição (3.45) define variáveis u_i para o intervalo de inteiros não-negativos.

3.2.3 Modelo baseado em *GG*

Este modelo também é baseado em uma formulação proposta por Goldbarg *et al.* (2017), devidamente corrigida, conforme descrito no Anexo II, com a adição de restrições relacionadas aos passageiros propostas por Calheiros (2017) reescritas de modo mais entendível. A função objetivo também é modificada. O problema é formulado em (3.46)–(3.64). Os parâmetros e variáveis são os mesmos apresentados na seção 3.2.1 com o acréscimo da seguinte variável.

Variável

$u_{i,j}$: inteiros não negativos arbitrários.

$$\min \sum_{c\in C}\sum_{i,j\in N}\frac{d_{ij}^c f_{ij}^c}{1+\sum_{l\in L} v_{lij}^c}+\sum_{c\in C}\sum_{i,j\in N}\gamma_{ij}^c\omega_{ij}^c \tag{3.46}$$

sujeito a

$$\sum_{c\in C}\sum_{j\in N} f_{ij}^c = 1, \qquad \forall i \in N \tag{3.47}$$

$$\sum_{c\in C}\sum_{i\in N} f_{ij}^c = 1, \qquad \forall j \in N \tag{3.48}$$

$$\sum_{\substack{j\in N\\ i\neq j}} u_{ij} - \sum_{\substack{j\in N\setminus\{1\}\\ i\neq j}} u_{ji} = 1, \qquad \forall i \in N\setminus\{1\} \tag{3.49}$$

$$u_{ij} \leq (n-1)\sum_{c\in C} f_{ij}^c, \qquad \forall i,j \in N, i \neq 1 \tag{3.50}$$

$$\sum_{\substack{i\in N\\ i\neq j}} f_{ji}^c - \sum_{\substack{i\in N\\ i\neq j}} f_{ij}^c \leq y_j^c, \qquad \forall j \in N\setminus\{1\}, \forall c \in C \tag{3.51}$$

$$\sum_{i\in N} f_{1i}^c = y_1^c, \qquad \forall c \in C \tag{3.52}$$

$$\sum_{\substack{i\in N\\ i\neq j}} f_{ij}^c - \sum_{\substack{i\in N\\ i\neq j}} f_{ji}^c \leq z_j^c, \qquad \forall j \in N\setminus\{1\}, \forall c \in C \tag{3.53}$$

$$\sum_{i\in N} f_{i1}^c = z_1^c, \qquad \forall c \in C \tag{3.54}$$

$$\omega_{ij}^c = y_j^c z_i^c, \qquad \forall c \in C, \forall i,j \in N, i \neq j \tag{3.55}$$

$$\sum_{l \in L} v_{lij}^c \leq k^c f_{ij}^c, \qquad \forall c \in C, \forall i,j \in N \quad (3.56)$$

$$\sum_{c \in C} \sum_{i \in N} v_{li1}^c + \sum_{c \in C} \sum_{i \in N} v_{l1i}^c = 0, \qquad \forall l \in L | org(l) \neq 1,\ dst(l) \neq 1 \quad (3.57)$$

$$\sum_{c \in C} \sum_{i \in N} v_{lri}^c = \sum_{c \in C} \sum_{i \in N} v_{lis}^c, \qquad \forall l \in L | r = org(l),\ s = dst(l) \quad (3.58)$$

$$\sum_{c \in C} \sum_{i \in N} v_{lir}^c + \sum_{c \in C} \sum_{i \in N} v_{lsi}^c = 0, \qquad \forall l \in L | r = org(l),\ s = dst(l) \quad (3.59)$$

$$\sum_{c \in C} \sum_{j \in N} v_{lij}^c = \sum_{c \in C} \sum_{j \in N} v_{lji}^c, \qquad \forall i \in N, \forall l \in L | i \neq org(l), i \neq dst(l) \quad (3.60)$$

$$\sum_{c \in C} \sum_{i,j \in N} \frac{d_{ij}^c f_{ij}^c}{1 + \sum_{p \in L} v_{pij}^c} \leq bud(l), \qquad \forall l \in L \quad (3.61)$$

$$\sum_{i \in N} y_i^c \leq 1, \qquad \forall c \in C \quad (3.62)$$

$$f_{ij}^c, \omega_{ij}^c, y_i^c, z_i^c, v_{lij}^c \in \{0,1\}, \qquad \forall i,j \in N, \forall c \in C, \forall l \in L \quad (3.63)$$

$$u_{ij} \in \mathbb{Z}_+, \qquad \forall i,j \in N \quad (3.64)$$

A função objetivo (3.46) tem duas partes. A primeira parte diz respeito ao custo, para o caixeiro, de percorrer as arestas do grafo usando carros diferentes. Ela soma o custo das arestas e divide pelo número de pessoas que estão no carro naquele trecho. A segunda parte da função objetivo diz respeito às taxas de retorno para devolver os carros alugados às suas origens. Restrições (3.47) e (3.48) asseguram que cada vértice é visitado apenas uma vez. Restrições (3.49) e (3.50) previnem a formação de *subtours* e corresponde à formulação do GG para o PCV (GAVISH; GRAVES, 1978). Restrição (3.51) garante que, se o carro c deixou o vértice j sem ter chegado até ele, então c foi alugado em j, $j \neq 1$. Restrição (3.52) assegura que um carro é alugado na cidade 1. Semelhante à equação (3.51), a restrição (3.53) garante que se um carro c chegou até o vértice j e não o deixou, então c foi devolvido em j, $j \neq 1$. Restrição (3.54) assegura que um carro é devolvido na cidade 1. Restrição (3.55) relaciona as variáveis ω_{ij}^c, y_j^c e z_i^c. Restrição (3.56) garante que a capacidade do carro é respeitada. Restrição (3.57) assegura que nenhum passageiro cuja origem e destino são diferentes de cidade 1 viaje de/para esta cidade. Restrição (3.58) assegura que se um passageiro embarca no veículo, ele deve desembarcar apenas em seu destino. Restrição (3.59) previne passageiros de retornar às suas origens e de deixarem seus destinos. Restrição (3.60) garante que existe uma rota da origem até o destino de cada passageiro. Restrição (3.61) assegura que o limite financeiro de cada passageiro será respeitado. Restrição (3.62) garante que cada carro é alugado no máximo

uma vez. Restrição (3.63) define as variáveis binárias e a restrição (3.64) define variáveis u_{ij} para o intervalo de inteiros não-negativos.

3.3 Linearização

Os modelos apresentados são de natureza não-linear. Foram utilizados dois métodos diferentes para eliminar a divisão de variáveis das funções objetivo [(3.1, 3.21) e 3.46)] e das restrições financeiras [(3.17, 3.43) e 3.55)]. O primeiro método, apresentado na seção 3.3.1, foi descrito por Rafeh e Jaberi (2016). O segundo, apresentado na seção 3.3.2, é uma adaptação baseada no trabalho de Calheiros (2017). Foi utilizado o método apresentado por Goldbarg *et al.* (2017), descrito na seção 3.3.3, para evitar o produto de variáveis nas restrições referentes aos aluguéis de devolução dos veículos.

3.3.1 Eliminação das Divisões (1º método)

A estratégia utilizada consiste em transformar a divisão em uma multiplicação por um número real. Em seguida, uma técnica de linearização para multiplicações entre números binários e reais é aplicada. Assim, para eliminar a operação de divisão, as seguintes variáveis são introduzidas.

Variáveis

α_{ij}^{c}: variável real cujo valor é inversamente proporcional ao número de passageiros no carro c atravessando a aresta (i,j) de i para j. Esta variável é utilizada para calcular os custos do caixeiro;

g_{lij}^{c}: variável real cujo valor é inversamente proporcional ao número de passageiros no carro c atravessando a aresta (i, j) de i para j. Esta variável é necessária para calcular os custos das viagens dos passageiros e é fruto da linearização da multiplicação entre α_{ij}^{c} e v_{lij}^{c} na equação (3.70), apresentada nas equações (3.71)–(3.73).

O valor de α_{ij}^{c} é dado pela equação (3.65) sujeito a (3.66).

$$\alpha_{ij}^{c}\left(1+\sum_{l\in L} v_{lij}^{c}\right)=1, \qquad \forall i,j\in N, \forall c\in C \quad (3.65)$$

$$0\leq \alpha_{ij}^{c}\leq 1, \qquad \forall i,j\in N, \forall c\in C \quad (3.66)$$

As funções objetivo [(3.1, 3.21) e 3.46)] e as restrições [(3.17, 3.43) e 3.55)] podem ser

reescritas como (3.67) e (3.68), respectivamente.

$$\min \sum_{c\in C}\sum_{i,j\in N} d_{ij}^c f_{ij}^c \alpha_{ij}^c + \sum_{c\in C}\sum_{i,j\in N} \gamma_{ij}^c \omega_{ij}^c \tag{3.67}$$

$$\sum_{c\in C}\sum_{i,j\in N} d_{ij}^c f_{ij}^c \alpha_{ij}^c \leq bud(l), \qquad \forall l \in L \tag{3.68}$$

Passageiros estão no veículo c atravessando a aresta (i,j) apenas nos casos onde (i,j) está ativa. Então, a restrição (3.65) pode ser reescrita como (3.69).

$$\alpha_{ij}^c \left(1 + \sum_{l\in L} v_{lij}^c\right) = f_{ij}^c, \qquad \forall i,j \in N, \forall c \in C \tag{3.69}$$

Utilizando a propriedade distributiva em (3.69), obtém-se a equação (3.70).

$$\alpha_{ij}^c + \alpha_{ij}^c \sum_{l\in L} v_{lij}^c = f_{ij}^c, \qquad \forall i,j \in N, \forall c \in C \tag{3.70}$$

Porém, esta simplificação causa o aparecimento de uma multiplicação entre uma variável binária e uma real na restrição (3.70). Portanto, é necessária a aplicação de uma técnica de linearização. O método descrito em Rafeh e Jaberi (2016) considera uma variável real x, uma variável binária d e os possíveis valores para os limites inferiores (L) e superiores (U) de x. Equação (3.71) é substituída por (3.72) e (3.73).

$$y = x * d \tag{3.71}$$

$$L * d \leq y \leq U * d \tag{3.72}$$

$$L * (1-d) \leq x - y \leq U * (1-d) \tag{3.73}$$

Restrição (3.70) pode ser reescrita como (3.74), sujeito a (3.75).

$$\alpha_{ij}^c + \sum_{l\in L} g_{lij}^c = f_{ij}^c, \qquad \forall i,j \in N, \forall c \in C \tag{3.74}$$

$$0 \leq g_{lij}^c \leq 1, \qquad \forall i,j \in N, \forall c \in C, \forall l \in L \tag{3.75}$$

Considerando os valores de L e U definidos em (3.75), são adicionadas as restrições (3.76)–(3.79).

$$g_{lij}^c \geq 0, \qquad \forall i,j \in N, \forall c \in C, \forall l \in L \tag{3.76}$$

$$g^c_{lij} \leq v^c_{lij}, \qquad \forall i,j \in N, \forall c \in C, \forall l \in L \quad (3.77)$$

$$g^c_{lij} \geq \alpha^c_{ij} - (1 - v^c_{lij}), \qquad \forall i,j \in N, \forall c \in C, \forall l \in L \quad (3.78)$$

$$g^c_{lij} \leq \alpha^c_{ij}, \qquad \forall i,j \in N, \forall c \in C, \forall l \in L \quad (3.79)$$

Finalmente, as funções objetivo [(3.1, 3.21) e 3.46)] e as restrições financeiras [(3.17, 3.43) e 3.55)] são reescritas como (3.80) e (3.81), respectivamente, sujeito a (3.82).

$$\min \sum_{c \in C} \sum_{i,j \in N} d^c_{ij} \alpha^c_{ij} + \sum_{c \in C} \sum_{i,j \in N} \gamma^c_{ij} \omega^c_{ij} \quad (3.80)$$

$$\sum_{c \in C} \sum_{i,j \in N} d^c_{ij} g^c_{lij} \leq bud(l), \qquad \forall l \in L \quad (3.81)$$

$$\alpha^c_{ij}, g^c_{lij} \in \mathbb{R}_+, \alpha^c_{ij}, g^c_{lij} \leq 1, \qquad \forall i,j \in N, \forall c \in C, \forall l \in L \quad (3.82)$$

3.3.2 Eliminação das Divisões (2º método)

A estratégia utilizada consiste em substituir as divisões por sucessivas subtrações. Considerando uma fração N/D, onde N é o numerador e D é o denominador, esta transformação é restrita para casos onde o D é um número inteiro, $D \geq 1$. A fração N/D pode ser reescrita como (3.83), cuja equivalência algébrica é comprovada no Apêndice VI.

$$\frac{N}{D} = N - \sum_{h=1}^{D-1} \frac{N}{h\ (h+1)} \quad (3.83)$$

Por exemplo, a equação (3.84) é reescrita como (3.85).

$$\frac{100}{8} = 12.5 \quad (3.84)$$

$$100 - \left(\frac{100}{1 \times 2} + \frac{100}{2 \times 3} + \frac{100}{3 \times 4} + \frac{100}{4 \times 5} + \frac{100}{5 \times 6} + \frac{100}{6 \times 7} + \frac{100}{7 \times 8} \right) = 12.5 \quad (3.85)$$

Visto que o custo de uma aresta percorrida é dividida entre o caixeiro e os passageiros embarcados naquele trecho, se considerarmos que um carro c é utilizado para atravessar uma aresta com todos os assentos preenchidos, ou seja, com o limite k^c atingido, a fração N/D pode ser reescrita como (3.86).

$$\frac{N}{D} = \frac{N}{1 + k^c} \quad (3.86)$$

Substituindo o denominador D por $1 + k^c$, a equação (3.83) pode ser reformulada

como (3.87).

$$\frac{N}{1+k^c} = N - \sum_{h=1}^{(1+k^c)-1} \frac{N}{h\ (h+1)} = N - \sum_{h=1}^{k^c} \frac{N}{h\ (h+1)} \tag{3.87}$$

Porém, nem sempre o carro vai estar com sua ocupação máxima, então a estratégia é enumerar os assentos h do carro utilizado, onde H é o conjunto de assentos e $h \in H$. Também é necessária a utilização de uma variável binária x_h^c, que indica se o assento h do carro c está ocupado ou não. Assim, é possível reescrever a equação (3.87) em (3.88), sujeito a (3.89) que assegura que a quantidade de passageiros deve respeitar a capacidade do veículo utilizado.

$$\frac{N}{1+\sum_{h\in H} x_h^c} = N - \sum_{h=1}^{k^c} \frac{x_h^c N}{h\ (h+1)} \tag{3.88}$$

$$\sum_{h\in H} x_h^c \leq k^c \tag{3.89}$$

Então, para linearizar as funções objetivo [(3.1, 3.21) e 3.46)] e as restrições [(3.17, 3.43) e 3.55)] usando este método, são introduzidas as seguintes váriaveis de decisão.

Variáveis

x_{ij}^{ch}: variável binária que indica se o assento h do carro c está ocupado enquanto atravessa a aresta (i, j) de i para j ($x_{ij}^{ch} = 1$) ou não ($x_{ij}^{ch} = 0$). Os assentos são ocupados sequencialmente, a partir do primeiro;

q_{li}: variável real não-negativa que indica o custo, para o passageiro l, de visitar a cidade i no percurso entre sua origem e seu destino. Como o procedimento utiliza sucessivas subtrações, esta variável impede a possibilidade de existência de custos negativos em cada visita i.

As funções objetivo [(3.1, 3.21) e 3.46)] e as restrições financeiras [(3.17, 3.43) e 3.55)] podem ser reescritas como (3.90) e (3.91), respectivamente, sujeito a (3.92) utilizada para relacionar as variáveis x_{ij}^{ch} e v_{lij}^c.

$$\min \sum_{c\in C} \sum_{i,j\in N} \left(d_{ij}^c f_{ij}^c - \sum_{h=1}^{k^c} \frac{x_{ij}^{ch} d_{ij}^c f_{ij}^c}{h(h+1)} \right) + \sum_{c\in C} \sum_{i,j\in N} \gamma_{ij}^c \omega_{ij}^c \tag{3.90}$$

$$\sum_{c\in C} \sum_{j\in N} \left(d_{ij}^c v_{lij}^c - \sum_{h=1}^{k^c} \frac{x_{ij}^{ch} d_{ij}^c v_{lij}^c}{h(h+1)} \right) \leq bud(l), \qquad \forall l \in L, \forall i \in N \tag{3.91}$$

$$\sum_{h=1}^{k^c} x_{ij}^{ch} = \sum_{l\in L} v_{lij}^c, \qquad \forall i, j \in N, \forall c \in C \tag{3.92}$$

Expressões (3.90) e (3.91) podem ser simplificadas por (3.93) e (3.94), respectivamente, a partir da eliminação da variável f_{ij}^c do numerador.

$$\min \sum_{c \in C} \sum_{i,j \in N} \left(d_{ij}^c f_{ij}^c - \sum_{h=1}^{k^c} \frac{x_{ij}^{ch} d_{ij}^c}{h(h+1)} \right) + \sum_{c \in C} \sum_{i,j \in N} \gamma_{ij}^c \omega_{ij}^c \tag{3.93}$$

$$\sum_{c \in C} \sum_{j \in N} \left(d_{ij}^c v_{lij}^c - \sum_{h=1}^{k^c} \frac{x_{ij}^{ch} d_{ij}^c}{h(h+1)} \right) \leq q_{li}, \qquad \forall l \in L, \forall i \in N \tag{3.94}$$

Restrições (3.95) e (3.96) asseguram que as restrições financeiras sejam respeitadas e que o valor pago pelo passageiro seja não negativo. Restrição (3.97) define x_{ij}^{ch} como uma variável binária. As funções objetivo [(3.1, 3.21) e 3.46)] e as restrições financeiras [(3.17, 3.43) e 3.55)] são reescritas como (3.93) e (3.94), respectivamente, sujeito a (3.92), (3.95)–(3.97).

$$\sum_{i \in N} q_{li} \leq bud(l), \qquad \forall l \in L \tag{3.95}$$

$$q_{li} \in \mathbb{R}_+, \qquad \forall l \in L, \forall i \in N \tag{3.96}$$

$$x_{ij}^{ch} \in \{0,1\}, \qquad \forall i,j \in N, \forall c \in C, h = 1, ..., k^c \tag{3.97}$$

3.3.3 Eliminação das Multiplicações

Para eliminar a multiplicação entre duas variáveis binárias, foi utilizada a técnica de linearização usual descrita por Robert (1960) e reformulada por Glover e Woosley (1974), onde uma restrição não-linear, como em (3.98), é substituída pelo conjunto de equações (3.99)–(3.102).

$$s = q \times r \tag{3.98}$$

$$s \leq q \tag{3.99}$$

$$s \leq r \tag{3.100}$$

$$s \geq r + q - 1 \tag{3.101}$$

$$q, r, s \in \{0,1\} \tag{3.102}$$

Portanto, para o modelo baseado em *DFJ* (seção 3.2.1), as restrições (3.103)–(3.111) substituem (3.6), (3.8) e (3.10). Para o modelo baseado em *GG* (seção 3.2.3), a restrição

(3.55) é substuída por (3.109)–(3.111).

$$y_i^c \leq \left(\sum_{j \in N} f_{ij}^c \right), \qquad \forall c \in C, \forall i \in N \quad (3.103)$$

$$y_i^c \leq \left(\sum_{c' \in C, c' \neq c} \sum_{h \in N} f_{hi}^{c'} \right), \qquad \forall c \in C, \forall i \in N \quad (3.104)$$

$$y_i^c \geq \left(\sum_{j \in N} f_{ij}^c \right) + \left(\sum_{c' \in C, c' \neq c} \sum_{h \in N} f_{hi}^{c'} \right) - 1, \qquad \forall c \in C, \forall i \in N \quad (3.105)$$

$$z_i^c \leq \left(\sum_{j \in N} f_{ij}^c \right), \qquad \forall c \in C, \forall i \in N \quad (3.106)$$

$$z_i^c \leq \left(\sum_{c' \in C, c' \neq c} \sum_{h \in N} f_{hi}^{c'} \right), \qquad \forall c \in C, \forall i \in N \quad (3.107)$$

$$z_i^c \geq \left(\sum_{j \in N} f_{ij}^c \right) + \left(\sum_{c' \in C, c' \neq c} \sum_{h \in N} f_{hi}^{c'} \right) - 1, \qquad \forall c \in C, \forall i \in N \quad (3.108)$$

$$\omega_{ij}^c \leq y_j^c, \qquad \forall c \in C, \forall i, j \in N, i \neq j \quad (3.109)$$

$$\omega_{ij}^c \leq z_i^c, \qquad \forall c \in C, \forall i, j \in N, i \neq j \quad (3.110)$$

$$\omega_{ij}^c \geq y_j^c + z_i^c - 1, \qquad \forall c \in C, \forall i, j \in N, i \neq j \quad (3.111)$$

Já para o modelo baseado em *MTZ* (seção 3.2.2), as restrições (3.112)–(3.120) substituem (3.29), (3.31) e (3.37).

$$z_i^c \leq h_i^c, \qquad \forall c \in C, \forall i \in N \quad (3.112)$$

$$z_i^c \leq 1 - b_i^c, \qquad \forall c \in C, \forall i \in N \quad (3.113)$$

$$z_i^c \geq h_i^c - b_i^c, \qquad \forall c \in C, \forall i \in N \quad (3.114)$$

$$y_i^c \leq b_i^c, \qquad \forall c \in C, \forall i \in N \quad (3.115)$$

$$y_i^c \leq 1 - h_i^c, \qquad \forall c \in C, \forall i \in N \quad (3.116)$$

$$y_i^c \geq b_i^c - h_i^c, \qquad \forall c \in C, \forall i \in N \quad (3.117)$$

$$\omega_{ji}^c \leq y_i^c, \qquad \forall c \in C, \forall i, j \in N, i \neq j \quad (3.118)$$

$$\omega_{ji}^c \leq z_j^c, \qquad \forall c \in C, \forall i, j \in N, i \neq j \quad (3.119)$$

$$\omega_{ji}^c \geq y_i^c + z_j^c - 1, \qquad \forall c \in C, \forall i, j \in N, i \neq j \quad (3.120)$$

3.4 Modelos Linearizados

Esta seção descreve a função objetivo e restrições que compõem os seis modelos linearizados que estão sendo propostos neste trabalho: *DFJ1*, *DFJ2*, *MTZ1*, *MTZ2*, *GG1* e *GG2*.

- ***DFJ1*** – baseado no modelo *DFJ*, apresentado na seção 3.2.1, utilizando as linearizações propostas nas seções 3.3.1 e 3.3.3. Consiste na função objetivo (3.80) sujeito às restrições (3.2)–(3.5), (3.11)–(3.16), (3.18)–(3.20), (3.74), (3.76)–(3.79), (3.81)–(3.82) e (3.103)–(3.111);

- ***DFJ2*** – baseado no modelo *DFJ*, apresentado na seção 3.2.1, utilizando as linearizações propostas nas seções 3.3.2 e 3.3.3. Consiste na função objetivo (3.93) sujeito às restrições (3.2)–(3.5), (3.11)–(3.16), (3.18)–(3.20), (3.92), (3.94)–(3.97) e (3.103)–(3.111);

- ***MTZ1*** – baseado no modelo *MTZ*, apresentado na seção 3.2.2, utilizando as linearizações propostas nas seções 3.3.1 e 3.3.3. Consiste na função objetivo (3.80) sujeito às restrições (3.22)–(3.28), (3.33)–(3.36), (3.38)–(3.45), (3.74), (3.76)–(3.79), (3.81)–(3.82) e (3.112)–(3.120);

- ***MTZ2*** – baseado no modelo *MTZ*, apresentado na seção 3.2.2, utilizando as linearizações propostas nas seções 3.3.2 e 3.3.3. Consiste na função objetivo (3.93) sujeito às restrições (3.22)–(3.28), (3.33)–(3.36), (3.38)–(3.45), (3.92), (3.94)–(3.97) e (3.112)–(3.120);

- ***GG1*** – baseado no modelo *GG*, apresentado na seção 3.2.3, utilizando as linearizações propostas nas seções 3.3.1 e 3.3.3. Consiste na função objetivo (3.80) sujeito às restrições (3.47)–(3.54), (3.56)–(3.60), (3.62)–(3.64), (3.74), (3.76)–(3.79), (3.81)–(3.82) e (3.109)–(3.111);

- ***GG2*** – baseado no modelo *GG*, apresentado na seção 3.2.3, utilizando as linearizações propostas nas seções 3.3.2 e 3.3.3. Consiste na função objetivo (3.93) sujeito às restrições (3.47)–(3.54), (3.56)–(3.60), (3.62)–(3.64), (3.92), (3.94)–(3.97) e (3.109)–(3.111).

4 Algoritmos Propostos

Este capítulo descreve as estratégias e heurísticas que foram estudadas e adaptadas ao *CaRSP*. A seção 4.1 define a representação de soluções para o problema. Na seção 4.2 são apresentadas duas técnicas de atribuição de passageiros: uma exata e uma heurística. A seção 4.3 descreve três heurísticas ingênuas. As seções 4.4, 4.5 e 4.6 apresentam, respectivamente, detalhes sobre as meta-heurísticas Algoritmo Memético, *GRASP + VNS + Path Relinking* e Algoritmo de Colônia de Formigas desenvolvidas para solucionar o *CaRSP*.

4.1 Representação da Solução

As soluções são representadas por três listas:

- **Cidades** – representa a sequência de cidades. Se inicia na cidade 1 que, por sua vez, não é representada na solução como destino final, o que fica subentendido. Possui tamanho $|N|$;
- **Carros** – representa a sequência de veículos utilizados. Cada veículo da solução é utilizado para trafegar a aresta subsequente do vetor de cidades. Quando há troca de veículos, significa que o carro que deixou de ser utilizado será retornado à sua cidade de origem. Possui tamanho $|N|$;
- **Embarques** – representa o embarque do passageiro l, ao qual associa-se uma origem, um destino e um limite financeiro. Se o passageiro l tiver embarcado é preenchido com valor 1, caso contrário é preenchido com valor 0. Possui tamanho $|L|$;

A Figura 10 ilustra um exemplo de representação de solução para um caso onde temos 6 cidades, 3 carros e 11 passageiros em potencial.

Figura 10: *CaRSP* – Representação da solução

Associado a cada passageiro em potencial, há uma origem, um destino e um limite financeiro, que por sua vez são parâmetros do problema. Na Figura 10 temos que o carro 3 é utilizado para visitar as arestas (1,3) e (3,6) e o carro 1 para trafegar os trechos (6,2), (2,4), (4,5) e (5,1). Na solução ilustrada temos que o carro 3 deverá ser retornado da cidade 6 para 1 e o carro 1 da cidade 1 para 6 e o caixeiro deve arcar com os custos destas despesas. Os passageiros 3, 5, 9 e 10 são transportados de suas origens até seus destinos, respeitando seus limites orçamentários. O carro 2 não foi utilizado e os passageiros 1, 2, 4, 6, 7, 8 e 11 não foram transportados nesta solução.

4.2 Problema de Atribuição de Passageiros

Embora a atribuição de passageiros faça parte do *CaRSP*, uma vez que uma rota é fornecida, este é por si só um problema de otimização (MARQUES *et al.*, 2019). Nas seções 4.2.1 e 4.2.2 são apresentadas, respectivamente, uma formulação exata e uma heurística para solucionar o Problema de Atribuição de Passageiros (*PAP*).

4.2.1 Formulação Exata

Para usar esta formulação, a cidade inicial também é inserida como cidade final, conforme ilustrado na Figura 11.

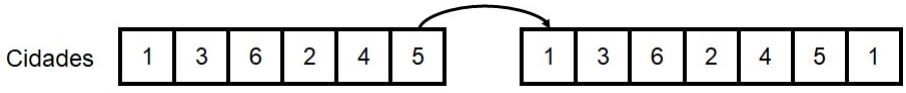

Figura 11: *PAP* – Inserção de cidade inicial como cidade final

Dada uma solução parcial *sol*, contendo a sequência de cidades visitadas (*sol.Cidades*) e veículos utilizados (*sol.Carros*), é possível obter o custo de cada visita. Antes de utilizar o carregamento exato de passageiros, existe uma etapa de pré-processamento que é descrita no Algoritmo 1. É feita uma análise, a partir de *sol*, capaz de eliminar passageiros cujos

destinos estejam antes de suas origens ou que o limite financeiro ($sol.Tarifas$) não seja suficiente para fazer sua viagem desde a origem até o destino, restringindo, assim, o conjunto de passageiros que estão aptos a embarcar.

Algoritmo 1: *PAP* – Etapa de Pré-Processamento

```
1  Entrada: sol, L
2  Saída: L*
3  início
4      L* ← ∅;
5      para (l = 1, ..., |L|) faça
6          indiceOrigem ← retornaIndiceOrigem (sol, l);
7          indiceDestino ← retornaIndiceDestino (sol, l);
8          se (indiceDestino = 1) então
9              indiceDestino ← |N|;
10         se (indiceOrigem < indiceDestino) então
11             custoTotal ← 0;
12             para (i = indiceOrigem, ..., indiceDestino) faça
13                 custo ← retornaCusto (sol.Carros[i], sol.Cidades[i], sol.Cidades[i + 1]);
14                 custoTotal ← custoTotal + (custo / (capacidade (sol.Carros[i]) + 1));
15             se (custoTotal ≤ sol.Tarifas[l]) então
16                 L* ← adicionaPassageiro (l);
17 retorne (L*);
18 fim
```

O Algoritmo 1 recebe como parâmetro de entrada uma solução parcial (sol), contendo a sequência de cidades visitadas e de veículos utilizados, e a lista de passageiros do problema (L). Na linha 4, temos que a lista de passageiros aptos a embarcar (L^*) está inicialmente vazia. Na linha 6, a função *retornaIndiceOrigem()* define o índice da origem do passageiro l, dada a sequência de cidades estabelecida na solução sol. Analogamente, a função *retornaIndiceDestino()*, mostrada na linha 7, define o índice do destino do passageiro l, de acordo com a solução sol. A estrutura condicional da linha 8 serve para corrigir passageiros cujo destino é a cidade inicial que, como mostrado anteriormente na Figura 1, agora também é colocada ao final da lista de cidades. A condição da linha 10 impõe que, apenas estão aptos a embarcar passageiros cujo índice da origem seja menor que o índice do destino. O laço da linha 12 simula a tarifa da viagem de um passageiro desde sua origem até seu destino, divindo o custo de cada aresta pela capacidade do veículo acrescida de uma unidade, que seria a parcela do caixeiro. Ao final, na linha 15, se a tarifa calculada para o passageiro for inferior ao seu limite orçamentário, então, conclui-se que

este passageiro está apto a embarcar e é adicionado à lista L^*. Na linha 17, a lista de passageiros aptos a embarcar é retornada.

As equações (4.1)–(4.7) apresentam a formulação matemática para o PAP em uma solução representada pela variável *sol*, contendo a sequência de cidades e veículos. Esta modelagem é uma adaptação, com as devidas correções, da proposta por Calheiros (2017) e da apresentada em Marques *et al.* (2019). As inconsistências e correções são apresentadas no Anexo III. A estratégia utilizada se baseia na linearização apresentada na seção 3.3.2. Os parâmetros e variáveis são definidos da seguinte forma.

Parâmetros

c_i: custo de usar o carro $sol.Carros[i]$ para viajar de $sol.Cidades[i]$ para $sol.Cidades[i+1]$;
$k^{sol.Carros[i]}$: limite de passageiros do carro $sol.Carros[i]$;
L^*: lista de passageiros aptos a embarcar após o pré-processamento;
bud(l): tarifa máximo que o passageiro l concorda em pagar, $l \in L^*$;
iorg(l): índice da cidade de origem do passageiro l em $sol.Cidades$, $l \in L^*$;
idst(l): índice da cidade de destino do passageiro l em $sol.Cidades$, $l \in L^*$.

Variáveis

e_l: variável binária que indica se o passageiro l embarcou ($e_l = 1$) ou não ($e_l = 0$);
g_{ih}: variável binária que indica se o assento h está ocupado na i-ésima cidade visitada de $sol.Cidades$ ($g_{ih} = 1$) ou não ($g_{ih} = 0$). Os assentos são ocupados sequencialmente, iniciando pelo primeiro;
q_{li}: variável real que indica o custo, para o passageiro l, para visitar a i-ésima cidade do *tour*.

$$\min \sum_{i \in N} \left(c_i - \sum_{h=1}^{k^{sol.Carros[i]}} \frac{g_{ih} c_i}{h(h+1)} \right) \tag{4.1}$$

sujeito a

$$\sum_{l \in P_i} e_l = \sum_{h=1}^{k^{sol.Carros[i]}} g_{ih}, \qquad P_i = \{ l \in L^* \mid \substack{\forall i \in N \\ iorg(l) \leq i < idst(l)} \} \tag{4.2}$$

$$\sum_{l \in P_i} e_l \leq k^{sol.Carros[i]}, \qquad P_i = \{ l \in L^* \mid \substack{\forall i \in N \\ iorg(l) \leq i < idst(l)} \} \tag{4.3}$$

$$c_i e_l - \sum_{h=1}^{k^{sol.Carros[i]}} \frac{g_{ih} c_i}{h(h+1)} \leq q_{li}, \qquad \substack{\forall i \in N \\ \forall l \in L^* \mid \\ iorg(l) \leq i < idst(l)} \tag{4.4}$$

$$\sum_{i \in N} q_{li} \leq bud(l), \qquad \forall l \in L^* \mid iorg(l) \leq i < idst(l) \tag{4.5}$$

$$g_{ih}, e_l \in \{0,1\}, \qquad \begin{matrix}\forall i \in N \\ l \in L^* \\ h=1,\ldots,k^{sol.Carros[i]}\end{matrix} \tag{4.6}$$

$$q_{li} \in \mathbb{R}_+ \tag{4.7}$$

Esta formulação assume que a sequência de cidades visitadas e carros utilizados estão armazenadas na variável *sol* (*sol.Cidades* e *sol.Carros*). A função objetivo (4.1) consiste em minimizar a soma dos custos de cada visita, i, que, por sua vez, é dado pelo custo da aresta percorrida dividido igualmente entre os ocupantes do veículo neste trecho. Restrição (4.2) conecta as variáveis e_l e g_{ih}, definindo quando um passageiro l embarcou, ou seja, quando l viaja de sua origem até seu destino ocupando um assento. Restrição (4.3) garante que a capacidade de cada carro utilizado nunca seja ultrapassada. Equação (4.4) e (4.5) garantem que o custo do percurso de cada passageiro a bordo nunca ultrapasse o seu limite financeiro. Restrições (4.6) e (4.7) garantem a integridade e não-negatividade das variáveis de decisão.

4.2.2 Heurística

Dada uma solução parcial *sol*, com a sequência de cidades visitadas (*sol.Cidades*) e de veículos utilizados (*sol.Carros*), o procedimento de embarque de passageiros, descrito no Algoritmo 2, analisa a demanda dos passageiros em cada arco da rota (linhas 4–10) e considera apenas os passageiros cuja origem esteja antes do destino (linha 12).

As linhas 13–24 calculam o custo de cada passageiro para viajar desde sua origem até seu destino. O custo de atravessar cada arco é calculado pela divisão do custo do arco (linha 15) pela demanda dos passageiros naquele trecho, incluindo o caixeiro (linha 19). Se esta demanda for maior que o limite do veículo (linha 16), então este custo é dividido pela capacidade do carro (linha 17).

Se o custo de viagem de um passageiro atender ao seu limite financeiro, ele será adicionado à lista de passageiros aptos a embarcar, L^* (linhas 21–22). Caso contrário, é removido da lista de demandas (linhas 23–24). O embarque heurístico (linha 25) consiste em embarcar cada passageiro $l \in L^*$.

Para cada trecho com mais passageiros embarcados do que a capacidade do veículo, os passageiros são removidos até que o limite do veículo seja respeitado. Passageiros com menos arcos percorridos entre sua origem e destino são mais propensos a serem removidos neste procedimento. Em seguida, o custo de cada passageiro a bordo é calculado e os que excedem o limite financeiro são removidos. Para cada pessoa removida, é necessário recal-

cular os custos dos outros passageiros e repetir o procedimento de retirada de passageiros até que o custo de cada passageiro a bordo não ultrapasse o limite financeiro estabelecido. Esta heurística gera uma solução viável que respeita todas as restrições do problema.

Algoritmo 2: *PAP* – Heurística

```
1  Entrada: N, L, sol;
2  Saída: sol;
3  início
4      para (l = 1, ..., |L|) faça
5          indicesOrigem[l] ← retornaIndiceOrigem (sol, l)
6          indicesDestino[l] ← retornaIndicesDestino (sol, l)
7          se (indicesDestino[l] = 1) então
8              indicesDestino[l] ← |N|
9          para (i = indicesOrigem[l], ..., indicesDestino[l]) faça
10             demandas[i] ← demandas[i] + 1
11     para (l = 1, ..., |L|) faça
12         se (indicesOrigem[l] < indicesDestino[l]) então
13             custoPassageiro ← 0;
14             para (i = indicesOrigem[l], ..., indicesDestino[l]) faça
15                 custo ← retornaCusto (sol.Carros[i], sol.Cidades[i], sol.Cidades[i + 1])
16                 se (demandas[i] >capacidade (sol.Carros[i]) então
17                     custo ← custo / (capacidade(sol.Carros[i]) + 1)
18                 senão
19                     custo ← custo / demandas[i] + 1)
20                 custoPassageiro ← custoPassageiro + custo
21             se (custoPassageiro ≤ tarifa[l]) então
22                 L* ← AdicionaPassageiro (l)
23             senão
24                 demandas ← RemoveDemandas (demandas, l)
25     sol ← EmbarqueHeuristico (sol, L*, demandas)
26 retorne (sol)
27 end
```

4.3 Heurísticas Ingênuas

Foram desenvolvidas três abordagens ingênuas para solucionar o *CaRSP* e servir como limites para o problema, denominadas *NAIVE1*, *NAIVE2* e *NAIVE3* e são apresentadas, respectivamente, nas seções 4.3.1, 4.3.2 e 4.3.3.

4.3.1 *NAIVE1*

A estratégia desta heurística é dividir a construção da solução em duas etapas: primeiro resolve-se o problema referente à sequência de cidades e veículos ($CaRS$), de forma exata. Em seguida, a partir da solução gerada, faz-se o carregamento exato de passageiros. A primeira fase é resolvida a partir do modelo apresentado na seção 2.2.4 do Capítulo 2, a partir das restrições (2.12)–(2.16) e (2.19)–(2.26). Já a segunda etapa é explicada na seção 4.2.1.

4.3.2 *NAIVE2*

A abordagem utilizada por esta heurística consiste em dividir a construção da solução em três etapas: primeiro, para cada carro, resolve-se apenas o problema da rota utilizando uma abordagem heurística. Em seguida, utilizando as rotas pré-estabelecidas na etapa anterior, seleciona-se a melhor combinação de aluguéis e devoluções de carros a partir de uma técnica exata. Por último, realiza-se o carregamento exato de passageiros em cada uma delas. A seguinte variável é introduzida.

Variável

sol_c: lista com $|C|$ soluções geradas pela heurística *NAIVE2*.

Na primeira etapa é resolvida a sequência de cidades, de forma heurística, utilizando cada carro c, $\forall c \in C$, e armazenadas em suas respectivas soluções (sol_c). Ao final desta etapa, são geradas $|C|$ rotas.

Na segunda etapa, para cada rota gerada e armazenada em sol_c, resolve-se, de forma exata, o problema de alocação dos veículos. Isto é, utiliza-se a formulação descrita em (2.12)–(2.16) e (2.19)–(2.26) com o acréscimo da equação (4.8), responsável por fixar a rota previamente calculada e obrigar que cada uma das arestas definidas em sol_c seja trafegada por um veículo. Concluída esta etapa, cada solução (sol_c) passa a utilizar a melhor combinação de automóveis possível, considerando tanto os aluguéis quanto as devoluções dos veículos.

$$\sum_{c \in C} f^c_{(sol_c.Cidades[i])(sol_c.Cidades[i+1])} = 1 \qquad \forall i \in N \quad (4.8)$$

Por último, para cada solução (sol_c) aplica-se o carregamento exato de passageiros, descrito previamente na seção 4.2.1. Considera-se como solução final, aquela cujo custo seja o menor entre as demais.

4.3.3 *NAIVE3*

Esta heurística divide a solução do *CaRSP* em duas etapas: primeiro resolve-se o problema referente à sequência de cidades e veículos (*CaRS*), a partir do uso de uma meta-heurística. Em seguida, a partir da solução gerada, faz-se o carregamento exato de passageiros. A primeira fase é solucionada por um Algoritmo Memético seguindo a mesma estrutura proposta neste trabalho, que será apresentada a seguir, na seção 4.4. Esta técnica faz uso dos mesmos procedimentos propostos para a manipulação apenas da sequência de cidades e de veículos, ou seja, os procedimentos que incorporam abordagens referentes aos passageiros não são utilizados. A segunda etapa consiste no carregamento exato de passageiros que, por sua vez, é explicado na seção 4.2.1.

4.4 Algoritmo Memético

Os Algoritmos Genéticos são inspirados no Darwinismo. É uma meta-heurística que possui inspiração biológica, prezando pela simulação de processos evolutivos em sistemas artificiais. Utiliza uma abordagem baseada em população, considerando os aspectos de reprodução, recombinação, mutação e seleção natural (MENEZES, 2014).

Os Algoritmos Meméticos se inspiram no Lamarckismo e podem ser definidos como algoritmos evolucionários híbridos, uma vez que incorporam ao Algoritmo Genético, estratégias de busca local para combinar as vantagens de heurísticas eficientes com a evolução baseada em população (DIGALAKIS; MARGARITIS, 2004). Este conceito foi proposto por Moscato (1989), e, de acordo com Menezes (2014), a abordagem memética considera um aperfeiçoamento não-genético na população com auto-reprodução, herança de características, mudança aleatória no genótipo e sobrevivência regulada por combinação de habilidades.

Alguns exemplos de aplicações de algoritmos meméticos voltado ao roteamento de veículos são vistos em Asconavieta (2011), Goldbarg, Asconavieta e Goldbarg (2012), Menezes (2014), Calheiros (2015), Sabry (2016), Araújo (2016), Silva (2017) e Calheiros (2017).

4.4.1 Algoritmo Memético Proposto

O pseudo-código do algoritmo memético proposto é apresentado em Algoritmo 3. Os parâmetros de entrada para o algoritmo são: *nomeInstancia* – identifica o problema que

será resolvido, *tamPop* – representa o tamanho da população, *txRep* – taxa de reprodução, *txMut* – taxa de mutação e *txBus* – taxa de busca local. A condição de parada é dada a partir do número de avaliações da função objetivo, *numAvaliacoes*.

Algoritmo 3: Algoritmo Memético para o *CaRSP*

```
1  Entrada: nomeInstancia, tamPop, txRep, txMut, txBus
2  Saída: melhorIndividuo
3  início
4      G ← leInstancia (nomeInstancia);
5      pop ← geraPopulacaoInicial (tamPop);
6      enquanto (numAvaliacoes < |N| × |C| × 500) faça
7          para (i = 1, ..., txRep) faça
8              pai, mae ← selecionaPais (pop)
9              filho1, filho2, filho3, filho4 ← reproducao (pai, mae);
10             pop ← insereFilhosPopulacao (pop, filho1, filho2, filho3, filho4);
11         para (i = 1, ..., txMut) faça
12             ind ← selecionaIndividuo (pop);
13             pop[ind] ← mutacao (pop[ind]);
14         para (i = 1, ..., txBus) faça
15             ind ← selecionaIndividuo (pop);
16             pop[ind] ← buscaLocal (pop[ind]);
17         pop ← selecaoNatural (pop, tamPop);
18     melhorIndividuo ← retornaMelhorIndividuo (pop);
19 retorne (melhorIndividuo);
20 fim
```

O processo de geração da população inicial, mostrado na linha 5, é explicado na seção 4.4.1.1. O procedimento da linha 9, cuja finalidade é a reprodução dos indivíduos, é detalhado na seção 4.4.1.2. A etapa de mutação, vista na linha 13, é descrita na seção 4.4.1.3. Já a função da linha 16, correspondente à fase de busca local, é definida na seção 4.4.1.4. Os indivíduos são selecionados nas etapas de reprodução (linha 8), mutação (linha 12) e busca local (linha 15) a partir dos procedimentos *selecionaPais()* e *selecionaIndividuo()* que, por sua vez, utilizam o método da roleta, onde são dadas maiores probabilidades para as soluções de menor custo. A função *selecionaIndividuos()*, presente nas linhas 12 e 15, não permite que sejam selecionados indíviduos que já sofreram o respectivo procedimento e não obtiveram melhorias. A função *selecaoNatural()*, mostrada na linha 17, consiste em manter na população apenas os *tamPop* melhores indivíduos. Por fim, o indivíduo da população com o menor custo é dado como a solução para o problema (linha 19).

4.4.1.1 População Inicial

A população inicial é gerada pelo procedimento *geraPopulacaoInicial()*, mostrado na linha 5 do Algoritmo 3, recebendo como parâmetro de entrada o tamanho da população (*tamPop*). Este procedimento é descrito a partir do Algoritmo 4.

Algoritmo 4: Geração da população inicial

```
1  Entrada: tamPop
2  Saída: pop
3  início
4      para (i = 1, ..., tamPop) faça
5          sol ← ∅
6          sol.Cidades[1] ← 1;
7          numCar ← sorteiaNumero (1, |C|);
8          sol.Carros[1] ← sorteiaNumero (1, |C|);
9          numCar ← numCar - 1;
10         indiceTroca ← sorteiaNumero (2, |N| - (numCar - 1));
11         para (j = 2, ..., |N|) faça
12             listaDestinos ← retornaListaDestinos (sol, sol.Cidades[j-1]);
13             cidadesNaoVisitadas ← retornaCidadesNaoVisitadas (sol);
14             se (j ≤ |N| / 2) então
15                 se (listaDestino ≠ ∅) então
16                     sol.Cidades[j] ← roletaPassageiros (sol, listaDestinos, j);
17                 senão
18                     sol.Cidades[j] ← roletaCustos (sol, cidadesNaoVisitadas, j);
19             senão
20                 sol.Cidades[j] ← roletaCustos (sol, cidadesNaoVisitadas, j);
21             se (j = indiceTroca) e (numCar > 0) então
22                 sol.Carros[j] ← sorteiaCarroNaoUtilizado (sol);
23                 numCar ← numCar - 1;
24                 indiceTroca ← sorteiaNumero (j + 1, |C| - (numCar - 1));
25             senão
26                 sol.Carros[j] ← sol.Carros[j-1];
27         sol ← carregamentoHeuristicoPassageiros (sol);
28         pop ← adicionaSolucao (sol);
29 retorne (pop);
30 fim
```

Para cada indíviduo que será gerado, este procedimento define a cidade inicial do *tour* (linha 6), sorteia a quantidade de carros que serão utilizados no percurso (linha

7), seleciona aleatoriamente qual será o carro utilizado no início do trajeto (linha 8) e define aonde ocorrerá a primeira troca de veículo, quando for o caso (linha 10). Então, para selecionar cada cidade que irá compor o *tour*, são utilizadas duas listas. A primeira (*listaDestinos*), mostrada na linha 12, gerada a partir da função *retornaListaDestinos()*, é composta pelos destinos dos passageiros cuja origem é a cidade atual e que ainda não foram visitados. Já a segunda (*cidadesNaoVisitadas*), mostrada na linha 13, gerada a partir da função *retornaCidadesNaoVisitadas()*, é constituída de todas as cidades não visitadas. Das linhas 14 a 20, o algoritmo alterna a ênfase construtiva da solução baseada na quantidade de passageiros aptos a embarcar (linha 16) ou nos custos envolvidos para trafegar as arestas (linha 18). Na linha 21, analisa-se se o índice de troca de veículo definido na linha 10 foi atingido e realiza-se a troca, se for o caso (linha 22). As linhas 25 e 26 definem que, quando não há troca de veículos, o carro utilizado na visita anterior será mantido. Por fim, o carregamento heurístico de passageiros, apresentado na seção 4.2.2, é realizado na linha 27 e o indivíduo é inserido na população na linha 28. Ao final, a população gerada é retornada (linha 29).

4.4.1.2 Reprodução

A etapa de reprodução é responsável, basicamente, por formar novos indivíduos. Isto se dá a partir da troca de informações de membros da população gerada inicialmente, chamados de pais. Existem várias regras que definem um cruzamento, a utilizada neste trabalho, descrita no Algoritmo 5, se baseia no operador de recombinação de um ponto.

A função *selecionaPais()*, mostrada na linha 4, seleciona dois indivíduos diferentes aleatoriamente, *pai* e *mae*, para gerar quatro filhos (*c1*, *c2*, *c3* e *c4*). *c1* e *c3* herdam material genético referente às cidades visitadas e carros utilizados por *mae* (linha 5). *c2* e *c4* herdam material genético referente às cidades visitadas e veículos usados por *pai* (linha 6). O ponto que limita quais cidades os filhos herdarão de seu outro pai é definido aleatoriamente, conforme mostrado na linha 7. *c1* e *c3* herdam cidades da segunda parte de *pai* (linha 8). *c2* e *c4* herdam cidades da segunda parte de *mae* (linha 9). *c1* herda a sequência de carros de *pai* (linha 10). *c2* herda a sequência de carros de *mae* (linha 11). A função *reparaCromossomos()*, mostrada na linha 12, substitui cada cidade repetida de *c1*, *c2*, *c3* e *c4* por uma que não foi visitada, de forma que resulte em custos mais baixos para percorrer os arcos incidentes àquele vértice. A função *reparaPassageiros()*, mostrada na linha 13, remove todos os passageiros embarcados que agora não podem mais viajar, seja devido à alteração das cidades visitadas ou que seu limite financeiro não

seja mais suficiente, e tenta embarcar outros passageiros em potencial, respeitando todas as restrições do problema.

Algoritmo 5: *CaRSP* – Reprodução

```
1  Entrada: pop
2  Saída: c1, c2, c3, c4
3  início
4      Cromossomo pai, mae ← selecionaPais (pop)
5      Cromossomo c1, c3 ← mae
6      Cromossomo c2, c4 ← pai
7      ponto ← geraNumeroAleatorio (2, |N| − 1)
8      c1, c3 ← herdaCidades (pai, ponto, |N|)
9      c2, c4 ← herdaCidades (mae, ponto, |N|)
10     c1.Carros ← pai.Carros
11     c2.Carros ← mae.Carros
12     c1, c2, c3, c4 ← reparaCromossomos (c1, c2, c3, c4)
13     c1, c2, c3, c4 ← reparaPassageiros (c1, c2, c3, c4)
14 retorne (c1, c2, c3, c4)
15 fim
```

4.4.1.3 Mutação

A variabilidade entre indíviduos de uma mesma população é dada a partir da recombinação genética e pode se dar por meio da reprodução ou através de mutações. A etapa de mutação utilizada neste trabalho se dá a partir do uso de cinco procedimentos, aplicados sequencialmente a uma dada solução, que são explicados nas seções 4.4.1.3.1, 4.4.1.3.2, 4.4.1.3.3, 4.4.1.3.4 e 4.4.1.3.5. Após cada um desses procedimentos, os passageiros são embarcados usando a heurística mostrada na seção 4.2.2, Algoritmo 2.

4.4.1.3.1 Remove Carro

Dada uma solução *sol*, este procedimento consiste em verificar qual carro utilizado pode ser removido de *sol* que resulte na solução de menor custo possível. Para cada carro c removido de *sol*, as cidades que antes eram visitadas por c passam a ser visitadas pelo carro anterior ou posterior, conforme descrito no Algoritmo 6.

O algoritmo recebe como parâmetro de entrada uma solução, *sol*. A função *estimaCusto()*, presente nas linhas 4, 18 e 19, calcula o custo estimado de uma dada solução a partir da análise de demanda dos passageiros ao longo do percurso desta. Quando se há uma demanda de passageiros abaixo da capacidade do veículo utilizado, o custo da aresta percorrida é dividido por esta demanda. Caso contrário, este custo é dividido pelo limite de passageiros deste automóvel. A cada iteração, as funções *retornaIndiceInicioCarro()*

Algoritmo 6: Mutação – Remove Carro

```
1  Entrada: sol
2  Saída: melhorSolucao
3  início
4      melhorSolucao ← estimaCusto (sol);
5      para (i = 1, ..., |N|) faça
6          indiceInicio ← retornaIndiceInicioCarro (sol, sol.Carros[i]);
7          indiceFim ← retornaIndiceFimCarro (sol, sol.Carros[i]);
8          se (indiceInicio > 1) então
9              carAnterior ← sol.Carros[indiceInicio − 1];
10             solAux1 ← sol;
11             para (j = indiceInicio, ..., indiceFim) faça
12                 solAux1.Carros[j] ← carAnterior;
13         se (indiceFim < |N|) então
14             carPosterior ← sol.Carros[indiceFim + 1];
15             solAux2 ← sol;
16             para (j = indiceInicio, ..., indiceFim) faça
17                 solAux2.Carros[j] ← carPosterior;
18         solAux1 ← estimaCusto (solAux1);
19         solAux2 ← estimaCusto (solAux2);
20         se (solAux1.Custo < melhorSolucao.Custo) então
21             melhorSolucao ← solAux1;
22         se (solAux2.Custo < melhorSolucao.Custo) então
23             melhorSolucao ← solAux2;
24         i ← indiceFim;
25 melhorSolucao ← reparaPassageiros (melhorSolucao);
26 retorne (melhorSolucao);
27 fim
```

e *retornaIndiceFimCarro()*, presentes nas linhas 6 e 7, retornam, respectivamente, os valores de *indiceInicio* e *indiceFim*, que definem o trecho em que o veículo correspondente àquela iteração é utilizado. Em seguida, nas linhas 8 a 12, este carro é substituído pelo veículo utilizado anteriormente. Já nas linhas 13 a 17, o veículo é substituído pelo utilizado posteriormente. As linhas 20 a 23 analisam qual foi a melhor solução gerada. A linha 24 atualiza o valor do índice i, responsável por controlar as iterações do laço, definindo que o próximo laço se dará após o valor de *indiceFim*, previamente calculado. Na linha 25, a função *reparaPassageiros()*, explicada anteriormente na seção 4.4.1.2, é aplicada sobre a melhor solução encontrada. Por fim, na linha 26, a melhor solução é retornada.

4.4.1.3.2 Adiciona Carro

Dada uma solução *sol*, este procedimento cria uma lista contendo todos os carros não utilizados em *sol*. Cada veículo desta lista é inserido antes e depois de cada automóvel utilizado em *sol*. A solução final é dada pela inserção de veículo que resulte no menor

custo possível, conforme descrito no Algoritmo 7.

Algoritmo 7: Mutação – Adiciona Carro

```
1  Entrada: sol
2  Saída: melhorSolucao
3  início
4      melhorSolucao ← estimaCusto(sol);
5      carrosNaoUtilizados ← retornaListaCarrosNaoUtilizados (sol);
6      para (i = 1, ..., |N|) faça
7          indiceInicio ← retornaIndiceInicioCarro (sol, sol.Carros[i]);
8          indiceFim ← retornaIndiceFimCarro (sol, sol.Carros[i]);
9          para (j = 1, ..., |carrosNaoUtilizados|) faça
10             se (indiceInicio > 1) ou ((indiceInicio = 1) e (indiceFim = |N|)) então
11                 solAux1 ← sol;
12                 para (k = indiceInicio, ..., indiceFim − 1) faça
13                     solAux1.Carros[k] ← carrosNaoUtilizados[j];
14                 solAux1 ← estimaCusto (solAux1);
15                 se (solAux1.Custo < melhorSolucao.Custo) então
16                     melhorSolucao ← solAux1;
17             se (indiceFim < |N|) ou ((indiceInicio = 1) e (indiceFim = |N|)) então
18                 solAux2 ← sol;
19                 para (k = indiceFim, ..., indiceInicio + 1) faça
20                     solAux2.Carros[k] ← carrosNaoUtilizados[j];
21                 solAux2 ← estimaCusto (solAux2);
22                 se (solAux2.Custo < melhorSolucao.Custo) então
23                     melhorSolucao ← solAux2;
24         i ← indiceFim;
25 melhorSolucao ← reparaPassageiros (melhorSolucao);
26 retorne (melhorSolucao);
27 fim
```

O algoritmo recebe como parâmetro de entrada uma solução, *sol*. Na linha 4, *melhorSolucao* recebe o custo estimado de *sol*. O procedimento *estimaCusto()* foi explicado na seção 4.4.1.3.1. A partir da função *retornaListaCarrosNaoUtilizados()*, presente na linha 5, obtém-se a lista de veículos não utilizados em *sol*. Nas linhas 7 e 8 são definidos, respetivamente, os índices a partir do qual se deu o ínicio (*indiceInicio*) e o fim (*indiceFim*) do trecho em que o veículo correspondente àquela iteração é utilizado. Conforme mostrado na linha 9, cada carro não utilizado, será inserido nas posições iniciais do carro atual (linhas 10 a 16) e nas posições finais (linhas 17 a 23), guardando apenas a melhor solução obtida. A variável *i*, cuja finalidade é controlar as iterações do laço, tem seu valor atualizado na linha 24. Na linha 25, a função *reparaPassageiros()*, explicada anteriormente na seção 4.4.1.2, é aplicada sobre a melhor solução encontrada. Por fim, na linha 26, a melhor solução é retornada.

4.4.1.3.3 Troca Interna de Carros

Dada uma solução *sol*, esta técnica se baseia em realizar, entre os veículos utilizados em *sol*, todas as trocas possíveis que envolvam dois carros. Este procedimento considera que se dois carros, a e b, são trocados, então as cidades que antes eram visitadas por a agora são visitadas por b e vice-versa. O processo é descrito no Algoritmo 8.

Algoritmo 8: Mutação – Troca Interna de Carros

```
1  Entrada: sol
2  Saída: melhorSolucao
3  início
4      melhorSolucao ← estimaCusto(sol);
5      indicesInicio ← retornaListaIndicesInicioCarros (sol);
6      indicesFim ← retornaListaIndicesFimCarros (sol);
7      para (i = 1, ..., |indicesInicio|) faça
8          para (j = i + 1, ..., |indicesInicio|) faça
9              solAux1 ← sol;
10             para (k = indicesInicio[i], ..., indicesFim[i]) faça
11                 solAux1.Carros[k] ← sol.Carros[indicesInicio[j]];
12             para (k = indicesInicio[j], ..., indicesFim[j]) faça
13                 solAux1.Carros[k] ← sol.Carros[indicesInicio[i]];
14             solAux1 ← estimaCusto (solAux1);
15             se (solAux1.Custo < melhorSolucao.Custo) então
16                 melhorSolucao ← solAux1;
17 melhorSolucao ← reparaPassageiros (melhorSolucao);
18 retorne (melhorSolucao);
19 fim
```

O algoritmo recebe como parâmetro de entrada uma solução, *sol*. Na linha 4, a estimativa de custo de *sol* é calculada, conforme apresentado na seção 4.4.1.3.1, e armazenada em *melhorSolucao*. Nas linhas 5 e 6, são definidas, respetivamente, as listas de índices a partir dos quais se deu o ínicio (*indicesInicio*) e o fim (*indicesFim*) da utilização de cada veículo. Os laços de repetição das linhas 7 e 8 definem qual par de carros será trocado. Na linha 9, a variável *solAux1* recebe uma cópia de *sol*. As linhas 10 a 13 realizam a troca dos veículos utilizados nos intervalos [*indicesInicio*[i], *indicesFim*[i]] e [*indicesInicio*[j], *indicesFim*[j]]. A solução gerada é submetida ao procedimento *estimaCusto()* na linha 14. A melhor configuração encontrada é armazenada, conforme mostrado nas linhas 15 e 16. Na linha 17, a função *reparaPassageiros()*, explicada anteriormente na seção 4.4.1.2, é aplicada sobre a melhor solução encontrada. Por fim, na linha 18, a melhor solução é retornada.

4.4.1.3.4 Troca Externa de Carros

Esta abordagem é semelhante à mostrada na seção 4.4.1.3.3, porém analisa a substituição de veículos utilizados por não utilizados, conforme descrito no Algoritmo 9.

Algoritmo 9: Mutação – Troca Externa de Carros

```
1  Entrada: sol
2  Saída: melhorSolucao
3  início
4      melhorSolucao ← estimaCusto(sol);
5      indicesInicio ← retornaListaIndicesInicioCarros (sol);
6      indicesFim ← retornaListaIndicesFimCarros (sol);
7      carrosNaoUtilizados ← retornaListaCarrosNaoUtilizados (sol);
8      para (i = 1, ..., |indicesInicio|) faça
9          para (j = 1, ..., |carrosNaoUtilizados|) faça
10             solAux ← sol;
11             para (k = indicesInicio[i], ..., indicesFim[i]) faça
12                 solAux.Carros[k] ← carrosNaoUtilizados[j];
13             solAux ← estimaCusto (solAux);
14             se (solAux.Custo < melhorSolucao.Custo) então
15                 melhorSolucao ← solAux;
16 melhorSolucao ← reparaPassageiros (melhorSolucao);
17 retorne (melhorSolucao);
18 fim
```

O algoritmo recebe como parâmetro de entrada uma solução, *sol*. Na linha 4, o procedimento *estimaCusto*, explicado na seção 4.4.1.3.1, é aplicado sobre *sol* e armazenado na em *melhorSolucao*. Nas linhas 5 e 6 são definidas, respectivamente *indicesInicio* e *indicesFim*, as listas de índices a partir dos quais se deu o ínicio e o fim da utilização de cada veículo. Na linha 7 é definida uma lista *carrosNaoUtilizados* que possui todos veículos que não estão sendo usados na solução *sol*. Os laços de repetição das linhas 8 e 9 definem qual par de carros será trocado. A variável *solAux* recebe uma cópia de *sol* na linha 10. As linhas 11 e 12 substituem a sequência de veículos definidos no intervalo [$indicesInicio[i]$, $indicesFim[i]$] por $carrosNaoUtilizados[j]$. O custo estimado da solução gerada é calculado na linha 13. Conforme mostrado nas linhas 14 e 15, apenas a melhor solução é armazenada. Na linha 16, a função *reparaPassageiros()*, explicada anteriormente na seção 4.4.1.2, é aplicada sobre a melhor solução encontrada. Por fim, na linha 17, a melhor solução é retornada.

4.4.1.3.5 Altera Aluguel/Devolução

Este procedimento consiste, basicamente, em analisar e avaliar a possibilidade de antecipar ou adiar os aluguéis e devoluções dos veículos de uma dada solução, conforme

descrito no Algoritmo 10.

Algoritmo 10: Mutação – Altera Aluguel/Devolução

```
1  Entrada: sol
2  Saída: melhorSolucao
3  início
4      melhorSolucao ← estimaCusto(sol);
5      indicesInicio ← retornaListaIndicesInicioCarros (sol);
6      indicesFim ← retornaListaIndicesFimCarros (sol);
7      para (i = 1, ..., |indicesInicio| − 1) faça
8          solAux1 ← melhorSolucao;
9          solAux2 ← melhorSolucao;
10         j ← indicesFim[i];
11         enquanto (j > indicesInicio[i]) faça
12             solAux1.Carros[j] ← melhorSolucao.Carros[indicesInicio[i + 1]];
13             solAux1 ← estimaCusto (solAux1);
14             se (solAux1.Custo < melhorSolucao.Custo) então
15                 melhorSolucao ← solAux1;
16             j ← j − 1;
17         para (j = indicesInicio[i + 1], ..., indicesFim[i + 1]) faça
18             solAux2.Carros[j] ← melhorSolucao.Carros[indicesFim[i]];
19             solAux2 ← estimaCusto (solAux1);
20             se (solAux2.Custo < melhorSolucao.Custo) então
21                 melhorSolucao ← solAux2;
22 melhorSolucao ← reparaPassageiros (melhorSolucao);
23 retorne (melhorSolucao);
24 fim
```

O algoritmo recebe como parâmetro de entrada uma solução, *sol*. Na linha 4, o procedimento *estimaCusto*, explicado na seção 4.4.1.3.1, é aplicado sobre *sol* e armazenado na em *melhorSolucao*. Nas linhas 5 e 6 são definidas, respectivamente *indicesInicio* e *indicesFim*, as listas de índices a partir dos quais se deu o ínicio e o fim da utilização de cada veículo. O laço de repetição da linha 7 define qual veículo está sendo analisado nesta iteração. Nas linhas 8 e 9, *solAux1* e *solAux2* recebem uma cópia de *melhorSolucao*. Nas linhas 10 a 16, é simulada a antecipação do aluguel do próximo veículo. Nas linhas 17 a 21, é simulada a prorrogação da devolução do veículo atual. Em ambos os casos, apenas a melhor configuração encontrada é salva, conforme mostram as linhas 14, 15, 20 e 21. Na linha 22, a função *reparaPassageiros()*, explicada anteriormente na seção 4.4.1.2, é aplicada sobre a melhor solução encontrada. Por fim, na linha 23, a melhor solução é retornada.

4.4.1.4 Busca Local

Um algoritmo de busca local explora um conjunto de vizinhanças de uma dada solução. Seu objetivo é gerar novas soluções, buscando melhorias. Neste trabalho foram desenvolvidas seis buscas locais, aplicadas sequencialmente em uma dada solução, que são explicadas nas seções 4.4.1.4.1, 4.4.1.4.2, 4.4.1.4.3, 4.4.1.4.4, 4.4.1.4.5 e 4.4.1.4.6. Após cada um desses procedimentos, os passageiros são embarcados usando a heurística mostrada na seção 4.2.2, Algoritmo 2.

4.4.1.4.1 Passageiros & Carros

Dada a sequência das cidades de uma solução *sol*, este procedimento analisa as demandas dos passageiros ao longo do *tour*, ou seja, passageiros cuja cidade de origem é visitada antes que a cidade de destino. Para cada arco percorrido, se a demanda daquele trecho for maior que a capacidade do veículo utilizado, é feita uma análise para verificar se há benefícios em trocar o carro por um não utilizado que possua mais assentos ou reorganizar os próprios carros que estão sendo utilizados para atender a essas demandas. A sequência de carros final é definida pela combinação que leva à melhor solução possível.

4.4.1.4.2 Passageiros & Cidades

Este procedimento consiste em uma busca local 2-*swap* que é aplicada na sequência de cidades de uma dada solução, *sol*. O objetivo é maximizar o total de passageiros capazes de embarcar, ou seja, passageiros cuja cidade de origem é visitada antes que a cidade de destino e que o custo da viagem desde sua origem até seu destino, considerando que todos os assentos de carro estão ocupados, respeita o seu limite financeiro. A sequência de veículos utilizados permanece inalterada. A sequência de cidades visitadas é definida pela combinação que leva à melhor solução possível.

4.4.1.4.3 *LKH* Passageiros

Dada uma solução *sol* que utiliza c veículos, esta técnica consiste em dividir a solução em subproblemas. O conjunto de cidades visitadas por cada carro c permanece inalterada, apenas a sequência de visitas sofrerá alteração.

Cada carro c vai resolver o problema do PCV em seu conjunto de cidades, em que a matriz de custos é composta pelo custo de cada aresta (i, j) dividido pela demanda de passageiros cuja origem é i e destino j e, nos casos aplicáveis, também pelo custo da taxa

de devolução do veículo, $i, j \in N, i \neq j$.

A heurística Lin-Kernighan (1973), considerada uma das melhores heurísticas para solucionar problemas do *PCV* (KARAPETYAN; GUTIN, 2011), é utilizada para solucionar os subproblemas gerados neste procedimento. O esquema geral desta busca local é ilustrado na Figura 12.

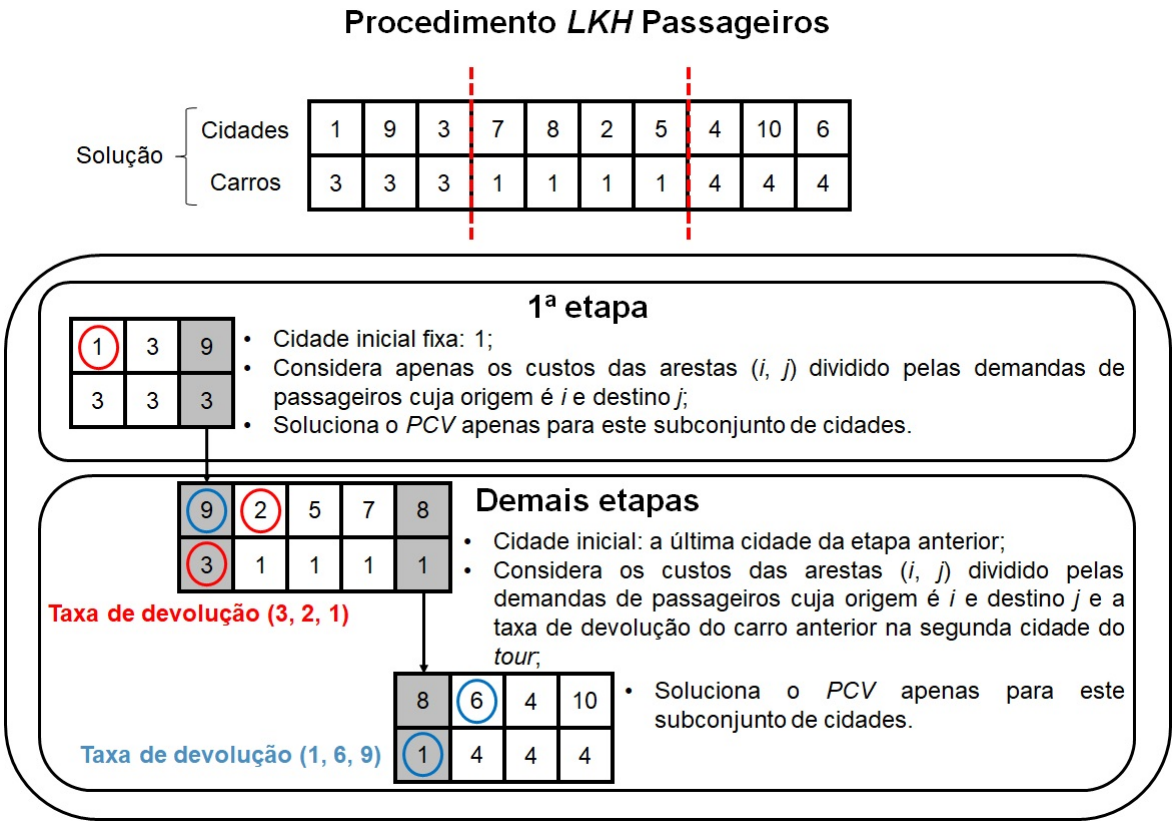

Figura 12: Busca Local – *LKH* Passageiros

4.4.1.4.4 2-*opt*

Consiste em uma versão adaptada do algoritmo de busca local tradicional 2-*opt*, proposto por Croes (1958). Dada uma solução, *sol*, este procedimento reorganiza cada par de arcos atravessados, mas mantém a ordem dos veículos usados. A sequência de arcos atravessados, ou seja a ordem em que as cidades são visitadas, é definida pela combinação que leva à melhor solução possível.

4.4.1.4.5 Altera Cidades sem Demandas

Dada uma solução, *sol*, este procedimento cria uma lista com todas as cidades onde não há passageiros sendo embarcados e nem desembarcados. Em seguida, uma busca local 2-*swap* é aplicada apenas nas cidades desta lista, sem alterar a ordem dos veículos usados.

O resultado é dado pela combinação de cidades que leva ao menor custo possível.

4.4.1.4.6 Altera Passageiros

Este procedimento substitui passageiros embarcados por passageiros não embarcados. Ele cria uma lista com passageiros a bordo que podem ser substituídos por passageiros que não embarcaram. Isso geralmente acontece em trechos do *tour* em que há uma alta demanda de passageiros, demanda esta que pode ser identificada a partir das origens e destinos dos passageiros do problema. Esta busca local realiza apenas a substituição de passageiros que levam a soluções viáveis, ou seja, que respeitam todas as restrições do problema. O esquema de embarque de passageiros é definido pela solução que leva à melhor solução possível.

4.5 *GRASP + VNS + Path Relinking*

Nesta seção são descritos os procedimentos *GRASP* (seção 4.5.1), *VNS* (seção 4.5.2) e *Path Relinking* (seção 4.5.3). Além disso, na seção 4.5.4 é apresentado o algoritmo proposto neste trabalho que baseia-se na utilização do *GRASP + VNS + Path Relinking*.

4.5.1 *GRASP*

O *GRASP*, também conhecido como *Greedy Randomized Adaptative Search Procedure*, proposto por Feo e Resende (1995), consiste em um procedimento de busca guloso, aleatório e adaptativo. É uma meta-heurística multi-partida, em que cada iteração é composta de duas fases: construção e busca local.

A fase construtiva consiste em gerar uma solução viável iterativamente, um elemento por vez, até que a solução esteja completa. A cada iteração, todos os elementos que podem ser adicionados a solução parcial formam uma Lista Restrita de Candidatos (LRC), formada por elementos candidatos de alta qualidade. O tamanho da LRC é determinado por um fator denominado α, $0 \leq \alpha \leq 1$. Por exemplo, para um problema como o PCV, o tamanho da LRC é dado pela equação (4.9).

$$|LRC| = 1 + \alpha \times (|N| - 1) \tag{4.9}$$

Valores de α próximos a zero conduzem à soluções mais gulosas, o que proporciona intensificação. Já valores de α próximos a um, tendem à aleatorizar as soluções geradas, o que

proporciona diversificação.

A fase de busca local consiste em melhorar as soluções encontradas anteriormente, através de um método de exploração da vizinhança. Algoritmos de busca local definem vizinhanças para cada solução, transformando a solução atual através de uma técnica ou heurística, gerando um conjunto de soluções com características aproximadas.

De acordo com Resende e Ribeiro (2005), um algoritmo básico com abordagem *GRASP* pode ser visto no Algoritmo 11.

Algoritmo 11: *GRASP* Genérico

```
1  Entrada: critério de parada
2  Saída: Solução x*
3  início
4      x* ← ∅;
5      f(x*) ← ∞;
6      enquanto critério de parada não satisfeito faça
7          x ← faseConstrutiva ();
8          x ← buscaLocal (x);
9          se f(x) < f(x*) então
10             x* ← x;
11             f(x*) ← f(x);
12 retorne (x*);
13 fim
```

Alguns exemplos de aplicações do *GRASP* voltado ao roteamento de veículos são vistos em Asconavieta (2011), Goldbarg, Asconavieta e Goldbarg (2012), Menezes (2014), Calheiros (2015), Sabry (2016), Silva (2017) e Filho (2019).

4.5.2 *Variable Neighborhood Search* (*VNS*)

A heurística *VNS* (*Variable Neighborhood Search*) foi proposta por Mladenović e Hansen (1997) como uma estrutura geral para solucionar problemas de otimização. Consiste na realização de mudanças sistemáticas de vizinhanças durante a exploração do espaço de busca.

Esta técnica busca explorar gradativamente todas as vizinhanças definidas sobre a solução corrente, percorrendo as mais próximas e dirigindo-se para as mais distantes. O objetivo é explorar inicialmente a vizinhança de menor custo computacional, e somente se não houver melhorias, partir para a próxima vizinhança. Sempre que uma vizinhança explorada acarretar em melhorias na solução, o procedimento reinicia a busca a partir da primeira vizinhança tomando como base a nova solução. Melhorias são definidas pelo valor

objetivo da solução. De acordo com Hansen e Mladenović (2001), um algoritmo básico com abordagem *VNS* pode ser visto no Algoritmo 12.

Algoritmo 12: *VNS* Genérico

```
1  Entrada: Solução x*, critério de parada, N_k - estruturas de vizinhança
2  Saída: Solução x*
3  início
4      enquanto critério de parada não satisfeito faça
5          k ← 1;
6          enquanto k ≤ k_max faça
7              x ← buscaLocal (x*, N_k);
8              se f(x) < f(x*) então
9                  x* ← x;
10                 f(x*) ← f(x);
11                 k ← 1;
12             senão
13                 k ← k + 1;
14 retorne (x*);
15 fim
```

Alguns exemplos de aplicações do *VNS* ou variantes voltado ao roteamento de veículos são vistos em Asconavieta (2011), Goldbarg, Asconavieta e Goldbarg (2012), Menezes (2014), Bastos (2017), Rios (2018) e Filho (2019).

4.5.3 *Path Relinking*

Heurística originalmente proposta por Glover (1996) como estratégia de intensificação para explorar trajetórias que conectam soluções de elite, ou seja, soluções de alta qualidade, obtidas pelos métodos Busca Tabu ou *Scatter Search* em problemas de otimização combinatória (RESENDE; RIBEIRO, 2016).

Considerando a solução base, x^b, e a solução alvo, x^a, calcula-se a diferença simétrica entre ambas, dada por $\Delta\ (x^b, x^a)$, ou seja, o conjunto de movimentos necessários para atingir x^a a partir de x^b. O caminho entre soluções é gerado ligando x^b e x^a. A melhor solução, x^*, encontrada neste caminho é retornada pelo procedimento.

O *Path Relinking* é bastante utilizado como estratégia complementar ao *GRASP*, conforme pode ser visto em Resende e Ribeiro (2005, 2016), Resende *et. al* (2010) e Ribeiro e Resende (2012).

Alguns exemplos de aplicações do *Path Relinking* voltado ao roteamento de veículos são vistos em Resende e Ribeiro (2003), Ho e Gendreau (2006), Usberti, França e França (2013), Menezes (2014) e Radojicić e Marić (2018).

4.5.4 *GRASP + VNS + Path Relinking* Proposto

A técnica que está sendo proposta neste trabalho é descrita no Algoritmo 13. Recebe como parâmetros de entrada: *nomeInstancia* – instância que será solucionada e α – fator que define o quão aleatório ou guloso será o procedimento. A variável x^* representa a melhor solução encontrada. As funções *geraSolucao()* (linha 8), *VNS()* (linha 9) e *PathRelinking()* (linha 10) são apresentadas, respectivamente, nas seções 4.5.4.1, 4.5.4.2 e 4.5.4.3. A condição de parada é dada a partir do número de avaliações da função objetivo, *numAvaliacoes*.

Algoritmo 13: Algoritmo *GRASP* para o *CaRSP*

```
1  Entrada: nomeInstancia, α
2  Saída: x*
3  início
4      G ← leInstancia (nomeInstancia);
5      x* ← ∅;
6      f(x*) ← ∞;
7      enquanto (numAvaliacoes < |N| × |C| × 500) faça
8          x ← geraSolucao (α);
9          x ← VNS (x);
10         x ← PathRelinking (x, x*);
11         se f(x) < f(x*) então
12             x* ← x;
13             f(x*) ← f(x);
14 retorne (x*);
15 fim
```

4.5.4.1 Procedimento *geraSolucao()*

Este procedimento recebe como parâmetro de entrada o fator α, responsável por definir a quantidade de elementos que vão compor a LRC a partir da equação (4.9). A solução é construída de forma iterativa, sendo a LRC a responsável por limitar os possíveis destinos partindo da cidade atual que, inicialmente, é a cidade 1. Considerando que a cidade atual é i, a cada iteração é necessário gerar uma LRC que leve em consideração o custo

de deslocamento de i para j dividido pela demanda de passageiros que existem de i para j, de forma que j ainda não tenha sido visitada e $j \in N, j \neq i$. Esta lista é ordenada de forma crescente e apenas os primeiros elementos vão compor a LRC, obedecendo o tamanho definido inicialmente. A próxima cidade a ser visitada é selecionada a partir de uma roleta que analisa os valores previamente calculados para formar a LRC e dá maiores probabilidades para visitar as cidades que resultaram em menores custos. Este processo é repetido até que todas as cidades sejam visitadas. O procedimento de geração da sequência de carros utilizados é completamente aleatório e segue a mesma estratégia apresentada anteriormente na seção 4.4.1.1, Algoritmo 4. Ao final, o carregamento heurístico de passageiros, apresentado na seção 4.2.2, é aplicado sobre a solução gerada.

4.5.4.2 Procedimento *VNS()*

O *VNS* proposto para o *CaRSP* segue a mesma estrutura apresentada na seção 4.5.2, Algoritmo 12. A estrutura de vizinhanças, N_k, consiste na sequência de procedimentos descrita a seguir e após cada um deles, os passageiros são embarcados usando a heurística mostrada na seção 4.2.2, Algoritmo 2.

1. **Troca Aleatória de Cidades** – apresentado na seção 4.5.4.2.1;
2. **Avalia Carros** – apresentado na seção 4.5.4.2.2;
3. **Passageiros & Cidades** – apresentado na seção 4.4.1.4.2;
4. ***LKH* Passageiros** – apresentado na seção 4.4.1.4.3;
5. **2-*swap* Custos & Devoluções** – apresentado na seção 4.5.4.2.3;
6. **2-*opt*** – apresentado na seção 4.4.1.4.4.

4.5.4.2.1 Troca Aleatória de Cidades

Dada uma solução s, este procedimento consiste, basicamente, em selecionar aleatoriamente uma cidade i de $s.Cidades$ e analisar qual cidade j pode ser trocada por i, de forma que resulte no menor *tour* possível, considerando que $j \in N, j \neq i$. A solução gerada, s', sofre o carregamento heurístico de passageiros, apresentado na seção 4.2.2, e se obtiver uma aptidão melhor que a solução inicial, s, então s' se torna a solução atual e o processo se repete. Caso contrário, o procedimento é encerrado. Ou seja, o procedimento se repete enquanto houver melhorias. A sequência de carros é mantida. Ao final, é retornada a melhor solução encontrada durante o procedimento.

4.5.4.2.2 Avalia Carros

A estratégia utilizada por este procedimento é descrita no Algoritmo 14. Uma solução s é dada como parâmetro de entrada. Dois carros distintos, *carroA* e *carroB*, são selecionados aleatoriamente na linha 5. O procedimento *defineTrechos()*, presente nas linhas 6 e 7, define o intervalo da solução s em que o *carroA* e o *carroB* estão sendo utilizados, ou seja, o índice de início e de fim de ambos.

Algoritmo 14: Busca Local – Avalia Carros

```
1  Entrada: s
2  Saída: s*
3  início
4      s* ← s;
5      carroA, carroB ← selecionaCarrosAleatoriamente (1, |C|);
6      inicioA, fimA ← defineTrechos (carroA);
7      inicioB, fimB ← defineTrechos (carroB);
8      se (inicioA = −1) e (inicioB = −1) então
9          s1 ← AdicionaCarro (s*, carroA);
10         s2 ← AdicionaCarro (s*, carroB);
11         s3 ← AdicionaCarro (s1, carroB);
12     senão se (inicioA ≠ −1) e (inicioB = −1) então
13         s1 ← TrocaExternaCarros (s*, inicioA, fimA, carroB);
14     senão se (inicioA = −1) e (inicioB ≠ −1) então
15         s1 ← TrocaExternaCarros (s*, inicioB, fimB, carroA);
16     senão
17         s1 ← TrocaInternaCarros (s*, inicioA, fimA, inicioB, fimB);
18         s2 ← RemoveCarro (s*, carroA);
19         s3 ← RemoveCarro (s*, carroB);
20     s* ← retornaMelhorSolucao (s1, s2, s3);
21     s* ← AlteraAluguelDevolucao (s*);
22 retorne (s*);
23 fim
```

O primeiro condicional, apresentado na linha 8, corresponde ao caso em que os carros selecionados não estão sendo utilizados na solução s. Nesta situação, criam-se três soluções: adicionando apenas o *carroA* (linha 9); adicionando apenas o *carroB* (linha

10) e adicionando ambos os carros (linha 11). A função *AdicionaCarro()* segue a mesma estratégia apresentada na seção 4.4.1.3.2. O segundo condicional, presente na linha 12, considera o caso onde o *carroA* é utilizado em s, mas o *carroB* não. Então, aplica-se o procedimento *TrocaExternaCarros()*, baseado no procedimento visto na seção 4.4.1.3.4. A situação contrária, onde apenas o *carroB* é utilizado, é descrita nas linhas 14 e 15. O último condicional, mostrado na linha 16, considera o caso onde ambos os carros estão sendo utilizado em s. Neste caso, criam-se três soluções: trocando os trechos onde se utilizam *carroA* e *carroB* (linha 17); removendo a utilização do *carroA* (linha 18) e removendo a utilização do *carroB* (linha 19). Os procedimentos *TrocaInternaCarros()* e *RemoveCarro()* são inspirados, respectivamente, nos procedimentos descritos nas seções 4.4.1.3.3 e 4.4.1.3.1. O procedimento *retornaMelhorSolucao()*, presente na linha 20, define qual foi a melhor solução encontrada e armazena em s^*. Em seguida, na linha 21, aplica-se em s^* o procedimento *AlteraAluguelDevolucao()*, que se baseia no método apresentado na seção 4.4.1.3.5. Por fim, a melhor solução encontrada é retornada.

4.5.4.2.3 2-*swap* Custos & Devoluções

Este procedimento é uma versão da tradicional busca local 2-*swap* que analisa a troca de cada par de cidades, i e j, de uma solução s e verifica se resulta em uma solução melhor, de forma que $i, j \in N, i \neq j$. Para isto, considera o custo das arestas percorridas e, quando há troca de veículos em pelo menos uma das cidades, também considera o custo da devolução dos carros. Um esquema deste procedimento é ilustrado na Figura 13.

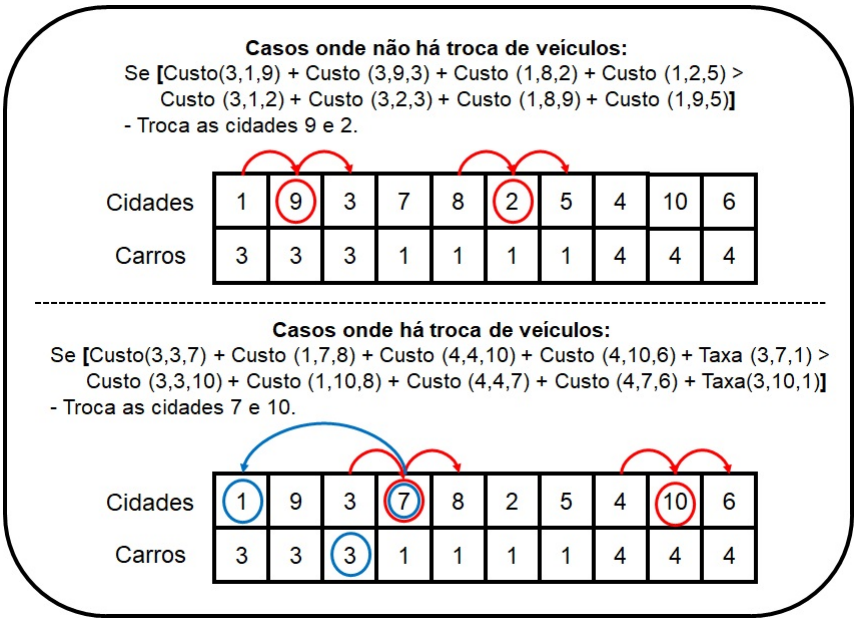

Figura 13: Busca Local – 2-*swap* Custos & Devoluções

4.5.4.3 Procedimento *PathRelinking()*

Este procedimento recebe duas soluções como parâmetro de entrada. A solução que possui melhor aptidão é considerada a solução alvo e a outra é definida como solução base. O conjunto de movimentos necessários para atingir a solução alvo, partindo da solução base, possui tamanho $|N| - 1$.

Inicialmente, a solução intermediária é igual a solução base. Soluções intermediárias são construídas iterativamente, herdando, a cada iteração, a cidade e o carro correspondente da solução alvo, conforme ilustrado na Figura 14.

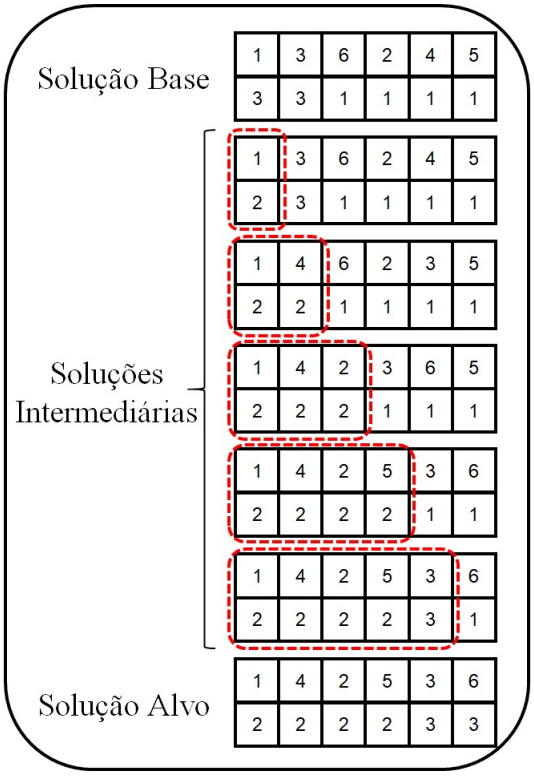

Figura 14: *CaRSP – Path Relinking*

Em seguida, é necessário reparar possíveis repetições de cidades e sequências inviáveis de veículos. A ordem de cidades é corrigida a partir da substituição de cada cidade repetida por uma que não foi visitada, de forma que resulte em custos mais baixos para percorrer os arcos incidentes àquele vértice.

Sequências inviáveis de veículos são caracterizadas pelo uso de um mesmo veículo em dois trechos não adjacentes. A solução para estes casos é analisar se é mais vantajoso prolongar o uso do carro herdado até a sua próxima aparição ou antecipar o uso do próximo carro até gerar uma solução viável. A opção que resultar em uma solução de menor custo é escolhida. Este processo ocorre conforme ilustrado na Figura 15. Para cada

solução intermediária gerada, os passageiros são embarcados usando a heurística mostrada na seção 4.2.2, Algoritmo 2.

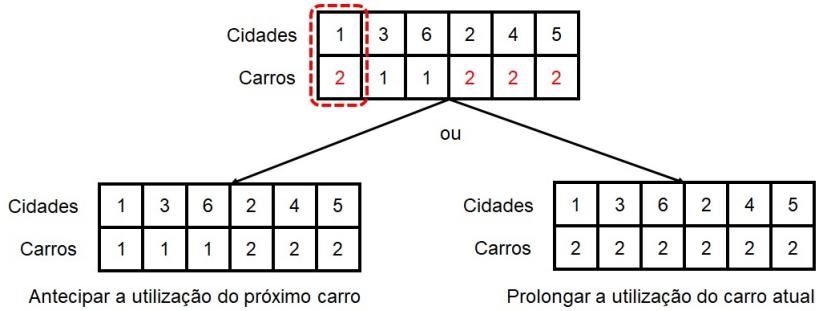

Figura 15: *CaRSP – Path Relinking* – Reparação de Carros Inviáveis

4.6 Algoritmo de Colônia de Formigas

O Algoritmo de Colônia de Formigas (ou *ACS – Ant Colony System*) foi originalmente introduzido por Dorigo, Maniezzo e Colorni (1991) e consiste em uma meta-heurística baseada em população inspirada no comportamento forrageiro de formigas.

As formigas possuem uma estratégia para percorrer o caminho mais curto entre o seu formigueiro e uma fonte de alimento que baseia-se na segregação de feromônios para marcar o caminho percorrido aos restantes membros da colônia.

Este método de comunicação entre as formigas, através de trilhas de feromônios, acaba coletivamente resolvendo problemas complexos. Este princípio da natureza tem sido útil na construção de algoritmos para resolução de problemas computacionais complexos, como por exemplo, o problema do caixeiro viajante, escalonamento, roteamento e quadrático de alocação (ASCONAVIETA, 2011). De acordo com Dorigo e Stützle (2004), a estrutura geral da meta-heurística *ACS* é descrita no Algoritmo 15.

Algoritmo 15: Algoritmo de Colônia de Formigas Genérico

```
1  Entrada: critério de parada
2  Saída: melhor solução
3  início
4      Iniciar trilha de feromônios;
5      enquanto critério de parada não satisfeito faça
6          Construir soluções com formigas;
7          Aplicar busca local; /*etapa opcional*/
8          Atualizar matriz de feromônios;
9  retorne (melhor solução);
10 fim
```

Cada formiga irá construir uma solução visitando os diversos vértices do grafo partindo de um vértice i. Ela escolhe probabilisticamente o próximo vértice j entre os vizinhos possíveis. A probabilidade da formiga k, que está no vértice i, de escolher j como seu próximo vértice é calculada pela equação (4.10).

$$p_{ij}^k = \begin{cases} \frac{(\tau_{ij})^\alpha (\eta_{ij})^\beta}{\sum_{l \in N_i^k} (\tau_{il})^\alpha (\eta_{il})^\beta}, & se\ j \in N_i^k \\ 0, & caso\ contrário \end{cases} \tag{4.10}$$

τ_{ij} e η_{ij} representam, respectivamente, a quantidade de feromônio associado à aresta (i, j) e o valor heurístico que representa a atratividade da formiga visitar o vértice j após visitar o vértice i. O valor de η_{ij} é inversamente proporcional à distância entre os vértices i e j, dada por d_{ij}, conforme mostrado na equação (4.11).

$$\eta_{ij} = \frac{1}{d_{ij}} \tag{4.11}$$

Os parâmetros α e β são ajustados para determinar a influência do feromônio e da informação heurística. A vizinhança factível da formiga k, ou seja, o conjunto de vértices que ainda não foram visitados pela formiga k, é representado por N_i^k. No final do *tour*, quando todas as formigas construíram a sua solução, ocorre a atualização do feromônio, τ_{ij}, caracterizada por duas fases: o depósito e a evaporação. Ao longo da trilha (i, j), a formiga deposita na aresta uma quantidade da substância definida conforme a equação (4.12).

$$\tau_{ij} = \underbrace{(1-\rho)\tau_{ij}}_{\text{Evaporação}} + \underbrace{\sum_{k=1}^{m} \Delta\tau_{ij}^k}_{\text{Depósito}} \tag{4.12}$$

A evaporação evita que acúmulos de feromônios cresçam indefinidamente e possibilita o esquecimento de decisões realizadas no passado. O parâmetro ρ é a uma informação heurística referente a taxa de evaporação do feromônio cujo valor está contido no intervalo $[0, 1]$. A quantidade de feromônio depositada pela formiga k na trilha entre i e j é representado por $\Delta\tau_{ij}^k$ e pode ser obtida a partir da equação (4.13).

$$\Delta\tau_{ij}^k = \begin{cases} \frac{Q}{L_k}, & se\ (i, j) \in S_k \\ 0, & caso\ contrário \end{cases} \tag{4.13}$$

O parâmetro Q é uma constante e L_k é o comprimento total do *tour* da k-ésima formiga, portanto o depósito é realizado caso o trecho (i, j) pertença a S_k, ou seja ao *tour*

construído pela formiga k. O critério de parada pode ser definido por um número máximo de iterações ou pelo evento da estagnação. O algoritmo torna-se estagnado a partir do momento em que todas as formigas convergem para o mesmo percurso em consequência do excessivo crescimento de feromônio nas arestas pertencentes a uma rota subótima.

Alguns exemplos de aplicações do *ACS* voltado ao roteamento de veículos são vistos em Asconavieta (2011), Calheiros (2015), Araújo (2016) e Filho (2019).

4.6.1 Algoritmo de Colônia de Formigas Proposto

O algoritmo que está sendo proposto neste trabalho é descrito no Algoritmo 16. Recebe como parâmetros de entrada: *nomeInstancia* – instância que será solucionada, α – controle de influência do feromônio, β – controle de influência da informação heurística da rota, γ – controle de influência da informação heurística dos passageiros e ρ – taxa de evaporação do feromônio. O número de formigas na colônia é $|N|$. A condição de parada é dada a partir do número de avaliações da função objetivo, *numAvaliacoes*.

Algoritmo 16: Algoritmo de Colônia de Formigas para o *CaRSP*

```
1  Entrada: nomeInstancia, α, β, γ, ρ
2  Saída: x*
3  início
4      G ← leInstancia (nomeInstancia);
5      τ ← iniciaFeromonio(G);
6      enquanto (numAvaliacoes < |N| × |C| × 500) faça
7          colonia ← geraFormigas (|N|, α, β, γ);
8          colonia ← buscaLocal (colonia);
9          x* ← retornaMelhorFormiga (colonia);
10         τ ← atualizaFeromonio (x*, τ, ρ);
11 retorne (x*);
12 fim
```

Os procedimentos *iniciaFeromonio()* (linha 5), *geraFormigas()* (linha 7), *buscaLocal()* (linha 8) e *atualizaFeromonio()* (linha 10) são apresentados, respectivamente, nas seções 4.6.1.1, 4.6.1.2, 4.6.1.3 e 4.6.1.4. Já o procedimento *retornaMelhorFormiga()*, presente na linha 9, retorna a formiga da colônia com a melhor aptidão.

Para o problema proposto, a decisão da formiga não se limita apenas à selecionar uma aresta (i, j) a ser percorrida, mas também define o carro, c, que será utilizado neste trecho. Assim, para algumas variáveis, surge um novo índice referente ao veículo escolhido: p_{ij}^{ck}, τ_{ij}^{c} e η_{ij}^{c}.

Ao longo do procedimento, gera-se uma lista chamada *listaElite*, composta de todos

os trechos adjacentes de soluções encontradas ao longo da execução que possuam lotação ultrapassando 50% da capacidade do veículo utilizado. Esta lista não permite a inserção de trechos repetidos e é utilizada pela busca local apresentada na seção 4.6.1.3.

4.6.1.1 Procedimento *iniciaFeromonio()*

Seguindo a proposta de Dorigo e Stützle (2004), o feromônio inicial é definido a partir da equação (4.14). Onde $n = |N|$, ou seja, o número de cidades e C^{nn} corresponde à média dos custos dos *tours* gerados pela heurística do vizinho mais próximo utilizando cada carro.

$$\tau_0 = \frac{1}{n \times C^{nn}} \tag{4.14}$$

4.6.1.2 Procedimento *geraFormigas()*

Este procedimento é responsável por gerar $|N|$ formigas que, por sua vez, vão compor a colônia. Cada uma destas formigas representa uma solução e começa seu *tour* em uma cidade diferente. Dada uma formiga f, o primeiro passo é sortear aleatoriamente, entre 1 e $|C|$, a quantidade de carros que serão utilizados nesta solução e armazenar em $numCarros$. A decisão referente ao carro que será utilizado e a próxima cidade visitada é dada a partir da equação (4.15).

$$p_{ij}^{ck} = \begin{cases} \frac{(\tau_{ij}^c)^\alpha (\eta_{ij}^c)^\beta (\sigma_{ij})^\gamma}{\sum_{l \in N_i^k} (\tau_{il}^c)^\alpha (\eta_{il}^c)^\beta (\sigma_{il})^\gamma}, & se\ j \in N_i^k \\ 0, & caso\ contrário \end{cases} \tag{4.15}$$

A variável η_{ij}^c corresponde ao valor heurístico que representa a atratividade da formiga visitar o vértice j após visitar o vértice i utilizando o carro c e é calculada a partir da equação (4.16). Onde δ_{ij} corresponde à quantidade de passageiros cuja origem e destino são, respectivamente, as cidades i e j.

$$\eta_{ij}^c = \frac{(1 + \delta_{ij})}{d_{ij}^c} \tag{4.16}$$

Já a variável σ_{ij} se refere ao valor heurístico que representa a demanda de passageiros indo da cidade i para j. Partindo da cidade i, este valor é definido para cada cidade não visitada, j, de acordo com o esquema ilustrado na Figura 16, de forma que $j \in N, i \neq j$.

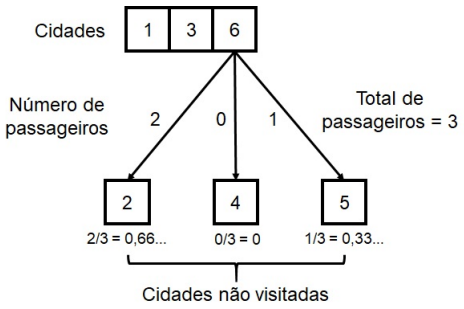

Figura 16: Algoritmo de Colônia de Formigas – Informação Heurística de Passageiros

Para cada cidade escolhida, j, há o processo de embarque dos passageiros aptos desde a cidade inicial até j. Se a quantidade de carros selecionados durante este procedimento atingir o limite definido por $numCarros$, a escolha das próximas cidades considera apenas as informações referentes ao último carro utilizado.

Este procedimento assegura que na cidade 1 seja, obrigatoriamente, alugado um carro. Isto se deve ao fato de que ao final do processo de geração de uma formiga é necessário haver um deslocamento da sequência de cidades e dos carros utilizados para que a cidade inicial seja a cidade 1. Isto impede o surgimento de formigas que não respeitem as características do $CaRSP$, ou seja, inviáveis.

4.6.1.3 Procedimento *buscaLocal()*

Em cada iteração, apenas uma das buscas locais propostas é aplicada sobre cada formiga da colônia e ao final do procedimento, os passageiros são embarcados usando a heurística mostrada na seção 4.2.2, Algoritmo 2. São propostas sete estratégias, são elas:

1. **Insere Trecho de Elite** – apresentação nesta seção;
2. **Troca Aleatória de Cidades** – apresentado na seção 4.5.4.2.1;
3. **Avalia Carros** – apresentado na seção 4.5.4.2.2;
4. **Passageiros & Cidades** – apresentado na seção 4.4.1.4.2;
5. ***LKH* Passageiros** – apresentado na seção 4.4.1.4.3;
6. **2-*swap* Custos & Devoluções** – apresentado na seção 4.5.4.2.3;
7. **2-*opt*** – apresentado na seção 4.4.1.4.4.

O método da roleta é utilizado para selecionar qual das estratégias será aplicada. Inicialmente, as abordagens possuem probabilidades iguais, mas à medida que conseguem melhorar as soluções originais, passam a ter maior prioridade no sorteio.

O procedimento ***Insere Trecho de Elite*** faz uso da variável *listaElite*, relatada na seção 4.6.1. Um elemento desta lista, t, contendo um conjunto de cidades com um alto índice de passageiros a serem embarcados é sorteada. A roleta utilizada considera uma relação proporcional entre o tamanho do trecho e a chance deste ser escolhido.

Dada uma formiga, f, o trecho escolhido, t, é inserido de forma a gerar duas soluções diferentes, f' e f'', seguindo a estratégia ilustrada na Figura 17. Para f', o trecho escolhido é inserido na sequência de cidades de f' a partir da posição da cidade inicial de t. Já para f'', o trecho escolhido é inserido na sequência de cidades de f'' até a cidade final de t. Ao final, há uma etapa de reparação para substituir cada cidade repetida de f' e f'' por uma que não foi visitada, de forma que resulte em custos mais baixos para percorrer os arcos incidentes àqueles vértices.

Figura 17: Algoritmo de Colônia de Formigas – Insere Trecho deElite

4.6.1.4 Procedimento *atualizaFeromonio()*

Este procedimento é feito apenas em função da melhor formiga da colônia, T^{bs}, cujo valor objetivo é dado por C^{bs}. Temos que $\Delta\tau_{ij}^{bs}$ é inversamente proporcional a C^{bs} conforme é mostrado na equação (4.17). A atualização do feromônio é aplicada apenas nos trechos e veículos utilizados por T^{bs} seguindo a equação (4.18). A evaporação, dada por ρ, evita que acúmulos de feromônios cresçam indefinidamente e possibilita o esquecimento de decisões realizadas no passado.

$$\Delta\tau_{ij}^{bs} = \frac{1}{C^{bs}} \tag{4.17}$$

$$\tau_{ij}^{c} = (1-\rho)\tau_{ij}^{c} + \rho\Delta\tau_{ij}^{bs} \qquad \forall i,j \in T^{bs} \tag{4.18}$$

5 Experimentos Computacionais

Os experimentos computacionais são apresentados com detalhes no presente capítulo. O banco de instâncias desenvolvido neste trabalho é explicado na seção 5.1. A seção 5.2 apresenta a metodologia utilizada na etapa experimental. Na seção 5.3 é apresentada uma comparação entre as abordagens exata e heurística para o *PAP*. Os resultados computacionais obtido pelos algoritmos propostos neste trabalho são apresentados nas seções 5.4 (Algoritmos Exatos) e 5.5 (Algoritmos Heurísticos). Uma análise comparativa dos resultados obtidos é apresentada na seção 5.6. Por fim, considerações sobre os resultados obtidos são tecidas na seção 5.7.

5.1 Banco de Instâncias

As instâncias criadas para o *CaRSP*, denominadas *CaRSPlib*, são baseadas nas do *CaRS*, descritas em Goldbarg, Asconavieta e Goldbarg (2012) e estão disponíveis em `http://www.dimap.ufrn.br/lae/downloads/Instances_CaRSP.rar`. As instâncias originais do *CaRS* contém o número de cidades, o número de carros, as matrizes de custos e de retornos de cada veículo. Para criar as instâncias do *CaRSP*, foram adicionadas a capacidade de cada veículo, a origem, o destino e o limite financeiro de cada passageiro.

A capacidade de cada veículo foi gerada a partir de uma distribuição uniforme entre [2,6]. As matrizes de custos e de retornos de cada carro tiveram seus valores originais alterados para balancear as capacidades dos veículos e seus custos operacionais. Esta modificação leva em consideração a capacidade do veículo como mostrado na Tabela 2, onde k^c representa a capacidade do carro c, e *f1* e *f2* são números reais gerados uniformemente entre os limites definidos na Tabela 2.

Tabela 2: Fatores multiplicativos dos elementos das matrizes de custo e de retorno

	Matriz de Custo	Matriz de Retorno
k^c	*f1*	*f2*
2	1,0	1,0
3	[1,1, 1,3]	[1,0, 1,1]
4	[1,3, 1,6]	[1,0, 1,2]
5	[1,6, 1,8]	[1,1, 1,3]
6	[1,8, 2,1]	[1,2, 1,4]

Sejam d_{ij}^c e γ_{ij}^c, respectivamente, o custo operacional e a taxa de retorno do carro c, considerando uma instância original do *CaRS*, $i, j \in N$. Para a instância correspondente do *CaRSP*, estes custos são calculados a partir das equações (5.1) e (5.2), respectivamente, onde $carsp(\cdot)$ corresponde ao novo valor do parâmetro para a instância do *CaRSP*, e $cars(\cdot)$ corresponde ao valor original da instância do *CaRS*.

$$carsp(d_{ij}^c) = cars(d_{ij}^c) \times f1 \tag{5.1}$$

$$carsp(\gamma_{ij}^c) = cars(\gamma_{ij}^c) \times f2 \tag{5.2}$$

A estratégia utilizada para calcular a demanda de passageiros necessitando de caronas em cada cidade é dividida em duas fases. Seja (i, j) a aresta de custo mais alto conectada ao vértice i para o carro c. Primeiramente, são contabilizadas, para cada carro c e vértice inicial i, a quantidade de vezes que j foi considerada a aresta de valor mais alto. Este mesmo processo se repete para calcular as arestas de custos mais baixos. Assim, são construídas duas listas, $EC\_indexes$ e $CC\_indexes$.

A primeira lista contém os índices das cidades ordenados de forma decrescente quanto ao número de vezes que a cidade j foi considerada a mais cara de se visitar. De forma análoga, a segunda lista contém os índices das cidades ordenados de forma decrescente quanto ao número de vezes que a cidade j foi considerada a mais barata de se visitar.

Já a segunda etapa é responsável pela distribuição de passageiros pelas cidades. Nenhum passageiro é associado à 20% das cidades mais baratas (lista $CC\_indexes$). Um passageiro é alocado às cidades cujo índice esteja entre as posições $[0.2n] + 1$ até $[0.3n]$ da lista $CC\_indexes$. Seja MC a capacidade média dos carros da instância em questão, calculado pela equação (5.3), onde $|C|$ representa o total de veículos disponíveis e k^c a capacidade de cada veículo. Um número aleatório de passageiros é selecionado entre $[MC, 2MC]$ para 30% das cidades mais caras (lista $EC\_indexes$). Para os 40% de cidades restantes, são alocados passageiros de forma aleatória entre os limites $[1, MC]$.

$$MC = \frac{\sum_{c=1}^{|C|} k^c}{|C|} \tag{5.3}$$

Resumindo, esta estratégia distribui uma menor quantidade de passageiros para as cidades mais atrativas de se visitar, ou seja, cujos custos sejam estatisticamente menores. Analogamente, um número maior de passageiros são alocados às cidades mais custosas de visitar. Isto se dá a partir da análise feita na primeira fase, onde são preenchidos os vetores $EC\_indexes$ e $CC\_indexes$.

O destino de cada passageiro l é definido por meio do método da roleta, considerando o custos das arestas para cada cidade i. A cidade de destino deve ser, obrigatoriamente, diferente da cidade de origem. Vale ressaltar que neste procedimento, são dadas maiores probabilidades para as arestas de maior custo.

O algoritmo de Dijkstra (1959) foi utilizado para gerar o limite financeiro de cada passageiro l, que é dado por λP onde λ é um número real distribuído uniformemente entre [1, 1.5], e P é o custo do caminho mais curto entre a origem e o destino do passageiro l, utilizando apenas um dos carros disponíveis.

As instâncias foram nomeadas como <id><$|N|$><tipo_distancia><tipo_custo>, onde *id* é uma sequência de caracteres, $|N|$ é o número de cidades, tipo_distancia "e" (Euclidiana) ou "n" (não-Euclidiana), e tipo_custo "s" (simétrico) ou "a" (assimétrico). Por exemplo, a instância nomeada Afeganistao4ns significa que *id* é "Afeganistao", o grafo possui 4 cidades, é uma instância não-Euclidiana e simétrica.

5.2 Metodologia dos Experimentos

Com a finalidade de validar os modelos propostos, os algoritmos exatos foram implementados na linguagem C++, utilizando o *solver* GUROBI 8.0. As execuções se deram em um computador com processador Intel Xeon E5-2698 2.3 GHz x 12, com 32 GB RAM DDR4 e sistema operacional CentOS Linux 6.5. Limitamos o tempo de execução a 80000 segundos.

As heurísticas ingênuas e as meta-heurísticas também foram implementadas em C++, mas para a etapa experimental foi utilizado um computador com processador Intel Core i7-4790 3.6 GHz x 8, com 16 GB RAM DDR4 e sistema operacional Linux Ubuntu 16.04 LTS. Todos os algoritmos heurísticos foram testados em trinta execuções independentes, exceto os algoritmos *NAIVE1* e *NAIVE2* que são executados apenas uma vez.

A condição de parada de todas as meta-heurísticas propostas é dada a partir do número de avaliações da função objetivo, definida empiricamente por $|N| \times |C| \times 500$.

A heurística ingênua *NAIVE2* e a busca local *LKH* Passageiros fazem uso do algoritmo Lin-Kernighan-Helsgaun (HELSGAUN, 2000), uma implementação eficiente da heurística Lin-Kernighan (1973). Foi utilizada a versão *LKH-3.0.3*, lançada em Julho de 2018 e disponível em `http://akira.ruc.dk/~keld/research/LKH/`.

O ajuste de parâmetros, descritos a seguir, foram definidos utilizando a ferramenta *irace*, versão 3.3, disponível em `http://iridia.ulb.ac.be/irace/`. Este *framework* implementa vários procedimentos de configuração automática e oferece procedimentos de ajuste de parâmetros que foram utilizados com sucesso para configurar automaticamente vários algoritmos do estado da arte (LÓPEZ-IBÁÑEZ *et al.*, 2016). Um grupo de instâncias foi gerado com as mesmas características da biblioteca *CaRSPlib* para servir exclusivamente como base para o *irace*.

A configuração da ferramenta se dá a partir da classificação de cada parâmetro associando-o a um dos seguintes domínios: inteiro, real, categórico ou ordinal. Os parâmetros utilizados foram do tipo inteiro e real. A configuração padrão do *irace* foi mantida. A Tabela 3 apresenta a descrição dos parâmetros utilizados. Já a Tabela 4 mostra as configurações sugeridas pelo *irace*. O símbolo "-" significa que esse parâmetro não foi utilizado no algoritmo em questão.

Tabela 3: Parâmetros utilizados pelo *irace* para o ajuste de parâmetros

Parâmetro	Intervalo	Tipo	Algoritmo
tamPop	[100, 200]	Inteiro	Memético
txRep	[0,1, 0,5]	Real	Memético
txMut	[0,1, 0,5]	Real	Memético
txBus	[0,1, 0,5]	Real	Memético
α_{GRASP}	[0,1, 0,9]	Real	*GRASP*
$\alpha_{Formigas}$	[0, 1]	Real	Formigas
β	[0, 1]	Real	Formigas
γ	[0, 1]	Real	Formigas
ρ	[0, 1]	Real	Formigas

Tabela 4: Parâmetros selecionados pelo *irace* para os algoritmos propostos

Algoritmo	*tamPop*	*txRep*	*txMut*	*txBus*	α_{GRASP}	$\alpha_{Formigas}$	β	γ	ρ
Memético	131	0,23	0,24	0,4	-	-	-	-	-
GRASP	-	-	-	-	0,81	-	-	-	-
Formigas	-	-	-	-	-	0,68	0,92	0,41	0,39

A análise do desempenho dos algoritmos usou como métricas a média das soluções e a média dos tempos obtidos nas execuções de cada instância. O teste de Friedman (FRIEDMAN, 1937), um teste estatístico não-paramétrico utilizado para detectar diferenças nos tratamentos em vários experimentos de teste, foi aplicado para verificar diferenças significativas. O nível de significância adotado foi de 0,05.

Uma vez que o teste de Friedman rejeitou a hipótese nula, utilizamos o teste de Nemenyi (NEMENYI, 1963), uma abordagem *post-hoc* destinada a encontrar os grupos de dados que diferem após um teste estatístico de múltiplas comparações, como o teste de Friedman. Este teste faz comparações entre cada par das heurísticas propostas em todos os grupos de instâncias.

5.3 Comparação entre os *PAPs* exato e heurístico

Esta seção tem o propósito de avaliar e comparar a formulações exata e heurística propostas para o *PAP*, apresentadas na seção 4.2. Para isso, dada uma instância, foram geradas 10000 soluções parciais completamente aleatórias, contendo apenas sequência de cidades visitadas e os carros utilizados. Para cada uma delas são aplicadas ambos procedimentos e os resultados obtidos são comparados em função da qualidade da solução encontrada e do tempo de processamento. Os resultados são apresentados no Apêndice I, a partir da Tabela 7.

No que se refere à qualidade das soluções obtidas, é possível observar que, de forma geral, ambas estratégias obtiveram comportamentos semelhantes. Os resultados heurísticos se aproximaram bastante dos exatos, obtendo *GAPs* médios consideravelmente baixos e altíssimas taxas de soluções iguais. Os *GAPs* médios ficaram abaixo de 1% para 88 das 90 instâncias. Já a taxa de soluções iguais ficou acima de 95% para 87 das 90 instâncias.

No que se refere ao tempo de execução, fica evidenciado que a heurística tem um desempenho melhor, principalmente, nas instâncias com até 70 cidades e aproximadamente 200 passageiros. Mas para instâncias com mais cidades e mais passageiros em potencial, a técnica exata passa a obter tempo de execução inferior.

5.4 Algoritmos Exatos

Esta seção descreve os resultados computacionais obtidos pelos seis modelos linearizados propostos neste trabalho (*DFJ1*, *DFJ2*, *MTZ1*, *MTZ2*, *GG1* e *GG2*) que, por sua

vez, foram implementados em um *solver*.

Os testes foram realizados em um grupo de 54 instâncias do *CaRSP* com até 50 cidades, 5 carros e 142 passageiros em potencial. As instâncias estão divididas em Euclidianas (18 instâncias) e não-Euclidianas (36 instâncias). Este mesmo grupo de instâncias também está dividido em simétricas (36 instâncias) e assimétricas (18 instâncias).

Os resultados são mostrados no Apêndice II, a partir das Tabelas 8 e 9. A primeira tabela apresenta os resultados obtidos pelas implementações dos modelos *DFJ1*, *MTZ1* e *GG1* e a segunda tabela relata os resultados obtidos pelas implementações dos modelos *DFJ2*, *MTZ2* e *GG2*.

O resumo dos resultados é apresentado na Tabela 5, composta pelas linhas: *#soluçõesÓtimas* – quantidade de soluções ótimas encontradas. Isto se dá quando o *GAP* é igual a zero; *Tempo Médio* – média dos tempos de execução, em segundos, nas instâncias onde foram capazes de encontrar alguma solução; *#melhoresSoluções* – número de vezes em que a solução encontrada foi melhor que as dos demais métodos; e, *#semSoluções* – total de casos em que não foi capaz de encontrar uma solução. Isto geralmente acontece em situações onde o limite de memória é excedido.

Tabela 5: Resumo dos resultados obtidos pelos algoritmos exatos

		DFJ1	*DFJ2*	*MTZ1*	*MTZ2*	*GG1*	*GG2*
Euclidianas	*#soluçõesÓtimas*	3	4	3	4	4	**5**
	Tempo Médio	70591,30	64289,88	71011,78	61633,20	70339,68	**57396,15**
	#melhoresSoluções	4	6	7	**8**	6	4
	#semSoluções	1	1	**0**	1	1	2
não-Euclidianas	*#soluçõesÓtimas*	6	**8**	6	**8**	7	**8**
	Tempo Médio	70203,56	62556,76	68726,84	63114,74	63434,41	**59776,56**
	#melhoresSoluções	15	**16**	9	**16**	8	10
	#semSoluções	3	2	2	**1**	4	6
Simétricas	*#soluçõesÓtimas*	6	8	6	8	8	**9**
	Tempo Médio	70282,76	63622,09	70051,91	62572,96	65407,11	**58746,75**
	#melhoresSoluções	13	14	11	**16**	11	10
	#semSoluções	3	2	**1**	**1**	4	5
Assimétricas	*#soluçõesÓtimas*	3	**4**	3	**4**	3	**4**
	Tempo Médio	70437,55	62159,21	68418,10	62748,62	66626,35	**59365,75**
	#melhoresSoluções	6	**8**	5	**8**	3	4
	#semSoluções	**1**	**1**	**1**	**1**	**1**	3

De forma geral, podemos dizer que o *GG2* se destacou, em todas as classes de instâncias, quanto aos critérios *#soluçõesÓtimas* e *Tempo Médio*. O *MTZ2* obteve melhor desempenho em todas as classes de instâncias em relação a quantidade de melhores solu-

ções encontradas. E, além disso, juntamente com o *DFJ2* e o *GG2*, também se sobressaiu quanto ao total de soluções ótimas encontradas para as instâncias não-Euclidianas e para as assimétricas. O *DFJ2*, juntamente com o *MTZ2*, obteve melhores resultados quanto ao critério *#melhoresSoluções* nas instâncias não-Euclidianas e nas assimétricas. Quanto ao critério *#semSoluções*, destacam-se o *MTZ1* e o *MTZ2*, obtendo, cada um, os melhores resultados em 3 das 4 classes de instâncias.

5.5 Algoritmos Heurísticos

Esta seção descreve os resultados computacionais obtidos pelos seis algoritmos heurísticos propostos neste trabalho: *NAIVE1*, *NAIVE2*, *NAIVE3*, Algoritmo Memético, *GRASP* e Algoritmo de Colônia de Formigas.

Os testes foram realizados em um grupo de 90 instâncias do *CaRSP* com até 200 cidades, 5 carros e 727 passageiros em potencial. As instâncias estão divididas em Euclidianas (30 instâncias) e não-Euclidianas (60 instâncias). Este mesmo grupo de instâncias também está dividido em simétricas (60 instâncias) e assimétricas (30 instâncias).

Os resultados são mostrados no Apêndice III, a partir das Tabelas 10 e 11. A primeira tabela apresenta os resultados obtidos pelas heurísticas ingênuas e a segunda tabela relata os resultados obtidos pelas meta-heurísticas. Informações sobre o desempenho dos procedimentos que compõem o Algoritmo Memético, o *GRASP*, o *VNS* e o Algoritmo de Colônia de Formigas propostos neste trabalho são apresentadas no Apêndice IV, a partir das Tabelas 12, 13, 14 e 15.

O resumo dos resultados é apresentado na Tabela 6, composta pelas linhas: *#melhoresResultados* – quantidade de vezes em que a solução encontrada foi melhor que as dos demais métodos; *Resultado Médio* – média das soluções obtidas; *#melhoresTempos* – total de vezes em que obteve menor tempo de execução que os demais métodos; e, *Tempo Médio* – média dos tempos de execução, em segundos, nas instâncias onde foram capazes de encontrar alguma solução. A heurística *NAIVE1* foi incapaz de solucionar 33 das 90 instâncias.

De forma geral, podemos dizer que o *GRASP* se destacou, em todas as classes de instâncias, quanto aos critérios *#melhoresResultados* e *Resultado Médio*. O *NAIVE3* obteve melhor desempenho em todas as classes de instâncias tanto em relação a quantidade de melhores tempos obtidos quanto em relação ao tempo médio de processamento. O *NAIVE2*, juntamente com o *NAIVE3*, obteve melhores resultados quanto ao critério *#melhores-*

Tempos nas instâncias não-Euclidianas. Além disso, as meta-heurísticas, comparadas às heurísticas ingênunas, se destacaram em relação às medias das soluções obtidas para todas as classes de instâncias.

Tabela 6: Resumo dos resultados obtidos pelos algoritmos heurísticos

		NAIVE1	*NAIVE2*	*NAIVE3*	Memético	*GRASP*	Formigas
Euclidianas	*#melhoresResultados*	3	3	0	1	**19**	5
	Resultado Médio	*	1898,87	4018,53	1837,86	**1634,10**	1895,84
	#melhoresTempos	0	**15**	**15**	0	0	0
	Tempo Médio	10441,29	265,63	**7,68**	105,36	246,14	115,36
não-Euclidianas	*#melhoresResultados*	7	2	1	5	**40**	6
	Resultado Médio	*	1729,13	1224,93	1048,01	**925,19**	1062,32
	#melhoresTempos	0	26	**30**	0	0	0
	Tempo Médio	9571,66	218,77	**2,79**	100,50	251,66	142,78
Simétricas	*#melhoresResultados*	6	4	0	3	**43**	6
	Resultado Médio	*	1980,38	2730,80	1514,02	**1332,77**	1554,35
	#melhoresTempos	0	29	**31**	0	0	0
	Tempo Médio	10430,20	266,49	**5,47**	111,38	273,81	138,59
Assimétricas	*#melhoresResultados*	4	1	1	3	**20**	5
	Resultado Médio	*	2027,69	1449,61	1230,51	**1110,06**	1262,48
	#melhoresTempos	0	12	**18**	0	0	0
	Tempo Médio	8730,44	196,00	**3,18**	106,88	248,92	159,62

* **A média dos resultados da heurística *NAIVE1* é desconsiderada pelo fato de só ter sido capaz de resolver 57 das 90 instâncias o que, consequentemente, interfere diretamente no valor obtido.**

5.6 Análise Comparativa de Resultados

Esta seção apresenta uma análise estatística acerca dos resultados obtidos pelos algoritmos heurísticos. Os critérios avaliados foram as médias das soluções obtidas e as médias dos tempos de execução dos métodos propostos. Os resultados das análises são apresentados no Apêndice V e estão divididos de acordo com as classes de instâncias: Euclidianas (Tabelas 16 e 17), não-Euclidianas (Tabelas 18 e 19), simétricas (Tabelas 20 e 21) e assimétricas (Tabelas 22 e 23).

De forma geral, a análise constata a superioridade do *GRASP* no que se refere à qualidade das soluções, obtendo *p-valores* inferiores a 0,05 em todos os casos, exceto na comparação com os Algoritmos Memético e de Colônia de Formigas nas instâncias assimétricas.

Além disso, também é possível observar que os Algoritmos Memético e de Colônia de Formigas apresentam desempenho superior em relação às soluções obtidas quando comparados com as heurísticas ingênuas em todos os casos, com exceção das instâncias

Euclidianas, onde resultados inconclusivos aparecem nas comparações entre: Algoritmo Memético × *NAIVE1*; Algoritmo Memético × *NAIVE2*; e, Algoritmo de Colônia de Formigas × *NAIVE1*.

No que se refere ao tempo de execução, o *NAIVE3* merece destaque por ter obtido resultados conclusivos em todos os casos quando comparados aos demais métodos, com exceção, apenas, da comparação com o *NAIVE2*.

O *NAIVE2* obteve melhor desempenho quanto ao tempo computacional quando comparado com os demais métodos em todas as classes de instâncias, com exceção das comparações com o *NAIVE3* e com o Algoritmo Memético nas instâncias Euclidianas e nas assimétricas.

Entre todos os métodos propostos, o *NAIVE1* apresentou a pior performance quanto ao tempo de execução em todos os casos.

5.7 Considerações sobre os Resultados

É possível observar que as implementações dos modelos exatos propostos neste trabalho só conseguem achar a solução ótima para instâncias muito pequenas, com até 14 cidades, 2 carros e 51 passageiros em potencial. Instâncias superiores passam a ter *GAPs* de pelo menos 60%, o que é considerado um valor alto.

Também vale ressaltar que as implementações de todas as modelagens propostas sofrem dificuldades extremas para solucionar as maiores instâncias utilizadas nos experimentos, com 50 cidades, 5 carros e 142 possíveis passageiros. Isto se deve, principalmente, ao limite de memória disponível e evidencia a grandiosidade de variáveis necessárias para solucionar estes problemas. O experimento deixa indícios de que instâncias maiores que as utilizadas não serão capazes de ser solucionadas pelas técnicas exatas apresentadas neste trabalho.

Os resultados heurísticos apresentados em todas as classes de instâncias foram, de certa forma, bastante similares tanto em função da qualidade das soluções obtidas, quanto em função do tempo de execução. É possível concluir que a estratégia de solucionar os subproblemas do *CaRSP* de forma isolada, adotada pelas heurísticas ingênuas propostas neste trabalho, comprometem a qualidade das soluções obtidas. Isto comprova que o problema proposto possui decisões que estão interligadas.

Quanto ao desempenho computacional, temos que o *NAIVE1* apresentou a pior per-

formance entre todos os métodos propostos. O *NAIVE2* apresentou tempos de execuções inferiores quando comparado com os demais métodos em todas as classes de instâncias, com exceção das comparações com o *NAIVE3* e com o Algoritmo Memético nas instâncias Euclidianas e nas assimétricas. Já o *NAIVE3* foi o destaque quanto ao tempo computacional.

Apesar dos testes estatísticos apresentarem resultados conclusivos para a comparação do *NAIVE2* com as meta-heurísticas propostas, é possível observar que isto se dá, principalmente, devido às instâncias com até 50 cidades, 5 carros e 142 passageiros em potencial. Para instâncias com características superiores, o método passou a apresentar uma forte degradação em seu tempo de execução.

Considerando a qualidade das soluções obtidas, é possível afirmar que as meta-heurísticas obtiveram melhores resultados que as heurísticas ingênuas. O *GRASP* merece destaque, superando os demais métodos em quase todos os casos analisados, com exceção, apenas, quando comparado com os Algoritmos Memético e de Colônia de Formigas nas instâncias assimétricas. A comparação entre os Algoritmos Memético e de Colônia de formigas apresentaram resultados inconclusivos.

Entre as meta-heurísticas propostas, o *GRASP* atingiu o tempo de processamento mais alto. Apesar de incomum, é possível observar que isto se deu pela utilização do *VNS* que, apesar de apresentar uma altíssima taxa de melhorias, também consome, em média, mais de 70% do tempo de execução, conforme mostrado na Tabela 14 (Apêndice IV).

6 Considerações finais

Este trabalho apresenta um novo problema, denominado Problema do Caixeiro Viajante com Passageiros (*CaRSP*), uma variação do Problema do Caixeiro Viajante ainda não relatada na literatura. Este estudo lida com o desafio de criar as primeiras abordagens, modelos, métodos, técnicas e algoritmos especializados em sua resolução, buscando garantir que as soluções propostas estejam de acordo com o que é definido para o problema proposto.

O estudo apresenta uma fundamentação teórica acerca de trabalhos dos quais o *CaRSP* se deriva, o Problema do Caixeiro Viajante com Passageiros (*PCV-P*) e o Problema do Caixeiro Viajante Alugador (*CaRS*). Para ambos os problemas foram apresentadas descrição, exemplificação, demonstração de variações, revisão bibliográfica e formulação matemática já existente na literatura.

O *CaRSP* pode representar um modelo de otimização importante para tratar situações típicas de transporte onde se há o aluguel e o compartilhamento de veículos, conceitos que atualmente estão sendo amplamente utilizados neste setor. Pode servir como uma aplicação relevante ao sistema de carona com compartilhamento de veículos, especialmente para carros particulares, uma vez que o modelo não é afetado pelas restrições impostas aos sistemas de carona que cobram taxas e permitem explorações lucrativas.

O problema proposto tem o potencial de aumentar a taxa de ocupação de veículos em estradas ou cidades, barateando custos de transporte causados por troca de carro e compartilhamento de despesas. Além disso, também minimiza os impactos ambientais causados pela poluição e melhora o conforto dos passageiros reduzindo os engarrafamentos e áreas ocupadas por carros estacionados.

São propostas três formulações matemáticas de natureza não-linear para o *CaRSP*, a partir das quais são desenvolvidos seis modelos linearizados (*DFJ1*, *DFJ2*, *MTZ1*, *MTZ2*, *GG1* e *GG2*) utilizando diferentes técnicas de linearização. Com a finalidade de verificar a efetividade das abordagens propostas, foi desenvolvido um banco composto por 90 instân-

cias dividas em Euclidianas, não-Euclidianas, simétricas e assimétricas. Estas instâncias possuem de 4 a 200 cidades, de 2 a 5 carros e de 10 a 727 passageiros em potencial.

Os testes dos modelos linearizados foram realizados em um grupo de 54 instâncias com até 50 cidades, 5 carros e 142 passageiros em potencial. De forma geral, podemos dizer que os métodos que utilizaram a linearização apresentada na seção 3.3.2, ou seja, *DFJ2*, *MTZ2* e *GG2*, obtiveram um desempenho melhor tanto em função da qualidade das soluções obtidas quanto em relação ao tempo médio de execução em todas as classes de instâncias.

Além disso, é possível observar que as modelagens propostas sofrem dificuldades extremas para solucionar as maiores instâncias utilizadas nos experimentos, deixando indícios de que instâncias com características superiores não serão capazes de ser solucionadas pelas técnicas exatas apresentadas neste trabalho. Isto se deve, principalmente, ao limite de memória disponível e evidencia a grandiosidade de variáveis necessárias para solucionar estes problemas.

Também são propostas três heurísticas ingênuas (*NAIVE1*, *NAIVE2* e *NAIVE3*) e três meta-heurísticas (*GRASP*, Algoritmos Memético e de Colônia de Formigas) para solucionar o *CaRSP*. Os algoritmos são descritos e um experimento computacional realizado sobre uma amostra de 90 instâncias.

Os resultados obtidos pelos algoritmos heurísticos evidenciam que em relação à qualidade das soluções encontradas, é possível afirmar que as meta-heurísticas obtiveram melhores resultados que as heurísticas ingênuas. O *GRASP*, por sua vez, foi o método que mais se destacou neste critério.

Por outro lado, temos que o *NAIVE3* obteve melhores resultados quanto ao tempo de execução. Porém, é possível observar que a estratégia de solucionar os subproblemas do *CaRSP* de forma isolada, adotada pelas heurísticas ingênuas propostas neste trabalho, comprometem a qualidade das soluções obtidas. Isto comprova que o problema possui decisões que estão interligadas.

O problema proposto neste trabalho é novo, logo, diversas questões de pesquisas ainda podem ser abordadas em pesquisas futuras, como a aplicação de outros métodos exatos, heurísticos ou até mesmo a investigação de novas variações.

6.1 Trabalhos Futuros

Em virtude da característica inovadora do modelo examinado, diversas questões de pesquisas podem ser investigadas em futuros trabalhos, dentre as quais destacam-se:

1. Definir outros modelos e estratégias de solução exatos mais eficientes que possam ser adaptadas ao problema;
2. Desenvolver novas heurísticas ingênuas baseadas nas melhores técnicas propostas para o *CaRS* relatadas na literatura. Podem ser utilizadas técnicas puramente meta-heurísticas como Algoritmos Científicos (FELIPE, 2014) e estratégias híbridas como Algoritmos Científicos com procedimentos de busca local adaptativos (RIOS; GOLDBARG; QUESQUÉN, 2017);
3. Estudar e adaptar outras meta-heurísticas ao *CaRSP*, tal como *Simulated Annealing*, Algoritmos Transgenéticos e Algoritmos Científicos;
4. Melhorar os operadores e soluções apresentadas no presente estudo;
5. Pesquisar e analisar estratégias que diminuam o tempo de execução das buscas locais propostas;
6. Permitir a utilização de outras formas de transporte durante o percurso, a exemplo do transporte público, carros elétricos ou até mesmo veículos autônomos;
7. Elaborar variações do *CaRSP* com características de demandas dinâmicas, aproximando ainda mais o problema da realidade.

Referências

ABLA. Anuário Brasileiro do Setor de Locação de Veículos 2019. Associação Brasileira das Locadoras de Automóveis, São Paulo/SP, 01 de fev. de 2019. Disponível em: http://www.virapagina.com.br/abla2019/. Acesso em: 15 de jan. 2020.

AGATZ, N. A. H.; ERERA, A.; SAVELSBERGH, M. W. P.; WANG, X. *Optimization for dynamic ridesharing: A review. European Journal of Operational Research*, v. 223, n. 2, p.29–303, 2012.

AGÊNCIA BRASIL. Efeito estufa: transporte responde por 25% das emissões globais. Agência Brasil, Brasília/DF, 11 de dez. de 2018. Disponível em: https://agenciabrasil.ebc.com.br/geral/noticia/2018-12/efeito-estufa-transporte-responde-por-25-das-emissoes-globais. Acesso em: 14 de nov. de 2019.

AMEY, A.; ATTANUCCI, J.; MISHALANI, R. *Real-time ridesharing: opportunities and challenges in using mobile phone technology to improve rideshare services. Transportation Research Record*, v. 2217, n. 1, p. 103–110, 2011.

APPLEGATE, D. L.; BIXBY, R. E.; CHVATAL, V.; COOK, W. J. *The Traveling Salesman Problem. Princeton University Press, New Jersey*, 2007.

ARAÚJO, G. F. Algoritmos Meta-heurísticos Para a Solução do Problema do Caixeiro Viajante com Múltiplas Caronas. Dissertação (Mestrado em Ciência da Computação) – Universidade Federal do Rio Grande do Norte, Natal/RN, 2016.

ASCONAVIETA, P. H. O Problema do Caixeiro Alugador: Um Estudo Algorítmico. Tese (Doutorado em Ciência da Computação) – Universidade Federal do Rio Grande do Norte, Natal/RN, 2011.

BASTOS, R. E. M. O Problema do Caixeiro Viajante com Passageiros e Lotação. Dissertação (Mestrado em Ciência da Computação) – Universidade Federal do Rio Grande do Norte, Natal/RN, 2017.

BRASILTURIS. Aluguel de automóveis aumenta na América Latina e Europa, diz *Mobility*. Brasilturis, São Paulo/SP, 28 de fev. de 2020. Disponível em: https://brasilturis.com.br/aluguel-de-automoveis-aumenta-na-america-latina-e-europa-diz-mobility/. Acesso em: 09 de mar. de 2020.

CALHEIROS, Z. S. A. Algoritmos Evolucionários na Solução do Caixeiro Viajante com Caronas. Monografia (Bacharelado em Ciência da Computação) – Universidade Federal do Rio Grande do Norte, Natal/RN, 2015.

CALHEIROS, Z. S. A. O Problema do Caixeiro Viajante com Passageiros. Dissertação (Mestrado em Ciência da Computação) – Universidade Federal do Rio Grande do Norte, Natal/RN, 2017.

CITYLAB. *The Great Divide in How Americans Commute to Work*. CityLab, Albany/NY, 22 de jan. de 2019. Disponível em: https://www.citylab.com/transportation/2019/01/commuting-to-work-data-car-public-transit-bike/580507/. Acesso em: 15 de jul. de 2019.

CROES, G. A. *A method for solving traveling salesman problems. Operations Research*, vol. 6(6), pp. 791–812, 1958.

DANTZIG, G. B.; FULKERSON, D. R.; JOHNSON, S. M. *Solution of a large-scale traveling salesman problem. Journal of the Operations Research* v. 2, n. 4, p. 393–410, 1954.

DIGALAKIS, J.; MARGARITIS, K. *Performance comparison of memetic algorithms. Journal of Applied Mathematics and Computation, Elsevier Science*, v. 158, p. 237–252, 2004.

DIJKSTRA, E. *A note on two problems in connexion with graphs. Numerische Mathematik*, v. 1, p. 269-271, 1959.

DORIGO, M.; MANIEZZO, V.; COLORNI, A. *The ant system: an autocatalytic optimizing process, Technical Report TR91-016, Politecnico di Milano*, 1991.

DORIGO, M.; STÜTZLE, T. *Ant Colony Optmization. Bradford Book, MIT Press*, 2004.

FEO, T. A.; RESENDE, M. G. *Greedy randomized adaptive search procedures. Journal of global optimization, Springer*, v. 6, n. 2, p. 109–133, 1995.

FELIPE, D. Algoritmos Científicos. Dissertação (Mestrado em Ciência da Computação) – Universidade Federal do Rio Grande do Norte, Natal/RN, 2014.

FILHO, J. G. L. Problema do Caixeiro Via jante com Coleta Opcional de Bônus, Tempo de Coleta e Passageiros. Tese (Doutorado em Ciência da Computação) – Universidade Federal do Rio Grande do Norte, Natal/RN, 2019.

FRIEDMAN, M. *The use of ranks to avoid the assumption of normality implicit in the analysis of variance, Journal of the American Statistical Association*, vol. 32(200), p. 675–701, 1937.

GAREY, M. R.; JOHNSON, D. S. *Computers and intractability: a guide to the theory of np-completeness. W. H. Freeman, San Francisco*, 1979.

GAVISH, B; GRAVES, S. C. *The Travelling Salesman Problem and Related Problems, Working paper GR-078-78, Operations Research Center, Massachusetts Institute of Technology*, 1978.

GLOVER, F.; WOOSLEY, E. *Converting the 0–1 polynomial programming problem to a 0–1 linear program. Operations Research* v. 22, n. 1, p. 180–182, 1974.

GLOVER, F. *Tabu search and adaptive memory programing – Advances, applications and challenges. In: Interfaces in Computer Science and Operations Research*, p. 1–75, 1996.

GOLDBARG, M. C.; ASCONAVIETA, P. H.; GOLDBARG, E. F. G. *Memetic algorithm for the traveling car renter problem: an experimental investigation. Memetic Computing, Springer-Verlag*, v. 4, n. 2, p. 89–108, 2012.

GOLDBARG, M. C.; GOLDBARG, E. F. G.; LUNA, H. P. L.; MENEZES, M. S.; CORRALES, L. *Integer programming models and linearizations for the traveling car renter problem. Optimization Letters* v. 12, n. 4, 743–761, 2017.

GOLDBARG, M. C.; GOLDBARG, ASCONAVIETA, P. H.; E. F. G.; MENEZES, M. S.; LUNA, H. P. L. *A transgenetic algorithm applied to the Traveling Car Renter Problem. Expert Systems with Applications*, v. 40, p. 6298–6310, 2013.

GOLDBARG, M. C.; LUNA, H. L. P. Otimização Combinatória e Programação Linear. 3ª ed., Elsevier, Rio de Janeiro/RJ, 2005.

HANSEN, P.; MLADENOVIĆ, N. *Variable neighborhood search: Principles and applications. European Journal of Operational Research*, v. 130, n. 3, p. 449–467, 2001.

HELSGAUN, K. *An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic. European Journal of Operational Research*. v. 126, n. 1, p. 106–130, 2000.

HO, S. C.; GENDREAU, M. *Path relinking for the vehicle routing problem. Journal of Heuristics*, v. 12, p. 55–72, 2006.

ILAVARASI, K.; JOSEPH, K. S. *Variants of travelling salesman problem: A survey, In: International Conference on Information Communication and Embedded Systems (ICICES)*, p. 1–7. 2014.

KARAPETYAN, D.; GUTIN, G. *Lin-Kernighan heuristic adaptations for the generalized traveling salesman problem, European Journal of Operational Research*, v. 208(3), p. 221–232, 2011.

LIN, S.; KERNIGHAN, B. W. *An Effective Heuristic Algorithm for the Traveling Salesman Problem, Operations Research*, v. 21, n. 2, p. 498–516, 1973.

LÓPEZ-IBÁÑEZ, M.; DUBOIS-LACOSTE, J.; CÁCERES, L. P.; STÜTZLE, M. B. T. *The irace package: iterated racing for automatic algorithm configuration, Operations Research Perspectives*, vol. 3, p. 43–58, 2016.

MARQUES, T. S.; CEZARIO, S. F.; GOLDBARG, E. F. G.; GOLDBARG, M. C.; MAIA, S. M. D. M. *Quota Traveling Salesman with Passengers and Collection Time. 8th Brazilian Conference on Intelligent Systems*, p. 299–304, 2019.

MLADENOVIĆ, N.; HANSEN, P. *Variable neighborhood search. Computers & operations research , Elsevier*, v. 24, n. 11, p. 1097–1100, 1997.

MENEZES, M. S. O Problema do Caixeiro Alugador com Coleta de Prêmios: Um Estudo Algorítmico. Tese (Doutorado em Ciência da Computação) – Universidade Federal do Rio Grande do Norte, Natal/RN, 2014.

MILLER, C. E.; TUCKER, A. W.; ZEMLIN, R. A. *Integer programming formulation of traveling salesman problems. Journal of the Association for Computing Machinery*, v. 7, n. 4, p. 326–329, 1960.

MOSCATO, P. *On evolution, search, optimization, genetic algorithms and martial arts: Towards Memetic Algorithm*. [S.l.], 1989.

NEMENYI, P. B. *Distribution-free multiple comparisons, PhD thesis, Princeton University*, 1963.

RADOJICIĆ, N.; MARIĆ, M. *Fuzzy GRASP with path relinking for the Risk-constrained Cash-in-Transit Vehicle Routing Problem. Applied Soft Computing*, v. 72, p. 486–497, 2018.

RAFEH, R.; JABERI, N.: *LinZinc: A Library for Linearizing Zinc Models. Iranian Journal of Science and Technology, Transactions of Electrical Engineering*, v. 40, n. 1, p. 63–73, 2016.

RESENDE, M. G. C; RIBEIRO, C. C. *A GRASP with path-relinking for private virtual circuit routing. Networks: an international journal*, v. 41(2), p. 104–114, 2003.

RESENDE, M. G. C.; RIBEIRO, C. C. *GRASP with Path-Relinking: Recent Advances and Applications. Metaheuristics: Progress as Real Problem Solvers, Springer*, v. 32, p. 29–63, 2005.

RESENDE, M. G. C.; RIBEIRO, C.C.; GLOVER, F. Glover; MARTÍ, R. *Scatter search and path-relinking: Fundamentals, advances, and applications. In: Handbook of metaheuristics, Springer*,v. 1, p. 87–107, 2010.

RESENDE, M. G. C.; RIBEIRO C. C. *Path-relinking. In: Optimization by GRASP. Springer, New York/NY*, v. 1, p. 167–188, 2016.

RIBEIRO, C.C.; RESENDE, M. G. C. *Path-relinking intensification methods for stochastic local search algorithms. Journal of Heuristics*, v. 18, p. 193–214, 2012.

RIOS, B. H. O.; GOLDBARG, E. F. G.; QUESQUÉN, G. Y. O. *A hybrid metaheuristic using a corrected formulation for the Traveling Car Renter Salesman Problem. 2017 IEEE Congress on Evolutionary Computing*, p. 2308–2314, 2017.

RIOS, B. H. O.; GOLDBARG, E. F. G.; GOLDBARG, M. C. *A Hybrid Metaheuristic for the Traveling Car Renter Salesman Problem. 6th Brazilian Conference on Intelligent Systems*, p. 276–281, 2017.

RIOS, B. H. O. Hibridização de Meta-Heurísticas com Métodos Baseados em Programação Linear para o Problema do Caixeiro Alugador. Dissertação (Mestrado em Ciência da Computação) – Universidade Federal do Rio Grande do Norte, Natal/RN, 2018.

ROBERT, F. *Applications de l'álgebre de Boole en recherche opérationnelle. Revue Française d'Informatique et de Rechercher Opérationnelle*, v. 4, p. 17–26, 1960.

SABRY, G. A.; GOLDBARG, M. C.; GOLDBARG, E. F. G. Problema do Caixeiro Viajante Alugador com Passageiros: Uma Abordagem Algorítimica. *In: Proceedings XVIII CLAIO, the Latin-Iberoamerican Conference on Operations Research*, p. 764–771, 2016.

SILVA, J. G. S. Algoritmos de Solução para o Problema do Caixeiro Viajante com Passageiros e Quota. Dissertação (Mestrado em Ciência da Computação) – Universidade Federal do Rio Grande do Norte, Natal/RN, 2017.

USBERTI, F. L.; FRANÇA, P. M.; FRANÇA, A. L. M. *GRASP with evolutionary path-relinking for the capacitated arc routing problem. Computers & Operations Research*, v. 40(12), p. 3206–3127, 2013.

ZHANG, Y.; ZHANG, Y. *Exploring the relationship between ridesharing and public transit use in the United States. 2018 Journal of Environmental Research and Public Health*, vol. 15(8), p. 1–23, 2018.

Apêndice I

A Tabela 7 apresenta uma análise comparativa entre os métodos heurístico e exato para solucionar o *PAP* cujos resultados são discutidos na seção 5.3. As colunas apresentam o nome da instância, o tempo total e médio de execução, em segundos, de cada método, a média entre os *GAPs* obtidos em cada teste, calculado por (6.1), e a porcentagem de vezes que as soluções foram iguais.

$$GAP = \frac{SolucaoHeuristica - SolucaoExata}{SolucaoExata} \times 100 \tag{6.1}$$

Tabela 7: Resultados obtidos para o *PAP*

			Exato		Heurístico		Estatísticas	
Instância	$\|C\|$	$\|L\|$	Tempo Total (s)	Tempo Médio (s)	Tempo Total (s)	Tempo Médio (s)	*GAP* Médio (%)	Resultados Iguais (%)
Afeganistao4e	2	10	17,65	0,001765	3,17	0,000317	10,19	66,95
Afeganistao4na	2	10	13,90	0,001390	2,23	0,000223	0,00	100,00
Afeganistao4ns	2	10	14,46	0,001446	2,26	0,000226	0,07	98,97
Chile6e	3	13	16,74	0,001674	2,77	0,000277	0,11	99,27
Chile6na	3	13	14,59	0,001459	2,62	0,000262	0,00	100,00
Chile6ns	3	13	13,70	0,001370	2,37	0,000237	0,09	98,54
Paquistao7e	3	21	19,28	0,001928	4,30	0,000430	0,45	97,28
Paquistao7na	3	21	14,56	0,001456	3,52	0,000352	0,01	99,84
Paquistao7ns	3	21	15,74	0,001574	3,82	0,000382	0,21	97,83
Nanibia8e	3	23	23,43	0,002343	5,92	0,000592	0,35	95,89
Nanibia8na	3	23	24,25	0,002425	6,70	0,000670	0,04	99,77
Nanibia8ns	3	23	19,60	0,001960	4,83	0,000483	0,03	99,26
Egito9e	4	34	29,69	0,002969	9,36	0,000936	0,97	89,85
Egito9na	4	34	20,84	0,002084	7,85	0,000785	0,01	99,83
Egito9ns	4	34	21,39	0,002139	7,11	0,000711	0,16	97,77
Tanzania9e	3	28	33,91	0,003391	8,62	0,000862	0,33	97,34
Tanzania9na	3	28	16,88	0,001688	5,50	0,000550	0,00	99,99
Tanzania9ns	3	28	17,54	0,001754	5,14	0,000514	0,03	98,83
AfricaSul11e	3	37	27,29	0,002729	10,76	0,001076	0,50	95,77
AfricaSul11na	3	37	19,60	0,001960	7,56	0,000756	0,01	99,74
AfricaSul11ns	3	37	20,66	0,002066	7,58	0,000758	0,01	99,64

Continuação da Tabela 7: Resultados obtidos para o *PAP*

			Exato		Heurístico		Estatísticas	
Instância	$\|C\|$	$\|L\|$	Tempo Total (s)	Tempo Médio (s)	Tempo Total (s)	Tempo Médio (s)	*GAP* Médio (%)	Resultados Iguais (%)
Niger12e	3	33	24,70	0,002470	7,60	0,000760	0,03	99,53
Niger12na	3	33	20,00	0,002000	6,51	0,000651	0,02	99,54
Niger12ns	3	33	21,33	0,002133	6,43	0,000643	0,02	99,01
Ira13e	4	44	28,02	0,002802	11,64	0,001164	0,05	99,14
Ira13na	4	44	19,54	0,001954	8,44	0,000844	0,00	99,96
Ira13ns	4	44	21,26	0,002126	8,45	0,000845	0,01	99,76
BrasilRJ14e	2	51	43,71	0,004371	17,61	0,001761	1,20	89,07
BrasilRJ14na	2	51	29,65	0,002965	14,97	0,001497	0,02	99,70
BrasilRJ14ns	2	51	28,87	0,002887	13,54	0,001354	0,16	98,16
Arabia14e	5	61	28,92	0,002892	17,76	0,001776	0,01	99,48
Arabia14na	5	61	23,15	0,002315	13,52	0,001352	0,00	99,97
Arabia14ns	5	61	24,11	0,002411	13,49	0,001349	0,00	100,00
Argelia15e	3	46	28,36	0,002836	11,72	0,001172	0,04	99,37
Argelia15na	3	46	21,37	0,002137	9,78	0,000978	0,00	99,98
Argelia15ns	3	46	25,59	0,002559	10,72	0,001072	0,01	99,50
Cazaquistao15e	5	58	29,97	0,002997	15,10	0,001510	0,03	99,29
Cazaquistao15na	5	58	21,94	0,002194	11,73	0,001173	0,01	99,79
Cazaquistao15ns	5	58	23,34	0,002334	11,85	0,001185	0,00	99,99
BrasilRN16e	2	47	32,56	0,003256	13,38	0,001338	0,16	97,73
BrasilRN16na	2	47	24,49	0,002449	10,88	0,001088	0,05	98,93
BrasilRN16ns	2	47	26,70	0,002670	11,31	0,001131	0,03	99,09
Australia16e	4	57	32,12	0,003212	16,22	0,001622	0,01	99,71
Australia16na	4	57	30,84	0,003084	15,03	0,001503	0,01	99,64
Australia16ns	4	57	32,91	0,003291	14,47	0,001447	0,01	99,74
China17e	3	51	26,82	0,002682	12,30	0,001230	0,00	99,97
China17na	3	51	21,52	0,002152	10,20	0,001020	0,00	100,00
China17ns	3	51	22,91	0,002291	10,05	0,001005	0,00	99,99
BrasilPR25e	3	65	39,39	0,003939	22,38	0,002238	0,01	99,79
BrasilPR25na	3	65	24,21	0,002421	12,88	0,001288	0,00	100,00
BrasilPR25ns	3	65	29,24	0,002924	15,38	0,001538	0,00	100,00
BrasilAM26e	3	65	38,40	0,003840	21,00	0,002100	0,00	99,95
BrasilAM26na	3	65	31,39	0,003139	14,32	0,001432	0,00	99,96
BrasilAM26ns	3	65	29,56	0,002956	14,17	0,001417	0,00	99,94
Livramento30e	3	89	43,40	0,004340	27,68	0,002768	0,03	98,51
Livramento30na	3	89	31,22	0,003122	18,48	0,001848	0,00	99,62
Livramento30ns	3	89	30,88	0,003088	18,45	0,001845	0,00	99,45
BrasilMG30e	4	94	49,60	0,004960	34,37	0,003437	0,05	97,90
BrasilMG30na	4	94	32,51	0,003251	20,66	0,002066	0,00	99,92

Continuação da Tabela 7: Resultados obtidos para o *PAP*

			Exato		Heurístico		Estatísticas	
Instância	$\|C\|$	$\|L\|$	Tempo Total (s)	Tempo Médio (s)	Tempo Total (s)	Tempo Médio (s)	*GAP* Médio (%)	Resultados Iguais (%)
BrasilMG30ns	4	94	37,52	0,003752	23,03	0,002303	0,00	99,91
BrasilSP32e	4	95	44,17	0,004417	31,06	0,003106	0,04	98,33
BrasilSP32na	4	95	28,60	0,002860	18,72	0,001872	0,00	99,81
BrasilSP32ns	4	95	29,59	0,002959	19,03	0,001903	0,00	99,65
BrasilRS32e	4	95	45,58	0,004558	31,66	0,003166	0,02	98,84
BrasilRS32na	4	95	30,03	0,003003	20,34	0,002034	0,00	99,88
BrasilRS32ns	4	95	33,67	0,003367	22,37	0,002237	0,00	99,90
BrasilCO40e	5	114	51,63	0,005163	39,08	0,003908	0,01	99,10
BrasilCO40na	5	114	32,76	0,003276	23,45	0,002345	0,00	99,89
BrasilCO40ns	5	114	34,19	0,003419	24,71	0,002471	0,00	99,92
BrasilNO45e	5	133	57,25	0,005725	47,61	0,004761	0,01	99,56
BrasilNO45na	5	133	34,12	0,003412	27,61	0,002761	0,00	100,00
BrasilNO45ns	5	133	38,29	0,003829	31,27	0,003127	0,00	99,99
BrasilNE50e	5	118	51,75	0,005175	36,83	0,003683	0,00	99,78
BrasilNE50na	5	142	40,02	0,004002	32,35	0,003235	0,00	99,94
BrasilNE50ns	5	142	41,11	0,004111	33,84	0,003384	0,00	99,91
st70eB	4	204	72,98	0,007298	67,63	0,006763	0,00	99,82
st70naB	4	204	46,13	0,004613	42,56	0,004256	0,00	99,99
st70nsB	4	192	60,67	0,006067	56,40	0,005640	0,00	100,00
rd100eB	4	290	126,50	0,012650	133,90	0,013390	0,00	99,98
rd100naB	4	290	66,55	0,006655	71,57	0,007157	0,00	99,98
rd100nsB	4	290	74,01	0,007401	78,91	0,007891	0,00	99,93
ch130e	5	375	164,54	0,016454	185,50	0,018550	0,00	100,00
ch130na	5	284	68,99	0,006899	71,61	0,007161	0,00	100,00
ch130ns	5	369	94,65	0,009465	106,69	0,010669	0,00	100,00
PortoVelho200e	3	586	235,38	0,023538	282,38	0,028238	0,00	99,89
Cuiaba200e	4	428	186,09	0,018609	180,61	0,018061	0,00	99,98
Aracaju200na	3	711	185,85	0,018585	268,61	0,026861	0,00	100,00
Aracaju200ns	3	686	200,78	0,020078	284,81	0,028481	0,00	100,00
Teresina200na	5	584	143,20	0,014320	190,63	0,019063	0,00	100,00
Teresina200ns	5	727	186,78	0,018678	276,23	0,027623	0,00	100,00

Apêndice II

As Tabelas 8 e 9 apresentam os resultados obtidos na execução da implementação das seis formulações linearizadas em um *solver*. As colunas destas tabelas são descritas a seguir:

- Instância: nome da instância;
- $|C|$: número de carros;
- $|L|$: número de passageiros em potencial;
- *Sol*: solução obtida na execução do algoritmo;
- *GAP*: desvio percentual, calculado pela equação (6.1), do valor mostrado na coluna *Resultado* para o limite inferior, *LB*, calculado pelo *solver*;
- T(s): tempo de execução em segundos.

$$GAP = \frac{Sol - LB}{LB} \times 100 \tag{6.1}$$

O valor "0" na coluna *GAP* indica que a instância foi resolvida de forma ótima. O valor "80000" na coluna T(s) indica que o *solver* encerrou a execução devido ao limite de tempo. O valor "-" nas colunas *Resultado*, *GAP* e T(s) indicam que o *solver* não foi capaz de encontrar uma solução para a instância. Os melhores resultados estão em negrito.

Tabela 8: Resultados obtidos pelo $DFJ1$, $MTZ1$ e $GG1$

			DFJ1			*MTZ1*			*GG1*		
Instância	$\|C\|$	$\|L\|$	*Sol*	*GAP*	T(s)	*Sol*	*GAP*	T(s)	*Sol*	*GAP*	T(s)
Chile6e	3	13	**264,15**	0	15,8	**264,15**	0	17,6	**264,15**	0	18,53
Chile6na	3	13	**458,66**	0	10,9	**458,66**	0	10,8	**458,66**	0	10,60
Chile6ns	3	13	**411,63**	0	13,2	**411,63**	0	9,9	**411,63**	0	12,18
Paquistao7e	3	21	**268,9**	0	219,2	**268,9**	0	191,4	**268,9**	0	205,94
Paquistao7na	3	21	**321,41**	0	63,4	**321,41**	0	75,4	**321,41**	0	50,25
Paquistao7ns	3	21	**230,6**	0	51,9	**230,6**	0	59,1	**230,6**	0	62,06
Egito9e	4	34	**277,6**	0	79817	**277,6**	0	78003	**277,6**	0	77337
Egito9na	4	34	508	33,8	80000	**476,36**	0	43021,6	**476,36**	0	12587,17
Egito9ns	4	34	**443,71**	0	79214	**443,71**	0	53535,7	**443,71**	0	12637,46
AfricaSul11e	3	37	449,5	91,6	80000	457,4	70,7	80000	**297,89**	0	78213

Continuação da Tabela 8: Resultados obtidos pelo $DFJ1$, $MTZ1$ e $GG1$

			DFJ1			*MTZ1*			*GG1*		
Instância	$\|C\|$	$\|L\|$	*Sol*	*GAP*	T(s)	*Sol*	*GAP*	T(s)	*Sol*	*GAP*	T(s)
AfricaSul11na	3	37	**612,91**	0	77364	753	91	80000	753	91,63	80000
AfricaSul11ns	3	37	562,3	95	80000	619,8	90,3	80000	**531,56**	0	76541,3
BrasilRJ14e	2	51	141,8	92,8	80000	**133,2**	78,5	80000	157,58	82,81	80000
BrasilRJ14na	2	51	**141,7**	96,1	80000	153,9	93,2	80000	182,71	94,63	80000
BrasilRJ14ns	2	51	**86,7**	94,1	80000	104,6	90,4	80000	87,13	87,86	80000
Argelia15e	3	46	**660,8**	95,7	80000	1021,4	89,7	80000	831,16	83,74	80000
Argelia15na	3	46	1116,6	97,8	80000	**1065,9**	89	80000	1153,66	91,72	80000
Argelia15ns	3	46	**652,8**	96,2	80000	869,6	87,3	80000	846	89,37	80000
Cazaquistao15e	5	58	2379,2	98,9	80000	997	90,9	80000	**902,31**	92,11	80000
Cazaquistao15na	5	58	1562,9	98,6	80000	2578,3	97,8	80000	**1365,5**	95,72	80000
Cazaquistao15ns	5	58	**1121,5**	98	80000	1572,2	93,8	80000	4662	97,99	80000
BrasilRN16e	2	47	372,8	96,6	80000	**297,7**	86,4	80000	379,98	89,75	80000
BrasilRN16na	2	47	**139,6**	94,9	80000	199,1	94,7	80000	139,65	92,16	80000
BrasilRN16ns	2	47	152,2	95,5	80000	127,2	91,9	80000	**109,65**	90,95	8000
Australia16e	4	57	1597,5	98,3	80000	1375,4	93,7	80000	**1273,75**	93,37	80000
Australia16na	4	57	**1331,9**	97,6	80000	1412,4	94,7	80000	1590,05	95,32	80000
Australia16ns	4	57	1146,7	97,5	80000	1188,1	94,1	80000	**1041,5**	92,95	80000
China17e	3	51	**1498,7**	98,1	80000	1521,4	95,9	80000	1918,5	96,75	80000
China17na	3	51	1352,5	98,1	80000	**1304,2**	93,8	80000	3357,5	97,87	80000
China17ns	3	51	**991,5**	97,4	80000	1610,3	95,7	80000	1998,5	96,33	80000
BrasilPR25e	3	65	877	98,7	80000	**787**	94,7	80000	808,66	95,31	80000
BrasilPR25na	3	65	**307**	98,4	80000	767,5	98,6	80000	350,66	97,58	80000
BrasilPR25ns	3	65	**283,5**	98,3	80000	348	97,1	80000	310,66	97,34	80000
BrasilAM26e	3	65	**637**	98,4	80000	1054	97	80000	927,33	96,44	80000
BrasilAM26na	3	65	**270,5**	98,3	80000	307	97,4	80000	2296	99,66	80000
BrasilAM26ns	3	65	**264**	98,2	80000	282,7	96,6	80000	275	96,43	80000
Livramento30e	3	89	1262,5	99,3	80000	**1069,7**	98	80000	-	-	-
Livramento30na	3	89	992,5	99,4	80000	**925,3**	98,8	80000	1075	98,95	80000
Livramento30ns	3	89	1025	99,4	80000	**989**	98,9	80000	1031,16	99,02	80000
BrasilMG30e	4	94	1016	99,3	80000	**1008**	97,2	80000	1057	97,19	80000
BrasilMG30na	4	94	**375**	99	80000	1055	99,3	80000	1174	99,38	80000
BrasilMG30ns	4	94	**393,5**	99,1	80000	1048,5	99,3	80000	**393,5**	98,17	80000
BrasilRS32e	4	95	967	99,5	80000	**784,5**	98,5	80000	1261,83	99,04	80000
BrasilRS32na	4	95	970	99,6	80000	**493,7**	98,9	80000	1156	99,54	80000
BrasilRS32ns	4	95	**386**	99	80000	962	99,4	80000	1107	99,13	80000
BrasilCO40e	5	114	**1889**	100	80000	**1889**	99,1	80000	2134	99,18	80000
BrasilCO40na	5	114	**1535**	100	80000	**1535**	100	80000	2085	99,46	80000
BrasilCO40ns	5	114	**1535**	100	80000	**1535**	99,3	80000	-	-	-
BrasilNO45e	5	133	-	-	-	**2574**	100	80000	2672	100	80000
BrasilNO45na	5	133	2284	100	80000	**2263**	100	80000	2326	99,55	80000
BrasilNO45ns	5	133	-	-	-	**2284**	99,5	80000	-	-	-
BrasilNE50e	5	118	3014	99,8	80000	2599	99,4	80000	**2411**	99,3	80000
BrasilNE50na	5	142	-	-	-	-	-	-	-	-	-
BrasilNE50ns	5	142	-	-	-	-	-	-	-	-	-

Tabela 9: Resultados obtidos pelo $DFJ2$, $MTZ2$ e $GG2$

			DFJ2			*MTZ2*			*GG2*		
Instância	$\|C\|$	$\|L\|$	*Sol*	*GAP*	T(s)	*Sol*	*GAP*	T(s)	*Sol*	*GAP*	T(s)
Chile6e	3	13	**264,15**	0	13,3	**264,15**	0	11,9	**264,15**	0	16,45

Continuação da Tabela 9: Resultados obtidos pelo *DFJ2*, *MTZ2* e *GG2*

			DFJ2			*MTZ2*			*GG2*		
Instância	$\|C\|$	$\|L\|$	*Sol*	*GAP*	T(s)	*Sol*	*GAP*	T(s)	*Sol*	*GAP*	T(s)
Chile6na	3	13	**458,66**	0	9,8	**458,66**	0	11,9	**458,66**	0	4,2
Chile6ns	3	13	**411,63**	0	5,1	**411,63**	0	3,9	**411,63**	0	4,67
Paquistao7e	3	21	**268,9**	0	46,4	**268,9**	0	28,8	**268,9**	0	25,63
Paquistao7na	3	21	**321,41**	0	17,5	**321,41**	0	20,1	**321,41**	0	17,72
Paquistao7ns	3	21	**230,6**	0	14,8	**230,6**	0	15,9	**230,6**	0	21,27
Egito9e	4	34	**277,6**	0	10504,6	**277,6**	0	3132	**277,6**	0	2593,54
Egito9na	4	34	**476,36**	0	3748,6	**476,36**	0	1111,5	**476,36**	0	1318,73
Egito9ns	4	34	**443,71**	0	7128,1	**443,71**	0	1762,8	**443,71**	0	1116,54
AfricaSul11e	3	37	**297,89**	0	42363,7	**297,89**	0	4591,7	**298,89**	0	7879,9
AfricaSul11na	3	37	**612,91**	0	12930,7	**612,91**	0	25583,1	**612,91**	0	9145,52
AfricaSul11ns	3	37	**531,56**	0	23075,2	**531,56**	0	20506,8	**531,56**	0	21668,23
BrasilRJ14e	2	51	127,1	28,8	80000	119,5	7,1	80000	**114,21**	0	27822,87
BrasilRJ14na	2	51	127	69	80000	**116,5**	60,6	80000	139,25	66,59	80000
BrasilRJ14ns	2	51	**85,63**	50,9	80000	**85,63**	43,9	80000	**85,63**	45,07	80000
Argelia15e	3	46	650,2	66,9	80000	**606,9**	51,4	80000	909,08	64,73	80000
Argelia15na	3	46	**929,9**	80,6	80000	1016,5	74,1	80000	972,23	71,71	80000
Argelia15ns	3	46	**669,7**	73,4	80000	757,2	66,5	80000	795,76	67,46	80000
Cazaquistao15e	5	58	1083,7	79,7	80000	988,2	73,5	80000	**968,58**	76,47	80000
Cazaquistao15na	5	58	**1059,4**	81,7	80000	1436,9	84,2	80000	1305,66	83,06	80000
Cazaquistao15ns	5	58	**863,8**	77,7	80000	1559,6	88,6	80000	1765,28	94,32	80000
BrasilRN16e	2	47	**297,5**	65,2	80000	319,9	59,7	80000	320,63	59,76	80000
BrasilRN16na	2	47	139,6	63,8	80000	**134,6**	56,9	80000	140	59,28	80000
BrasilRN16ns	2	47	**85,2**	39,3	80000	114,3	50,9	80000	136,35	59,4	80000
Australia16e	4	57	1151,5	79,7	80000	**894,9**	66,5	80000	1450	95,79	80000
Australia16na	4	57	**1220,3**	81,4	80000	1306,5	80,1	80000	1566,48	83,88	80000
Australia16ns	4	57	1217	81,7	80000	**970,9**	73,4	80000	1612,63	92,04	80000
China17e	3	51	1372,7	83,6	80000	**1014,7**	75,4	80000	1410,31	82,19	80000
China17na	3	51	**1173**	82,3	80000	1209,4	76,7	80000	1272,5	77,5	80000
China17ns	3	51	1051,9	80,1	80000	947,1	69,8	80000	**929,83**	69,13	80000
BrasilPR25e	3	65	1014	117,2	80000	817	119,8	80000	**814,5**	129,91	80000
BrasilPR25na	3	65	401	114,3	80000	**360**	109,9	80000	494,66	88,54	80000
BrasilPR25ns	3	65	367,5	123,3	80000	368	116,8	80000	**347**	85,84	80000
BrasilAM26e	3	65	1033,5	126,4	80000	**864**	117,3	80000	978	127,54	80000
BrasilAM26na	3	65	374,5	117,2	80000	**289,3**	119,3	80000	407,5	113,66	80000
BrasilAM26ns	3	65	359,5	118,6	80000	340,9	125,6	80000	**339,25**	96,77	80000
Livramento30e	3	89	**1212,3**	127,7	80000	1729	128,1	80000	1761	132,93	80000
Livramento30na	3	89	960,7	117,1	80000	**551**	129,2	80000	655,5	128,33	80000
Livramento30ns	3	89	1128	113,5	80000	**811,5**	116,5	80000	1016	118,7	80000
BrasilMG30e	4	94	1483,5	365,1	80000	**1033,5**	494,9	80000	1377	132,82	80000
BrasilMG30na	4	94	1055	198,3	80000	**904,7**	227,9	80000	1577	112,2	80000
BrasilMG30ns	4	94	1057	192,4	80000	**794,5**	194,1	80000	1228	115,97	80000
BrasilRS32e	4	95	1408	403,9	80000	**989**	546,5	80000	2552	121,98	80000
BrasilRS32na	4	95	1380	218,8	80000	1308	231,9	80000	**678**	137,99	80000
BrasilRS32ns	4	95	1346	233	80000	1390	253,2	80000	**1053**	125,66	80000
BrasilCO40e	5	114	**1880**	593,2	80000	1889	609,8	80000	4640,5	294,66	80000
BrasilCO40na	5	114	1535	497	80000	**1527**	493,9	80000	-	-	-
BrasilCO40ns	5	114	3449	276,2	80000	**2956**	305,5	80000	-	-	-
BrasilNO45e	5	133	-	-	-	**2536**	624,8	80000	-	-	-
BrasilNO45na	5	133	**2284**	383,4	80000	**2284**	379,4	80000	-	-	-
BrasilNO45ns	5	133	**2278**	390,1	80000	2284	386,2	80000	-	-	-
BrasilNE50e	5	118	**3014**	488	80000	-	-	-	-	-	-

Continuação da Tabela 9: Resultados obtidos pelo $DFJ2$, $MTZ2$ e $GG2$

			DFJ2			*MTZ2*			*GG2*		
Instância	$\|C\|$	$\|L\|$	*Sol*	*GAP*	T(s)	*Sol*	*GAP*	T(s)	*Sol*	*GAP*	T(s)
BrasilNE50na	5	142	-	-	-	-	-	-	-	-	-
BrasilNE50ns	5	142	-	-	-	**2228**	451,6	80000	-	-	-

Apêndice III

As 90 instâncias foram executadas trinta vezes para as heurísticas *NAIVE3*, Algoritmo Memético, *GRASP* e Algoritmo Colônia de Formigas. Já o *NAIVE1* e o *NAIVE2*, foram executado apenas uma vez. As Tabelas 10 e 11 apresentam os resultados obtidos pelos algoritmos, onde as colunas *Resultado* e *Tempo (s)* são calculadas, respectivamente, a partir das médias das soluções e dos tempos obtidos, em segundos. O valor "-" nas colunas *Resultado*, *Tempo (s)* indicam que o método não foi capaz de encontrar uma solução para a instância. Os melhores resultados estão em negrito.

Tabela 10: Resultados obtidos pelas heurísticas ingênuas

			NAIVE1		*NAIVE2*		*NAIVE3*	
Instância	$\|C\|$	$\|L\|$	Resultado	Tempo (s)	Resultado	Tempo (s)	Resultado	Tempo (s)
Afeganistao4e	2	10	423,00	0,11	**367,53**	0,05	423,00	0,20
Afeganistao4na	2	10	**310,47**	0,08	**310,47**	0,03	**310,47**	0,21
Afeganistao4ns	2	10	306,07	0,05	**262,90**	0,03	306,07	0,21
Chile6e	3	13	459,67	0,37	459,67	0,16	**400,76**	0,35
Chile6na	3	13	**458,67**	0,73	490,43	0,10	502,83	0,35
Chile6ns	3	13	502,33	0,66	555,00	0,21	**476,15**	0,35
Paquistao7e	3	21	400,80	0,28	**312,25**	0,08	370,54	1,23
Paquistao7na	3	21	374,33	0,54	374,33	0,08	**369,41**	1,57
Paquistao7ns	3	21	463,00	0,65	**322,20**	0,09	392,60	1,33
Nanibia8e	3	23	667,30	1,81	978,17	0,11	**618,67**	1,10
Nanibia8na	3	23	**621,92**	2,18	1798,37	0,11	763,97	1,12
Nanibia8ns	3	23	**623,28**	2,45	1081,70	0,11	626,50	1,01
Egito9e	4	34	630,50	4,45	618,25	0,26	**469,57**	1,26
Egito9na	4	34	**573,34**	2,44	1333,50	0,24	727,66	1,38
Egito9ns	4	34	722,62	6,91	1022,46	0,28	**654,81**	1,14
Tanzania9e	3	28	**393,82**	2,10	1054,17	0,13	572,27	0,42
Tanzania9na	3	28	766,00	4,50	972,00	0,13	**697,92**	0,42
Tanzania9ns	3	28	612,00	4,91	1252,00	0,18	**539,25**	0,42
AfricaSul11e	3	37	561,17	3,07	**528,50**	0,37	771,82	0,59
AfricaSul11na	3	37	**936,67**	17,66	1396,00	0,34	1043,82	0,56
AfricaSul11ns	3	37	**754,42**	24,91	1223,00	0,34	1007,17	0,56
Niger12e	3	33	**627,83**	3,59	963,57	0,21	881,39	1,39
Niger12na	3	33	**1099,92**	12,97	1878,00	0,24	1220,60	1,18

Continuação da Tabela 10: Resultados obtidos pelas heurísticas ingênuas

			NAIVE1		*NAIVE2*		*NAIVE3*	
Instância	$\|C\|$	$\|L\|$	Resultado	Tempo (s)	Resultado	Tempo (s)	Resultado	Tempo (s)
Niger12ns	3	33	**987,75**	27,77	1634,75	0,23	1122,59	1,45
Ira13e	4	44	**699,73**	4,02	1353,43	0,50	1190,96	1,68
Ira13na	4	44	**1220,00**	167,51	1925,00	0,51	1527,64	1,51
Ira13ns	4	44	**862,30**	250,51	2506,83	0,51	1247,42	1,64
BrasilRJ14e	2	51	**256,93**	6,06	287,03	0,13	264,15	0,77
BrasilRJ14na	2	51	**217,00**	373,97	333,75	0,66	239,64	0,50
BrasilRJ14ns	2	51	**94,00**	359,99	212,17	0,12	210,92	0,50
Arabia14e	5	61	**849,68**	9,25	1690,40	1,98	1775,30	0,84
Arabia14na	5	61	**1198,05**	81,23	3069,00	1,62	2060,48	0,79
Arabia14ns	5	61	**948,20**	63,44	2913,00	2,11	1790,88	0,80
Argelia15e	3	46	**724,82**	53,85	1694,08	0,70	1254,48	1,55
Argelia15na	3	46	**1165,33**	3129,33	2238,00	0,69	1404,64	1,56
Argelia15ns	3	46	1277,00	3045,16	2342,40	0,52	**1198,76**	1,75
Cazaquistao15e	5	58	**877,13**	19,17	1969,50	2,73	1484,00	0,87
Cazaquistao15na	5	58	**1645,00**	5891,29	2745,00	2,88	1910,22	0,83
Cazaquistao15ns	5	58	**1505,00**	2815,79	2427,00	3,15	1666,53	0,83
BrasilRN16e	2	47	**365,10**	8,37	**365,10**	0,13	510,09	0,91
BrasilRN16na	2	47	**243,70**	5,82	410,00	0,17	310,67	0,49
BrasilRN16ns	2	47	**231,17**	2,88	418,00	0,12	258,87	0,50
Australia16e	4	57	**1432,33**	802,77	1499,33	1,87	1641,77	2,30
Australia16na	4	57	**1456,30**	49996,75	2245,33	1,45	1844,41	1,98
Australia16ns	4	57	**1365,08**	80000	2333,47	1,40	1704,45	2,00
China17e	3	51	**1213,22**	6234,41	2001,00	1,16	1635,93	0,65
China17na	3	51	-	-	2101,00	0,78	**1535,07**	0,64
China17ns	3	51	-	-	1905,00	3,74	**1427,77**	0,63
BrasilPR25e	3	65	**627,58**	440,10	777,00	3,08	901,69	2,02
BrasilPR25na	3	65	**311,33**	80000	611,33	3,19	463,04	0,91
BrasilPR25ns	3	65	**319,50**	80000	615,00	2,79	458,18	0,95
BrasilAM26e	3	65	691,00	8218,29	**504,05**	3,97	884,01	4,49
BrasilAM26na	3	65	-	-	582,00	4,07	**351,33**	2,40
BrasilAM26ns	3	65	-	-	574,00	3,30	**344,87**	2,18
Livramento30e	3	89	766,50	1786,92	**714,67**	2,15	1015,05	5,30
Livramento30na	3	89	-	-	833,00	2,47	**696,05**	2,44
Livramento30ns	3	89	-	-	826,50	2,29	**685,57**	2,49
BrasilMG30e	4	94	642,00	170,73	**614,50**	10,07	773,07	5,20
BrasilMG30na	4	94	-	-	618,00	11,94	**515,54**	3,30
BrasilMG30ns	4	94	-	-	561,50	8,79	**509,69**	3,49
BrasilSP32e	4	95	679,00	127,11	**653,50**	9,28	800,77	3,28
BrasilSP32na	4	95	-	-	483,50	10,53	**414,36**	1,69

Continuação da Tabela 10: Resultados obtidos pelas heurísticas ingênuas

			NAIVE1		*NAIVE2*		*NAIVE3*	
Instância	$\|C\|$	$\|L\|$	Resultado	Tempo (s)	Resultado	Tempo (s)	Resultado	Tempo (s)
BrasilSP32ns	4	95	-	-	480,50	11,08	**407,75**	1,62
BrasilRS32e	4	95	**446,33**	3135,40	**446,33**	9,61	672,04	3,26
BrasilRS32na	4	95	-	-	596,50	10,92	**493,87**	1,54
BrasilRS32ns	4	95	-	-	605,00	12,15	**494,93**	1,52
BrasilCO40e	5	114	901,50	80000	**883,00**	45,34	1109,99	7,93
BrasilCO40na	5	114	-	-	1231,50	40,03	**971,22**	5,86
BrasilCO40ns	5	114	-	-	1263,00	49,62	**962,32**	5,83
BrasilNO45e	5	133	-	-	**1200,00**	80,93	1588,56	5,82
BrasilNO45na	5	133	-	-	1576,00	86,60	**1312,02**	2,69
BrasilNO45ns	5	133	-	-	1625,00	76,49	**1335,09**	2,68
BrasilNE50e	5	118	**1003,00**	80000	1044,00	110,78	1390,97	10,25
BrasilNE50na	5	142	-	-	1618,67	92,01	**1315,23**	2,71
BrasilNE50ns	5	142	-	-	1635,00	97,56	**1311,68**	2,67
st70eB	4	204	**2541,00**	80000	3021,00	300,84	3986,42	3,68
st70naB	4	204	-	-	4155,00	92,44	**2703,33**	3,51
st70nsB	4	192	-	-	4096,00	91,59	**2542,94**	3,63
rd100eB	4	290	-	-	**10635,63**	825,20	41367,29	7,46
rd100naB	4	290	-	-	5465,00	386,27	**4077,79**	9,99
rd100nsB	4	290	-	-	5223,00	204,20	**3960,42**	8,81
ch130e	5	375	-	-	**13610,50**	3236,44	36960,03	10,70
ch130na	5	284	-	-	7094,00	925,58	**4401,80**	9,68
ch130ns	5	369	-	-	8176,00	1182,73	**5114,40**	9,75
PortoVelho200e	3	586	-	-	**3323,00**	956,53	7233,11	66,01
Cuiaba200e	4	428	-	-	**3397,00**	2364,23	7608,22	78,84
Aracaju200na	3	711	-	-	7173,00	1208,15	**5679,43**	11,53
Aracaju200ns	3	686	-	-	7193,00	1942,99	**5592,99**	18,73
Teresina200na	5	584	-	-	5173,00	2995,76	**3623,95**	22,12
Teresina200ns	5	727	-	-	6571,00	4321,76	**4940,25**	18,42

Tabela 11: Resultados obtidos pelas meta-heurísticas

			Algoritmo Memético		*GRASP*		Colônia de Formigas	
Instância	$\|C\|$	$\|L\|$	Resultado	Tempo (s)	Resultado	Tempo (s)	Resultado	Tempo (s)
Afeganistao4e	2	10	**286,00**	0,38	318,43	0,49	**286,00**	0,36
Afeganistao4na	2	10	**310,47**	0,45	320,50	0,49	**310,47**	0,31
Afeganistao4ns	2	10	**262,90**	0,38	317,36	0,46	266,71	0,41
Chile6e	3	13	357,76	1,08	345,14	2,35	**299,31**	2,62
Chile6ns	3	13	489,92	1,13	433,09	2,36	**414,97**	2,22

Continuação da Tabela 11: Resultados obtidos pelas meta-heurísticas

			Algoritmo Memético		*GRASP*		Colônia de Formigas	
Instância	$\|C\|$	$\|L\|$	Resultado	Tempo (s)	Resultado	Tempo (s)	Resultado	Tempo (s)
Chile6na	3	13	483,97	1,05	481,98	2,41	**458,66**	2,43
Paquistao7e	3	21	277,34	1,47	275,45	2,85	**273,90**	1,58
Paquistao7na	3	21	323,08	1,52	**322,19**	2,96	322,53	3,68
Paquistao7ns	3	21	**231,54**	1,42	232,56	2,63	240,10	2,19
Nanibia8e	3	23	638,16	1,70	556,81	2,36	**544,56**	2,02
Nanibia8na	3	23	901,60	1,81	594,92	2,44	**544,26**	2,24
Nanibia8ns	3	23	628,44	1,78	**495,57**	2,58	526,78	3,92
Egito9e	4	34	487,38	3,18	**387,74**	3,87	452,07	3,12
Egito9na	4	34	835,72	3,36	**661,36**	4,12	828,22	3,69
Egito9ns	4	34	659,69	3,35	**579,12**	4,13	716,80	5,46
Tanzania9e	3	28	535,14	2,24	**402,73**	2,85	412,64	2,43
Tanzania9na	3	28	574,55	2,26	485,72	2,99	**467,39**	2,27
Tanzania9ns	3	28	449,29	2,10	**418,32**	3,08	427,22	2,92
AfricaSul11e	3	37	618,62	2,95	**422,85**	3,48	574,81	3,73
AfricaSul11na	3	37	979,73	3,27	**779,12**	4,27	926,44	4,81
AfricaSul11ns	3	37	971,52	3,18	**721,16**	4,11	913,29	4,98
Niger12e	3	33	812,64	3,33	**587,98**	4,85	715,39	6,21
Niger12na	3	33	1034,78	3,40	**824,70**	4,49	1000,66	4,56
Niger12ns	3	33	875,84	3,34	**670,13**	4,69	935,05	6,26
Ira13e	4	44	898,44	5,53	**639,39**	6,90	962,93	7,13
Ira13na	4	44	1371,60	5,82	**1084,64**	7,39	1245,36	7,29
Ira13ns	4	44	992,18	5,66	**781,11**	7,28	884,59	6,33
BrasilRJ14e	2	51	194,64	2,90	**165,99**	3,50	182,00	3,28
BrasilRJ14na	2	51	196,62	3,16	**156,84**	3,97	177,09	4,71
BrasilRJ14ns	2	51	128,91	3,12	**99,57**	3,85	123,32	3,91
Arabia14e	5	61	1397,54	9,32	**1047,38**	11,64	1519,44	11,52
Arabia14na	5	61	1830,42	10,03	**1336,53**	11,92	1925,16	14,48
Arabia14ns	5	61	1603,88	9,93	**1098,33**	12,03	1833,55	19,06
Argelia15e	3	46	1007,43	4,79	**740,47**	6,53	1065,61	6,62
Argelia15na	3	46	1255,92	5,15	**1087,03**	6,73	1339,70	8,89
Argelia15ns	3	46	1061,58	5,05	**833,63**	6,55	1055,19	9,10
Cazaquistao15e	5	58	1233,31	10,00	**828,60**	12,38	1276,12	9,85
Cazaquistao15na	5	58	1705,58	10,37	**1321,11**	12,60	2033,98	21,45
Cazaquistao15ns	5	58	1606,14	10,37	**1051,07**	12,88	1574,85	12,15
BrasilRN16e	2	47	355,58	3,33	352,76	4,60	**322,42**	5,65
BrasilRN16na	2	47	208,08	3,41	**163,92**	4,65	213,82	5,78
BrasilRN16ns	2	47	208,55	3,38	**142,44**	4,96	157,50	5,31
Australia16e	4	57	1210,97	8,04	**821,64**	11,10	1168,91	10,07

Continuação da Tabela 11: Resultados obtidos pelas meta-heurísticas

			Algoritmo Memético		*GRASP*		Colônia de Formigas	
Instância	$\|C\|$	$\|L\|$	Resultado	Tempo (s)	Resultado	Tempo (s)	Resultado	Tempo (s)
Australia16na	4	57	1542,92	8,40	**1235,09**	10,69	1544,14	9,82
Australia16ns	4	57	1288,69	8,33	**1044,14**	10,55	1283,27	12,83
China17e	3	51	1322,32	5,86	**945,53**	7,83	1240,98	9,77
China17na	3	51	1464,66	6,10	**1262,80**	8,35	1423,42	9,69
China17ns	3	51	1323,80	5,76	**968,76**	7,43	1145,51	9,55
BrasilPR25e	3	65	634,33	9,87	**535,21**	13,71	605,13	16,85
BrasilPR25na	3	65	418,19	10,79	**346,21**	17,32	432,10	18,88
BrasilPR25ns	3	65	402,63	10,75	**325,64**	18,50	423,21	17,35
BrasilAM26e	3	65	560,20	9,82	**464,41**	12,71	576,19	13,69
BrasilAM26na	3	65	316,37	11,12	**269,69**	17,92	294,81	19,78
BrasilAM26ns	3	65	289,39	11,05	**256,53**	17,32	277,83	21,59
Livramento30e	3	89	690,23	16,35	**628,19**	25,90	705,15	19,18
Livramento30na	3	89	577,61	15,75	**520,28**	23,82	559,98	34,44
Livramento30ns	3	89	536,50	16,97	**493,58**	25,64	521,36	33,01
BrasilMG30e	4	94	575,65	21,41	**509,34**	27,53	559,11	27,88
BrasilMG30na	4	94	426,62	23,57	**362,03**	37,45	440,27	44,49
BrasilMG30ns	4	94	416,84	22,98	**350,42**	35,61	426,76	43,63
BrasilSP32e	4	95	560,67	26,74	**494,56**	36,82	557,19	45,23
BrasilSP32na	4	95	321,32	27,25	**284,08**	46,01	320,48	53,49
BrasilSP32ns	4	95	306,37	27,22	**284,61**	44,13	320,26	53,08
BrasilRS32e	4	95	401,06	27,45	**363,84**	41,70	417,38	43,49
BrasilRS32na	4	95	434,77	27,27	**369,96**	42,87	413,15	49,57
BrasilRS32ns	4	95	414,46	26,96	**354,41**	41,78	387,27	35,81
BrasilCO40e	5	114	819,29	49,91	**755,26**	63,90	798,36	74,13
BrasilCO40na	5	114	835,99	51,04	**765,58**	79,60	830,34	95,08
BrasilCO40ns	5	114	842,49	49,34	**759,91**	84,08	851,77	92,57
BrasilNO45e	5	133	1049,04	63,43	**1008,53**	78,62	1060,31	80,41
BrasilNO45na	5	133	1060,71	68,24	**967,39**	126,74	1187,95	120,50
BrasilNO45ns	5	133	1045,54	65,81	**950,59**	111,79	1154,64	107,91
BrasilNE50e	5	118	1022,26	65,60	**987,14**	100,26	1016,89	159,09
BrasilNE50na	5	142	1077,84	75,18	**1009,78**	131,52	1181,72	130,32
BrasilNE50ns	5	142	1110,79	72,11	**1008,51**	127,74	1173,14	132,89
st70eB	4	204	2653,04	120,52	**2431,74**	188,95	2683,72	229,19
st70naB	4	204	1940,35	114,86	**1685,90**	219,54	2162,68	168,57
st70nsB	4	192	1826,03	108,38	**1631,56**	204,27	1930,26	222,07
rd100eB	4	290	13293,05	253,73	**11957,41**	340,64	14329,43	227,80
rd100naB	4	290	2974,62	235,25	**2854,82**	493,73	3309,95	341,68
rd100nsB	4	290	2999,07	240,25	**2561,11**	494,76	3117,54	344,01

Continuação da Tabela 11: Resultados obtidos pelas meta-heurísticas

			Algoritmo Memético		*GRASP*		Colônia de Formigas	
Instância	$\|C\|$	$\|L\|$	Resultado	Tempo (s)	Resultado	Tempo (s)	Resultado	Tempo (s)
ch130e	5	375	14052,36	525,19	**13015,33**	920,14	14667,49	503,83
ch130na	5	284	**3396,05**	398,82	3430,99	980,81	3791,78	553,50
ch130ns	5	369	3829,64	505,05	**3755,81**	1233,50	4320,02	649,14
PortoVelho200e	3	586	3627,60	944,65	**3618,88**	2594,16	3738,34	968,77
Cuiaba200e	4	428	3563,82	960,18	**3414,22**	2851,58	3863,49	965,39
Aracaju200na	3	711	5046,89	828,39	5103,36	1931,61	**4965,18**	1063,27
Aracaju200ns	3	686	4864,96	796,19	**4396,49**	2523,82	4791,16	1022,36
Teresina200na	5	584	**3068,22**	1249,29	3213,22	3228,29	3225,97	1988,94
Teresina200ns	5	727	4037,83	1500,72	**3927,62**	3991,86	4191,62	1972,36

Apêndice IV

As Tabelas 12, 13, 14 e 15 apresentam informações gerais sobre as execuções dos procedimentos que compõem, respectivamente, o Algoritmo Memético, o *GRASP*, o *VNS* e o Algoritmo de Colônia de Formigas propostos neste trabalho. Estas tabelas possuem dados obtidos ao solucionar todas as 90 instâncias do *CaRSPlib*, cujos resultados são apresentados na Tabela 11. As colunas destas tabelas são descritas a seguir:

- *Procedimento*: nome do procedimento;
- *Tempo Médio (s)*: média do tempo de execução do procedimento, em segundos. É calculada a partir do tempo total de execução deste procedimento dividido pelo número de execuções;
- *Tempo (%)*: porcentagem em relação ao tempo total de execução do procedimento dividido pela soma do tempo total de todos os procedimentos;
- *#execuções*: total de execuções do procedimento;
- *Melhorias (%)*: proporção entre a quantidade de vezes em que o procedimento proporcionou melhorias à solução original e o total de execuções (*#execuções*).

Tabela 12: Dados sobre os procedimentos utilizados no Algoritmo Memético

Procedimento	*Tempo Médio (s)*	*Tempo (%)*	*#execuções*	*Melhorias (%)*
Reprodução	9,07	0,0820	592638	0,37
Mutação – Remove Carro	1,90	0,0172	397420	0,54
Mutação – Adiciona Carro	2,10	0,0190	451194	0,13
Mutação – Troca Interna de Carros	1,71	0,0155	358783	0,07
Mutação – Troca Externa de Carros	1,92	0,0173	432238	0,11
Mutação – Altera Aluguel/Devolução	4,38	0,0396	392603	0,52
Passageiros & Carros	2,48	0,0224	722233	0,09
Passageiros & Cidades	2,57	0,0232	721828	0,36
LKH Passageiros	4,21	0,0380	721605	0,33
2-*opt*	72,45	0,6546	721313	0,49
Altera Cidades sem Demandas	3,51	0,0317	718888	0,27
Altera Passageiros	4,37	0,0395	717534	0,05

Tabela 13: Dados sobre os procedimentos utilizados no *GRASP*

Procedimento	*Tempo Médio (s)*	*Tempo (%)*	*#execuções*	*Melhorias (%)*
VNS	278,12	0,7391	395127	0,72
Path Relinking	98,16	0,2609	23217	0,38

Tabela 14: Dados sobre os procedimentos utilizados no *VNS*

Procedimento	*Tempo Médio (s)*	*Tempo (%)*	*#execuções*	*Melhorias (%)*
Troca Aleatória de Cidades	87,68	0,2690	2449626	0,28
Avalia Carros	53,74	0,1649	1703928	0,19
Passageiros & Cidades	57,02	0,1750	1423611	0,26
LKH Passageiros	23,01	0,0706	982854	0,08
2-*swap* Custos & Devoluções	42,33	0,1299	903120	0,26
2-*opt*	62,12	0,1906	589593	0,30

Tabela 15: Dados sobre os procedimentos utilizados no Algoritmo de Colônia de Formigas

Procedimento	*Tempo Médio (s)*	*Tempo (%)*	*#execuções*	*Melhorias (%)*
Insere Trecho de Elite	0,16	0,0007	55535	0,22
Troca Aleatória de Cidades	46,88	0,1942	225474	0,57
Avalia Carros	1,99	0,0083	262987	0,55
Passageiros & Cidades	6,75	0,0280	553451	0,64
LKH Passageiros	14,86	0,0616	2049082	0,38
2-*swap* Custos & Devoluções	25,72	0,1065	1172506	0,79
2-*opt*	145,06	0,6009	484797	0,85

Apêndice V

Os resultados das análise estão divididos de acordo com as classes de instâncias: Euclidianas (Tabelas 16 e 17), não-Euclidianas (Tabelas 18 e 19), simétricas (Tabelas 20 e 21) e assimétricas (Tabelas 22 e 23). Os critérios avaliados foram as médias das soluções obtidas e as média dos tempos de execução dos métodos propostos. Os resultados conclusivos estão em negrito.

Tabela 16: *p-valores* para instâncias Euclidianas em relação às soluções

	Formigas	*GRASP*	Memético	*NAIVE1*	*NAIVE2*	*NAIVE3*
Formigas	-	**0,0394**	0,9000	0,1188	**0,0234**	**0,0010**
GRASP	-	-	**0,0040**	**0,0010**	**0,0010**	**0,0010**
Memético	-	-	-	0,4490	0,1508	**0,0010**
NAIVE1	-	-	-	-	0,9000	0,1001
NAIVE2	-	-	-	-	-	0,3417
NAIVE3	-	-	-	-	-	-

Tabela 17: *p-valores* para instâncias Euclidianas em relação ao tempo de execução

	Formigas	*GRASP*	Memético	*NAIVE1*	*NAIVE2*	*NAIVE3*
Formigas	-	0,9000	0,0530	**0,0010**	**0,0010**	**0,0010**
GRASP	-	-	**0,0289**	**0,0010**	**0,0010**	**0,0010**
Memético	-	-	-	**0,0010**	0,2338	**0,0150**
NAIVE1	-	-	-	-	**0,0010**	**0,0010**
NAIVE2	-	-	-	-	-	0,9000
NAIVE3	-	-	-	-	-	-

Tabela 18: *p-valores* para instâncias não-Euclidianas em relação às soluções

	Formigas	*GRASP*	Memético	*NAIVE1*	*NAIVE2*	*NAIVE3*
Formigas	-	**0,0034**	0,9000	**0,0010**	**0,0010**	**0,0010**
GRASP	-	-	**0,0010**	**0,0010**	**0,0010**	**0,0010**
Memético	-	-	-	**0,0010**	**0,0010**	**0,0010**
NAIVE1	-	-	-	-	0,0888	0,9000
NAIVE2	-	-	-	-	-	**0,0175**
NAIVE3	-	-	-	-	-	-

Tabela 19: *p-valores* para instâncias não-Euclidianas em relação ao tempo de execução

	Formigas	*GRASP*	Memético	*NAIVE1*	*NAIVE2*	*NAIVE3*
Formigas	-	0,9000	**0,0010**	**0,0010**	**0,0010**	**0,0010**
GRASP	-	-	**0,0010**	**0,0010**	**0,0010**	**0,0010**
Memético	-	-	-	**0,0010**	**0,0299**	**0,0010**
NAIVE1	-	-	-	-	**0,0010**	**0,0010**
NAIVE2	-	-	-	-	-	0,8608
NAIVE3	-	-	-	-	-	-

Tabela 20: *p-valores* para instâncias simétricas em relação às soluções

	Formigas	*GRASP*	Memético	*NAIVE1*	*NAIVE2*	*NAIVE3*
Formigas	-	**0,0010**	0,9000	**0,0010**	**0,0010**	**0,0010**
GRASP	-	-	**0,0010**	**0,0010**	**0,0010**	**0,0010**
Memético	-	-	-	**0,0010**	**0,0010**	**0,0010**
NAIVE1	-	-	-	-	0,6776	0,7903
NAIVE2	-	-	-	-	-	0,9000
NAIVE3	-	-	-	-	-	-

Tabela 21: *p-valores* para instâncias simétricas em relação ao tempo de execução

	Formigas	*GRASP*	Memético	*NAIVE1*	*NAIVE2*	*NAIVE3*
Formigas	-	0,9000	**0,0010**	**0,0010**	**0,0010**	**0,0010**
GRASP	-	-	**0,0010**	**0,0010**	**0,0010**	**0,0010**
Memético	-	-	-	**0,0010**	**0,0221**	**0,0010**
NAIVE1	-	-	-	-	**0,0010**	**0,0010**
NAIVE2	-	-	-	-	-	0,8044
NAIVE3	-	-	-	-	-	-

Tabela 22: *p-valores* para instâncias assimétricas em relação às soluções

	Formigas	*GRASP*	Memético	*NAIVE1*	*NAIVE2*	*NAIVE3*
Formigas	-	0,2033	0,9000	**0,0083**	**0,0010**	**0,0074**
GRASP	-	-	0,0642	**0,0010**	**0,0010**	**0,0010**
Memético	-	-	-	**0,0394**	**0,0010**	**0,0356**
NAIVE1	-	-	-	-	0,2033	0,9000
NAIVE2	-	-	-	-	-	0,2182
NAIVE3	-	-	-	-	-	-

Tabela 23: *p-valores* para instâncias assimétricas em relação ao tempo de execução

	Formigas	*GRASP*	Memético	*NAIVE1*	*NAIVE2*	*NAIVE3*
Formigas	-	0,9000	**0,0094**	**0,0010**	**0,0010**	**0,0010**
GRASP	-	-	**0,0074**	**0,0010**	**0,0010**	**0,0010**
Memético	-	-	-	**0,0010**	0,3031	**0,0356**
NAIVE1	-	-	-	-	**0,0010**	**0,0010**
NAIVE2	-	-	-	-	-	0,9000
NAIVE3	-	-	-	-	-	-

Apêndice VI

A linearização apresentada na seção 3.3.2 consiste em substituir a fração N/D, pela seguinte expressão:

$$\frac{N}{D} = N - \sum_{h=1}^{D-1} \frac{N}{h(h+1)}$$

Proposição: *Mostraremos, por indução matemática, que a expressão a seguir é válida para todos os inteiros positivos* $n \geq 1$.

$$\frac{1}{1\times 2} + \frac{1}{2\times 3} + \frac{1}{3\times 4} + \cdots + \frac{1}{n\times(n+1)} = \frac{n}{n+1}$$

Demonstração. Vamos mostrar o passo base e o passo indutivo da expressão.

Passo base: Para $n = 1$ temos que:

$$\frac{1}{1\times 2} = \frac{1}{2}$$

logo, o passo base é verdadeiro.

Passo indutivo: *Assumindo que a fórmula é verdadeira para* $n = k$, *mostraremos que é verdadeira também para* $n = k + 1$. *Dessa forma, temos a seguinte hipótese indutiva:*

$$\frac{1}{1\times 2} + \frac{1}{2\times 3} + \frac{1}{3\times 4} + \cdots + \frac{1}{n\times(n+1)} = \frac{n}{n+1}$$

Queremos mostrar que é válido para $(n+1)$, *ou seja:*

$$\frac{1}{1\times 2} + \frac{1}{2\times 3} + \frac{1}{3\times 4} + \cdots + \frac{1}{n\times(n+1)} + \frac{1}{(n+1)\times(n+2)} = \frac{n+1}{n+2}$$

Daí, segue que:

$$\begin{aligned}\frac{1}{1\times 2}+\frac{1}{2\times 3}+\frac{1}{3\times 4}+\cdots+\frac{1}{n\times(n+1)}+\frac{1}{(n+1)\times(n+2)} &= \frac{n}{n+1}+\frac{1}{(n+1)\times(n+2)}\\ &= \frac{n(n+2)+1}{(n+1)(n+2)}\\ &= \frac{n^2+2n+1}{(n+1)(n+2)}\\ &= \frac{(n+1)^2}{(n+1)(n+2)}\\ &= \frac{(n+1)}{(n+2)}\end{aligned}$$

logo, o passo indutivo também é verdadeiro.

Proposição: *A fórmula* $\frac{N}{D} = N - \sum_{h=1}^{D-1} \frac{N}{h(h+1)}$ *possui equivalência algébrica.*

Demonstração. Utilizando o resultado mostrado anteriormente, temos que:

$$\begin{aligned}\frac{N}{D} &= N - \sum_{h=1}^{D-1} \frac{N}{h(h+1)}\\ &= N - N\sum_{h=1}^{D-1} \frac{1}{h(h+1)}\\ &= N - N\frac{(D-1)}{D}\\ &= N - \frac{ND}{D} + \frac{N}{D}\\ &= N - N + \frac{N}{D}\\ &= \frac{N}{D}\end{aligned}$$

Então, a partir desta verificação fica comprovado que a linearização possui equivalência algébrica. Porém, vale ressaltar os aspectos númericos, visto que se trata de uma sequência de operações algébricas de divisão pode acarretar em propagação de erros. Como o trabalho utiliza valores baixos para D, isto não ocorre. Também vale resssaltar que o *solver* geralmente trata o tipo de variável, dificultando que erros deste tipo venham a ocorrer.

Anexo I

O modelo matemático, para o *CaRS*, proposto por Rios, Goldbarg e Quesquén (2017) é formulado em (8.1)–(8.16).

$$\min \sum_{c\in C}\sum_{i,j\in N} d_{ij}^c f_{ij}^c + \sum_{c\in C}\sum_{i,j\in N} \gamma_{ij}^c \omega_{ij}^c \tag{8.1}$$

sujeito a

$$u_i - u_j + (n-1)f_{ij}^c + (n-3)f_{ji}^c \leq n-2, \tag{8.2}$$

$$\forall i,j = 2,...,n \mid i \neq j, \forall c \in C \tag{8.3}$$

$$h_j^c = \sum_{i\in N, i\neq j} f_{ij}^c, \qquad \forall j \in N, \forall c \in C \tag{8.4}$$

$$\sum_{c\in C} h_j^c = 1, \qquad \forall j \in N \tag{8.5}$$

$$b_i^c = \sum_{j\in N, i\neq j} f_{ij}^c, \qquad \forall i \in N, \forall c \in C \tag{8.6}$$

$$\sum_{c\in C} b_i^c = 1, \qquad \forall i \in N \tag{8.7}$$

$$z_i^c = h_i^c(1-b_i^c), \qquad \forall i \in N, \forall c \in C \tag{8.8}$$

$$\sum_{i\in N} z_i^c \leq 1, \qquad \forall c \in C \tag{8.9}$$

$$y_i^c = b_i^c(1-h_i^c), \qquad \forall i \in N, \forall c \in C \tag{8.10}$$

$$\sum_{i\in N} y_i^c \leq 1, \qquad \forall c \in C \tag{8.11}$$

$$\sum_{c\in C} z_1^c = 1 \tag{8.12}$$

$$\sum_{c\in C} y_1^c = 1 \tag{8.13}$$

$$\omega_{ij}^c = y_i^c z_j^c \qquad \forall c \in C, \forall i,j \in N \tag{8.14}$$

$$f_{ij}^c, \omega_{ij}^c, y_i^c, z_i^c \in \{0,1\}, \qquad \forall i,j \in N, \forall c \in C \tag{8.15}$$

$$u_i \in \mathbb{N}, \qquad i \in N\backslash\{1\} \tag{8.16}$$

A partir da análise das equações (8.10) e (8.13) deste modelo, é possível perceber uma inconsistência. A restrição (8.13) obriga que um carro seja alugado na cidade 1, mas a restrição (8.10) impõe que para que um carro c seja alugado em uma cidade i, o caixeiro não pode ter chegado em i utilizando o mesmo carro c. Ou seja, desta maneira o caixeiro não seria capaz de realizar um *tour* utilizando apenas um veículo. Analogamente, a restrição (8.8) também causa a mesma inconsistência no processo de devolução de um veículo. Esta inconsistência também aparece nos modelos propostos por Rios, Goldbarg e Goldbarg (2017) e Rios (2018). Para adequar o modelo, as equações (8.8) e (8.10) devem ser substuídas por (8.17)–(8.20), descritas a seguir.

$$z_i^c = h_i^c(1 - b_i^c), \qquad \forall i \in N\backslash\{1\}, \forall c \in C \quad (8.17)$$

$$z_1^c = h_1^c, \qquad \forall c \in C \quad (8.18)$$

$$y_i^c = b_i^c(1 - h_i^c), \qquad \forall i \in N\backslash\{1\}, \forall c \in C \quad (8.19)$$

$$y_1^c = b_1^c, \qquad \forall c \in C \quad (8.20)$$

Anexo II

Um dos modelos matemáticos, para o *CaRS*, proposto por Goldbarg *et al.* (2017) é formulado em (9.1)–(9.12).

$$\min \sum_{c\in C}\left(\sum_{(i,j)\in M} d_{ij}^{c} f_{ij}^{c} + \sum_{c\in C} \gamma_{ij}^{c}\omega_{ij}^{c}\right) \tag{9.1}$$

sujeito a

$$\sum_{c\in C}\sum_{i\in N} f_{ij}^{c} = 1, \qquad \forall j \in N \tag{9.2}$$

$$\sum_{c\in C}\sum_{j\in N} f_{ij}^{c} = 1, \qquad \forall i \in N \tag{9.3}$$

$$\sum_{\substack{j\in N\\ i\neq j}} u_{ij} - \sum_{\substack{j\in N\setminus\{1\}\\ i\neq j}} u_{ji} = 1, \qquad \forall i \in N\setminus\{1\} \tag{9.4}$$

$$u_{ij} \leq (n-1)\sum_{c\in C} f_{ij}^{c}, \qquad \forall i,j \in N, i \neq 1 \tag{9.5}$$

$$\sum_{\substack{i\in N\\ i\neq j}} f_{ji}^{c} - \sum_{\substack{i\in N\\ i\neq j}} f_{ij}^{c} \leq y_{j}^{c}, \qquad \forall j \in N\setminus\{1\}, \forall c \in C \tag{9.6}$$

$$\sum_{i\in N} f_{1i}^{c} = y_{1}^{c}, \qquad \forall c \in C \tag{9.7}$$

$$\sum_{\substack{i\in N\\ i\neq j}} f_{ij}^{c} - \sum_{\substack{i\in N\\ i\neq j}} f_{ji}^{c} \leq z_{j}^{c}, \qquad \forall j \in N\setminus\{1\}, \forall c \in C \tag{9.8}$$

$$\sum_{i\in N} f_{i1}^{c} = z_{1}^{c}, \qquad \forall c \in C \tag{9.9}$$

$$y_{i}^{c} + z_{j}^{c} - 1 \leq \omega_{ij}^{c}, \qquad \forall i,j \in N, i \neq j, \forall c \in C \tag{9.10}$$

$$f_{ij}^{c}, \omega_{ij}^{c}, y_{i}^{c}, z_{i}^{c} \in \{0,1\}, \qquad \forall i,j \in N, \forall c \in C \tag{9.11}$$

$$u_{ij} \in \mathbb{Z}_{\geq 0}, \qquad \forall i,j \in N \tag{9.12}$$

Assim como a primeira parcela da soma da função objetivo (9.1), a segunda também deveria percorrer os índices $(i,j) \in M$ ao invés de percorrer $c \in C$.

Anexo III

A modelagem matemática para o PAP proposta por Calheiros (2017) é formulada em (10.1)–(10.6). Para facilitar a compreensão, utilizamos os mesmos parâmetros e variáveis já definidos neste trabalho.

$$\sum_{1 \leq i \leq 1} c_i - \sum_{2 \leq h \leq k} \frac{g_{ih}}{h(h-1)} \tag{10.1}$$

sujeito a

$$\sum_{org(j) \leq i \leq dst(j)} e_j = \sum_{h=2}^{k} g_{ih}, \qquad (1 \leq i \leq n) \tag{10.2}$$

$$\sum_{i=1}^{n} q_{li} \leq bud(l), \qquad (1 \leq l \leq n) \tag{10.3}$$

$$c_i e_j - \sum_{h=2}^{k} \frac{c_i g_{ih}}{h(h-1)} \leq q_{li}, \qquad (1 \leq i \leq n, org(j) \leq i \leq dst(j)) \tag{10.4}$$

$$q_{li} \in \mathbb{R}_{\geq 0}, \qquad (1 \leq l, i \leq n) \tag{10.5}$$

$$z_{ih} \leq 1, z_{ih} \in \mathbb{R}_{\geq 0}, \qquad (1 \leq i \leq n, 2 \leq h \leq k) \tag{10.6}$$

Na função objetivo (10.1) é possível observar que no numerador do somatório está faltando multiplicar a variável g_{ih} pelo custo c_i, o que inviabilizaria o cálculo do custo. Além disso, o índice do somatório está variando de $2 \leq h \leq k$, o que causa uma inconsistência no que está sendo calculado. Vamos considerar um exemplo em que o carro c, cuja capacidade é 3, ou seja, $k = 3$, está atravessando uma aresta de custo 100 com todos os assentos ocupados. Assim, utilizando o a estratégia definida na função objetivo (10.1), a equação (10.7) acaba resultando em (10.8), demonstrando que o valor calculado é incorreto.

$$\frac{100}{(1+3)} = 25 \tag{10.7}$$

$$\frac{100}{(1+3)} = \ 100 - \sum_{h=2}^{k} \frac{100}{h\ (h-1)} = 100 - \left(\frac{100}{2 \times 1} + \frac{100}{3 \times 2} \right) = 33,33... \tag{10.8}$$

A modelagem matemática para o *PAP* proposta por Marques *et. al* (2019) é formulada em (10.9)–(10.15).

$$\sum_{i \in N} \left(c_i - \sum_{h=2}^{k} \frac{c_i g_{i(h-1)}}{h(h+1)} \right) \tag{10.9}$$

sujeito a

$$\sum_{l \in P(i)} e_l = \sum_{h=2}^{k} g_{ih}, \qquad \forall i \in N, P(i) = \{l \in L \mid iorg(l) \leq i < idst(l)\} \tag{10.10}$$

$$c_i - \sum_{h=2}^{k} \frac{c_i g_{i(h-1)}}{h(h+1)} \leq q_{li}, \qquad \forall i \in N, \forall l \in L \mid iorg(l) \leq i < idst(l) \tag{10.11}$$

$$\sum_{i \in N} q_{li} \leq bud(l), \qquad \forall l \in L \tag{10.12}$$

$$0 \leq g_{ih} \leq 1, \qquad \forall i \in N, \forall j \in \{2, ..., k\} \tag{10.13}$$

$$q_{li} \in \mathbb{R}^{\geq 0}, \qquad \forall l \in L, \forall i \in N \tag{10.14}$$

$$e_l \in \{0, 1\}, \qquad \forall l \in L \tag{10.15}$$

O índice do somatório da função objetivo (10.9) está variando de $2 \leq h \leq k$, o que causa uma inconsistência no que está sendo calculado. Vamos considerar um exemplo em que o carro c, cuja capacidade é 3, ou seja, $k = 3$, está atravessando uma aresta de custo 100 com todos os assentos ocupados. Assim, utilizando a estratégia definida na função objetivo (10.9), a equação (10.16) acaba resultando em (10.17), demonstrando que o valor calculado é incorreto.

$$\frac{100}{(1+3)} = 25 \tag{10.16}$$

$$\frac{100}{(1+3)} = \ 100 - \sum_{h=2}^{k} \frac{100}{h\ (h+1)} = 100 - \left(\frac{100}{2 \times 3} + \frac{100}{3 \times 4} \right) = 75 \tag{10.17}$$

Além disso, a restrição (10.11) deve, necessariamente, estar ativa apenas para os passageiros que estão sendo transportados, ou seja, deve relacionar a variável e_l e o custo c_i. Esta inconsistência leva à soluções incorretas.