

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Lucas Correia Bernardes

**Metaheurísticas para o Problema de
Roteamento de Veículos Capacitados: o
Algoritmo Híbrido de Otimização por Colônia
de Formigas e a Busca Híbrida em Grande
Vizinhança**

Uberlândia, Brasil

2018

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Lucas Correia Bernardes

**Metaheurísticas para o Problema de Roteamento de
Veículos Capacitados: o Algoritmo Híbrido de Otimização
por Colônia de Formigas e a Busca Híbrida em Grande
Vizinhança**

Trabalho de conclusão de curso apresentado
à Faculdade de Computação da Universidade
Federal de Uberlândia, Minas Gerais, como
requisito parcial exigido à obtenção do grau
de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Alexsandro Santos Soares

Universidade Federal de Uberlândia – UFU

Faculdade de Computação

Bacharelado em Ciência da Computação

Uberlândia, Brasil

2018

Lucas Correia Bernardes

Metaheurísticas para o Problema de Roteamento de Veículos Capacitados: o Algoritmo Híbrido de Otimização por Colônia de Formigas e a Busca Híbrida em Grande Vizinhaça

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como requisito parcial exigido à obtenção do grau de Bacharel em Ciência da Computação.

Trabalho aprovado. Uberlândia, Brasil, 2 de julho de 2018:

Prof. Dr. Alexsandro Santos Soares
Orientador

Profa. Dra. Márcia Aparecida Fernandes

Prof. Dr. Wendel Alexandre Xavier de Melo

Uberlândia, Brasil
2018

Agradecimentos

Primeiramente, agradeço a Deus pela minha vida, pela saúde e pela sabedoria para atravessar mais essa etapa.

A todos os professores que participaram nas etapas de minha formação como profissional e cidadão, não somente pelo ensino do conhecimento teórico, como também por ajudarem a moldar meu caráter e minha visão de mundo.

Ao meu orientador, Alexsandro, pela paciência, incentivo e ajuda nesse processo final de minha graduação.

Aos meus familiares, em especial a meus pais, pelo apoio e amor incondicional, sendo minha base em todos os momentos da vida.

Ao grupo de amigos que adquiri durante a graduação, pela parceria de permanecer comigo tanto nos melhores momentos, quanto nos momentos de dificuldade, desânimo e cansaço do processo.

Meus sinceros agradecimentos a todos que de forma direta ou indireta me ajudaram a cumprir mais esta etapa da minha vida.

“Para criaturas pequenas como nós, a vastidão só é suportável por meio do amor.”
(Carl Sagan)

Resumo

Bernardes, L.C. **Análise de metaheurísticas para o Problema de Roteamento de Veículos Capacitados**. Curso de Ciência da Computação, Campus Uberlândia, UFU, Uberlândia, Brasil, 2018, 97p.

Este trabalho apresenta a comparação de duas técnicas de metaheurísticas para o Problema de Roteamento de Veículos Capacitados (PRVC) - o Algoritmo Híbrido de Otimização por Colônia de Formigas (AHOCF) e a Busca Híbrida em Grande Vizinhança (BHGV). Além disso, é mostrada uma visão geral das técnicas mais recentes e das variações mais utilizadas do problema.

Será descrita a implementação dos métodos na linguagem de programação Clojure, e a performance dos algoritmos será testada em um conjunto das instâncias mais utilizadas do repositório CVRPLib. Os resultados obtidos serão então contrastados com os resultados apresentados pelos autores e entre si, para, por fim, verificar que o método AHOCF obteve um desempenho superior nos resultados, em comparação com os alcançados pela BHGV.

Palavras-chave: Problema de Roteamento de Veículos Capacitados. Metaheurística. Problema de Otimização. Busca em Grande Vizinhança. Otimização por Colônia de Formigas. Recozimento Simulado.

Abstract

This paper presents a comparison between two metaheuristics techniques to the Capacitated Vehicle Routing Problem (CVRP) - the Hybrid Ant Colony Optimization (HACO) and Hybrid Large Neighborhood Search (HLNS). Furthermore, an overview of the most recent techniques and most used problem variants is shown.

The implementation of both methods using the Clojure programming language is described, and the algorithms performance tested against the most used set of instances from the CVRPLib repository. The results obtained are then contrasted with the results presented by the authors and with each other, to finally verify that the HACO metaheuristic obtained a superior performance in the results, in comparison with those achieved by the HLNS.

Keywords: Capacitated Vehicle Routing Problem. Metaheuristics. Optimization Problems. Large Neighborhood Search. Ant Colony Optimization. Simulated Annealing.

Lista de ilustrações

Figura 1 – Representação gráfica de um PRVC	21
Figura 2 – Representação gráfica da solução composta das rotas $x_1 = \langle v_0, v_4, v_2, v_5, v_0 \rangle$, $x_2 = \langle v_0, v_3, v_6, v_0 \rangle$ e $x_3 = \langle v_0, v_1, v_7, v_0 \rangle$	23
Figura 3 – Representação gráfica da solução ótima composta das rotas $x_1 = \langle$ $v_0, v_1, v_2, v_0 \rangle$, $x_2 = \langle v_0, v_3, v_4, v_0 \rangle$ e $x_3 = \langle v_0, v_5, v_6, v_7, v_0 \rangle$	24
Figura 4 – Representação esquemática da metodologia da revisão sistemática	29
Figura 5 – Proporção entre os tipos de métodos encontrados na revisão sistemática .	31
Figura 6 – Representação gráfica da instância E-n22-k4, presente na classe E, onde cada ponto azul representa um consumidor a ser visitado e cada ponto vermelho representa um depósito	53
Figura 7 – Fluxo de execução do analisador <i>CVRPlib_{parser}</i>	55
Figura 8 – Fluxo de execução das instâncias	57
Figura 9 – Arquitetura geral da implementação do analisador <i>CVRPlib_{parser}</i>	60
Figura 10 – Representação em lista da solução da Figura 3	61
Figura 11 – Representação da solução da Figura 3 por meio de uma matriz de adjacências	61
Figura 12 – Arquitetura geral da implementação do método AHOCF	62
Figura 13 – Arquitetura geral da implementação do método BHGV	63
Figura 14 – Melhoria da população de soluções no algoritmo Otimização por Colô- nia de Formigas para a instância A-n32-k5	67
Figura 15 – Melhoria da melhor solução até o momento pelo algoritmo Recozimento Simulado para a instância A-n32-k5	67
Figura 16 – Comparação entre o tempo de execução dos algoritmos AHOCF e BHGV .	73

Lista de Algoritmos

4.1	PSEUDOCÓDIGO DO AHOCF	41
4.2	PSEUDOCÓDIGO DA BUSCA LOCAL DO AHOCF	43
4.3	PSEUDOCÓDIGO DO RECOZIMENTO SIMULADO NO AHOCF	46
4.4	PSEUDO-CÓDIGO BGV	47
4.5	PSEUDOCÓDIGO BHGV	48
4.6	PSEUDOCÓDIGO DA HEURÍSTICA DA PIOR REMOÇÃO	49
4.7	PSEUDOCÓDIGO DA HEURÍSTICA ARREPENDE-2	50
4.8	PSEUDOCÓDIGO DA OTIMIZAÇÃO POR COLÔNIA DE FORMIGAS NA BHGV	52

Lista de listagens

Figura 5.1 – Esquema parcial da instância E-n22-k4 no formato TSPLIB95	54
Figura 5.2 – Esquema da solução ótima da instância E-n22-k4 no formato TSPLIB95	55
Figura 5.3 – Arquivo JSON parcial da instância E-n22-k4	57
Figura 5.4 – Arquivo de resultados da instância E-n22-k4	59

Lista de tabelas

Tabela 1 – Periódicos de maior impacto na área	31
Tabela 2 – Informações de base, periódico e código-fonte dos artigos selecionados na revisão sistemática	38
Tabela 3 – Informações de método e tipo de instância utilizada dos artigos selecionados	39
Tabela 4 – Propriedades do JSON de saída	56
Tabela 5 – Propriedades do arquivo de resultados	58
Tabela 6 – Parâmetros utilizados no algoritmo BGVH	64
Tabela 7 – Parâmetros utilizados no algoritmo AHOCF	64
Tabela 8 – Comparação entre os resultados do AHOCF	65
Tabela 9 – Comparação entre os resultados da BHGV. Os desvios em negrito mostram as instâncias em que o ótimo foi obtido	68
Tabela 10 – Solução encontrada e tempo gasto em cada algoritmo por instância . .	71
Tabela 11 – Instâncias do repositório CVRPLib, onde n é o número de consumidores, K o número de veículos, Q a capacidade máxima dos veículos e MSC a melhor solução conhecida	84

Lista de abreviaturas e siglas

PRV	Programação de Roteamento de Veículos
PRVC	Programação de Roteamento de Veículos Capacitados
PCV	Problema do Caixeiro Viajante
BGVH	Busca em Grande Vizinhaça Híbrida

Sumário

1	INTRODUÇÃO	15
1.1	Questão de pesquisa	15
1.2	Hipótese	16
1.3	Objetivo	16
1.3.1	Objetivos Específicos	16
1.4	Justificativa	16
1.5	Organização do Trabalho	17
2	FUNDAMENTAÇÃO TEÓRICA	18
2.1	Definição do Problema	18
2.1.1	Exemplo de PRVC	21
2.2	Variantes	23
2.3	Métodos de Soluções	25
2.3.1	Métodos exatos	25
2.3.2	Métodos heurísticos	26
2.3.3	Metaheurísticas	27
3	REVISÃO SISTEMÁTICA DA LITERATURA	28
3.1	Planejamento da Revisão	29
3.1.1	Questões da Pesquisa	29
3.1.2	Protocolo de Revisão	29
3.2	Realização da Revisão	31
3.3	Documentação da Revisão	32
3.3.1	Artigos levantados	32
3.3.2	Escolha das metaheurísticas	37
4	METAHEURÍSTICAS ESCOLHIDAS	40
4.1	Algoritmo Híbrido de Otimização por Colônia de Formigas	40
4.1.1	Mecanismo de Construção	40
4.1.2	Busca Local	42
4.1.3	Atualização dos Feromônios	42
4.1.4	Perturbação dos Feromônios	44
4.1.5	Recozimento Simulado	44
4.2	Busca Híbrida em Grande Vizinhaça	45
4.2.1	Busca em Grande Vizinhaça	45
4.2.2	Heurística de Destruição	48

4.2.3	Heurística de Reparação	49
4.2.4	Mecanismo de Construção	50
5	DESENVOLVIMENTO	53
5.1	Analizador das instâncias do problema	53
5.2	Arquivo de resultados	57
5.3	Implementação	59
5.3.1	Analizador das instâncias	59
5.3.2	Algoritmo Híbrido de Otimização por Colônia de Formigas	60
5.3.3	Busca Híbrida em Grande Vizinhança	62
5.4	Experimentos	63
5.5	Resultados e Análise	65
5.5.1	Algoritmo Híbrido de Otimização por Colônia de Formigas	65
5.5.2	Busca Híbrida em Grande Vizinhança	68
5.5.3	Comparação entre as metaheurísticas	70
6	CONCLUSÃO	74
Conclusão		74
6.1	Conclusões	74
6.2	Trabalhos futuros	75
	REFERÊNCIAS	76
	APÊNDICES	83
	APÊNDICE A – INSTÂNCIAS DO REPOSITÓRIO CVRPLIB	84
	APÊNDICE B – CÓDIGOS DO ANALISADOR DE INSTÂNCIAS DO REPOSITÓRIO CVRPLIB	91
B.1	Arquivo project.clj	91
B.2	Arquivo core.clj	91
B.3	Arquivo parser.clj	93
B.4	Arquivo distances.clj	95
B.5	Arquivo optimal.clj	97
	APÊNDICE C – CÓDIGOS DO ALGORITMO HÍBRIDO DE OTI- MIZAÇÃO POR COLÔNIA DE FORMIGAS	99
C.1	Arquivo project.clj	99
C.2	Arquivo core.clj	99

C.3	Arquivo instance.clj	101
C.4	Arquivo solution.clj	103
C.5	Arquivo saco.clj	105
C.6	Arquivo aco.clj	106
C.7	Arquivo local_search.clj	108
C.8	Arquivo pheromones.clj	109
C.9	Arquivo sa.clj	110
C.10	Arquivo utils.clj	112

APÊNDICE D – CÓDIGOS DA BUSCA HÍBRIDA EM GRANDE VIZINHANÇA

D.1	Arquivo project.clj	113
D.2	Arquivo core.clj	113
D.3	Arquivo hybrid_ins.clj	114

1 Introdução

O **Problema de Roteamento de Veículos Capacitados** (PRVC), ou *Capacitated Vehicle Routing Problem* (CVRP), é um clássico problema de otimização combinatória proposto inicialmente por [Dantzig e Ramser \(1959\)](#) como Problema de Despacho de Caminhões.

Esse problema é composto de um conjunto de consumidores dispostos em um grafo ponderado, cada um com uma quantidade de demanda a ser atendida, e uma frota de veículos de capacidade fixa localizados em um depósito na rede. O objetivo, então, é definir rotas para cada veículo partindo e retornando ao depósito, de tal forma que cada consumidor da rede seja visitado uma vez, que a soma das demandas dos consumidores nas rotas não ultrapasse a capacidade dos veículos, e que a distância percorrida por cada veículo seja minimizada.

O PRVC é uma generalização do Problema do Caixeiro Viajante (PCV), onde cada veículo pode ser visto como uma instância do PCV com a restrição de capacidade, e trata-se de um problema de complexidade NP-difícil [2]. Portanto, métodos exatos para obter a melhor solução do problema apresentam uma grande limitação por possuírem um tempo de execução elevado e serem eficientes somente em pequenas instâncias.

Como a maior parte dos problemas reais e interessantes possuem grafos complexos com uma grande quantidade de consumidores e veículos, são utilizados métodos advindos da Inteligência Artificial, como heurísticas e metaheurísticas, para se obterem as soluções ótimas ou próximas das ótimas em um tempo de execução razoável.

Nesse sentido, este trabalho apresenta uma análise de dois trabalhos do estado-da-arte que propõem métodos metaheurísticos - o algoritmo de Busca Híbrida em Grande Vizinhança proposto por [Akpınar \(2016\)](#), e o algoritmo híbrido de Otimização por Colônia de Formigas proposto por [Sun, Fu e Liu \(2017\)](#) - com o objetivo de comparar o desempenho em se encontrar a solução ótima e o tempo de execução de ambos em um conjunto de instâncias do repositório CVRPLib [5].

1.1 Questão de pesquisa

Qual é o desempenho do algoritmo Busca em Vizinhança Grande Híbrida (BVGH) para encontrar soluções para o Problema de Roteamento de Veículos Capacitados, em comparação com o Algoritmo Híbrido de Otimização por Colônia de Formigas (AHOCF)?

1.2 Hipótese

Por ser ligeiramente mais recente, o Algoritmo Híbrido de Otimização por Colônia de Formigas possui resultados, no mínimo, equivalentes aos obtidos pelo método de Busca Híbrida em Grande Vizinhaça, com resultados melhores em casos específicos.

1.3 Objetivo

O objetivo desta pesquisa é comparar quantitativamente duas técnicas aplicadas ao Problema de Roteamento de Veículos Capacitados, a metaheurística Busca Híbrida em Grande Vizinhaça (BHGV) e o Algoritmo Híbrido de Otimização por Colônia de Formigas (AHOCF), analisando o tempo de processamento e a qualidade das soluções encontradas.

1.3.1 Objetivos Específicos

Esse objetivo pode ser desmembrado em três etapas específicas:

1. Confrontar os resultados do método Busca Híbrida em Grande Vizinhaça apresentados pelo autor em [3] com os obtidos pela implementação do algoritmo;
2. Confrontar os resultados do método Algoritmo Híbrido de Otimização por Colônia de Formigas apresentados pelo autor em [4] com os obtidos pela implementação do algoritmo;
3. Confrontar os métodos implementados entre si, analisando o desempenho de ambos com os resultados obtidos.

1.4 Justificativa

O PRV é um dos problemas de otimização combinatória mais recorrentes na literatura. Devido à sua alta aplicabilidade em situações do cotidiano [6], e com modelos refletindo cada vez mais aspectos da realidade, há uma demanda crescente por soluções mais simples e eficientes.

Além desse amplo interesse prático, o PRV também é um problema teórico aberto, pertencente à classe de problemas NP-Difícil [2]. Devido a isso, há uma inviabilidade de se utilizarem métodos exatos, fazendo com que grande parte das pesquisas por soluções concentrem-se em soluções metaheurísticas devido à sua vantagem de encontrar boas soluções em um tempo hábil.

1.5 Organização do Trabalho

Este trabalho está organizado da seguinte forma: o capítulo 2 apresenta a fundamentação teórica para compreender este estudo, definindo o problema e suas variações, por meio de uma análise da complexidade do problema e, ainda, explicando os problemas de otimização e os tipos de métodos existentes. O capítulo 3, por sua vez, apresenta uma revisão sistemática na literatura com um levantamento dos principais métodos propostos no estado-da-arte, além do processo de escolha das duas metaheurísticas. Em seguida, o capítulo 4 descreve em detalhes os dois métodos escolhidos. No capítulo 5, mostram-se os resultados obtidos e é feita uma análise relativa aos testes executados. O sexto capítulo traz conclusões e propostas para trabalhos futuros e, por fim, o capítulo 7 apresenta as referências bibliográficas consultadas.

2 Fundamentação Teórica

Neste capítulo, são apresentados os conceitos fundamentais para melhor entendimento deste trabalho.

2.1 Definição do Problema

Um problema é dito ser de **otimização combinatória** quando consiste em encontrar uma solução ótima de um conjunto finito de soluções e é composto por três componentes: (I) as variáveis de decisão, (II) uma função objetivo e (III) um conjunto de restrições. Os valores possíveis das variáveis de decisão são delimitados pelas restrições impostas e formam um conjunto de soluções factíveis do problema, denominado espaço de busca. A solução para o problema de otimização, denominada ótimo global, é dada pelo maior valor possível para a função objetivo, caso seja um problema de maximização, ou menor valor, caso seja um problema de minimização, desde que os valores atribuídos não violem as restrições do problema.

Na teoria da computação, um problema é dito pertencente à classe P quando existir um algoritmo que o resolva de maneira eficiente, e é dito pertencente à classe NP quando sua solução puder ser verificada por um algoritmo de maneira eficiente. Segundo a tese de Cobham-Edmonds [7], é considerado *eficiente* o algoritmo que utilize uma quantidade polinomial de recursos computacionais para resolver determinado problema.

Um problema pertencente à classe P é também pertencente à classe NP ($P \subseteq NP$), já que é possível utilizar o próprio algoritmo que o resolva eficientemente para verificar a solução encontrada. Entretanto, não se sabe se um problema pertencente à classe NP é também pertencente à classe P, sendo o principal problema em aberto na ciência da computação atualmente [8].

Um problema é dito ser **NP-completo** quando pertencer à classe NP e qualquer problema pertencente à esta classe puder ser reduzido a ele de maneira eficiente. Portanto, caso exista um algoritmo que resolva um problema NP-completo de maneira eficiente, toda a classe de problemas NP também poderá ser resolvida de maneira eficiente. Da mesma forma, um problema é definido **NP-difícil** quando qualquer problema em NP puder ser reduzido a ele de maneira eficiente, mas podendo não necessariamente pertencer à classe NP. Portanto, um problema NP-completo é também NP-difícil; mas um problema NP-difícil pode não ser NP-completo.

A complexidade dos problemas de otimização combinatória são, em geral, NP-difíceis e está diretamente relacionada ao enorme tamanho do espaço de busca desses

problemas. Assim, uma busca exaustiva pelo espaço de soluções é inviável, e é possível que, em muitos casos, fazendo-se a busca não exaustiva pelo espaço, não se consiga melhorar a solução encontrada, de modo que esta não seja um ótimo global. Tal situação faz dela, portanto, um ótimo local. Sendo assim, os algoritmos para encontrar as melhores soluções de um problema são projetados de modo a evitar que fiquem presos nesses ótimos locais e devolvam a melhor solução possível, ou uma solução próxima da melhor, em um tempo razoável.

O **Problema de Roteamento de Veículos (PRV)**, também conhecido como **Problema de Roteirização de Veículos** e **Problema de Roteamento de Veículos com Frotas Homogêneas (PRVFH)**, é um problema clássico de otimização combinatória proposto por [Dantzig e Ramser \(1959\)](#) como Problema de Despacho de Caminhões. Este problema especificamente modelava como uma frota homogênea de caminhões poderia suprir um conjunto de demandas de combustíveis por parte de consumidores a partir de um depósito central, percorrendo a menor distância possível. Alguns anos depois, [Clarke e Wright \(1964\)](#) generalizaram o problema para a definição em que ele é encontrado hoje: como suprir um conjunto de demandas de consumidores a partir de um depósito central usando uma frota de veículos com capacidade máxima.

[Lenstra e Kan \(1981\)](#) mostraram que esse problema possui complexidade NP-difícil, já que é considerado uma generalização do Problema do Caixeiro Viajante (PCV), onde a rota de um veículo representa uma instância de um problema do PCV. Portanto, métodos que devolvam soluções exatas são inviáveis por possuírem tempo de execução elevados e só são eficientes em pequenas instâncias do problema.

Segundo [Labadie, Prins e Prodhon \(2016\)](#), um modelo de um PRV é constituído por quatro componentes:

1. A rede, composta por vértices e arestas, geralmente representada por um grafo;
2. Os nós a serem visitados, denominados consumidores e que possuem, cada um, uma demanda;
3. A frota de veículos que realizará a tarefa;
4. Os depósitos, onde os veículos iniciam e terminam.

A rede representa os caminhos ou estradas que ligam os consumidores e os depósitos. Uma demanda pode ser um produto a ser entregue ou um serviço a ser realizado. Um depósito, por sua vez, representa o local onde os veículos estão estacionados ou são recarregados.

Os modelos atuais de PRV são ainda mais complexos por possuírem um conjunto de características específicas para incorporar problemas do mundo real, como janelas de

tempo para entregas e coletas nos consumidores, custos dinâmicos de caminhos para simular congestionamentos, quantidade de poluição emitida por um veículo etc.

O modelo clássico e mais simples do PRV é denominado **Problema de Roteamento de Veículos Capacitados (PRVC)** ou, ainda, PRV com capacidade, onde o grafo que representa a rede é completo e não direcionado, a frota de veículos possui capacidade homogênea e há apenas um depósito.

O PRVC pode ser definido formalmente como um grafo não direcionado e completo $G = (V, E)$, onde o conjunto $V = \{0, \dots, n-1\}$ representa os N vértices, sendo o vértice v_0 correspondente ao depósito e os vértices $v_{1..N}$ ao conjunto de consumidores ou clientes. No depósito v_0 está locado um conjunto K de veículos de capacidade Q_k . Cada consumidor possui uma demanda q_i , que deve ser satisfeita. O conjunto das arestas $E = \{(i, j) : i, j \in v\}$ representa o menor caminho entre cada vértice (i, j) do grafo, e para cada aresta existe um custo d_{ij} associado. Este custo pode ser simétrico, ou seja, $d_{ij} = d_{ji}$ caso o custo de visitar o vértice i e depois o j seja o mesmo que visitar j e depois o i , ou assimétrico, dependendo da modelagem do problema.

Uma solução do PRVC é composta de um conjunto de K partições X_{ij}^k , onde $i, j \in E$ e $k \in \{1, \dots, K\}$, e $X_{ij}^k = 1$ se o veículo k viaja do vértice i até o vértice j , e $X_{ij}^k = 0$, caso contrário. O objetivo do PRVC é encontrar tais partições, tal que:

- A distância total percorrida pelos veículos seja minimizada:

$$\min \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \sum_{k=1}^K d_{ij} X_{ij}^k \quad (2.1)$$

- Cada consumidor seja servido por apenas um veículo:

$$\sum_{k=1}^K \sum_{i=0}^{N-1} X_{ij}^k = 1, j \in \{1, \dots, N-1\} : i \neq j \quad (2.2)$$

$$\sum_{k=1}^K \sum_{j=0}^{N-1} X_{ij}^k = 1, i \in \{1, \dots, N-1\} : i \neq j \quad (2.3)$$

- A soma das demandas q_i dos consumidores não exceda a capacidade Q_k do veículo:

$$\sum_{i=0}^{N-1} \sum_{j=0}^{N-1} q_i X_{ij}^k \leq Q_k, k \in \{1, \dots, K\} \quad (2.4)$$

- Cada partição inicie e termine no depósito:

$$\sum_{j=1}^{N-1} X_{ij}^k = \sum_{j=1}^{N-1} X_{ji}^k \leq 1 \text{ para } i = 0 \text{ e } k \in \{1, \dots, K\} \quad (2.5)$$

- Haja, no máximo, K rotas para servir os consumidores:

$$\sum_{k=1}^K \sum_{j=1}^{N-1} X_{ij}^k \leq K \text{ para } i = 0 \quad (2.6)$$

É possível que a quantidade de veículos utilizadas também seja otimizada, buscando-se, então, a menor quantidade possível de veículos para suprir a demanda dos consumidores dentre os K veículos disponíveis.

2.1.1 Exemplo de PRVC

Seja o modelo de PRVC com $N = 8$ vértices representado na figura 1, cujo depósito dispõe de $K = 3$ veículos de capacidade máxima $Q = 10$. A demanda de cada vértice é dada pelo vetor:

$$q = [0 \quad 4 \quad 3 \quad 5 \quad 2 \quad 3 \quad 4 \quad 3]$$

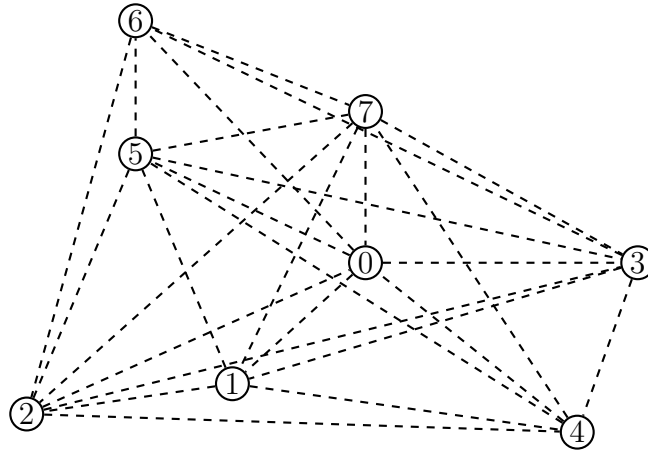


Figura 1 – Representação gráfica de um PRVC

Nota-se que o vértice v_0 representa o depósito e, portanto, sua demanda é $q_0 = 0$. A menor distância entre cada vértice é dada pela matriz 8×8 simétrica:

$$d = \begin{bmatrix} 0 & 3 & 6 & 5 & 4 & 4 & 6 & 3 \\ 3 & 0 & 3 & 7 & 7 & 4 & 6 & 5 \\ 6 & 3 & 0 & 10 & 11 & 5 & 7 & 8 \\ 5 & 7 & 10 & 0 & 3 & 8 & 9 & 5 \\ 4 & 7 & 11 & 3 & 0 & 7 & 7 & 4 \\ 4 & 4 & 5 & 8 & 7 & 0 & 2 & 4 \\ 6 & 6 & 7 & 9 & 7 & 2 & 0 & 4 \\ 3 & 5 & 8 & 5 & 4 & 4 & 4 & 0 \end{bmatrix}$$

Seja x_k o vetor que representa os vértices percorridos pela rota k . Uma solução possível para este modelo é a representada na Figura 2, composta das rotas $x_1 = \langle 0, 2, 4, 5, 0 \rangle$, $x_2 = \langle 0, 3, 6, 0 \rangle$ e $x_3 = \langle 0, 1, 7, 0 \rangle$ e que são representadas pelas matrizes:

$$X^1 = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, X^2 = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, X^3 = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Aplicando-se a restrição 2.4, é possível verificar que as demandas designadas a cada veículo não excedem sua capacidade:

$$\begin{aligned} q_0 \cdot X_{0,2}^1 + q_2 \cdot X_{2,4}^1 + q_4 \cdot X_{4,5}^1 + q_5 \cdot X_{5,0}^1 &\leq Q \\ 0 \cdot 1 + 3 \cdot 1 + 2 \cdot 1 + 3 \cdot 1 &\leq 10 \\ 8 &\leq 10 \end{aligned}$$

$$\begin{aligned} q_0 \cdot X_{0,3}^2 + q_3 \cdot X_{3,6}^2 + q_6 \cdot X_{6,0}^2 &\leq Q \\ 0 \cdot 1 + 5 \cdot 1 + 4 \cdot 1 &\leq 10 \\ 9 &\leq 10 \end{aligned}$$

$$\begin{aligned} q_0 \cdot X_{0,1}^3 + q_1 \cdot X_{1,7}^3 + q_7 \cdot X_{7,0}^3 &\leq Q \\ 0 \cdot 1 + 4 \cdot 1 + 3 \cdot 1 &\leq 10 \\ 7 &\leq 10 \end{aligned}$$

Além disso, como cada consumidor está contido em apenas uma rota, há exatamente K rotas para suprir as demandas e todas iniciam e terminam no depósito; nenhuma outra restrição é quebrada, fazendo com que esta seja uma solução *factível* para o problema. A distância total percorrida é encontrada aplicando-se a equação 2.1:

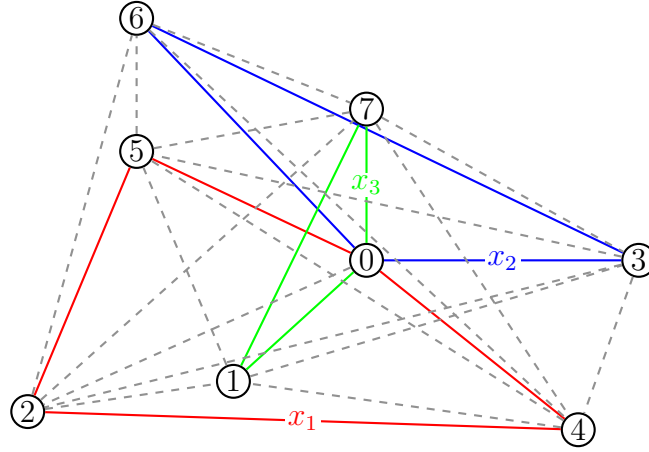


Figura 2 – Representação gráfica da solução composta das rotas $x_1 = \langle v_0, v_4, v_2, v_5, v_0 \rangle$, $x_2 = \langle v_0, v_3, v_6, v_0 \rangle$ e $x_3 = \langle v_0, v_1, v_7, v_0 \rangle$

$$\begin{aligned}
 \sum_{i=0}^7 \sum_{j=0}^7 \sum_{k=1}^3 d_{ij} X_{ij}^k &= (d_{0,2} \cdot X_{0,2}^1 + d_{2,4} \cdot X_{2,4}^1 + d_{4,5} \cdot X_{4,5}^1 + d_{5,0} \cdot X_{5,0}^1) \\
 &\quad + (d_{0,3} \cdot X_{0,3}^2 + d_{3,6} \cdot X_{3,6}^2 + d_{6,0} \cdot X_{6,0}^2) \\
 &\quad + (d_{0,1} \cdot X_{0,1}^3 + d_{1,7} \cdot X_{1,7}^3 + d_{7,0} \cdot X_{7,0}^3) \\
 &= (6 + 11 + 7 + 4) + (5 + 9 + 6) + (3 + 5 + 3) \\
 &= 28 + 20 + 11 = 59
 \end{aligned}$$

Apesar de factível, a solução não possui a menor distância possível a ser percorrida pelos veículos. Observando a Figura 3, pode-se verificar que uma solução possível é composta das rotas $x_1 = \langle v_0, v_1, v_2, v_0 \rangle$, $x_2 = \langle v_0, v_3, v_4, v_0 \rangle$ e $x_3 = \langle v_0, v_5, v_6, v_7, v_0 \rangle$. Aplicando-se todas as equações nesta solução, tem-se que a distância total percorrida é $12 + 12 + 15 = 39$ e que se trata de uma solução factível, portanto uma solução melhor que a anterior. De fato, não há nenhuma outra combinação de rotas com menor distância do que essa, e esta solução é denominada **ótimo global** do problema.

Como pode ser visto, é relativamente fácil encontrar soluções factíveis para um PRVC, mas o mesmo não se pode afirmar para os ótimos globais. Isso porque não há um método simples e direto para encontrar a melhor solução como existe para verificar se uma dada solução é factível, sendo necessário fazer uma busca exaustiva dentre todas as soluções possíveis para encontrá-la.

2.2 Variantes

Assim como outros problemas de otimização combinatória, o PRV representa uma modelagem matemática de um problema real do cotidiano. São exemplos de instâncias

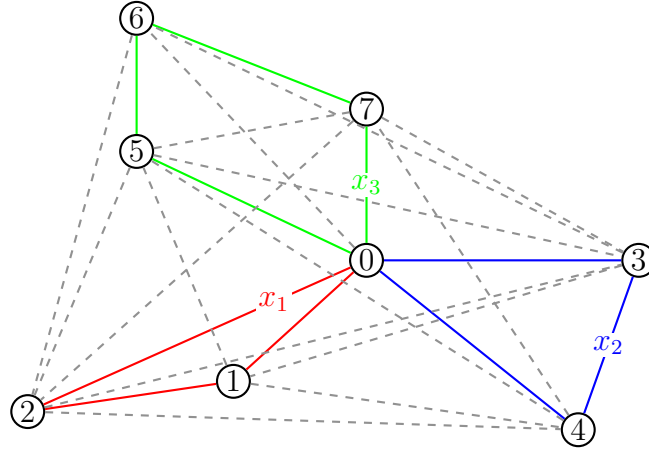


Figura 3 – Representação gráfica da solução ótima composta das rotas $x_1 = \langle v_0, v_1, v_2, v_0 \rangle$, $x_2 = \langle v_0, v_3, v_4, v_0 \rangle$ e $x_3 = \langle v_0, v_5, v_6, v_7, v_0 \rangle$

reais do problema o abastecimento de combustíveis para postos de gasolina e companhias, a coleta de lixo, o serviço de correios ou entrega de encomendas, o transporte de funcionários de empresas e a distribuição de alimentos [11].

Para aumentar a fidelidade do modelo, diversas variantes do PRV surgiram na literatura durante os anos, incorporando mais complexidades que ocorrem nessas situações do cotidiano. Algumas das mais conhecidas são:

- **PRV com multi-depósitos (PRVMD):** nesta variação, pode haver mais de um depósito no problema, cada um com seus veículos.
- **PRV com janelas de tempo (PRVJT):** cada consumidor ou depósito possui uma janela de horários em que podem ser visitados, e os veículos devem respeitar esta restrição em suas rotas.
- **PRV com coleta e entrega (PRVCE):** um consumidor pode ser um ponto de coleta ou entrega. A soma dos produtos coletados pelo veículo não deve exceder sua capacidade.
- **PRV com backhauls (PRVB):** similar ao PRVCE, com a diferença de que todas as entregas devem ser feitas na ida, e as coletas na volta para o depósito.
- **PRV com frotas heterogêneas (PRVFH):** cada veículo pode possuir capacidades de carga diferentes.
- **PRV dinâmico (PRVD):** o custo dos arcos que ligam os vértices altera-se em tempo de execução. Essas alterações podem simular engarrafamentos, acidentes etc.
- **PRV com rotas abertas (PRVRA):** os veículos não precisam retornar para o depósito ao servir todos os consumidores.

- **PRV verde**¹ (**PRVV**): incorpora problemas ambientais ao PRV, como poluição gerada, gasto de combustíveis etc.

Esses problemas geralmente são combinados, ou seja, não são mutualmente exclusivos. Como o objetivo primário de grande parte dos trabalhos relacionados ao PRV envolve a resolução de algum problema encontrado pelos pesquisadores no dia a dia, poucos abordam apenas um subproblema específico para solucionar, mas sim um conjunto deles que representa um modelo mais fiel da realidade [10]. Essas combinações de diversas variantes em um único problema são geralmente denominadas de **PRV rico (PRVR)**² [12], ou **PRV multi-atributo (PRVMA)** [13].

Outro fator comum em artigos que abordam o problema é a utilização de mais de um objetivo para ser atingido. A família de problemas com essas características é denominada **PRV multi-objetivo (PRVMO)**. Os objetivos mais comumente abordados são: minimização da distância total percorrida pelos veículos, o tempo total gasto em uma rota, o custo monetário, o tamanho da frota, a qualidade do serviço ou ainda o lucro coletado na rota [12].

Durante o passar dos anos, vários autores apresentaram classificações sistemáticas das diversas variações existentes na literatura, como os trabalhos de Bodin e Golden (1981), Ronen (1988) e Eksioglu, Vural e Reisman (2009).

2.3 Métodos de Soluções

Dada a complexidade de se encontrar o ótimo global em um problema de otimização combinatória, vários tipos de métodos foram desenvolvidos ao longo do tempo procurando minimizar o tempo gasto na busca e maximizar a qualidade da solução encontrada.

2.3.1 Métodos exatos

Os algoritmos que sempre conseguem encontrar o ótimo global independentemente da instância são denominados **métodos exatos**. Nestes algoritmos, geralmente é realizada uma busca exaustiva no espaço de soluções possíveis, algumas vezes aproveitando certas características do problema para diminuir o espaço de busca. Métodos exatos podem ser divididos em dois tipos [17]:

- **Enumeração explícita**: todas as soluções factíveis são construídas para se obter a solução ótima. Exemplo: busca exaustiva.

¹ Tradução direta do inglês de *Green VRP*

² Tradução direta do inglês de *Rich VRP*

- **Enumeração implícita:** todas as soluções factíveis são consideradas e implicitamente avaliadas, porém não são explicitamente construídas. Exemplo: *Branch-and-bound*, método Simplex.

Em problemas de otimização cujas modelagens consistem em uma programação linear inteira, a técnica denominada **relaxamento** pode ser aplicada para otimizar a busca pelo ótimo global. Nessa técnica, algumas restrições do problema original são desconsideradas, gerando um problema relacionado que pode ser resolvido em tempo polinomial. Entretanto, esse problema relacionado pode não ter o mesmo ótimo global que o do problema original, e, portanto, deve-se levar em consideração a distância do ótimo original que o relaxamento pode gerar.

Outra técnica utilizada para resolver problemas de programação linear inteira são os procedimentos denominados **métodos de plano de corte**. Neles, a função objetivo é iterativamente refinada por inequações lineares, chamadas cortes [18]. Estão contidos nessa técnica os métodos *Branch-and-bound* e *Branch-and-cut*.

Laporte e Nobert (1987) apresenta alguns dos métodos exatos mais conhecidos na literatura para o PRVC. Como problemas de otimização combinatória em geral apresentam complexidade NP-complexo ou NP-difícil, há uma grande limitação nos métodos exatos por conta do alto custo computacional para se chegar ao ótimo global, graças ao grande espaço de busca desses problemas. Isso faz com que sejam pouco abordados na literatura para resolver instâncias mais complexas do problema.

2.3.2 Métodos heurísticos

Os métodos denominados **heurísticos**, ao contrário dos algoritmos exatos, não fornecem o ótimo global ou ótimos com um limite formal de qualidade. A ideia desses métodos é fornecer soluções satisfatórias, com a menor distância possível do ótimo global, e com um custo computacional mais baixo. Isso faz com que se tornem métodos mais viáveis na prática, sendo muito utilizados comercialmente para resolver rapidamente problemas de otimização [10].

Alguns dos tipos de heurísticas encontradas na literatura são as denominadas **construtivas**. O Algoritmo *Savings* de Clarke & Wright [9] é uma das heurísticas construtivas mais conhecidas para o PRVC, devido à sua simplicidade e à qualidade das soluções geradas. Nesse método, a solução é construída iterativamente baseada no valor de *savings* entre cada consumidor, realizando sucessivas combinações nas rotas.

Outro tipo bastante difundido na literatura são as **heurísticas de melhoria**. Dentre essas, encontra-se o mecanismo λ -opt para o PRVC, onde λ arestas são removidas das rotas e todos os λ segmentos restantes são reconectados de todas as combinações possíveis. Seus custos são, então, recalculados, e, se alguma melhoria é identificada

com as conexões realizadas, elas são mantidas. Caso não haja nenhuma melhoria por sucessivas iterações, o algoritmo é finalizado [10].

Os trabalhos de Laporte et al. (2000) e Cordeau et al. (2005) apresentam uma visão mais abrangente de vários tipos de heurísticas propostas durante os anos para o PRVC. Todas elas possuem em comum uma incapacidade de realizar melhorias em determinadas soluções do espaço de busca, denominadas ótimos locais do problema. Assim, foi necessário o desenvolvimento de algum tipo de mecanismo para evitá-los e garantir a qualidade das soluções encontradas.

2.3.3 Metaheurísticas

Com o passar dos anos, foram desenvolvidas as técnicas denominadas **metaheurísticas** para definir formas de contornar o problema enfrentado pelas heurísticas no melhoramento das soluções. São consideradas “heurísticas de heurísticas” por apresentarem algum mecanismo heurístico de escape para que o algoritmo não fique preso nos ótimos locais, de modo a possuírem maiores chances de encontrar o ótimo global.

O termo metaheurística foi cunhado por Glover (1986 apud Sörensen[23], 2013). Segundo a definição do autor:

Metaheurísticas são um *framework* algorítmico em alto-nível e independente de problema que provém um conjunto de instruções ou estratégias para desenvolver algoritmos heurísticos de otimização. O termo é também utilizado para referenciar uma implementação de um algoritmo heurístico de otimização específica para um problema, de acordo com as instruções expressadas no *framework*.

Assim como os métodos heurísticos, metaheurísticas não possuem garantias de encontrar o ótimo global do problema, tampouco apresentam um limite formal na distância encontrada do ótimo. Ainda assim, esses métodos são os mais difundidos na literatura, graças ao custo computacional mais baixo que em métodos exatos e aos bons resultados encontrados na prática por eles [24].

Grande parte desses métodos baseia-se em metáforas do mundo real, como processos naturais e comportamento coletivo de diferentes espécies na natureza.

Algumas soluções mais recorrentes da literatura que se enquadram nessa categoria são Algoritmos Genéticos, Busca Tabu, Otimização baseada em Colônia de Formigas, Busca Variável em Vizinhança ³, Busca Local Iterada, Procedimento de Busca Gulosa Aleatória e Adaptável ⁴, dentre muitas outras [6].

³ Tradução direta do inglês de *Variable Neighborhood Search* (VNS)

⁴ Tradução direta do inglês de *Greedy Randomized Adaptive Search Procedure* (GRASP)

3 Revisão Sistemática da Literatura

Uma revisão da literatura é a forma pelo qual os pesquisadores identificam o conhecimento científico em uma determinada área de pesquisa, fazendo, assim, um planejamento do escopo da pesquisa para evitar redundâncias com o conhecimento já existente e repetição de erros. Entretanto, uma revisão feita de maneira *ad hoc*, ou seja, sem um protocolo pré-estabelecido, pode ser dirigida por interesse pessoal dos pesquisadores, levando a resultados poucos confiáveis e não repetíveis, e, conseqüentemente, com pouco valor científico.

Dessa forma, uma revisão sistemática é uma maneira de identificar, avaliar, interpretar e documentar toda pesquisa disponível e relevante de uma determinada área, com o intuito de que os futuros leitores possam avaliar o rigor, a completude e o grau de repetibilidade do processo por meio da definição de um **protocolo de revisão sistemática**. Trata-se de uma forma de estudo secundário, ou seja, analisa um conjunto de estudos primários para agregar os resultados a fim de produzir evidências mais fortes de um determinado fenômeno [25].

Diversas revisões da literatura foram realizadas para o PRV, como os trabalhos de Laporte (1992), Toth e Vigo (2002), Cordeau et al. (2005), Golden, Raghavan e Wasil (2008), Kumar e Panneerselvam (2012) e Braekers, Ramaekers e Nieuwenhuyse (2016). Porém, nem todos trataram especificamente do PRVC nem focaram métodos metaheurísticos. Além disso, o trabalho de revisão mais recente analisa os artigos publicados entre 2009 e 2013, sendo necessária uma revisão mais atualizada e específica a metaheurísticas aplicadas ao PRVC.

Este capítulo tem como objetivo, portanto, realizar uma revisão sistemática das publicações mais recentes relacionadas ao PRVC. Para tanto, será criado um protocolo de revisão utilizando as diretrizes propostas por Kitchenham (2007), que envolve três fases principais: (I) planejamento da revisão, (II) realização da revisão e (III) documentação da revisão.

Na Figura 4, mostra-se que a fase I é composta pela especificação das questões da pesquisa e pelo desenvolvimento do protocolo da revisão. A fase II é composta por quatro etapas: identificação e seleção das pesquisas, avaliação da qualidade das pesquisas selecionadas, da extração dos dados das pesquisas e, por fim, da sintetização dos dados. A última fase, por sua vez, é composta da análise dos resultados obtidos e da elaboração de um relatório da revisão realizada.

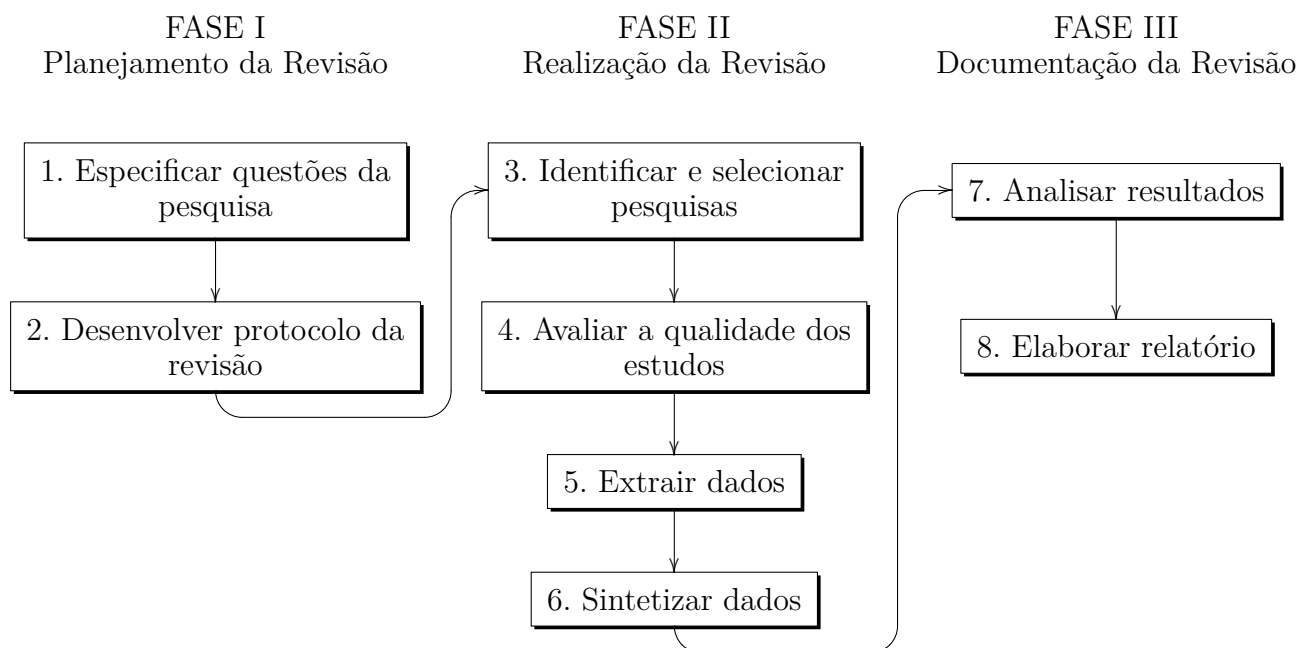


Figura 4 – Representação esquemática da metodologia da revisão sistemática

Fonte: Adaptado de [Arpini e Rosa \(2017\)](#)

3.1 Planejamento da Revisão

O objetivo desta seção é realizar a fase I da revisão sistemática, onde as questões da pesquisa serão especificadas, e, por fim, o protocolo será especificado para a realização da revisão.

3.1.1 Questões da Pesquisa

O primeiro passo na definição do protocolo é definir quais são as questões da pesquisa a serem respondidas durante o processo de revisão. Considerando o objetivo deste trabalho apresentado na seção 1.3, a revisão sistemática da literatura teve como objetivo a resposta à seguinte questão:

Quais serão as metaheurísticas utilizadas para a realização da análise proposta por este trabalho?

3.1.2 Protocolo de Revisão

Nesta seção, são descritas as diretrizes que conduziram o processo da revisão da literatura por meio do protocolo de revisão. Segundo o modelo de revisão proposto por [Kitchenham \(2007\)](#), o protocolo é constituído de cinco etapas: (I) identificação dos trabalhos, (II) seleção dos trabalhos, (III) avaliação da qualidade dos trabalhos, (IV) forma de extração dos dados e (V) forma de sintetizar os dados.

Na fase I, para identificar os trabalhos, efetuaram-se buscas nas bases [Science Direct](#)¹, [Springer Link](#)², [IEEE Xplore Digital Library](#)³, [ACM Digital Library](#)⁴ e [BDTD](#)⁵, utilizando-se as palavras-chave “capacitated vehicle routing problem”, “problema roteamento veículos capacitados” e “problema roteirização veículos capacitados” no título, resumo e palavras-chave dos estudos e limitadas ao período de 2014 a 2017.

Para a seleção dos trabalhos na fase II, classificaram-se os artigos por ordem de relevância nas bases de busca e analisaram-se apenas os 200 primeiros artigos dos resultados da base, para restringir a quantidade de trabalhos. Como critério de inclusão, consideraram-se as pesquisas de qualquer país de origem, porém somente nos idiomas inglês e português. Além disso, foram incluídos apenas artigos científicos publicados em periódicos e congressos, teses e dissertações. Foram excluídos artigos repetidos, trabalhos que tratam somente de revisão e de taxonomia da literatura, além de artigos sobre variações do PRVC.

Na fase III, como forma de determinar a qualidade dos trabalhos encontrados para escolha dos métodos a serem implementado, levou-se em consideração as seguintes características, ordenadas decrescentemente em ordem de relevância:

1. Utilizou instâncias públicas nos experimentos
2. Possui código-fonte disponível
3. Foi publicado em uma revista ou periódico de alto impacto

Foi levado em consideração a utilização de instâncias públicas como maior relevância para que fosse possível contrastar os resultados obtidos pelo autor com os obtidos pelos experimentos deste trabalho. Em seguida, a disponibilidade de código-fonte foi a segunda característica de maior relevância, para que houvesse a possibilidade de implementar o método proposto pelo autor de maneira mais fiel o possível. Por fim, levou-se em consideração o periódico em que o artigo foi publicado, para selecionar os artigos que tenham sido mais rigorosamente avaliados e revisados. Um periódico foi considerado de alto impacto caso ele esteja contido dentre os periódicos da tabela 1.

Já na fase IV, foram extraídas das publicações informações básicas como autor, título da pesquisa, local e ano de publicação, além de informações da pesquisa como as características do PRVC estudado, os métodos descritos para solucionar o problema e os *benchmarks* utilizados nos experimentos, para serem enfim sintetizados na fase V.

¹ Disponível em: <<https://www.sciencedirect.com>>

² Disponível em: <<https://link.springer.com>>

³ Disponível em: <<http://ieeexplore.ieee.org>>

⁴ Disponível em: <<https://dl.acm.org/>>

⁵ Disponível em: <<http://bdttd.ibict.br>>

Tabela 1 – Periódicos de maior impacto na área

Fonte: Adaptado de [Jaegere, Defraeye e Nieuwenhuyse \(2013\)](#)

Periódico	Fator de Impacto
Expert Systems with Applications	2,203
Transportation Research	1,957
European Journal of Operational Research	1,815
Computers & Operations Research	1,720
Decision Support Systems	1,687
Computers & Industrial Engineering	1,589
Transportation Science	1,507
Applied Mathematics and Computation	1,317

3.2 Realização da Revisão

Nesta seção, é descrito o resultado de realização do protocolo de revisão sistemática definido anteriormente.

Na fase I do protocolo, aplicando-se as palavra-chaves nas bases de busca e utilizando-se o período determinado, foram obtidos 983 artigos na base Science Direct, 808 na Springer Link, 53 na IEEE Xplore Digital Library, 1568 na ACM Digital Library e 37 na BDTD.

Para filtrar os trabalhos na fase II, foram selecionados, empregando-se os critérios de inclusão e exclusão determinados, 10 artigos na base Science Direct, 5 na Springer Link, 10 na IEEE Xplore Digital Library, 2 na ACM Digital Library e 2 na BDTD. Os artigos selecionados estão dispostos nas tabelas 2 e 3, assim como as características para determinar a qualidade dos artigos apresentadas na fase III. A Figura 5 mostra a proporção entre cada tipo de métodos encontrados na revisão.

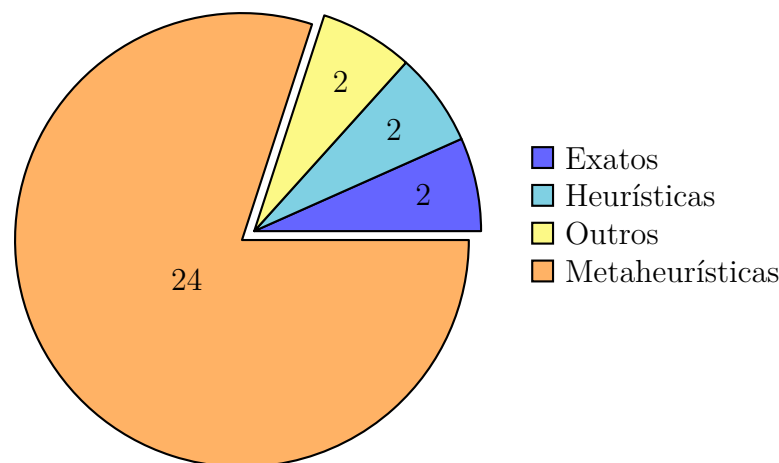


Figura 5 – Proporção entre os tipos de métodos encontrados na revisão sistemática

3.3 Documentação da Revisão

Nesta seção, são apresentados os trabalhos levantados na revisão sistemática para se ter uma visão geral acerca deles, além do processo de escolha das metaheurísticas utilizadas para a realização deste trabalho.

3.3.1 Artigos levantados

Jin, Crainic e Løkketangen (2014) introduz, em seu artigo, uma metaheurística paralela e cooperativa, que consiste em múltiplas instâncias de Buscas Tabu funcionando paralelamente e que cooperam entre si por meio de trocas assíncronas de soluções ótimas em um *pool* de soluções. Foram propostos alguns diferenciais para atingir uma melhor performance no algoritmo, como alterações na Busca Tabu que se baseia em agrupamento de soluções em suas etapas de intensificação e diversificação, quatro variantes na vizinhança de reinserção e a possibilidade de envio de soluções não factíveis para o *pool* de soluções.

Dentre o grupo dos métodos híbridos e distribuídos, Akpinar (2016) propõe um algoritmo utilizando uma combinação de duas metaheurísticas. O algoritmo consiste em fazer uma Busca em Grande Vizinhança (BGV) com uma etapa adicional de uma Otimização por Colônia de Formigas (OCF) para diversificar mais a vizinhança da busca. O método foi, então, testado em quatro *benchmarks* contra duas variantes da BGV e sete outros métodos da literatura, com um resultado satisfatório do algoritmo desenvolvido.

Teymourian et al. (2016) desenvolve duas metaheurísticas para o PRVC: o Gotejamento de Água Inteligente (GAI) e a Busca do Cuco Avançada (BCA) para formar quatro variações de algoritmos híbridos: Algoritmo de Busca Local Híbrida (BLH) com o GAI e com a BCA, e o Algoritmo de Pós-Otimização Híbrida (APOH), também com ambas metaheurísticas.

Vidal (2016) apresenta, em seu trabalho, uma melhoria no algoritmo de *Split*, amplamente presente na literatura em Algoritmos Genéticos e heurísticas *Route-first Cluster-second*. Esse algoritmo é utilizado para particionar uma solução em rotas distintas de custo minimizado. Utilizando propriedades de grafos, o autor propôs uma alternativa simples e com um menor custo computacional $O(n)$ à já estabelecida abordagem por algoritmo de Bellman.

Zhao et al. (2016) utiliza uma hibridização de uma Busca em Vizinhança Variável (BVV) com uma metaheurística denominada Otimização por Erva Invasiva (OEI), que consiste em um algoritmo de busca aleatória para simular o processo de reprodução das ervas invasivas.

Grande parte dos trabalhos concentra-se nos métodos metaheurísticos, como o de Amous et al. (2017). Em sua pesquisa, foi proposta uma aplicação da Busca em Vizinhança

Variável para o PRVC utilizando quatro tipo de vizinhanças diferentes para aumentar a diversidade em cada ciclo de busca.

Bouزيد, Haddadene e Salhi (2017) propõe outra Busca em Vizinhança Variável combinando com um método para particionar *tours* gigantes denominado Divisão Lagrangiana, juntamente com um mecanismo de reparação.

Hosseinabadi et al. (2017) aplica no PRVC um método denominado Algoritmo de Busca Local por Emulação Gravitacional (BLEG). Nesse método, cada solução é considerada como um “corpo” que possui massa cujo valor é sua função objetivo, ou seja, quanto melhor a solução, maior é sua massa e maior é a força gravitacional que esse corpo exerce nos outros. A força gravitacional é calculada aplicando as Leis de Newton, e a busca ocorre por meio de um “ponteiro” que viaja através do espaço de soluções à procura do corpo que exerce a maior atração gravitacional possível.

Mohammed et al. (2017a) publicou dois trabalhos no período da revisão sistemática. No artigo em questão, foi desenvolvido um Algoritmo Genético para resolver um problema real de serviço de traslado de ônibus de uma universidade, que foi modelado como um problema de roteamento de veículos capacitados.

Já em Mohammed et al. (2017b), um Algoritmo do *K-ésimo* Vizinho mais Próximo (KVP) é implementado e aplicado em instâncias de *benchmarks* da literatura. O KVP é um algoritmo de classificação e regressão, onde a entrada são casos de teste e a saída é a classe relacionada ao objeto ou a estimativa para o objeto. No algoritmo descrito para o PRVC, o KVP é utilizado para calcular a estimativa de melhor distância entre cada consumidor e os depósitos para ser utilizado posteriormente na construção de uma boa solução.

Yu et al. (2017) propõe um Algoritmo de Busca por Organismos Simbióticos (BOS). Nesta metaheurística, são simuladas as estratégias de interações simbióticas por um organismo para sobreviver em um ecossistema. Foram desenvolvidos dois tipos de representação de soluções e duas estratégias de interação, competição e amensalismo, formando seis variações do algoritmo, que foram testadas em instâncias da literatura.

Niu et al. (2014) propõe um tipo de Algoritmo de Membrana, que consiste em uma estrutura paralela, distribuída e não determinística, onde algoritmos de Otimização por Colônia de Formigas são utilizados de forma distribuída nos dispositivos computacionais das membranas mais básicas para gerar uma solução para o PRVC.

Já Wodecki et al. (2014) apresenta um Algoritmo Memético implementado em CUDA, uma API utilizada para computação paralela, que combina Algoritmo Genético com Busca por Arrefecimento Simulado em um ambiente de computação paralela empregando o Modelo de Ilha. Cada GPU contém uma população em que são aplicados os operadores genéticos e uma busca local para melhoramento da solução. Como cada “ilha”

é computada paralelamente, há um ganho substancial em performance e escalabilidade no algoritmo.

Há também um quarto grupo, que consiste em métodos diversos, como métodos de aprendizado de máquina e de agrupamento e classificação. Dentre eles, está o trabalho de [Ewbank, Wanke e Hadi-Vencheh \(2015\)](#), que propõem uma Abordagem de Agrupamento *Fuzzy* não supervisionado para o PRVC. Nele, um algoritmo de atribuição redistribui as demandas entre os grupos baseado em suas notas de associação, respeitando a capacidade dos veículos. Além disso, uma rede neural é treinada para prever o parâmetro de *fuzzificação* mais adequado para o algoritmo.

[Pecin et al. \(2016\)](#) propõe uma versão melhorada do algoritmo de programação inteira *Branch-and-cut-and-price*. É introduzida uma técnica para redução do impacto dos Cortes em Subconjuntos de Linhas, muito utilizados na literatura para restringir a quantidade de cortes realizados nas variáveis da programação linear do problema. O algoritmo proposto incorpora também algumas melhorias de trabalhos anteriores, como Enumeração de Rotas e Ramificação Forte. Com as melhorias propostas, os autores obtiveram boa performance em instância com até 360 consumidores, antes só resolvidas por métodos heurísticos.

[Kır, Yazgan e Tüncel \(2017\)](#) apresenta, em seu artigo, uma abordagem de Busca por Grande Vizinhança Adaptável (BGVA) com Busca Tabu para o PRVC. O algoritmo foi testado tanto em uma instância real de um problema de uma empresa que produz móveis de escritório quanto em instâncias da literatura.

No trabalho de [Santillan et al. \(2017\)](#), é explorada uma implementação do algoritmo de Busca Cuco, utilizando Voos de Lévy com as operações *2-opt* e Ponte-Dupla. Em seus experimentos, os autores constataram que, à medida que as instâncias testadas aumentam, mais lenta é a convergência do algoritmo para a solução ótima, tendo obtido o ótimo conhecido em poucas instâncias dos testes.

[Akhand et al. \(2015a\)](#) introduz um método composto de duas etapas. A primeira compreende a atribuição de consumidores para os veículos utilizando um Algoritmo de Agrupamento por Varredura, que realiza o agrupamento a partir de um ângulo definido pelo usuário. A segunda etapa consiste em uma busca pela rota de custo ótimo em cada veículo, utilizando um método de inteligência coletiva denominado Método do Produtor-Aproveitador ⁶.

Já em [Akhand et al. \(2015b\)](#), os autores estendem seu trabalho anterior aplicando outras variações de algoritmos de inteligência coletiva na etapa de otimização de cada rota, como Otimização por Colônia de Formigas (ACO), Otimização por Enxame de Partículas (PSO), Método do Produtor-Aproveitador (PSM) e Otimização por Enxame

⁶ Tradução direta do inglês de *Producer-Scrounger Method* (PSM)

de Partículas com Tentativa de Velocidade (VTPSO)⁷, além de uma implementação de Algoritmo Genético (AG). Nos testes realizados pelos autores, a variação utilizando VTPSO foi a que produziu melhores resultados para o PRVC.

Chen et al. (2015) apresenta um Algoritmo Híbrido de Varredura com Dois Estágios. No primeiro estágio do Algoritmo de Varredura, os clientes são agrupados, e cada agrupamento é considerado como um veículo. No segundo estágio, cada agrupamento vizinho é recombinado para evitar ótimos locais. Em seguida, um Algoritmo Guloso é, então, utilizado para minimizar o custo de cada veículo.

Steinhaus, Shirazi e Sodhi (2015) propõe a utilização de uma Rede Neural Artificial (RNA) para o PRVC, utilizando um Mapeamento Auto-Organizado. Os autores estendem métodos existentes, fortalecendo o parâmetro de *bias* e introduzindo um mecanismo restrito para forçar a rede a explorar um número maior de soluções factíveis. Nos testes realizados pelos autores, o método conseguiu superar as já estabelecidas heurísticas construtivas Algoritmo de Varredura e o Algoritmo de Clarke & Wright.

Zhang e Lee (2015) apresenta o Algoritmo de Colônia de Abelhas Artificiais com Direcionamento de Rotas (RABC), uma adaptação do Algoritmo de Colônia de Abelhas Artificiais (ABC) para o PRVC com algumas melhorias no balanço dos efeitos das etapas de diversificação e intensificação do algoritmo. O método é comparado com o algoritmo original do ABC presente na literatura e obteve soluções com custos melhores do que os encontrados pelo antecessor.

Ruttanateerawichien, Kurutach e Pichpibul (2016) introduz uma versão mais eficiente e efetiva do Algoritmo da Bola Dourada. Esse método é uma metaheurística relativamente recente, baseada em múltiplas populações simulando o processo de busca de diferentes conceitos relacionados ao futebol. Uma solução do PRVC é representada por um time, e cada jogador representa uma rota. Todos os jogadores dos times são gerados aleatoriamente, exceto um, que é gerado utilizando o método *Savings* de Clarke & Wright. Em cada iteração, os jogadores podem melhorar sua qualidade por meio de procedimentos de treinamento, utilizando algoritmos *inter-rotas* e *intrarrotas*. Além disso, os jogadores podem mudar de um time para outro por um procedimento de transferência.

Em seu artigo, Yeh, Liu e Liao (2016) introduz uma Busca Local Iterada em Dois Estágios para o PRVC. O método é baseado no algoritmo de Busca Local Iterada com Descida Variável de Vizinhança e Ordenação Aleatória de Vizinhança (ILS-RVND), onde o método é quebrado em dois estágios. No estágio um, é realizada a geração da solução inicial. Utilizando o ILS-RVND, é gerado um conjunto de soluções factíveis. No estágio dois, utilizando as soluções geradas no estágio anterior, o ILS-RVND é novamente aplicado para realizar a busca por soluções melhores.

⁷ Tradução direta do inglês de *Velocity Tentative Producer-Scrounger Method* (VTPSM)

No artigo de [Mingprasert e Masuchun \(2017\)](#), é desenvolvido um Algoritmo Adaptativo de Colônia de Abelhas Artificiais ⁸. Utilizando como base o Algoritmo de Colônia de Abelhas Artificiais (ABC) da literatura, os autores estenderam o método para torná-lo adaptativo, utilizando o Algoritmo da Inserção Mais Próxima e o Método de Trocas *2-opt* na fase da criação da solução inicial e em fases de melhoramento das soluções.

Dois algoritmos híbridos baseados no Algoritmo de Otimização por Colônia de Formigas (ACO) são apresentados no trabalho de [Sun, Fu e Liu \(2017\)](#). Segundo os autores, como o ACO possui a limitação de ficar preso com uma certa frequência em ótimos locais, o método de Arrefecimento Simulado (SA) é utilizado para ajudar o algoritmo a fugir dessas situações e explorar novas soluções, formando, assim, o algoritmo híbrido denominado SACO. Além disso, outra versão, denominada DSACO e baseada no Algoritmo Decomposto de Otimização por Colônia de Formigas, é apresentada no artigo para obter soluções de melhor qualidade e diminuir o custo computacional.

[Xiao, Hajjam-El-Hassani e Dridi \(2017\)](#) apresenta o algoritmo bio-inspirado de Busca do Cuco por Voos de Lévy. O método é baseado no comportamento parasita de algumas espécies de Cuco, que utilizam ninhos de outros pássaros como hospedeiro para suas crias. Os Cucos voam seguindo um padrão denominado Voos de Lévy, vasculhando o espaço de busca e depositando seus ovos nos ninhos hospedeiros escolhidos. O algoritmo proposto é uma extensão de versões já existentes do método para PRVC, onde um ovo em um hospedeiro representa uma solução, e cada ninho, um indivíduo da população.

[Oliveira e Delgado \(2015\)](#) propõe, em seu artigo, um algoritmo para resolução do PRVC utilizando Simulações de Monte Carlo com amostragem aleatória simples em conjunto com a heurística *Savings* de Clarke & Wright. A técnica proposta pelos autores tem como ideia manter o algoritmo original de *Savings* para construção das soluções, porém utilizando as simulações para variar a ordem em que as arestas são percorridas na heurística, alterando, assim, as soluções finais e permitindo que diferentes rotas sejam geradas em cada simulação.

Em sua dissertação, [Lima \(2015\)](#) apresenta variações de Algoritmos Genéticos com representações cromossômicas e heurísticas alternativas. Foram propostas pelo autor três diferentes estratégias de representação cromossômica para codificação da solução, além do emprego da heurística de Gillet & Miller na geração das soluções incluídas na população inicial do AG e os algoritmos de Subida/Descida de Encosta para refinamento das soluções após certo número de gerações sem melhoria.

Já na dissertação de [Masutti \(2016\)](#), é proposto um *framework* inspirado no comportamento de abelhas, denominado *VRoptBees*. O *framework* apresentado é relacionado a outro algoritmo presente na literatura denominado *optBees*, um algoritmo de otimiza-

⁸ Tradução direta do inglês de *Adaptive Artificial Bee Colony Algorithm* (AABC)

ção baseado no comportamento coletivo das tomadas de decisões em colônias de abelhas e adaptado para solução de problemas de roteamento de veículos e problemas do caixeiro viajante.

3.3.2 Escolha das metaheurísticas

Esta seção apresenta a definição das duas metaheurísticas para a realização deste trabalho, com base nas três características de qualidade definidas na fase III do protocolo.

Considerando a primeira característica, temos que apenas o artigo de [Mohammed et al. \(2017a\)](#) não utilizava instâncias públicas em seus experimentos, e portanto foi desconsiderado do processo de escolha. Na segunda característica, apenas os artigos de [Vidal \(2016\)](#), [Sun, Fu e Liu \(2017\)](#) e [Oliveira e Delgado \(2015\)](#) disponibilizavam o código fonte implementado e utilizado nos experimentos. Desses três, apenas o artigo de [Sun, Fu e Liu \(2017\)](#) se tratava de uma metaheurística, sendo o primeiro escolhido para implementação.

Nesse artigo, o autor propõe dois tipos de metaheurísticas para o PRVC: o **Algoritmo Híbrido de Otimização por Colônia de Formigas (AHO CF)** e o **Algoritmo Decomposto de Otimização por Colônia de Formigas (ADO CF)**. O primeiro utiliza uma combinação das metaheurísticas Otimização por Colônia de Formigas (OCF) e Recozimento Simulado (RS) na busca de soluções para o PRVC. O segundo utiliza de mecanismos paralelos na combinação das metaheurísticas OCF e RS, e portanto, não foi utilizado na análise deste trabalho.

A segunda metaheurística foi definida com base na terceira característica de qualidade do protocolo - o impacto do periódico em que o artigo foi publicado. Segundo a relação de fatores de impacto apresentado na tabela 1, o periódico *Expert Systems with Application* detém o maior fator de impacto na área de otimização combinatória. Dos artigos levantados, apenas o trabalho de [Akpinar \(2016\)](#) foi publicado neste periódico no período da revisão. O método de **Busca Híbrida em Grande Vizinhança (BHGV)** proposto pelo autor foi então escolhido como a segunda metaheurística para a realização deste trabalho.

Tabela 2 – Informações de base, periódico e código-fonte dos artigos selecionados na revisão sistemática

Base	Artigo	Periódico publicado	Código-fonte
Science Direct	[32]	Computers & Operations Research	Não
	[3]	Expert Systems with Applications	Não
	[33]	Information Sciences	Não
	[34]	Computers & Operations Research	Sim
	[35]	Procedia Computer Science	Não
	[36]	Electronic Notes in Discrete Mathematics	Não
	[37]	Computers & Operations Research	Não
	[38]	Applied Mathematical Modelling	Não
	[39]	Journal of Computational Science	Não
	[40]	Journal of Computational Science	Não
	[41]	Applied Soft Computing	Não
Springer Link	[42]	Soft Computing	Não
	[43]	Parallel Processing and Applied Mathematics	Não
	[44]	Neural Computing and Applications	Não
	[45]	Mathematical Programming Computation	Não
	[46]	Journal of Industrial Engineering International	Não
	[47]	Journal of Industrial Engineering International	Não
IEEE	[48]	Electrical and Computer Engineering	Não
	[49]	Electrical Information and Communication Technology	Não
	[50]	Control, Automation and Robotics	Não
	[51]	Computational Science and Engineering	Não
	[52]	Systems, Man and Cybernetics	Não
	[53]	Computer and Information Science	Não
	[54]	Computer, Consumer and Control	Não
	[55]	Knowledge and Smart Technology	Não
	[4]	Advanced Information Technology, Electronic and Automation Control Conference	Sim
	[56]	Logistics and Supply Chain Management	Não
ACM	[57]	Brazilian Symposium on Information Systems	Sim
BDTD	[58]	Tese	Não
	[59]	Tese	Não

Tabela 3 – Informações de método e tipo de instância utilizada dos artigos selecionados

Artigo	Método(s) desenvolvido(s)	Tipo de instâncias
[32]	Metaheurística paralela e cooperativa	Públicas
[3]	Busca em Grande Vizinhança Híbrida	Públicas
[33]	Gotejamento Inteligente de Água e Busca Cuco	Públicas
[34]	Algoritmo <i>Split</i>	Públicas
[35]	Otimização Híbrida Discreta por Erva Invasiva	Públicas
[36]	Busca em Vizinhança Variável	Públicas
[37]	BVV com Relaxação Lagrangiana	Públicas
[38]	Busca Local por Emulação Gravitacional	Públicas
[39]	Algoritmo Genético	Reais
[40]	K-ésimo Vizinho Mais Próximo	Públicas
[41]	Busca por Organismo Simbiótico	Públicas
[42]	Algoritmo de Membrana	Públicas
[43]	Algoritmo Memético Paralelo Multi-GPU	Públicas
[44]	Abordagem por Agrupamento Fuzzy Não Supervisionado	Públicas
[45]	<i>Branch-and-cut-and-price</i>	Públicas
[46]	Busca Adaptativa em Grande Vizinhança com Busca Tabu	Reais, Públicas
[47]	Busca Cuco via Voos de Lévy	Públicas
[48]	Método de Produtor-Aproveitador	Públicas
[49]	Diversas metaheurísticas de Inteligência Coletiva	Públicas
[50]	Algoritmo Híbrido de Varredura em Dois Estágios	Públicas
[51]	Rede Neural Auto-Organizável	Públicas
[52]	Algoritmo por Colônia Artificial de Abelhas	Públicas
[53]	Técnica baseada na Bola Dourada	Públicas
[54]	Busca Local Iterativa em Dois-Estágios	Públicas
[55]	Algoritmo Adaptativo por Colônia Artificial de Abelhas	Públicas
[4]	Otimização por Colônia de Formigas Híbrida	Públicas
[56]	Busca Cuco Estendida	Públicas
[57]	Métodos de Monte Carlo	Públicas
[58]	Algoritmo Genético	Públicas
[59]	Framework baseado no comportamento de abelhas	Públicas

4 Metaheurísticas escolhidas

Neste capítulo, são apresentados em detalhe os dois métodos determinados no capítulo anterior para a implementação e análise deste trabalho: o Algoritmo Híbrido de Otimização por Colônia de Formigas e a Busca Híbrida em Grande Vizinhança.

4.1 Algoritmo Híbrido de Otimização por Colônia de Formigas

Nesta seção, é descrito o Algoritmo Híbrido de Otimização por Colônia de Formigas (AHOCF), proposto por [Sun, Fu e Liu \(2017\)](#). O algoritmo consiste na hibridização de duas metaheurísticas bem conhecidas na literatura: a Otimização por Colônia de Formigas (OCF) e o Recozimento Simulado (RS).

O algoritmo inicia-se construindo uma população de N_{ants} soluções por meio do OCF. A seguir, uma busca local é aplicada na população utilizando operadores de Troca (*Swap*) e Mudança 2-opt (*2-opt Exchange*) para tentar gerar uma pequena melhora nas soluções encontradas.

A matriz de feromônios é, então, atualizada com base nas N_{elite} melhores soluções da população. Caso haja uma estagnação nas soluções encontradas pelo OCF, ou seja, caso a melhor solução encontrada pela metaheurística seja a mesma por K_t iterações, é realizada uma perturbação na matriz de feromônios para fazer com que o OCF tenha mais liberdade para explorar o espaço de busca nas próximas iterações.

Além disso, se a melhor solução encontrada até o momento (S_{best}) não se alterar por K_b iterações, é utilizada a metaheurística Recozimento Simulado na solução S_{best} para realizar uma busca por soluções melhores na vizinhança da melhor solução.

O algoritmo segue até que o número máximo de iterações ou o tempo máximo de execução seja atingido. As etapas do algoritmo são mostradas no pseudocódigo 4.1. Nas subseções seguintes, são detalhadas cada uma dessas etapas do algoritmo.

4.1.1 Mecanismo de Construção

Na variação da OCF utilizada pelo algoritmo, cada formiga representa uma solução com veículos que iniciam no depósito e vão construindo suas rotas consecutivamente.

Em cada passo da construção, a formiga k aplica a fórmula probabilística 4.1 para definir o próximo consumidor a ser visitado j com base no atual i , onde N_i^k são os possíveis consumidores da formiga k no consumidor i , $\eta_{(i,j)}$ representa a informação heurística entre os consumidores i e j , $\tau_{(i,j)}$ representa a quantidade de feromônio na aresta (i,j) , e α e

Algoritmo 4.1: PSEUDOCÓDIGO DO AHO CF

Entrada: $\tau_{inicial}$, N_{elite} , K_t , K_b
Saída: A melhor solução encontrada s_{best}

```

1  início
2  |  $\tau \leftarrow$  Inicializa matriz de feromônios( $\tau_{inicial}$ )
3  |  $t_{cnt} \leftarrow 0$ 
4  |  $b_{cnt} \leftarrow 0$ 
5  | enquanto diferente do critério de parada faça
6  |   |  $s \leftarrow$  Gera a população de soluções
7  |   | se melhor solução de  $s = s_{best}$  então
8  |   |   |  $t_{cnt} \leftarrow t_{cnt} + 1$ 
9  |   | fim
10 |   |  $s' \leftarrow$  Aplica Busca Local em cada indivíduo de  $s$ 
11 |   | se  $s' = s_{best}$  então
12 |   |   |  $b_{cnt} \leftarrow b_{cnt} + 1$ 
13 |   | senão se  $aptidão(s') < aptidão(s_{best})$  então
14 |   |   |  $s_{best} \leftarrow s'$ 
15 |   | fim
16 |   |  $\tau' \leftarrow$  Atualiza  $\tau$  com base nos  $N_{elite}$  indivíduos de  $s'$ 
17 |   | se  $t_{cnt} = K_t$  então
18 |   |   |  $\tau'' \leftarrow$  Aplica perturbação em  $\tau'$ 
19 |   |   |  $t_{cnt} \leftarrow t_{cnt} - 2$ 
20 |   | fim
21 |   | se  $b_{cnt} = K_b$  então
22 |   |   |  $s'' \leftarrow$  Aplica Recozimento Simulado em  $S_{best}$ 
23 |   |   | se  $aptidão(s'') < aptidão(s_{best})$  então
24 |   |   |   |  $s_{best} \leftarrow s''$ 
25 |   |   | fim
26 |   |   |  $b_{cnt} \leftarrow 0$ 
27 |   | fim
28 | fim
29 fim
30 retorna  $s_{best}$ 

```

β são os parâmetros que determinam, respectivamente, a influência dos feromônios e da informação heurística.

$$p_{(i,j)}^k = \begin{cases} \frac{\tau_{ij}^\alpha \times \eta_{ij}^\beta}{\sum_{l \in N_i^k} \tau_{il}^\alpha \times \eta_{il}^\beta}, & \text{se } j \in N_i^k \\ 0, & \text{caso contrário} \end{cases} \quad (4.1)$$

Utilizando um mecanismo de roleta, o consumidor com maior valor probabilístico $p_{(i,j)}^k$ possui maior chance de ser escolhido. Esse mecanismo é repetido até que todos os consumidores tenham sido visitados, e esse processo é repetido k vezes, gerando uma população de k soluções.

4.1.2 Busca Local

Após a fase de construção da população, são aplicados dois procedimentos de busca local em cada solução encontrada para tentar melhorá-las, baseados nas operações de Movimento de Troca e Movimento de Troca 2-opt.

O procedimento de busca local do AHOCF determina qual a solução seguinte baseada nas soluções geradas a partir da operação aplicada, denominadas vizinhança da solução. A melhor solução da vizinhança é escolhida, e o algoritmo continua com a busca até que não haja nenhuma melhoria na vizinhança.

No primeiro procedimento de busca, utiliza-se o Movimento de Troca, onde dois consumidores da solução são escolhidos aleatoriamente e suas posições são trocadas entre si. Esses consumidores podem estar tanto na mesma rota quanto em rotas da solução diferentes.

Já na segunda busca, o Movimento de Troca 2-opt, também conhecido como Movimento de Inversão, é utilizado para gerar a vizinhança da solução. Nesse movimento, dois consumidores de uma mesma rota são escolhidos aleatoriamente, e todos os consumidores entre os dois têm suas posições invertidas. A rota da solução em que a operação é realizada também é definida aleatoriamente.

O esquema geral da busca local pode ser visto no pseudocódigo 4.2.

4.1.3 Atualização dos Feromônios

Após a construção e o melhoramento das soluções, os feromônios τ são atualizados para reforçar as arestas mais presentes nas melhores soluções encontradas. Primeiramente, todos os feromônios são evaporados segundo a equação 4.2, onde ρ é o coeficiente de evaporação.

$$\tau(i, j) = (1 - \rho)\tau(i, j) \quad (4.2)$$

Em seguida, utiliza-se o sistema de formigas baseado em classificação (AS_{rank})¹ para atualizar os feromônios, onde apenas as n_e formigas mais bem classificadas, denominadas formigas de elite, depositam seus feromônios na matriz τ . O primeiro lugar da

¹ Tradução direta do inglês *Rank-based Ant System*

Algoritmo 4.2: PSEUDOCÓDIGO DA BUSCA LOCAL DO AHOCF

Entrada: Solução s
Saída: A solução melhorada s^*

```

1 início
2    $s^* \leftarrow s$ 
3   repita
4      $v \leftarrow \text{Troca}(s^*)$ 
5      $s' \leftarrow \text{Melhor solução de } v$ 
6     se  $\text{aptidão}(s') < \text{aptidão}(s^*)$  então
7        $s^* \leftarrow s'$ 
8     fim
9   até  $\text{aptidão}(s') < \text{aptidão}(s^*)$ ;
10  repita
11     $v \leftarrow \text{Troca 2-opt}(s^*)$ 
12     $s' \leftarrow \text{Melhor solução de } v$ 
13    se  $\text{aptidão}(s') < \text{aptidão}(s^*)$  então
14       $s^* \leftarrow s'$ 
15    fim
16  até  $\text{aptidão}(s') < \text{aptidão}(s^*)$ ;
17 fim
18 retorna  $s^*$ 

```

classificação fica com a melhor solução até o momento do algoritmo (S_{best}), seguido pelas $n_e - 1$ melhores soluções encontradas após o processo de busca local.

$$\tau_{ij} = \left[\sum_{r=1}^{n_e-1} (n_e - r) \Delta \tau_{ij}^r \right] + n_e \Delta \tau_{ij}^* \quad (4.3)$$

$$\Delta \tau_{ij}^r = 1/C^r$$

$$\Delta \tau_{ij}^* = 1/C^*$$

A atualização dos feromônios é definida pela equação 4.3, onde n_e é o número de formigas de elite, C^r é a aptidão da r -ésima melhor solução encontrada pelo processo de busca e C^* é a aptidão da solução S_{best} . Os feromônios são depositados com peso $(n_e - r)$ para as melhores formigas e n_e para a melhor solução até o momento.

A equação 4.4 condensa todo o processo de atualização realizado no AHOCF.

$$\tau_{ij} = (1 - \rho) \tau_{ij} + \left[\sum_{r=1}^{n_e-1} (n_e - r) \Delta \tau_{ij}^r \right] + n_e \Delta \tau_{ij}^* \quad (4.4)$$

4.1.4 Perturbação dos Feromônios

Como apenas as formigas de elite estão permitidas a depositar feromônios na matriz, pode haver um rápido aumento na quantidade de feromônios nas arestas das soluções construídas por elas. Isso pode levar à estagnação do processo, fazendo com que as formigas sempre construam soluções similares às anteriores.

Para evitar isso, o AHOCF emprega a estratégia de perturbação nos feromônios da equação 4.5 para gerar maior diversidade na formação das soluções.

$$\tau_{ij} = \delta\bar{\tau} + (1 - \delta)\tau_{ij} \quad (4.5)$$

Nessa estratégia, δ é a taxa de perturbação ($0 \leq \delta \leq 1$) e $\bar{\tau}$ é a quantidade média de feromônio na matriz τ . Quanto maior o valor de δ , mais perto da média ficará cada feromônio τ_{ij} após a perturbação, fazendo com que a OCF funcione como se estivesse sendo recomeçada. Por outro lado, caso δ seja muito pequeno, a perturbação não será evidente.

4.1.5 Recozimento Simulado

Apesar de o mecanismo de perturbação ajudar no escape da OCF de ótimos locais, ele não garante que as próximas soluções geradas serão melhores. Nesse contexto, o AHOCF emprega a metaheurística Recozimento Simulado (RS) para ajudar na diversificação da exploração por soluções melhores.

Baseada no processo de recozimento de um sólido, o algoritmo inicia a uma elevada temperatura inicial T_0 , resfriando-a com a busca na vizinhança da solução até que a temperatura final T_f seja alcançada. No movimento entre vizinhanças, melhores soluções são sempre aceitas, enquanto soluções piores são aceitas com a probabilidade $P(S')$ definida na equação 4.6.

$$P(S') = e^{-\frac{\Delta}{T(t)}} \quad (4.6)$$

$$\Delta = C(S') - C(S)$$

Nesse processo, S é a solução atual e $C(S)$ sua aptidão; S' é a nova solução e $C(S')$ sua aptidão; e $T(t)$ a temperatura atual. Em cada temperatura $T(t)$, são realizadas Z buscas na vizinhança da solução. Em seguida, a temperatura é reduzida utilizando a equação 4.7, onde λ é a taxa de redução da temperatura ($0 < \lambda < 1$).

$$T(t+1) = \lambda T(t) \quad (4.7)$$

Em cada iteração do processo de busca pela vizinhança, um dentre três tipos de operações é escolhido aleatoriamente para ser realizado: o Movimento de Troca, o

Movimento de Inversão (ou Movimento de Troca 2-opt) ou o Movimento de Inserção. Os Movimentos de Troca e Troca 2-opt são os mesmos utilizados no processo de Busca Local apresentados na subseção 4.1.2. No Movimento de Inserção, um consumidor é escolhido aleatoriamente e inserido em outra posição aleatória da solução, podendo ser da mesma rota ou de uma rota diferente.

Quando uma solução melhor que S_{best} é encontrada no movimento entre vizinhanças, é depositada a quantidade de feromônio da equação 4.8 nas arestas da solução na matriz τ , onde C_s é a aptidão da solução encontrada e n_{SA} é o peso da solução gerada pelo RS nos feromônios.

$$\tau_{ij} = \tau_{ij} + n_{SA} \times \frac{1}{C_s} \quad (4.8)$$

Emprega-se também uma lista Tabu para evitar buscas repetidas em uma solução da vizinhança que já tenha sido visitada anteriormente em um período recente. Nessa estrutura, guardam-se as SZ_{tabu} últimas soluções visitadas pelo algoritmo, fazendo com que o RS só aceite soluções que também não estejam presentes na lista no momento do movimento entre vizinhanças.

As etapas do algoritmo RS podem ser vistas no pseudocódigo 4.3.

4.2 Busca Híbrida em Grande Vizinhança

Nesta seção, é detalhado o algoritmo de Busca Híbrida em Grande Vizinhança (BHGv), proposto por Akpınar (2016) e implementado neste trabalho. O BHGV é um método híbrido composto de uma Busca em Grande Vizinhança (BGV) com uma etapa que utiliza o algoritmo de Otimização por Colônia de Formigas (OCF). As etapas e os detalhes do método são descritos nas subseções.

4.2.1 Busca em Grande Vizinhança

O algoritmo Busca em Grande Vizinhança (BGV)² é uma metaheurística de busca proposta inicialmente por Shaw (1998), onde uma solução inicial é gradualmente melhorada por meio de sua destruição e reparação.

O algoritmo começa com uma solução inicial s , definida aleatoriamente, que é também considerada a melhor solução até o momento s^* . O mecanismo de geração da vizinhança depende de duas operações sucessivas: a destruição da solução por uma heurística de remoção, na etapa de relaxamento, e a sua reparação por meio de uma heurística de inserção, na etapa de reotimização.

² Tradução direta do inglês de *Large Neighborhood Search* (LNS)

Algoritmo 4.3: PSEUDOCÓDIGO DO RECOZIMENTO SIMULADO NO AHO CF

Entrada: Melhor solução S_{best} , feromônios τ , temperatura inicial T_0 , temperatura final T_f , Z , λ , SZ_{tabu} , n_{SA}

Saída: A solução S_{best} melhorada, matriz τ de feromônios atualizada

```

1 início
2    $S \leftarrow S_{best}$ 
3    $tabu \leftarrow$  Lista Tabu vazia de tamanho  $SZ_{tabu}$ 
4    $T \leftarrow T_0$ 
5    $cnt \leftarrow 0$ 
6   enquanto  $T > T_f$  faça
7      $S' \leftarrow$  Aplica uma das três operações para gerar um vizinho de  $S$ 
8     se  $tabu$  não contém  $S'$  então
9        $tabu \leftarrow$  Adiciona solução  $S'$  em  $tabu$ 
10       $\Delta \leftarrow C(S') - C(S)$ 
11      se  $\Delta < 0$  então
12         $S' \leftarrow S$ 
13      senão
14         $rnd \leftarrow$  Gera um valor aleatório no intervalo uniforme  $(0, 1]$ 
15        se  $rnd < e^{-\frac{\Delta}{T(t)}}$  então
16           $S' \leftarrow S$ 
17        fim
18      fim
19      se  $C(S) < C(S_{best})$  então
20         $S_{best} \leftarrow S$ 
21         $\tau \leftarrow$  Atualiza feromônios( $\tau$ ,  $S_{best}$ ,  $n_{SA}$ )
22      fim
23      se  $cnt = Z$  então
24         $T \leftarrow \lambda T$ 
25         $cnt \leftarrow 0$ 
26      fim
27       $cnt \leftarrow cnt + 1$ 
28    fim
29  fim
30 fim
31 retorna  $S_{best}$ ,  $\tau$ 

```

Essas heurísticas variam com o problema que se está tentando resolver. Na variação Busca Adaptativa em Grande Vizinhança (BAGV), diferentes tipos de heurísticas podem ser utilizados na mesma operação de remoção e reparação, e um método é utilizado para decidir qual a melhor heurística a ser empregada no momento.

Alguns procedimentos de destruição conhecidos para o PRVC são a Heurística de

Remoção de Shaw, o Algoritmo de Remoção Aleatória e a Heurística da Pior Remoção ³. Dentre os procedimentos de reparação, estão a Heurística Gulosa Básica e a Heurística do Arrependimento ⁴ [61].

A etapa de relaxamento gera uma solução parcialmente destruída s' e um conjunto de consumidores removidos L^- . Esses consumidores são, então, enviados para a etapa de reparação para serem reinseridos na solução s' , formando a solução s'' . O movimento entre vizinhanças é, então, realizado, onde a solução s'' é avaliada por uma função de aceitação para se tornar a solução atual s , ou, ainda, a melhor solução até o momento s^* .

O algoritmo termina quando o critério de parada é alcançado. Ele pode ser um número máximo de iterações ou um número máximo de iterações sem melhoria da solução s^* . As etapas do algoritmo são mostradas no pseudocódigo 4.4.

Algoritmo 4.4: PSEUDO-CÓDIGO BGV

Entrada: Solução inicial $s \in \{\text{Soluções}\}$, $q \in \mathbb{N}$
Saída: A melhor solução encontrada s^*

```

1 início
2    $s^* \leftarrow s$ 
3   enquanto diferente do critério de parada faça
4      $s', L^- \leftarrow \text{destrói}(s)$ 
5      $s'' \leftarrow \text{reconstrói}(s', L^-)$ 
6     se  $\text{aceita}(s'')$  então
7        $s \leftarrow s''$ 
8     fim
9     se  $\text{aptidão}(s) < \text{aptidão}(s^*)$  então
10       $s^* \leftarrow s$ 
11    fim
12  fim
13 fim
14 retorna  $s^*$ 

```

Na Busca Híbrida em Grande Vizinhança, o autor propõe uma melhoria com um mecanismo híbrido na função de aceitação, utilizando a metaheurística de Otimização por Colônia de Formigas para construir soluções melhores caso a solução encontrada na BGV não seja melhor que a solução atual.

Dessa forma, o algoritmo modificado aceita apenas soluções que apresentem melhoria e constrói uma nova solução caso não seja satisfatória, fazendo com que o espaço de busca seja expandido e evitando paradas em ótimos locais. O pseudocódigo 4.5 apresenta o esquema geral do algoritmo.

³ Tradução direta do inglês de *Worst Removal Heuristic*

⁴ Tradução direta do inglês de *Regret Heuristic*

Algoritmo 4.5: PSEUDOCÓDIGO BHGV

Entrada: Solução inicial $s \in \{\text{Soluções}\}$, $q \in \mathbb{N}$
Saída: A melhor solução encontrada s^*

```

1  início
2       $s^* \leftarrow s$ 
3      repita
4           $s', L^- \leftarrow \text{PiorRemoção}(s)$ 
5           $s'' \leftarrow \text{Arrepende-2}(s', L^-)$ 
6          se  $\text{aptidão}(s'') < \text{aptidão}(s)$  então
7               $s \leftarrow s''$ 
8          senão
9               $s \leftarrow \text{constrói solução utilizando OCF}(S^*)$ 
10         fim
11         se  $\text{aptidão}(s) < \text{aptidão}(s^*)$  então
12              $s^* \leftarrow s$ 
13         fim
14     até número máximo de iterações não seja ultrapassado;
15 fim
16 retorna  $s^*$ 

```

O critério de parada ocorre quando o número máximo de iterações é atingido, e, nas etapas de destruição e reparação, o algoritmo utiliza as heurísticas da Pior Remoção e Arrepende-2, explicadas em detalhe nas próximas seções.

4.2.2 Heurística de Destruição

Na BHGV, a Heurística da Pior Remoção é utilizada como ferramenta para destruição da solução em cada iteração do algoritmo. Essa heurística baseia-se na ideia de remover os consumidores que induzem a um alto custo em sua posição atual da rota para serem posteriormente reinseridos em uma posição melhor.

Para tanto, o custo de servir o consumidor i em sua posição atual na solução s é determinado pela equação 4.9, onde $f(s)$ é o custo da solução completa e $f_{-i}(s)$ é o custo da solução sem o consumidor i .

$$\text{custo}(i, s) = f(s) - f_{-i}(s) \quad (4.9)$$

Os custos são ordenados de forma não crescente e utiliza-se um mecanismo de roleta para selecionar aleatoriamente um consumidor, onde quanto maior o $\text{custo}(i, s)$, maior a chance de ser removido. O número de consumidores a serem removidos pela heurística é parametrizada pelo valor q , e eles são inseridos em um conjunto de consumidores

removidos L^- . As etapas do algoritmo são descritas no pseudocódigo 4.6.

Algoritmo 4.6: PSEUDOCÓDIGO DA HEURÍSTICA DA PIOR REMOÇÃO

Entrada: $s \in \{\text{Soluções}\}$, $q \in \mathbb{N}$, $p \in \mathbb{R}_+$

```

1 início
2    $s' \leftarrow s$ 
3   enquanto  $q > 0$  faça
4      $L \leftarrow$  todos os consumidores ordenados de ordem decendente por
        $cost(i, s)$ 
5      $y \leftarrow$  número aleatório no intervalo  $[0, 1)$ 
6      $r \leftarrow L[y^p | L|]$ 
7      $s' \leftarrow$  remove  $r$  da solução  $s'$ 
8      $L^- \leftarrow$  adiciona  $r$  ao conjunto de consumidores removidos
9      $q \leftarrow q - 1$ 
10  fim
11 fim
12 retorna  $s', L^-$ 

```

4.2.3 Heurística de Reparação

A heurística de reinserção ou reparação utilizada no BHGV denomina-se Heurística Arrepende-2.⁵ Esse algoritmo é um melhoramento da Heurística Gulosa Básica, onde os consumidores são inseridos no lugar em que o aumento no custo da solução é o menor possível.

Para selecionar a melhor posição de reinserção de cada consumidor, é calculado o valor $\Delta f_{i, x_{i1}}$ como o custo de inserir o consumidor na melhor posição possível, e o valor $\Delta f_{i, x_{i2}}$ como o custo de inserir o consumidor na segunda melhor posição.

Em seguida, calcula-se o valor de “arrependimento” utilizando a equação 4.10, que representa a distância do custo gerado em inserir o consumidor nas duas posições. Quanto maior a distância, maior a necessidade de inserir o consumidor antes dos outros para que o custo final da solução não seja muito elevado.

$$r_i^* = \Delta f_{i, x_{i2}} - \Delta f_{i, x_{i1}} \quad (4.10)$$

Portanto, em cada iteração da Heurística Arrepende-2, é escolhido para inserção na melhor posição possível o consumidor que maximiza a equação $\max_{i \in U} r_i^*$, até que a solução seja inteiramente reconstruída. As etapas do algoritmo são mostradas no pseudocódigo 4.7.

⁵ Tradução livre do inglês *Regret-2 Heuristic*

Algoritmo 4.7: PSEUDOCÓDIGO DA HEURÍSTICA ARREPENDE-2

Entrada: Solução parcialmente destruída s^- , consumidores removidos L^-

```

1 início
2   para cada  $i \in L^-$  faça
3      $\Delta f_{i, x_{i1}} \leftarrow$  custo de inserir o consumidor  $i$  na primeira melhor posição
4      $\Delta f_{i, x_{i2}} \leftarrow$  custo de inserir o consumidor  $i$  na segunda melhor posição
5      $r_i^* \leftarrow \Delta f_{i, x_{i2}} - \Delta f_{i, x_{i1}}$ 
6   fim
7 fim
8 retorna  $s'$ 

```

4.2.4 Mecanismo de Construção

A BHGV utiliza a Otimização por Colônia de Formigas como heurística construtiva para gerar novas soluções quando não há melhoria no processo de destruição e reparação.

O algoritmo começa com a melhor solução até o momento S^* , utilizada para inicializar a matriz de feromônios η com o valor de $1/\text{aptidão}(S^*)$. Além disso, a distância d entre cada consumidor é usada para construir a matriz da informação heurística τ , baseado no valor de *Savings* da equação 4.11.

$$S_{ij} = d_{i0} + d_{0j} - d_{ij} \quad \forall i, j \in \{1, \dots, N\} : i \neq j \quad (4.11)$$

Em seguida, N_{ants} soluções são geradas pelo algoritmo, iniciando pelo depósito e selecionando o próximo consumidor j com base no consumidor atual i e nos mecanismos de construção apresentados na equação 4.12. Um número aleatório r determina qual o mecanismo utilizado para determinar cada próximo consumidor.

$$j = \begin{cases} \text{argmax}_{j \in A_i} [\tau_{ij}]^\alpha \times [\eta_{ij}]^\beta, & \text{se } r \leq r_1 & (I) \\ p(i, j_2) = \frac{[\tau_{ij_2}]^\alpha \times [\eta_{ij_2}]^\beta}{\sum_{j \in A_i} ([\tau_{ij}]^\alpha \times [\eta_{ij}]^\beta)} & \text{se } r_1 < r \leq r_1 + r_2 & (II) \\ \text{seleção aleatória de algum } j \in A_i, & \text{se } r_1 + r_2 < r \leq r_1 + r_2 + r_3 & (III) \end{cases} \quad (4.12)$$

Onde:

- (I) Neste mecanismo, denominado *exploração*⁶, o próximo consumidor j é aquele que possui maior feromônio e maior valor heurístico dentre todos os consumidores disponíveis para serem inseridos A_i .

⁶ Tradução direta do inglês *Exploitation*.

- (II) Denominado *exploração enviesada* ⁷, o próximo consumidor j é escolhido com base em um mecanismo de roleta. Cada possível consumidor j_2 possui uma probabilidade associada $p(i, j_2)$, calculada com base no valor relativo de feromônio e heurística. Quanto maior esse valor, maior é a probabilidade de o consumidor ser escolhido como próximo.
- (III) O próximo consumidor é escolhido aleatoriamente dentre todos os consumidores disponíveis A_i .

Os parâmetros α e β determinam respectivamente a importância dos feromônios e das informações heurísticas em cada mecanismo. Cada formiga constrói cada uma de suas K rotas de forma paralela, selecionando os consumidores até que todos sejam devidamente servidos.

Após o procedimento de construção em cada formiga, a matriz de feromônios η é atualizada com base na avaliação da aptidão da solução encontrada. Primeiramente, uma porção do valor de feromônio existente é evaporado em toda a matriz, de acordo com a equação 4.13, onde ρ é o coeficiente de evaporação ($0 \leq \rho < 1$).

$$\tau(i, j) = (1 - \rho)\tau(i, j) \quad (4.13)$$

A partir de então, cada formiga deposita uma quantidade de feromônio nas arestas das rotas que compõem a solução. Essa quantidade é dada pela equação 4.14 [62], onde Z é a avaliação da função de aptidão na solução encontrada pela formiga.

$$\Delta\tau_{(i,j)}^n = \begin{cases} \frac{1}{Z}, & \text{se a aresta (i,j) está presente na solução n} \\ 0, & \text{caso contrário} \end{cases} \quad (4.14)$$

A etapa de atualização dos feromônios pode, então, ser resumida na equação 4.15.

$$\tau(i, j) = (1 - \rho)\tau(i, j) + \sum_{n=1}^N \Delta\tau_{(i,j)}^n \quad (4.15)$$

Na BHGV, só são aceitas as soluções construídas pelo OCF que possuírem um padrão mínimo de qualidade, determinado pelo parâmetro ψ . Caso exista alguma rota da solução construída cujo comprimento com o fator ψ seja maior que o maior comprimento da melhor solução até o momento, essa solução não é aceita e é feito uma nova tentativa de construção.

Por fim, a melhor solução encontrada pelo OCF até o momento é atualizada, caso a solução de menor custo construída pela colônia na iteração seja melhor. O algoritmo

⁷ Tradução direta do inglês *Biased Exploitation*.

Algoritmo 4.8: PSEUDOCÓDIGO DA OTIMIZAÇÃO POR COLÔNIA DE FORMIGAS NA BHGV

Entrada: Melhor solução até o momento s^* , ρ , ψ , N_{iter} , distâncias d

```

1  início
2  |  $\eta \leftarrow$  inicializa matriz de informação heurística( $d$ )
3  |  $\tau \leftarrow$  inicializa matriz de feromônios( $s^*$ )
4  |  $s' \leftarrow$  vazio
5  | enquanto  $N_{iter} > 0$  faça
6  |   para cada  $n \in N_{ants}$  faça
7  |   |  $s_n \leftarrow$  constrói solução( $\eta$ ,  $\tau$ )
8  |   | se factível( $s_n$ ) e
9  |   |   aptidão( $s_n$ ) < aptidão( $s'$ ) e
10 |   |   comprimento( $s_n$ ) < ( $\psi \times$  comprimento( $s'$ )) então
11 |   |   |  $s' \leftarrow s_n$ 
12 |   | fim
13 |   fim
14 |    $\tau \leftarrow$  atualiza feromônios( $\tau$ ,  $\rho$ ,  $s$ )
15 |    $N_{iter} \leftarrow N_{iter} - 1$ 
16 | fim
17 fim
18 retorna  $s'$ 
  
```

finaliza após N_{iter} iterações, e a melhor solução encontrada é devolvida para o BHGV. O processo do OCF na BHGV é esquematizado no pseudocódigo 4.8.

5 Desenvolvimento

Neste capítulo, são apresentados a implementação de um analisador de instâncias do repositório CVRPLib e das metaheurísticas escolhidas do capítulo anterior, além dos experimentos dos algoritmos em cima das instâncias para atender os objetivos especificados na seção 1.3.

5.1 Analisador das instâncias do problema

Desde que o PRVC foi enunciado, vários benchmarks foram utilizados para provar a qualidade dos métodos desenvolvidos na literatura. As primeiras instâncias a ganharem popularidade foram as propostas e utilizadas por [Augerat et al. \(1995\)](#), contendo as classes A, B e P. Posteriormente, as classes E, F e M foram propostas por [Christofides e Eilon \(1969\)](#), [Christofides \(1979\)](#) e [Fisher \(1994\)](#), respectivamente.

Essas seis classes formam o benchmark ABEFMP e foram exaustivamente utilizadas até a metade da década de 2000 para testar o desempenho de métodos exatos e heurísticos. Entretanto, esse benchmark é considerado fácil de se resolver nos dias atuais, sendo que 90 das 95 instâncias possuem menos de 100 consumidores na rede, e todos foram resolvidos por métodos exatos.

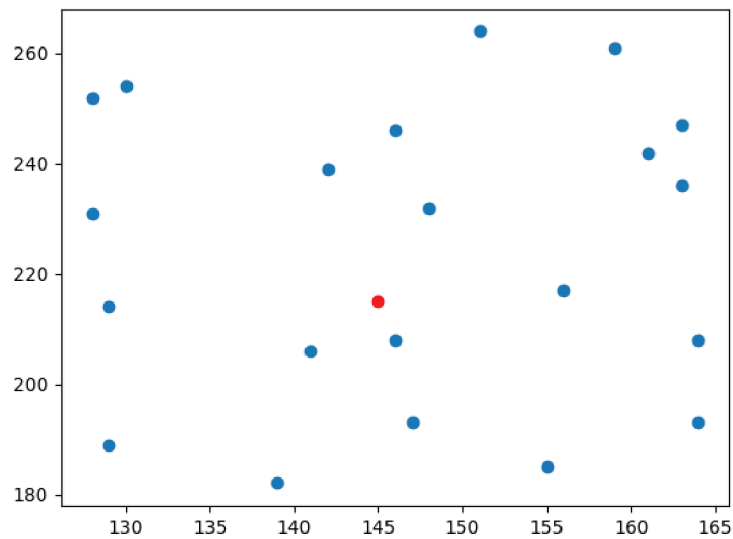


Figura 6 – Representação gráfica da instância E-n22-k4, presente na classe E, onde cada ponto azul representa um consumidor a ser visitado e cada ponto vermelho representa um depósito

Sendo assim, vários outros conjuntos foram propostos com o objetivo de aumentar a fidelidade da complexidade dos problemas da vida real. Dentre esses, [Uchoa et al. \(2017\)](#) propuseram um conjunto de 205 instâncias variando de 100 a 1000 consumidores, utilizando métodos variados na construção da rede.

Além disso, os autores criaram um repositório denominado [CVRPlib](#)¹, que centraliza as informações dos benchmarks e dispõe de mecanismos para verificar as melhores soluções encontradas e visualizar graficamente as instâncias mais utilizadas do PRVC, como as classes ABEFMP.

A maior parte das instâncias presentes no CVRPlib segue o mesmo formato de especificação de instâncias do Problema do Caixeiro Viajante, denominado TSPLIB95. Cada instância é descrita em um arquivo **.vrp**, contendo informações como as coordenadas, as demandas de cada consumidor e a capacidade dos veículos.

Nesse formato, o nome de uma instância é composto do conjunto a qual ela pertence, seguido do número de consumidores da rede e pelo número de veículos disponíveis. A listagem 5.1 apresenta um exemplo de instância no formato TSPLIB95.

```
NAME : E-n22-k4
COMMENT : (Christophides and Eilon, Min no of trucks: 4, Optimal value: 375)
TYPE : CVRP
DIMENSION : 22
EDGE_WEIGHT_TYPE : EUC_2D
CAPACITY : 6000
NODE_COORD_SECTION
1 145 215
2 151 264
3 159 261
4 130 254
5 128 252
...
DEMAND_SECTION
1 0
2 1100
3 700
4 800
5 1400
...
DEPOT_SECTION
1
-1
EOF
```

Listagem 5.1 – Esquema parcial da instância E-n22-k4 no formato TSPLIB95

Além disso, cada instância possui, ainda, um arquivo **.opt** ou **.sol** contendo, respectivamente, a solução ótima e a melhor solução conhecida, caso a instância não tenha

¹ Disponível em <http://vrp.atd-lab.inf.puc-rio.br/index.php>

sido resolvida ainda. Esses arquivos são preenchidos com as rotas percorridas por cada veículo e o custo da solução. A listagem 5.2 apresenta um exemplo de solução ótima no formato do TSPLIB95.

```
Route #1: 17 20 18 15 12
Route #2: 16 19 21 14
Route #3: 13 11 4 3 8 10
Route #4: 9 7 5 2 1 6
Cost 375
```

Listagem 5.2 – Esquema da solução ótima da instância E-n22-k4 no formato TSPLIB95

Este formato possui uma complexidade na leitura dos arquivos devido à não padronização de acordo com alguns formatos de dados mais utilizados nas linguagens de programação, como XML ou JSON. Além disso, não dispõe imediatamente de alguns dados importantes para o processamento da instância, como a matriz de distâncias, sendo necessário o cálculo da distância entre cada consumidor com base em suas coordenadas para montá-la.

Portanto, foi desenvolvido um analisador dos arquivos no repositório CVRPlib denominado *CVRPlib_{parser}*, convertendo do formato TSPLIB95 para o formato JSON para facilitar posteriormente a leitura pelos algoritmos. As coordenadas foram pré-processadas para formar a matriz de distâncias, e os arquivos da solução ótima e da melhor solução conhecida também foram analisados para compor o arquivo final. O fluxo do analisador pode ser visto na Figura 7.

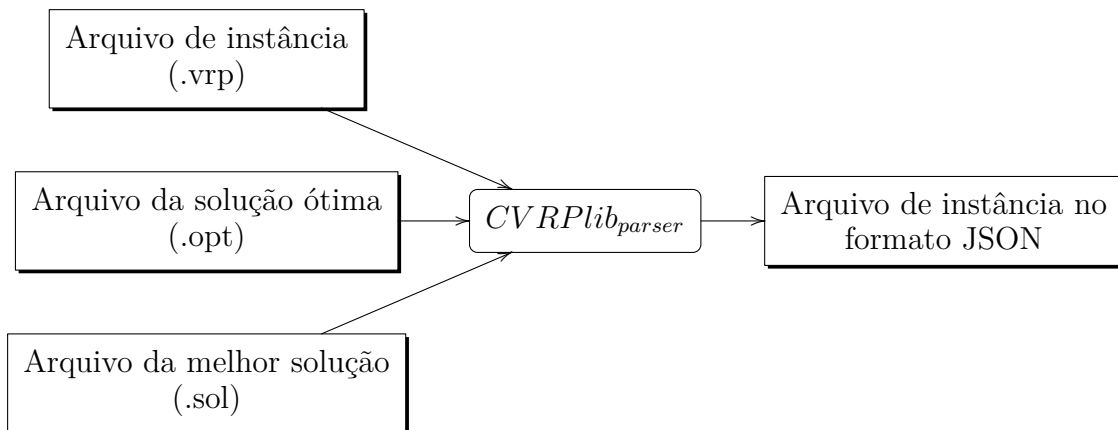


Figura 7 – Fluxo de execução do analisador *CVRPlib_{parser}*

O arquivo JSON de saída contém todos os campos presentes no arquivo de instância original no formato TSPLib, com a adição de dois novos campos: *distances* e *optimal*. A especificação de cada propriedade do arquivo de saída pode ser vista na tabela 4.

A propriedade *vehicles* representa a quantidade de veículos disponíveis no depósito, quando o problema modelado considerar também o número de veículos como uma

Tabela 4 – Propriedades do JSON de saída

Propriedade	Tipo	Descrição
name	String	Nome da instância
type	String	O tipo de problema da instância (CVRP)
comment	String	Comentários adicionais, como nome do autor
dimension	Number	Número total de nós
capacity	Number	Capacidade máxima de um veículo
vehicles	Number	Número de veículos, caso exista
edge_weight_type	String	Especifica como as distâncias são apresentadas
edge_weight_format	String	O formato em que as distâncias são especificadas
node_coord_section	Array	Coordenada de cada nó no formato [<i><no></i> , <i><x></i> , <i><y></i>]
demand_section	Array	Demanda de cada consumidor no formato [<i><consumidor></i> , <i><demanda></i>]
edge_weight_section	Array	Distância entre cada nó no formato especificado em <i>edge_weight_format</i>
distances	Array	Matriz de distâncias construída com base em <i>node_coord_section</i> ou <i>edge_weight_section</i>
optimal	Object	Solução ótima da instância, caso exista
best_known	Number	Melhor solução conhecida até o momento, caso exista

restrição, ou a quantidade mínima possível de veículos para aquela instância. Alguns algoritmos, como a BHGV, utilizam esse campo para definir um limitante superior como guia no processo.

O campo *distances* contém a matriz de distâncias da instância e é processado de acordo com os valores dos campos *edge_weight_type* e *edge_weight_format*. Caso a instância esteja definida por meio de suas coordenadas, a propriedade *edge_weight_type* será preenchida como o tipo de fórmula utilizada para calcular as distâncias. Nas instâncias do CVRPLib, os pesos de cada aresta são definidos como distâncias Euclidianas em 2 dimensões (“EUC_2D”), utilizando a equação 5.1, onde $[x]$ é a função de arredondamento ao inteiro mais próximo.

$$d_{ij} = \left\lceil \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \right\rceil \quad (5.1)$$

A matriz da instância pode também estar pré-definida por meio de uma matriz triangular superior ou inferior no campo *edge_weight_section*. Caso seja esse o caso, a propriedade *edge_weight_format* será preenchida com “EXPLICIT”, e será necessário transpor a matriz triangular para formar a matriz de distâncias completa.

Um exemplo de arquivo de instância resultante com consumidores representados por coordenadas Euclidianas é apresentado na listagem 5.3.

```

{
  "name" : "E-n22-k4",
  "type" : "CVRP",
  "comment" : "(Christophides and Eilon, Min no of trucks: 4, Optimal value: 375)",
  "dimension" : 22,
  "capacity" : 6000,
  "vehicles" : 4,
  "edge_weight_type" : "EUC_2D",
  "node_coord_section" : [ [ 1, 145, 215 ], [ 2, 151, 264 ], [ 3, 159, 261 ], [ 4,
↪ 130, 254 ], [ 5, 128, 252 ], [ 6, 163, 247 ], [ 7, 146, 246 ], [ 8, 161, 242 ],
↪ [ 9, 142, 239 ], [ 10, 163, 236 ], [ 11, 148, 232 ], [ 12, 128, 231 ], [ 13,
↪ 156, 217 ], [ 14, 129, 214 ], [ 15, 146, 208 ], [ 16, 164, 208 ], [ 17, 141, 206
↪ ], [ 18, 147, 193 ], [ 19, 164, 193 ], [ 20, 129, 189 ], [ 21, 155, 185 ], [ 22,
↪ 139, 182 ] ],
  "demand_section" : [ [ 1, 0 ], [ 2, 1100 ], [ 3, 700 ], [ 4, 800 ], [ 5, 1400 ], [
↪ 6, 2100 ], [ 7, 400 ], [ 8, 800 ], [ 9, 100 ], [ 10, 500 ], [ 11, 600 ], [ 12,
↪ 1200 ], [ 13, 1300 ], [ 14, 1300 ], [ 15, 300 ], [ 16, 900 ], [ 17, 2100 ], [
↪ 18, 1000 ], [ 19, 900 ], [ 20, 2500 ], [ 21, 1800 ], [ 22, 700 ] ],
  "distances" : [ [ 0, 49, 48, 42, 41, 37, 31, 31, 24, 28, 17, 23, 11, 16, 7, 20,
↪ 10, 22, 29, 31, 32, 34 ], [ 49, 0, 9, 23, 26, 21, 19, 24, 27, 30, 32, 40, 47,
↪ 55, 56, 57, 59, 71, 72, 78, 79, 83 ], [ 48, 9, 0, 30, 32, 15, 20, 19, 28, 25,
↪ 31, 43, 44, 56, 55, 53, 58, 69, 68, 78, 76, 81 ], [ 42, 23, 30, 0, 3, 34, 18,
↪ 33, 19, 38, 28, 23, 45, 40, 49, 57, 49, 63, 70, 65, 73, 73 ], [ 41, 26, 32, 3,
↪ 0, 35, 19, 34, 19, 38, 28, 21, 45, 38, 48, 57, 48, 62, 69, 63, 72, 71 ], [ 37,
↪ 21, 15, 34, 35, 0, 17, 5, 22, 11, 21, 38, 31, 47, 43, 39, 47, 56, 54, 67, 63, 69
↪ ], [ ... ] ],
  "optimal" : {
    "routes" : [ [ 17, 20, 18, 15, 12 ], [ 16, 19, 21, 14 ], [ 13, 11, 4, 3, 8, 10
↪ ], [ 9, 7, 5, 2, 1, 6 ] ],
    "cost" : 375
  }
}

```

Listagem 5.3 – Arquivo JSON parcial da instância E-n22-k4

5.2 Arquivo de resultados

Após a execução dos algoritmos, um arquivo de saída (.out) no formato JSON é produzido, conforme fluxo mostrado na imagem 8.

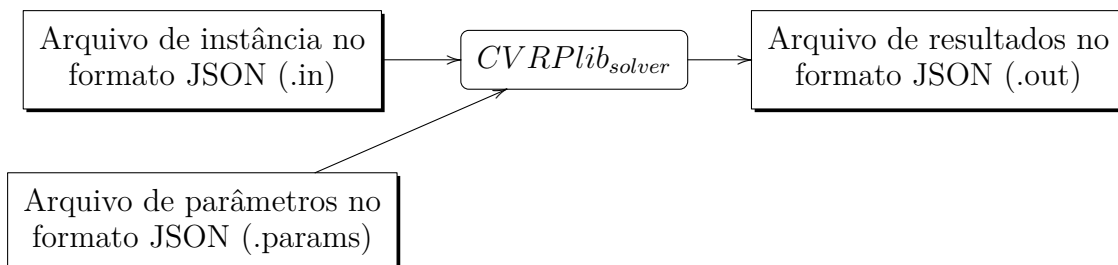


Figura 8 – Fluxo de execução das instâncias

Esse arquivo contém as informações da solução obtida, tempo gasto e parâmetros utilizados pelos algoritmos, e são utilizados para gerar as tabelas dos experimentos na seção 5.5. As propriedades do JSON deste arquivo podem ser vistas na tabela 5.

Tabela 5 – Propriedades do arquivo de resultados

Propriedade	Tipo	Descrição
instance	String	Nome da instância
parameters	Object	Os parâmetros recebidos pelo algoritmo
initial_solution	Object	Solução inicial, caso haja
solution	Object	Solução encontrada pelo algoritmo
start	String	Data de início no formato dd-mm-yyyy hh:mm:ss
end	String	Data de fim no formato dd-mm-yyyy hh:mm:ss
evaluations	Number	Número de avaliações da função objetivo
elapsed_time	Number	Tempo gasto pelo algoritmo, em segundos
optimal	Number	A solução ótima da instância, caso haja
best_known	Number	A melhor solução conhecida para a instância
dev	Number	Desvio relativo entre a solução encontrada e a melhor solução conhecida

A propriedade *parameters* contém o valor dos parâmetros recebidos em arquivo pelos algoritmos, além dos valores definidos em tempo de execução. Algumas propriedades podem não estar presentes no resultado, como é o caso da *initial_solution* em algoritmos que não utilizam o conceito de solução inicial e *optimal*, em instâncias em que o ótimo não é conhecido.

A listagem 5.4 mostra um exemplo de um arquivo de resultado da instância A-n22-k4, obtido na execução do algoritmo AHOCF.

```

{
  "instance" : "E-n22-k4",
  "optimal" : 375,
  "parameters" : {
    "max_iter" : 100,
    "num_elite_ants" : 6,
    "beta" : 2,
    "perturbation_ratio" : 0.7,
    "lambda" : 0.95,
    "initial_temperature" : 5.0,
    "sa_max_tries" : 250,
    "initial_trail" : 0.5,
    "max_best_so_far_not_improved" : 5,
    "num_ants" : 22,
    "max_runtime" : 600,
    "max_best_unchanged" : 22,
    "evaporation_rate" : 0.9,
    "tabu_size" : 3,
    "alpha" : 1
  },
  "start" : "04-06-2018 03:07:50",
  "end" : "04-06-2018 03:08:17",
  "elapsed_time" : 27.362290269,
  "solution" : {
    "tour" : [ 0, 15, 18, 20, 16, 0, 12, 9, 7, 5, 10, 0, 8, 6, 2, 1, 3, 4, 11, 0,
    ↪ 14, 17, 21, 19, 13, 0 ],
    "cost" : 382.0
  },
  "dev" : 1.87
}

```

Listagem 5.4 – Arquivo de resultados da instância E-n22-k4

5.3 Implementação

Nesta seção são apresentados os detalhes da implementação dos componentes deste trabalho: o analisador *CVRplib_{parser}* e os algoritmos AHOCF e BHGV.

Os algoritmos foram implementados utilizando programação funcional com a linguagem Clojure ² 1.8, seguindo uma arquitetura declarativa que descreve *como* os dados de entrada (instâncias) serão processados pelos algoritmos, ao invés de modelar várias entidades e definir a forma com que elas se relacionam, como é o caso de uma arquitetura imperativa de uma programação orientada a objetos.

5.3.1 Analisador das instâncias

O código fonte da implementação do analisador *CVRplib_{parser}* está disponível no repositório <https://github.com/lucascb/cvrplib_parser> e no apêndice B, assim como

² Disponível em <<https://clojure.org/>>

o arquivo JSON de todas as instâncias do repositório CVRPLib.

Na implementação, o analisador foi dividido em quatro arquivos: *core*, *parser*, *distances* e *optimal*, como pode ser visto na Figura 9.

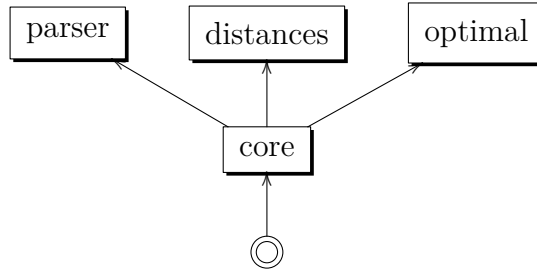


Figura 9 – Arquitetura geral da implementação do analisador *CVRPLib_{parser}*

O arquivo *core* contém o ponto de entrada da execução do processo, e é responsável por receber as instâncias a serem analisadas e executar cada etapa do analisador, além de gravar o resultado em um arquivo JSON.

O arquivo fonte *parser* contém os fluxos de análise de todas as seções do arquivo de instância .vrp no formato TSPLIB95. O ponto de entrada é a função *parse*, que recebe o conteúdo em texto do arquivo .vrp e extrai as informações gerais no cabeçalho do arquivo através de expressões regulares, e em seguida, o corpo do arquivo com o restante das informações. O resultado é uma estrutura de *hash-map* com todos os dados da instância.

O arquivo *distances* contém os códigos responsáveis pela geração da matriz de distâncias da instância, através das coordenadas Euclidianas ou da matriz triangular definida no arquivo. A função inicial *create-distances* recebe a estrutura de *hash-map* gerada na etapa anterior, retornando uma outra estrutura de *hash-map* contendo a matriz calculada.

Por fim, no arquivo *optimal* estão contidos os códigos referentes à análise dos arquivos de solução .opt e .sol no formato TSPLIB95, através das funções *parse-optimal* e *parse-best-known*, respectivamente. Essas funções recebem o conteúdo em texto desses arquivos, e retornam uma estrutura de *hash-map* contendo as rotas e o custo da solução ótima, ou apenas o custo da melhor solução conhecida, quando for o caso.

5.3.2 Algoritmo Híbrido de Otimização por Colônia de Formigas

Durante os procedimentos do AHOCF, utilizou-se uma representação da solução como uma lista unidimensional cujo tamanho é $N + K - 1$. Essa lista representa uma permutação dos consumidores, onde estão contidas consecutivamente as K rotas intercaladas pelo depósito, como pode ser visto na Figura 10.

Outro tipo de representação utilizado é a matriz de adjacências m de todas as rotas da solução, usada para facilitar o cálculo dos feromônios a serem depositados no algoritmo

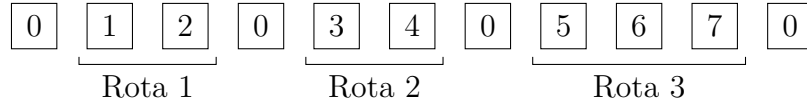


Figura 10 – Representação em lista da solução da Figura 3

de Otimização por Colônia de Formigas. A Figura 11 apresenta a mesma solução anterior representada por meio de matriz, construída por meio das regras definidas na equação 5.2.

$$\begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figura 11 – Representação da solução da Figura 3 por meio de uma matriz de adjacências

$$m_{ij} = \begin{cases} 1, \text{ caso a aresta } (i, j) \text{ esteja presente na rota de algum veículo da solução} \\ 0, \text{ caso contrário} \end{cases} \quad (5.2)$$

Os arquivos da implementação estão organizados segundo o esquema representado na Figura 12. O arquivo [core](#) contém o ponto de entrada do algoritmo, recebendo como parâmetro as instâncias a serem processadas, convertendo-as para uma estrutura de *hash-map*.

Essa estrutura é recebida então no arquivo [saco](#), que contém a esquema geral de funcionamento da metaheurística AHOCF. O ponto de entrada é a função *solve-instance*, que posteriormente invoca as etapas do mecanismo de construção, da busca local, da atualização dos feromônios e da busca pelo Recozimento Simulado.

Primeiramente, a população de soluções é construída utilizando o mecanismo de construção da OCF, presente no arquivo [aco](#). Através da função *construct-solutions*, a OCF recebe a matriz heurística e de feromônios e em seguida retorna uma lista de soluções construída com base nas matrizes.

Em seguida, essas soluções são repassadas para o mecanismo de busca local no arquivo [local_search](#), que retorna uma lista de soluções melhoradas pelo método.

As formigas de elite são definidas por meio das melhores soluções encontradas, e são passadas para os procedimentos de atualização e perturbação dos feromônios, presentes

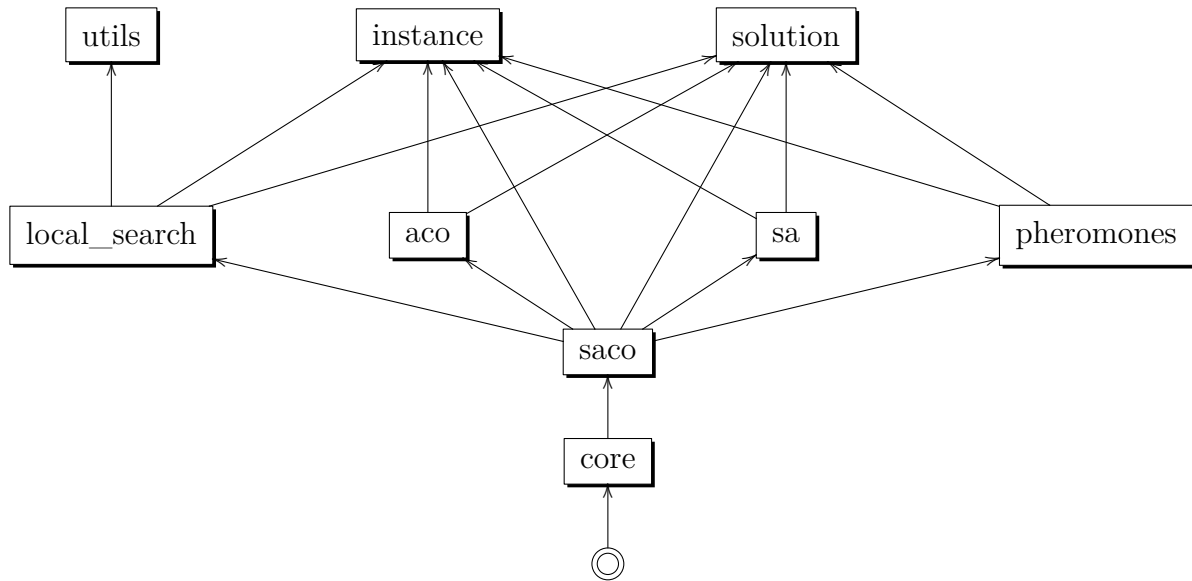


Figura 12 – Arquitetura geral da implementação do método AHOCF

no arquivo [pheromones](#).

Após a atualização dos feromônios, o mecanismo de Recozimento Simulado no arquivo [sa](#) é invocado, caso não haja melhorias nas últimas iterações do algoritmo. A função *improve-solution* é responsável por realizar tal feito, retornando uma solução possivelmente melhorada pelo método e a matriz de feromônios atualizada.

Estão contidas no arquivo [instance](#) as propriedades da instância sendo executada e os parâmetros do algoritmo, além das funções responsáveis pela geração do arquivo de resultado no formato JSON. Já no arquivo [solution](#), estão contidas a definição da estrutura de solução utilizada pelo algoritmo, além de funções auxiliares com o intuito de realizar o processamento da solução. Por fim, o arquivo [utils](#) contém a implementação da função *rand-map*, utilizada no algoritmo de busca local e retirada de <https://github.com/bodil/unpredictable>. A biblioteca foi inserida diretamente no fonte deste projeto para que a aleatoriedade dos algoritmos levassem em consideração a semente definida nos experimentos.

O código fonte da implementação do AHOCF está disponível no repositório https://github.com/lucasb/hybrid_aco e no apêndice C.

5.3.3 Busca Híbrida em Grande Vizinhança

Uma solução do PRVC na BHGV é representada por uma lista de rotas que iniciam e terminam no depósito da instância:

$$[[0, 1, 2, 0], [0, 3, 4, 0], [0, 5, 6, 7, 0]]$$

Da mesma forma, é também utilizada a representação por matriz da Figura 11, nos processos de atualização dos feromônios e cálculo da função objetivo.

A arquitetura da implementação é apresentada na Figura 13, onde o código fonte foi dividido nos arquivos `core` e `hybrid-lns`. O primeiro é responsável por receber as instâncias do PRVC a serem processadas e os parâmetros, e repassar a informação lida para o segundo arquivo pela função `start`, onde estão contidas toda as lógicas da BHGV. Por fim, o resultado obtido é gravado em um arquivo no mesmo formato JSON especificado na seção 5.2.

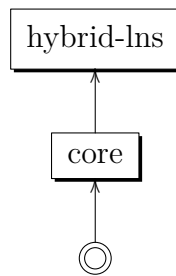


Figura 13 – Arquitetura geral da implementação do método BHGV

O código fonte da implementação da BHGV está disponível no repositório <<https://github.com/lucascb/hybrid-lns>> e no apêndice D.

5.4 Experimentos

Nesta seção, as metaheurísticas descritas no capítulo anterior serão experimentadas e analisadas em cima das instâncias do repositório CVRPlib em uma implementação baseada nos artigos de Akpınar (2016) e Sun, Fu e Liu (2017).

Primeiramente, cada instância do benchmark foi analisada e convertida para o formato de arquivo JSON, utilizando o algoritmo $CVRPlib_{parser}$ descrito na seção 5.1. Em seguida, os arquivos de instância em JSON foram executados em cada algoritmo implementado e com o respectivo arquivo de parâmetros, produzindo o arquivo de saída JSON (.out) com os resultados obtidos, conforme Figura 8.

Os parâmetros utilizados nos experimentos do algoritmo BGVH foram coletados de [3] e [62] e estão dispostos na tabela 6. Os parâmetros utilizados no algoritmo AHOCF estão dispostos na tabela 7 e foram retirados do artigo e do código-fonte disponibilizado pelo autor no repositório <https://github.com/i-sunny/cvrp_aco>. Alguns parâmetros são definidos em tempo de execução e são baseados nos valores de cada instância, como o número de consumidores N .

Nas etapas dos algoritmos em que se utiliza métodos aleatórios, foi utilizado uma *semente*, com o intuito de transformar esses processos estocásticos em determinísticos,

Tabela 6 – Parâmetros utilizados no algoritmo BGVH

Parâmetro	Valor
N_{iter}	100
P	6
$r1$	0.5
$r2$	0.3
$r3$	0.2
α	1
β	1
ψ	1.5
p	0.15
N_{iter}^{ACO}	50
N_{ants}^{ACO}	20

Tabela 7 – Parâmetros utilizados no algoritmo AHOCF

Parâmetro	Valor
N_{ants}	N
α	1
β	2
ρ	0.1
δ	0.7
η_e	6
n_{SA}	12
τ_{init}	0.5
T_{init}	5
T_{end}	$T_{init} \div 50$
λ	0.95
SZ_{tabu}	3
Z	$\max(4 \times N, 250)$
t_{cnt}	100
K_t	N
K_b	5
max_{run}	600

fazendo com que os resultados possam ser verificados posteriormente em uma execução posteriora. Utilizou-se como semente nos experimentos deste trabalho o valor 1.

Para realizar a análise dos métodos, utilizou-se o desvio relativo $\sigma\%$ entre a melhor solução da instância conhecida (BKS) e a melhor solução encontrada pelo algoritmo (BF), utilizando a equação 5.3 e a média de tempo de processamento gasto por cada algoritmo para chegar ao resultado, em segundos.

$$\sigma\% = \frac{BF - BKS}{BKS} * 100 \quad (5.3)$$

Os experimentos deste trabalho foram todos obtidos executando os códigos em uma máquina Ubuntu Linux 16.04 LTS com CPU Intel® Core™ i5 3.1 GHz e 8 GB de memória RAM.

5.5 Resultados e Análise

Nesta seção, são feitas a apresentação e a análise dos resultados obtidos pelas metaheurísticas nas instâncias do repositório CVRPLib. Os resultados obtidos serão comparados com os apresentados pelos autores no artigo, além de contrastados entre si para verificar a qualidade dos métodos propostos.

5.5.1 Algoritmo Híbrido de Otimização por Colônia de Formigas

Em [Sun, Fu e Liu \(2017\)](#), os autores avaliam o desempenho do AHOCF em cima de 14 instâncias do benchmark CMT de [Christofides \(1979\)](#), cujos tamanhos variam de 50 até 199 consumidores.

A tabela 8 apresenta os resultados da execução da implementação deste trabalho e os resultados apresentados pelo autor, onde MSC é a melhor solução conhecida no benchmark e $\sigma\%$ é o desvio relativo definido pela equação 5.3. As soluções marcadas com um asterisco (*) ainda não foram resolvidas até os dias atuais e, portanto, não possuem ótimo conhecido.

Tabela 8 – Comparação entre os resultados do AHOCF

Instância		Resultados do autor			Resultados encontrados		
Nome	MSC	Melhor	$\sigma\%$	Tempo (s)	Melhor	$\sigma\%$	Tempo (s)
CMT1	524,61	524,61	<u>0,0</u>	1,60	538,0	2,55	36,82
CMT2	835,26	835,26	<u>0,0</u>	139,76	878,0	5,11	600,0
CMT3	826,14	832,01	0,71	186,12	878,0	6,27	600,0
CMT4	1028,42	1062,33	3,29	192,91	1151,0	11,91	267,35
CMT5	1291,29	1398,11	8,27	416,21	1803,0	39,62	600,01
CMT6*	555,43	555,43	<u>0,0</u>	40,88	555,0	<u>0,0</u>	394,57
CMT7*	909,68	909,68	<u>0,0</u>	75,10	909,0	<u>0,0</u>	72,38
CMT8*	865,95	865,94	<u>0,0</u>	238,77	873,0	0,81	603,02
CMT9*	1162,55	1164,08	0,13	392,24	1162,0	<u>0,0</u>	269,48
CMT10*	1395,85	1446,68	5,08	393,62	1763,0	26,30	600,0
CMT11	1042,12	1042,11	<u>0,0</u>	384,17	1216,0	16,68	605,14
CMT12	819,56	819,56	<u>0,0</u>	30,67	973,0	18,72	600,02
CMT13*	1541,14	1547,80	0,43	321,22	1541,0	<u>0,0</u>	600,0
CMT14*	866,37	866,37	<u>0,0</u>	50,85	911,0	5,15	132,09

Segundo o autor, o algoritmo obteve a melhor solução em 8 das 14 instâncias do benchmark, e nas instâncias em que isso não foi possível, o algoritmo obteve na média um desvio relativo $\sigma\%$ de 2,98%.

Já nos experimentos executados neste trabalho, foi possível obter a melhor solução em apenas 4 das mesmas 14 instâncias. O algoritmo obteve um resultado melhor que o do autor nas instâncias CMT9 e CMT13, onde foi possível chegar à melhor solução.

Entretanto, em algumas instâncias, obteve-se um desvio relativo superior a 25% nos experimentos, tendo um resultado bastante inferior ao do autor. A média dos desvios relativos do algoritmo implementado nas instâncias CMT ficou em 9,5%.

Já com relação ao tempo de processamento gasto pelo algoritmo, a implementação deste trabalho obteve um tempo superior na maioria das instâncias testadas, em muitas chegando até ao tempo máximo permitido de execução, muito embora isso se deva à escolha da forma em que os algoritmos foram implementados e a diferenças entre as máquinas usadas para teste.

Por se tratar de um método híbrido, o algoritmo é composto de vários sub-algoritmos trabalhando em conjunto para buscar novas soluções para o problema. Isolando cada um desses componentes, tem-se que o AHOCF é composto de duas metaheurísticas, o Algoritmo de Otimização por Colônia de Formigas (OCF) e o Recozimento Simulado (RS).

Fez-se, então, uma análise isolada de cada um desses sub-algoritmos para verificar o desempenho no melhoramento das soluções e o seu tempo gasto. Assim, é possível ter uma visão de qual metaheurística mais influencia nas soluções encontradas e no tempo gasto pelos métodos.

A primeira etapa consiste em aplicar a OCF para gerar várias soluções baseadas nas matrizes heurística e de feromônios. As melhores soluções geradas são utilizadas para atualizar os feromônios, levando em consideração as rotas percorridas pelos veículos. As soluções são, então, geradas novamente baseadas nas informações atualizadas, assim por diante.

Na Figura 14, é apresentada a melhor, a pior e a média das soluções em cada iteração do algoritmo OCF para uma instância A-n32-k5 do benchmark. Observando o traçado do gráfico, é possível verificar que as soluções vão lentamente melhorando a cada iteração, obtendo uma melhora no custo de aproximadamente 10% ao final da execução do algoritmo. Em relação ao tempo de processamento, o tempo médio por instância no benchmark foi de 0,87 segundo.

A segunda etapa do algoritmo consiste em aplicar o RS na melhor solução até o momento para tentar melhorá-la e diversificar o espaço de busca. Analisando o desempenho do RS na instância A-n32-k5 na Figura 15, é possível verificar que há uma melhora

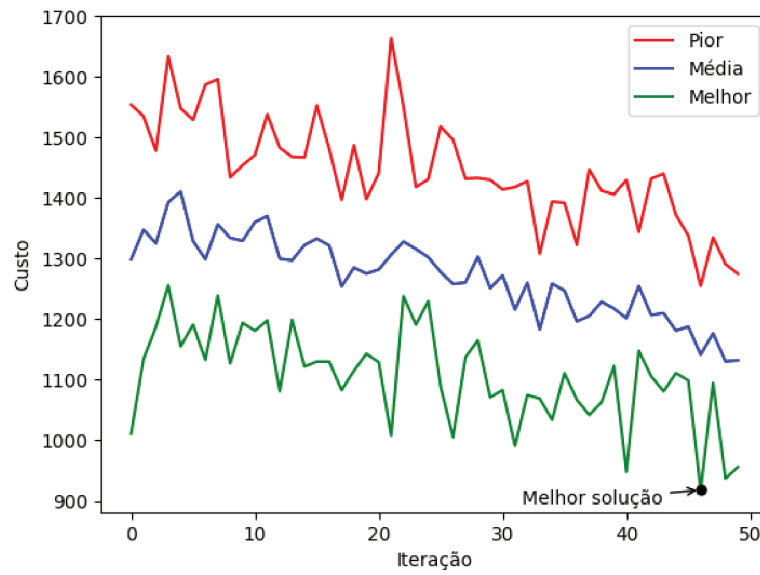


Figura 14 – Melhoria da população de soluções no algoritmo Otimização por Colônia de Formigas para a instância A-n32-k5

substantial na qualidade da solução após a execução do algoritmo, tendo uma queda de aproximadamente 25% no custo da solução.

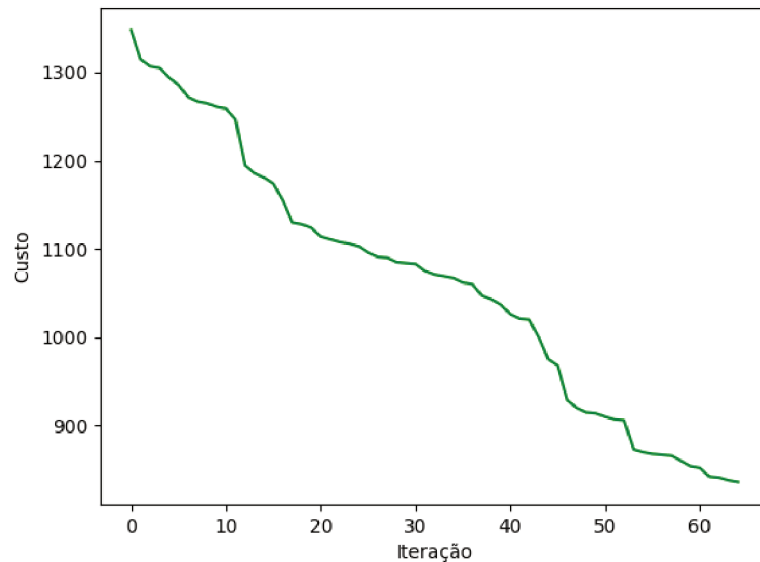


Figura 15 – Melhoria da melhor solução até o momento pelo algoritmo Recozimento Simulado para a instância A-n32-k5

Assim, grande parte do tempo consumido pelo AHOCF vem da execução do RS, visto que ele é executado em várias iterações quando há estagnação na qualidade das soluções encontradas pelo OCF. Além disso, nota-se que o resultado do método é altamente responsável pela diversidade e pela qualidade das soluções obtidas no RS.

5.5.2 Busca Híbrida em Grande Vizinhaça

No artigo de [Akpinar \(2016\)](#), o autor avalia o desempenho do algoritmo proposto em 88 instâncias das classes A, B, E, P e CMT, que variam de 15 até 100 consumidores. O algoritmo implementado neste trabalho foi executado nas mesmas instâncias que o autor, com exceção das instâncias da classe CMT, que apresentaram divergências de formato com as encontradas no repositório CVRPLib.

O resultado dos experimentos realizados neste trabalho é apresentado na tabela 9, contrastando com os resultados apresentados pelo autor em seu artigo, onde MSC é a melhor solução conhecida até o momento e $\sigma\%$, o desvio relativo.

Tabela 9 – Comparação entre os resultados da BHGV. Os desvios em negrito mostram as instâncias em que o ótimo foi obtido

Instância		Resultados do autor		Resultados encontrados	
Nome	MSC	$\sigma\%$	Tempo (s)	$\sigma\%$	Tempo (s)
A-n32-k5	784	0,00	856,21	18,11	131,66
A-n33-k5	661	0,00	900,06	7,56	150,03
A-n33-k6	742	0,00	947,86	17,52	162,93
A-n34-k5	778	0,00	909,04	27,76	156,91
A-n36-k5	799	0,00	1055,60	22,27	169,94
A-n37-k5	669	0,00	1103,70	10,76	168,86
A-n37-k5	949	0,00	1113,40	46,15	217,15
A-n38-k5	730	0,00	1139,20	36,98	197,39
A-n39-k5	822	0,00	1202,80	34,30	209,39
A-n39-k6	831	0,00	1266,00	64,25	230,20
A-n44-k6	937	0,00	1567,80	43,75	277,92
A-n45-k6	944	1,48	1728,10	58,79	64,25
A-n45-k7	1146	0,00	1741,70	27,22	315,87
A-n46-k7	914	0,00	1804,50	21,77	348,94
A-n48-k7	1073	1,03	1978,60	23,39	323,99
A-n53-k7	1010	0,00	2323,70	85,54	422,62
A-n54-k7	1167	0,00	2497,40	62,55	462,40
A-n55-k9	1073	0,00	2771,00	87,23	554,13
A-n60-k9	1354	0,00	3345,40	73,41	678,82
A-n61-k9	1034	3,19	3355,70	208,80	679,30
A-n62-k8	1288	1,55	3363,30	58,15	705,92
A-n63-k10	1314	1,14	3800,10	173,51	804,57
A-n63-k9	1616	2,04	3651,20	159,28	732,22
A-n64-k9	1401	1,00	3831,80	52,31	647,00

A-n65-k9	1174	0,94	3854,20	163,45	775,83
A-n69-k9	1159	0,95	4460,90	81,27	905,06
A-n80-k10	1763	2,95	6493,60	118,32	1221,85
B-n31-k5	672	0,00	828,20	12,20	139,48
B-n34-k5	788	0,00	908,51	8,37	148,79
B-n35-k5	955	0,00	998,94	5,34	155,49
B-n38-k6	805	0,00	1219,90	10,68	212,72
B-n39-k5	549	0,00	1241,00	41,89	206,12
B-n41-k6	829	0,00	1392,10	41,85	250,34
B-n43-k6	742	0,00	1502,00	29,38	285,10
B-n44-k7	909	0,00	1623,30	33,22	304,47
B-n45-k5	751	0,00	1621,70	58,85	263,15
B-n45-k6	678	0,00	1657,70	78,61	304,99
B-n50-k7	741	0,00	2174,80	47,63	371,66
B-n50-k8	1312	0,53	2169,20	120,27	434,66
B-n51-k7	1016	0,00	2228,50	105,32	428,71
B-n52-k7	747	0,00	2289,30	49,39	454,00
B-n56-k7	707	0,00	2709,80	25,74	506,64
B-n57-k7	1140	0,00	2844,40	167,04	492,38
B-n57-k9	1598	0,00	3055,80	29,78	558,86
B-n63-k10	1496	1,20	3750,90	71,39	760,04
B-n64-k9	861	1,51	3834,60	269,68	804,61
B-n66-k9	1316	1,06	4242,60	59,57	747,92
B-n67-k10	1032	1,74	4523,00	65,31	933,13
B-n68-k9	1272	1,42	4395,20	75,47	929,39
B-n78-k10	1221	0,57	6049,20	92,95	1182,79
P-n16-k8	450	0,35	737,30	24,00	48,87
P-n19-k2	212	0,00	363,90	8,01	37,07
P-n20-k2	216	0,00	353,04	0,92	44,64
P-n21-k2	211	0,00	399,88	2,84	42,78
P-n22-k2	216	0,00	413,12	3,24	45,52
P-n22-k8	590	0,00	562,84	33,16	105,17
P-n23-k8	529	0,00	615,53	92,87	92,87
P-n40-k5	458	0,00	1227,50	1,31	196,59
P-n45-k5	510	0,00	1569,40	4,31	251,74
P-n50-k7	554	0,00	2025,60	41,69	388,79
P-n50-k8	631	1,90	2142,30	179,71	451,37
P-n50-k10	696	0,00	2347,80	89,22	478,42
P-n51-k10	741	0,81	2449,10	26,04	424,05

P-n55-k7	568	0,00	2532,90	30,80	432,33
P-n55-k8	588	0,00	2648,85	25,17	486,23
P-n55-k10	694	0,00	2876,60	32,27	603,34
P-n55-k15	989	0,00	3053,50	60,36	736,12
P-n60-k10	744	1,48	3392,70	75,80	724,36
P-n60-k15	968	0,93	3997,20	105,68	954,94
P-n65-k10	792	1,01	3883,00	56,18	903,03
P-n70-k10	827	1,21	4640,60	194,43	1031,10
P-n76-k4	593	0,84	5054,70	77,06	814,50
P-n76-k5	627	2,87	4940,00	77,03	905,25
E-n22-k4	375	0,00	447,39	5,06	60,42
E-n23-k3	569	0,00	393,08	136,37	62,01
E-n30-k3	503	0,00	698,98	20,59	97,85
E-n33-k4	835	0,00	818,03	17,60	124,94
E-n76-k14	1021	0,88	6190,70	173,35	1497,13
E-n76-k7	682	1,91	5186,30	50,29	906,25
E-n76-k8	735	1,22	5289,70	77,82	1062,38
E-n101-k14	1067	1,41	12039,00	213,58	2494,76

Nos experimentos realizados pelo autor, o algoritmo chegou no ótimo global em 51 das 82 instâncias do benchmark, com uma média dos desvios relativos $\sigma\%$ de 0,51%. Entretanto, nos testes realizados na implementação deste trabalho, o algoritmo obteve um desempenho médio de 70,81% de desvio em relação ao ótimo, não conseguindo chegar no ótimo em nenhuma instância do experimento.

Essa diferença de resultados pode estar relacionada ao conjunto de parâmetros utilizados nos experimentos, já que nem todos estavam explicitamente disponíveis no artigo do autor, além de possíveis divergências nos detalhes da implementação, por não haver código fonte disponível publicamente.

Isso pode ser verificado também no tempo de execução do algoritmo, que nos experimentos do autor obteve um tempo médio de 2513,75 segundos, contra a média de 474 segundos alcançada pela implementação deste trabalho. Entretanto, o tempo pode divergir por terem sido executados em máquinas diferentes e por terem sido implementados utilizando linguagens diferentes, não sendo uma fonte de comparação confiável.

5.5.3 Comparação entre as metaheurísticas

Nesta subseção, é realizada a comparação entre o desempenho das duas metaheurísticas em no conjunto de 77 instâncias das classes A, B, E e P do repositório CVRPLib. O resultado desse experimento pode ser observado na tabela 10.

Tabela 10 – Solução encontrada e tempo gasto em cada algoritmo por instância

Instância		AHOCF			BHGVS		
Nome	Ótimo	Melhor	$\sigma\%$	Tempo (s)	Melhor	$\sigma\%$	Tempo (s)
A-n32-k5	784	829,0	5,7	275,0	926,0	18,1	131,7
A-n33-k5	661	661,0	0,0	26,7	711,0	7,6	150,0
A-n33-k6	742	775,0	4,4	253,0	872,0	17,5	162,9
A-n34-k5	778	786,0	1,0	260,5	994,0	27,8	156,9
A-n36-k5	799	820,0	2,6	278,9	977,0	22,3	169,9
A-n37-k5	669	679,0	1,5	29,5	741,0	10,8	168,9
A-n37-k6	949	983,0	3,6	219,7	1387,0	46,2	217,2
A-n38-k5	730	747,0	2,3	30,8	1000,0	37,0	197,4
A-n39-k5	822	855,0	4,0	29,9	1104,0	34,3	209,4
A-n39-k6	831	835,0	0,5	316,7	1365,0	64,3	230,2
A-n44-k6	937	949,0	1,3	330,7	1347,0	43,8	277,9
A-n45-k6	944	986,0	4,4	330,8	1499,0	58,8	304,3
A-n45-k7	1146	1161,0	1,3	371,4	1458,0	27,2	315,9
A-n46-k7	914	967,0	5,8	357,3	1113,0	21,8	348,9
A-n48-k7	1073	1125,0	4,8	332,0	1324,0	23,4	324,0
A-n53-k7	1010	1076,0	6,5	47,8	1874,0	85,5	422,6
A-n54-k7	1167	1222,0	4,7	436,1	1897,0	62,6	462,4
A-n55-k9	1073	1101,0	2,6	479,0	2009,0	87,2	554,1
A-n60-k9	1354	1394,0	3,0	470,2	2348,0	73,4	678,8
A-n61-k9	1034	1092,0	5,6	511,3	3193,0	208,8	679,3
A-n62-k8	1288	1379,0	7,1	477,4	2037,0	58,2	705,9
A-n63-k10	1314	1397,0	6,3	517,3	3594,0	173,5	804,6
A-n63-k9	1616	1700,0	5,2	471,2	4190,0	159,3	732,2
A-n64-k9	1401	1483,0	5,9	514,6	2134,0	52,3	647,0
A-n65-k9	1174	1213,0	3,3	537,5	3093,0	163,5	775,8
A-n69-k9	1159	1231,0	6,2	600,0	2101,0	81,3	905,1
A-n80-k10	1763	1873,0	6,2	600,0	3849,0	118,3	1221,9
B-n31-k5	672	672,0	0,0	266,6	754,0	12,2	139,5
B-n34-k5	788	788,0	0,0	297,5	854,0	8,4	148,8
B-n35-k5	955	969,0	1,5	259,6	1006,0	5,3	155,5
B-n38-k6	805	806,0	0,1	324,6	891,0	10,7	212,7
B-n39-k5	549	554,0	0,9	305,3	779,0	41,9	206,1
B-n41-k6	829	839,0	1,2	351,4	1176,0	41,9	250,3
B-n43-k6	742	758,0	2,2	351,9	960,0	29,4	285,1
B-n44-k7	909	924,0	1,7	342,6	1211,0	33,2	304,5

B-n45-k5	751	774,0	3,1	351,0	1193,0	58,9	263,2
B-n45-k6	678	729,0	7,5	385,4	1211,0	78,6	305,0
B-n50-k7	741	742,0	0,1	393,9	1094,0	47,6	371,7
B-n50-k8	1312	1328,0	1,2	418,3	2890,0	120,3	434,7
B-n52-k7	747	753,0	0,8	437,1	1116,0	49,4	454,0
B-n56-k7	707	714,0	1,0	453,9	889,0	25,7	506,6
B-n57-k9	1598	1639,0	2,6	307,2	2074,0	29,8	558,9
B-n63-k10	1496	1560,0	4,3	506,9	2564,0	71,4	760,0
B-n64-k9	861	894,0	3,8	53,8	3183,0	269,7	804,6
B-n66-k9	1316	1366,0	3,8	360,0	2100,0	59,6	747,9
B-n67-k10	1032	1071,0	3,8	541,8	1706,0	65,3	933,1
B-n68-k9	1272	1303,0	2,4	592,4	2232,0	75,5	929,4
B-n78-k10	1221	1249,0	2,3	600,0	2356,0	93,0	1182,8
E-n101-k14	1067	1180,0	10,6	600,0	3346,0	213,6	2494,8
E-n22-k4	375	375,0	<u>0,0</u>	153,6	394,0	5,1	60,4
E-n23-k3	569	569,0	<u>0,0</u>	205,4	1345,0	136,4	62,0
E-n33-k4	835	862,0	3,2	26,4	982,0	17,6	124,9
E-n76-k14	1021	1070,0	4,8	600,0	2791,0	173,4	1497,1
E-n76-k7	682	734,0	7,6	600,0	1025,0	50,3	906,3
E-n76-k8	735	755,0	2,7	600,0	1307,0	77,8	1062,4
P-n101-k4	681	759,0	11,5	600,0	878,0	28,9	1076,4
P-n16-k8	450	450,0	<u>0,0</u>	20,6	558,0	24,0	48,9
P-n19-k2	212	212,0	<u>0,0</u>	14,8	229,0	8,0	37,1
P-n20-k2	216	216,0	<u>0,0</u>	159,6	218,0	0,9	44,6
P-n21-k2	211	211,0	<u>0,0</u>	169,7	217,0	2,8	42,8
P-n22-k2	216	216,0	<u>0,0</u>	170,9	223,0	3,2	45,5
P-n23-k8	529	529,0	<u>0,0</u>	242,6	772,0	45,9	92,9
P-n40-k5	458	462,0	0,9	21,1	464,0	1,3	196,6
P-n45-k5	510	519,0	1,8	322,7	532,0	4,3	251,7
P-n50-k10	696	715,0	2,7	398,7	1317,0	89,2	478,4
P-n50-k7	554	571,0	3,1	407,7	785,0	41,7	388,8
P-n50-k8	631	645,0	2,2	423,9	1765,0	179,7	451,4
P-n51-k10	741	770,0	3,9	34,4	934,0	26,0	424,1
P-n55-k10	694	706,0	1,7	445,1	918,0	32,3	603,3
P-n55-k7	568	571,0	0,5	39,8	743,0	30,8	432,3
P-n60-k10	744	778,0	4,6	506,4	1308,0	75,8	724,4
P-n60-k15	968	986,0	1,9	50,8	1991,0	105,7	954,9
P-n65-k10	792	835,0	5,4	51,1	1237,0	56,2	903,0
P-n70-k10	827	857,0	3,6	578,8	2435,0	194,4	1031,1

P-n76-k4	593	664,0	12,0	600,3	1050,0	77,1	814,5
P-n76-k5	627	713,0	13,7	601,4	1110,0	77,0	905,3

Fazendo uma análise do desvio relativo $\sigma\%$ alcançado pelas metaheurísticas, é possível verificar que foi alcançado um resultado muito superior pelo AHOCF em grande parte das instâncias do repositório CVRPlib, com uma média de 0,53% de desvio em relação ao ótimo.

Por outro lado, chegou-se a um desvio relativo médio de 64,46% na BHGV, tendo obtido soluções satisfatórias com desvio abaixo de 10% apenas em instâncias pequenas, de até 35 consumidores. Também houve um número maior de ótimos encontrados pela primeira metaheurística, obtendo-se 11 desvios relativos zero, contra nenhum ótimo da segunda.

Analisando o tempo de execução de cada metaheurística, o AHOCF também obteve uma vantagem sobre o BVGH em quase todas as instância do benchmark, tendo um desempenho, em média, 33% mais rápido. Na BHGV, foi possível notar um aumento quase linear no tempo gasto à medida que o número de consumidores das instâncias iam aumentando, enquanto o AHOCF obteve um comportamento constante, devido à seu limitante superior de 600 segundos de execução.

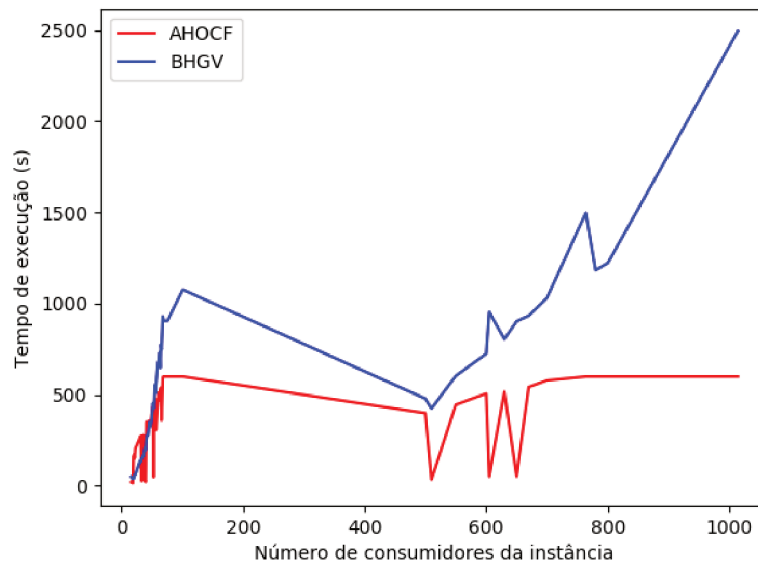


Figura 16 – Comparação entre o tempo de execução dos algoritmos AHOCF e BHGV

6 Conclusão

Neste capítulo, são apresentadas as conclusões que foram alcançadas por meio deste trabalho e, ainda, algumas propostas de trabalhos futuros com o intuito de dar continuidade a esta pesquisa.

6.1 Conclusões

Primeiramente, foi apresentado neste trabalho um levantamento dos métodos publicados em um período recente para o Problema de Roteamento de Veículos Capacitados, através de um protocolo de revisão sistemática definido no capítulo 3. Constatou-se que grande parte dos métodos recentes propostos estão contidos no grupo das metaheurísticas, sendo um tópico bastante em alta atualmente.

Desses métodos, foram escolhidas duas metaheurísticas para a realização dos experimentos deste trabalho, com base na avaliação dos artigos em três características principais: o uso de instâncias públicas nos experimentos do método, a disponibilidade de código fonte e a presença do trabalho em um periódico de alto impacto.

Chegou-se à definição de duas metaheurísticas para serem implementadas e analisadas no trabalho: o Algoritmo Híbrido de Otimização por Colônia de Formigas (AHO CF) e a Busca Híbrida em Grande Vizinhança (BHGV), detalhadas no capítulo 4. Para simplificar a execução dos experimentos, foi especificado na seção 5.1 a implementação de um analisador de instâncias do repositório CVRPLib, convertendo os arquivos no formato TSPLIB95 para o formato JSON.

Os algoritmos foram implementados na linguagem Clojure, e nos experimentos foi feita a comparação dos resultados apresentados pelos autores com os alcançados pela implementação deste trabalho, e posteriormente contrastados entre si.

Nos experimentos do AHO CF apresentados na subseção 5.5.1, foram obtidos resultados ligeiramente inferiores aos apresentados pelo autor do método, chegando ao ótimo global em 8 das 14 instâncias do benchmark nos experimentos do autor, contra 4 de 14 nos experimentos deste trabalho. Além disso, obteve um desvio médio de 2,98% nos resultados do autor, contra 9,5% nos resultados obtidos neste trabalho.

Já nos testes realizados na subseção 5.5.2 com a BHGV, os resultados encontrados nos experimentos foram muito inferiores aos apresentados pelo autor do método, com um desvio médio em relação ao ótimo de 70,81%, contra o 0,51% de desvio médio do autor. Foi observado que a divergência pode estar relacionada aos parâmetros utilizados pelo autor com os que foram utilizados no trabalho, já que muitos deles não foram explicitamente

disponibilizados. Além disso, a falta de código fonte público pode ter contribuído para que houvesse possíveis diferenças na implementação do método.

Por fim, na subseção 5.5.3 foi contrastado os métodos entre si, havendo uma nítida diferença na performance obtida pelo AHOCF em relação à BHGV. O primeiro obteve resultados superiores tanto nas soluções encontradas, quanto no tempo de processamento das instâncias. Foi possível notar também que a metaheurística Recozimento Simulado foi determinante no desempenho do AHOCF, sendo responsável pela qualidade das soluções encontradas pelo método.

6.2 Trabalhos futuros

Apresenta-se por fim como propostas para trabalhos futuros:

- O ajuste nos parâmetros da BHGV para chegar em resultados mais próximos aos apresentados pelo autor;
- O uso de técnicas paralelas na implementação das metaheurísticas, a fim de otimizar o tempo de execução;
- A criação de um *framework* único em Clojure contendo a implementação de diversas outras metaheurísticas para o PRVC, seguindo os formatos de arquivo sugeridos neste trabalho;
- O desenvolvimento de um *front-end* para representação gráfica em tempo real das rotas sendo construídas durante a execução das metaheurísticas.

Referências

- 1 DANTZIG, G. B.; RAMSER, J. H. The truck dispatching problem. *Management Science*, Institute for Operations Research and the Management Sciences (INFORMS), v. 6, n. 1, p. 80–91, oct 1959. Disponível em: <http://dx.doi.org/10.1287/mnsc.6.1.80>. Acesso em: 4 abr. 2017. Citado 2 vezes nas páginas 15 e 19.
- 2 LENSTRA, J. K.; KAN, A. H. G. R. Complexity of vehicle routing and scheduling problems. *Networks*, Wiley-Blackwell, v. 11, n. 2, p. 221–227, 1981. Disponível em: <https://doi.org/10.1002/net.3230110211>. Acesso em: 14 maio 2017. Citado 3 vezes nas páginas 15, 16 e 19.
- 3 AKPINAR, S. Hybrid large neighbourhood search algorithm for capacitated vehicle routing problem. *Expert Systems with Applications*, Elsevier BV, v. 61, p. 28–38, nov 2016. Disponível em: <http://dx.doi.org/10.1016/j.eswa.2016.05.023>. Acesso em: 16 ago. 2017. Citado 9 vezes nas páginas 15, 16, 32, 37, 38, 39, 45, 63 e 68.
- 4 SUN, X.; FU, Y.; LIU, T. A hybrid ACO algorithm for capacitated vehicle routing problems. In: *2017 IEEE 2nd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*. IEEE, 2017. Disponível em: <http://dx.doi.org/10.1109/iaeac.2017.8054067>. Acesso em: 16 ago. 2017. Citado 9 vezes nas páginas 15, 16, 36, 37, 38, 39, 40, 63 e 65.
- 5 UCHOA, E. et al. New benchmark instances for the capacitated vehicle routing problem. *European Journal of Operational Research*, Elsevier BV, v. 257, n. 3, p. 845–858, mar 2017. Disponível em: <http://dx.doi.org/10.1016/j.ejor.2016.08.012>. Acesso em: 27 ago. 2017. Citado 2 vezes nas páginas 15 e 54.
- 6 JAEGERE, N. D.; DEFRAEYE, M.; NIEUWENHUYSE, I. V. The vehicle routing problem: state of the art classification and review. 2013. Disponível em: <https://lirias.kuleuven.be/handle/123456789/457452>. Acesso em: 6 abr. 2017. Citado 3 vezes nas páginas 16, 27 e 31.
- 7 GOLDREICH, O. *Computational Complexity*. Cambridge University Press, 2008. Disponível em: <https://doi.org/10.1017/cbo9780511804106>. Acesso em: 25 mar. 2018. Citado na página 18.
- 8 GAREY, M. R.; JOHNSON, D. S. *Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)*. W. H. Freeman, 1979. ISBN 0716710455. Disponível em: <https://www.amazon.com/Computers-Intractability-NP-Completeness-Mathematical-Sciences/dp/0716710455?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0716710455>. Acesso em: 14 mar. 2018. Citado na página 18.
- 9 CLARKE, G.; WRIGHT, J. W. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, Institute for Operations Research and the Management Sciences (INFORMS), v. 12, n. 4, p. 568–581, aug 1964. Disponível em: <http://dx.doi.org/10.1287/opre.12.4.568>. Acesso em: 4 abr. 2017. Citado 2 vezes nas páginas 19 e 26.

- 10 LABADIE, N.; PRINS, C.; PRODHON, C. *Metaheuristics for Vehicle Routing Problems*. Wiley, 2016. v. 3. (Computer Engineer Series, v. 3). Disponível em: <http://dx.doi.org/10.1002/9781119136767>. Acesso em: 22 abr. 2017. Citado 4 vezes nas páginas 19, 25, 26 e 27.
- 11 COELHO, L. C.; RENAUD, J.; LAPORTE, G. Road-based goods transportation: a survey of real-world logistics applications from 2000 to 2015. *INFOR: Information Systems and Operational Research*, Informa UK Limited, v. 54, n. 2, p. 79–96, apr 2016. Disponível em: <http://dx.doi.org/10.1080/03155986.2016.1167357>. Acesso em: 24 maio 2017. Citado na página 24.
- 12 LAHYANI, R.; KHEMAKHEM, M.; SEMET, F. Rich vehicle routing problems: From a taxonomy to a definition. *European Journal of Operational Research*, Elsevier BV, v. 241, n. 1, p. 1–14, feb 2015. Disponível em: <http://dx.doi.org/10.1016/j.ejor.2014.07.048>. Acesso em: 10 abr. 2017. Citado na página 25.
- 13 VIDAL, T. et al. Heuristics for multi-attribute vehicle routing problems: A survey and synthesis. *European Journal of Operational Research*, Elsevier BV, v. 231, n. 1, p. 1–21, nov 2012. Disponível em: <http://dx.doi.org/10.1016/j.ejor.2013.02.053>. Acesso em: 12 abr. 2017. Citado na página 25.
- 14 BODIN, L.; GOLDEN, B. Classification in vehicle routing and scheduling. *Networks*, Wiley-Blackwell, v. 11, n. 2, p. 97–108, 1981. Disponível em: <http://dx.doi.org/10.1002/net.3230110204>. Acesso em: 6 abr. 2017. Citado na página 25.
- 15 RONEN, D. Perspectives on practical aspects of truck routing and scheduling. *European Journal of Operational Research*, Elsevier BV, v. 35, n. 2, p. 137–145, may 1988. Disponível em: [http://dx.doi.org/10.1016/0377-2217\(88\)90022-7](http://dx.doi.org/10.1016/0377-2217(88)90022-7). Acesso em: 4 abr. 2017. Citado na página 25.
- 16 EKSIOGLU, B.; VURAL, A. V.; REISMAN, A. The vehicle routing problem: A taxonomic review. *Computers & Industrial Engineering*, Elsevier BV, v. 57, n. 4, p. 1472–1483, nov 2009. Disponível em: <http://dx.doi.org/10.1016/j.cie.2009.05.009>. Acesso em: 10 abr. 2017. Citado na página 25.
- 17 VANDERBEI, R. J. *Linear Programming*. Springer US, 2014. Disponível em: <http://dx.doi.org/10.1007/978-1-4614-7630-6>. Acesso em: 16 ago. 2017. Citado na página 25.
- 18 MARCHAND, H. et al. Cutting planes in integer and mixed integer programming. *Discrete Applied Mathematics*, Elsevier BV, v. 123, n. 1-3, p. 397–446, nov 2002. Disponível em: [https://doi.org/10.1016/s0166-218x\(01\)00348-1](https://doi.org/10.1016/s0166-218x(01)00348-1). Acesso em: 16 ago. 2017. Citado na página 26.
- 19 LAPORTE, G.; NOBERT, Y. Exact algorithms for the vehicle routing problem. In: *Surveys in Combinatorial Optimization*. Elsevier, 1987. p. 147–184. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0304020808732353>. Acesso em: 2 set. 2017. Citado na página 26.
- 20 LAPORTE, G. et al. Classical and modern heuristics for the vehicle routing problem. *International Transactions in Operational Research*, Wiley-Blackwell, v. 7, n. 4-5, p. 285–300, sep 2000. Disponível em: <http://dx.doi.org/10.1111/j.1475-3995.2000.tb00200.x>. Acesso em: 2 jun. 2017. Citado na página 27.

- 21 CORDEAU, J.-F. et al. New heuristics for the vehicle routing problem. In: *Logistics Systems: Design and Optimization*. Springer-Verlag, 2005. p. 279–297. Disponível em: http://dx.doi.org/10.1007/0-387-24977-x_9. Acesso em: 24 abr. 2017. Citado 2 vezes nas páginas 27 e 28.
- 22 GLOVER, F. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, Elsevier BV, v. 13, n. 5, p. 533–549, jan 1986. Disponível em: [https://doi.org/10.1016/0305-0548\(86\)90048-1](https://doi.org/10.1016/0305-0548(86)90048-1). Acesso em: 2 set. 2017. Citado na página 27.
- 23 SÖRENSEN, K. Metaheuristics-the metaphor exposed. *International Transactions in Operational Research*, Wiley-Blackwell, v. 22, n. 1, p. 3–18, feb 2013. Disponível em: https://www.researchgate.net/publication/237009138_Metaheuristics_-_the_metaphor_exposed. Acesso em: 29 set. 2017. Citado na página 27.
- 24 GOLDEN, B.; RAGHAVAN, S.; WASIL, E. *The Vehicle Routing Problem: Latest advances and new challenges*. Springer US, 2008. Disponível em: <http://dx.doi.org/10.1007/978-0-387-77778-8>. Acesso em: 13 abr. 2017. Citado 2 vezes nas páginas 27 e 28.
- 25 BUDGEN, D.; KITCHENHAM, B. A.; BRERETON, P. *Evidence-Based Software Engineering and Systematic Reviews*. Apple Academic Press Inc., 2015. ISBN 1482228653. Disponível em: http://www.ebook.de/de/product/23848469/david_budgen_barbara_ann_kitchenham_pearl_brereton_evidence_based_software_engineering_and_systematic_reviews.html. Acesso em: 4 abr. 2017. Citado na página 28.
- 26 LAPORTE, G. The vehicle routing problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, Elsevier BV, v. 59, n. 3, p. 345–358, jun 1992. Disponível em: [http://dx.doi.org/10.1016/0377-2217\(92\)90192-c](http://dx.doi.org/10.1016/0377-2217(92)90192-c). Acesso em: 2 set. 2017. Citado na página 28.
- 27 TOTH, P.; VIGO, D. Models, relaxations and exact approaches for the capacitated vehicle routing problem. *Discrete Applied Mathematics*, Elsevier BV, v. 123, n. 1-3, p. 487–512, nov 2002. Disponível em: [http://dx.doi.org/10.1016/s0166-218x\(01\)00351-1](http://dx.doi.org/10.1016/s0166-218x(01)00351-1). Acesso em: 2 set. 2017. Citado na página 28.
- 28 KUMAR, S. N.; PANNEERSELVAM, R. A survey on the vehicle routing problem and its variants. *Intelligent Information Management*, Scientific Research Publishing, Inc., v. 04, n. 03, p. 66–74, 2012. Disponível em: <http://dx.doi.org/10.4236/iim.2012.43010>. Acesso em: 24 abr. 2017. Citado na página 28.
- 29 BRAEKERS, K.; RAMAEKERS, K.; NIEUWENHUYSE, I. V. The vehicle routing problem: State of the art classification and review. *Computers & Industrial Engineering*, Elsevier BV, v. 99, p. 300–313, sep 2016. Disponível em: <http://dx.doi.org/10.1016/j.cie.2015.12.007>. Acesso em: 24 abr. 2017. Citado na página 28.
- 30 KITCHENHAM, B. A. Guidelines for performing systematic literature reviews in software engineering. *EBSE Technical Report EBSE-2007-01*, 2007. Disponível em: <http://dx.doi.org/10.1.1.117.471>. Acesso em: 14 ago. 2017. Citado 2 vezes nas páginas 28 e 29.

- 31 ARPINI, B. P.; ROSA, R. A. Uma revisão sistemática da literatura sobre o problema de roteirização de veículos capacitados com restrições de carregamento bidimensional (2l-cvrp). *TRANSPORTES* v. 25, n. 1, 2017. Disponível em: <https://www.revistatransportes.org.br/anpet/article/view/867>. Acesso em: 14 ago. 2017. Citado na página 29.
- 32 JIN, J.; CRAINIC, T. G.; LØKKETANGEN, A. A cooperative parallel metaheuristic for the capacitated vehicle routing problem. *Computers & Operations Research*, Elsevier BV, v. 44, p. 33–41, apr 2014. Disponível em: <http://dx.doi.org/10.1016/j.cor.2013.10.004>. Acesso em: 17 ago. 2017. Citado 3 vezes nas páginas 32, 38 e 39.
- 33 TEYMOURIAN, E. et al. Enhanced intelligent water drops and cuckoo search algorithms for solving the capacitated vehicle routing problem. *Information Sciences*, Elsevier BV, v. 334-335, p. 354–378, mar 2016. Disponível em: <http://dx.doi.org/10.1016/j.ins.2015.11.036>. Acesso em: 16 ago. 2017. Citado 3 vezes nas páginas 32, 38 e 39.
- 34 VIDAL, T. Technical note: Split algorithm in $O(n)$ for the capacitated vehicle routing problem. *Computers & Operations Research*, Elsevier BV, v. 69, p. 40–47, may 2016. Disponível em: <http://dx.doi.org/10.1016/j.cor.2015.11.012>. Acesso em: 17 ago. 2017. Citado 4 vezes nas páginas 32, 37, 38 e 39.
- 35 ZHAO, Y. et al. A discrete hybrid invasive weed optimization algorithm for the capacitated vehicle routing problem. *Procedia Computer Science*, Elsevier BV, v. 91, p. 978–987, 2016. Disponível em: <http://dx.doi.org/10.1016/j.procs.2016.07.127>. Acesso em: 16 ago. 2017. Citado 3 vezes nas páginas 32, 38 e 39.
- 36 AMOUS, M. et al. A variable neighborhood search algorithm for the capacitated vehicle routing problem. *Electronic Notes in Discrete Mathematics*, Elsevier BV, v. 58, p. 231–238, apr 2017. Disponível em: <http://dx.doi.org/10.1016/j.endm.2017.03.030>. Acesso em: 16 ago. 2017. Citado 3 vezes nas páginas 32, 38 e 39.
- 37 BOUZID, M. C.; HADDADENE, H. A.; SALHI, S. An integration of lagrangian split and VNS: The case of the capacitated vehicle routing problem. *Computers & Operations Research*, Elsevier BV, v. 78, p. 513–525, feb 2017. Disponível em: <http://dx.doi.org/10.1016/j.cor.2016.02.009>. Acesso em: 16 ago. 2017. Citado 3 vezes nas páginas 33, 38 e 39.
- 38 HOSSEINABADI, A. A. R. et al. A new efficient approach for solving the capacitated vehicle routing problem using the gravitational emulation local search algorithm. *Applied Mathematical Modelling*, Elsevier BV, mar 2017. Disponível em: <http://dx.doi.org/10.1016/j.apm.2017.02.042>. Acesso em: 16 ago. 2017. Citado 3 vezes nas páginas 33, 38 e 39.
- 39 MOHAMMED, M. A. et al. Solving vehicle routing problem by using improved genetic algorithm for optimal solution. *Journal of Computational Science*, Elsevier BV, v. 21, p. 255–262, jul 2017. Disponível em: <http://dx.doi.org/10.1016/j.jocs.2017.04.003>. Acesso em: 16 ago. 2017. Citado 4 vezes nas páginas 33, 37, 38 e 39.
- 40 MOHAMMED, M. A. et al. Solving vehicle routing problem by using improved k-nearest neighbor algorithm for best solution. *Journal of Computational Science*, Elsevier BV, v. 21, p. 232–240, jul 2017. Disponível em: <http://dx.doi.org/10.1016/j.jocs.2017.04.012>. Acesso em: 16 ago. 2017. Citado 3 vezes nas páginas 33, 38 e 39.

- 41 YU, V. F. et al. Symbiotic organisms search and two solution representations for solving the capacitated vehicle routing problem. *Applied Soft Computing*, Elsevier BV, v. 52, p. 657–672, mar 2017. Disponível em: <http://dx.doi.org/10.1016/j.asoc.2016.10.006>. Acesso em: 16 ago. 2017. Citado 3 vezes nas páginas 33, 38 e 39.
- 42 NIU, Y. et al. A novel membrane algorithm for capacitated vehicle routing problem. *Soft Computing*, Springer Nature, v. 19, n. 2, p. 471–482, mar 2014. Disponível em: <http://dx.doi.org/10.1007/s00500-014-1266-0>. Acesso em: 17 ago. 2017. Citado 3 vezes nas páginas 33, 38 e 39.
- 43 WODECKI, M. et al. Multi-GPU parallel memetic algorithm for capacitated vehicle routing problem. In: *Parallel Processing and Applied Mathematics*. Springer Berlin Heidelberg, 2014. p. 207–214. Disponível em: https://doi.org/10.1007/978-3-642-55195-6_19. Acesso em: 17 ago. 2017. Citado 3 vezes nas páginas 33, 38 e 39.
- 44 EWBANK, H.; WANKE, P.; HADI-VENCHEH, A. An unsupervised fuzzy clustering approach to the capacitated vehicle routing problem. *Neural Computing and Applications*, Springer Nature, v. 27, n. 4, p. 857–867, apr 2015. Disponível em: <http://dx.doi.org/10.1007/s00521-015-1901-4>. Acesso em: 16 ago. 2017. Citado 3 vezes nas páginas 34, 38 e 39.
- 45 PECIN, D. et al. Improved branch-cut-and-price for capacitated vehicle routing. *Mathematical Programming Computation*, Springer Nature, v. 9, n. 1, p. 61–100, jun 2016. Disponível em: <http://dx.doi.org/10.1007/s12532-016-0108-8>. Acesso em: 16 ago. 2017. Citado 3 vezes nas páginas 34, 38 e 39.
- 46 KIR, S.; YAZGAN, H. R.; TÜNCEL, E. A novel heuristic algorithm for capacitated vehicle routing problem. *Journal of Industrial Engineering International*, Springer Nature, v. 13, n. 3, p. 323–330, feb 2017. Disponível em: <http://dx.doi.org/10.1007/s40092-017-0187-9>. Acesso em: 16 ago. 2017. Citado 3 vezes nas páginas 34, 38 e 39.
- 47 SANTILLAN, J. H. et al. Cuckoo search via lévy flights for the capacitated vehicle routing problem. *Journal of Industrial Engineering International*, Springer Nature, aug 2017. Disponível em: <http://dx.doi.org/10.1007/s40092-017-0227-5>. Acesso em: 17 ago. 2017. Citado 3 vezes nas páginas 34, 38 e 39.
- 48 AKHAND, M. A. H. et al. Optimization of capacitated vehicle routing problem using producer-scrounger method. In: *2015 IEEE International WIE Conference on Electrical and Computer Engineering (WIECON-ECE)*. IEEE, 2015. Disponível em: <http://dx.doi.org/10.1109/wiecon-ece.2015.7443922>. Acesso em: 16 ago. 2017. Citado 3 vezes nas páginas 34, 38 e 39.
- 49 AKHAND, M. A. H. et al. Solving capacitated vehicle routing problem with route optimization using swarm intelligence. In: *2015 2nd International Conference on Electrical Information and Communication Technologies (EICT)*. IEEE, 2015. Disponível em: <http://dx.doi.org/10.1109/eict.2015.7391932>. Acesso em: 16 ago. 2017. Citado 3 vezes nas páginas 34, 38 e 39.
- 50 CHEN, M.-H. et al. A hybrid two-stage sweep algorithm for capacitated vehicle routing problem. In: *2015 International Conference on Control, Automation and*

- Robotics*. IEEE, 2015. Disponível em: <<http://dx.doi.org/10.1109/iccar.2015.7166030>>. Acesso em: 16 ago. 2017. Citado 3 vezes nas páginas 35, 38 e 39.
- 51 STEINHAUS, M.; SHIRAZI, A. N.; SODHI, M. Modified self organizing neural network algorithm for solving the vehicle routing problem. In: *2015 IEEE 18th International Conference on Computational Science and Engineering*. IEEE, 2015. Disponível em: <<http://dx.doi.org/10.1109/cse.2015.56>>. Acesso em: 16 ago. 2017. Citado 3 vezes nas páginas 35, 38 e 39.
- 52 ZHANG, S.; LEE, C. An improved artificial bee colony algorithm for the capacitated vehicle routing problem. In: *2015 IEEE International Conference on Systems, Man, and Cybernetics*. IEEE, 2015. Disponível em: <<http://dx.doi.org/10.1109/smc.2015.371>>. Acesso em: 16 ago. 2017. Citado 3 vezes nas páginas 35, 38 e 39.
- 53 RUTTANATEERAWICHIEEN, K.; KURUTACH, W.; PICHPIBUL, T. A new efficient and effective golden-ball-based technique for the capacitated vehicle routing problem. In: *2016 IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS)*. IEEE, 2016. Disponível em: <<http://dx.doi.org/10.1109/icis.2016.7550764>>. Acesso em: 16 ago. 2017. Citado 3 vezes nas páginas 35, 38 e 39.
- 54 YEH, C.-C.; LIU, D.-Y.; LIAO, Y.-K. Two-stage iterated local search for solving capacitated vehicle routing problems. In: *2016 International Symposium on Computer, Consumer and Control (IS3C)*. IEEE, 2016. Disponível em: <<http://dx.doi.org/10.1109/is3c.2016.22>>. Acesso em: 16 ago. 2017. Citado 3 vezes nas páginas 35, 38 e 39.
- 55 MINGPRASERT, S.; MASUCHUN, R. Adaptive artificial bee colony algorithm for solving the capacitated vehicle routing problem. In: *2017 9th International Conference on Knowledge and Smart Technology (KST)*. IEEE, 2017. Disponível em: <<http://dx.doi.org/10.1109/kst.2017.7886072>>. Acesso em: 16 ago. 2017. Citado 3 vezes nas páginas 36, 38 e 39.
- 56 XIAO, L.; HAJJAM-EL-HASSANI, A.; DRIDI, M. An application of extended cuckoo search to vehicle routing problem. In: *2017 International Colloquium on Logistics and Supply Chain Management (LOGISTIQUA)*. IEEE, 2017. Disponível em: <<http://dx.doi.org/10.1109/logistiqua.2017.7962869>>. Acesso em: 16 ago. 2017. Citado 3 vezes nas páginas 36, 38 e 39.
- 57 OLIVEIRA, R. A. D. C.; DELGADO, K. V. Capacitated vehicle routing system applying monte carlo methods. In: *Proceedings of the Annual Conference on Brazilian Symposium on Information Systems: Information Systems: A Computer Socio-Technical Perspective - Volume 1*. Porto Alegre, Brazil, Brazil: Brazilian Computer Society, 2015. (SBSI 2015). Disponível em: <<http://dl.acm.org/citation.cfm?id=2814058.2814060>>. Acesso em: 20 ago. 2017. Citado 4 vezes nas páginas 36, 37, 38 e 39.
- 58 LIMA, S. J. D. A. *Otimização do problema de roteamento de veículos capacitado usando algoritmos genéticos com heurísticas e representações cromossômicas alternativas*. Dissertação (Mestrado) — Universidade Nove de Julho, 2015. Disponível em: <<http://bibliotecatede.uninove.br/handle/tede/1124>>. Acesso em: 20 ago. 2017. Citado 3 vezes nas páginas 36, 38 e 39.

- 59 MASUTTI, T. A. S. *Um Framework Inspirado no Comportamento Coletivo de Abelhas para a Solução de Problemas de Roteamento de Veículos*. Dissertação (Mestrado) — Universidade Presbiteriana Mackenzie, 2016. Disponível em: <http://tede.mackenzie.br/jspui/handle/tede/3123>. Acesso em: 20 ago. 2017. Citado 3 vezes nas páginas 36, 38 e 39.
- 60 SHAW, P. Using constraint programming and local search methods to solve vehicle routing problems. In: *Principles and Practice of Constraint Programming — CP98*. Springer Berlin Heidelberg, 1998. p. 417–431. Disponível em: http://dx.doi.org/10.1007/3-540-49481-2_30. Acesso em: 17 ago. 2017. Citado na página 45.
- 61 ROPKE, S.; PISINGER, D. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, Institute for Operations Research and the Management Sciences (INFORMS), v. 40, n. 4, p. 455–472, nov 2006. Disponível em: <https://doi.org/10.1287/trsc.1050.0135>. Acesso em: 17 ago. 2017. Citado na página 47.
- 62 AKPINAR, S.; BAYHAN, G. M.; BAYKASOGLU, A. Hybridizing ant colony optimization via genetic algorithm for mixed-model assembly line balancing problem with sequence dependent setup times between tasks. *Applied Soft Computing*, Elsevier BV, v. 13, n. 1, p. 574–589, jan 2013. Disponível em: <http://dx.doi.org/10.1016/j.asoc.2012.07.024>. Acesso em: 9 nov. 2017. Citado 2 vezes nas páginas 51 e 63.
- 63 AUGERAT, P. et al. Computational results with a branch and cut code for the capacitated vehicle routing problem. 1995. Disponível em: https://www.researchgate.net/publication/239062886_Computational_results_with_a_branch_and_cut_code_for_the_capacitated_vehicle_routing_problem. Acesso em: 17 ago. 2017. Citado na página 53.
- 64 CHRISTOFIDES, N.; EILON, S. An algorithm for the vehicle-dispatching problem. *Journal of the Operational Research Society*, Informa UK Limited, v. 20, n. 3, p. 309–318, sep 1969. Disponível em: <https://link.springer.com/article/10.1057/jors.1969.75>. Acesso em: 2 mar. 2018. Citado na página 53.
- 65 CHRISTOFIDES, N. *Combinatorial Optimization*. John Wiley & Sons Ltd, 1979. ISBN 0471997498. Disponível em: <https://www.amazon.com/Combinatorial-Optimization-Nicos-Christofides/dp/0471997498?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0471997498>. Acesso em: 24 set. 2017. Citado 2 vezes nas páginas 53 e 65.
- 66 FISHER, M. L. Optimal solution of vehicle routing problems using minimum k-trees. *Operations Research*, Institute for Operations Research and the Management Sciences (INFORMS), v. 42, n. 4, p. 626–642, aug 1994. Disponível em: <https://doi.org/10.1287/opre.42.4.626>. Acesso em: 2 mar. 2018. Citado na página 53.

Apêndices

APÊNDICE A – Instâncias do repositório CVRPLib

Tabela 11 – Instâncias do repositório CVRPLib, onde n é o número de consumidores, K o número de veículos, Q a capacidade máxima dos veículos e MSC a melhor solução conhecida

Instância	n	K	Q	MSC	Ótimo
A-n32-k5	31	5	100	784	sim
A-n33-k5	32	5	100	661	sim
A-n33-k6	32	6	100	742	sim
A-n34-k5	33	5	100	778	sim
A-n36-k5	35	5	100	799	sim
A-n37-k5	36	5	100	669	sim
A-n37-k6	36	6	100	949	sim
A-n38-k5	37	5	100	730	sim
A-n39-k5	38	5	100	822	sim
A-n39-k6	38	6	100	831	sim
A-n44-k6	43	6	100	937	sim
A-n45-k6	44	6	100	944	sim
A-n45-k7	44	7	100	1146	sim
A-n46-k7	45	7	100	914	sim
A-n48-k7	47	7	100	1073	sim
A-n53-k7	52	7	100	1010	sim
A-n54-k7	53	7	100	1167	sim
A-n55-k9	54	9	100	1073	sim
A-n60-k9	59	9	100	1354	sim
A-n61-k9	60	9	100	1034	sim
A-n62-k8	61	8	100	1288	sim
A-n63-k9	62	9	100	1616	sim
A-n63-k10	62	10	100	1314	sim
A-n64-k9	63	9	100	1401	sim
A-n65-k9	64	9	100	1174	sim
A-n69-k9	68	9	100	1159	sim
A-n80-k10	79	10	100	1763	sim
B-n31-k5	30	5	100	672	sim

B-n34-k5	33	5	100	788	sim
B-n35-k5	34	5	100	955	sim
B-n38-k6	37	6	100	805	sim
B-n39-k5	38	5	100	549	sim
B-n41-k6	40	6	100	829	sim
B-n43-k6	42	6	100	742	sim
B-n44-k7	43	7	100	909	sim
B-n45-k5	44	5	100	751	sim
B-n45-k6	44	6	100	678	sim
B-n50-k7	49	7	100	741	sim
B-n50-k8	49	8	100	1312	sim
B-n51-k7	50	7	100	1032	sim
B-n52-k7	51	7	100	747	sim
B-n56-k7	55	7	100	707	sim
B-n57-k7	56	7	100	1153	sim
B-n57-k9	56	9	100	1598	sim
B-n63-k10	62	10	100	1496	sim
B-n64-k9	63	9	100	861	sim
B-n66-k9	65	9	100	1316	sim
B-n67-k10	66	10	100	1032	sim
B-n68-k9	67	9	100	1272	sim
B-n78-k10	77	10	100	1221	sim
E-n13-k4	12	4	6000	247	sim
E-n22-k4	21	4	6000	375	sim
E-n23-k3	22	3	4500	569	sim
E-n30-k3	29	3	4500	534	sim
E-n31-k7	30	7	140	379	sim
E-n33-k4	32	4	8000	835	sim
E-n51-k5	50	5	160	521	sim
E-n76-k7	75	7	220	682	sim
E-n76-k8	75	8	180	735	sim
E-n76-k10	75	10	140	830	sim
E-n76-k14	75	14	100	1021	sim
E-n101-k8	100	8	200	815	sim
E-n101-k14	100	14	112	1067	sim
F-n45-k4	44	4	2010	724	sim
F-n72-k4	71	4	30000	237	sim
F-n135-k7	134	7	2210	1162	sim
M-n101-k10	100	10	200	820	sim

M-n121-k7	120	7	200	1034	sim
M-n151-k12	150	12	200	1015	sim
M-n200-k16	199	16	200	1274	sim
M-n200-k17	199	17	200	1275	sim
P-n19-k2	18	2	160	212	sim
P-n20-k2	19	2	160	216	sim
P-n21-k2	20	2	160	211	sim
P-n22-k2	21	2	160	216	sim
P-n22-k8	21	8	3000	603	sim
P-n23-k8	22	8	40	529	sim
P-n40-k5	39	5	140	458	sim
P-n45-k5	44	5	150	510	sim
P-n50-k7	49	7	150	554	sim
P-n50-k8	49	8	120	631	sim
P-n50-k10	49	10	100	696	sim
P-n51-k10	50	10	80	741	sim
P-n55-k7	54	7	170	568	sim
P-n55-k10	54	10	115	694	sim
P-n55-k15	54	15	70	989	sim
P-n60-k10	59	10	120	744	sim
P-n60-k15	59	15	80	968	sim
P-n65-k10	64	10	130	792	sim
P-n70-k10	69	10	135	827	sim
P-n76-k4	75	4	350	593	sim
P-n76-k5	75	5	280	627	sim
P-n101-k4	100	4	400	681	sim
CMT1	50	5	160	524.61	sim
CMT2	75	10	140	835.26	sim
CMT3	100	8	200	826.14	sim
CMT4	150	12	200	1028.42	sim
CMT5	199	17	200	1291.29	sim
CMT6	50	6	160	555.43	não
CMT7	75	11	140	909.68	não
CMT8	100	9	200	865.95	não
CMT9	150	14	200	1162.55	não
CMT10	199	18	200	1395.85	não
CMT11	120	7	200	1042.12	sim
CMT12	100	10	200	819.56	sim
CMT13	120	11	200	1541.14	não

CMT14	100	11	200	866.37	não
tai75a	75	10	1445	1618.36	sim
tai75b	75	9	1679	1344.62	sim
tai75c	75	9	1122	1291.01	sim
tai75d	75	9	1699	1365.42	sim
tai100a	100	11	1409	2041.34	sim
tai100b	100	11	1842	1939.9	sim
tai100c	100	11	2043	1406.2	sim
tai100d	100	11	1297	1580.46	sim
tai150a	150	15	1544	3055.23	sim
tai150b	150	14	1918	2727.03	sim
tai150c	150	14	2021	2358.66	sim
tai150d	150	14	1874	2645.39	sim
tai385	385	46	65	24366.41	não
Golden_1	240	9	550	5623.47	não
Golden_2	320	10	700	8404.61	não
Golden_3	400	9	900	11036.2	não
Golden_4	480	10	1000	13590	não
Golden_5	200	5	900	6460.98	não
Golden_6	280	7	900	8412.9	não
Golden_7	360	8	900	10102.7	não
Golden_8	440	10	900	11635.3	não
Golden_9	255	14	1000	579.71	não
Golden_10	323	16	1000	735.66	não
Golden_11	399	17	1000	912.03	não
Golden_12	483	19	1000	1101.5	não
Golden_13	252	26	1000	857.19	sim
Golden_14	320	29	1000	1080.55	sim
Golden_15	396	33	1000	1337.87	não
Golden_16	480	36	1000	1611.56	não
Golden_17	240	22	200	707.76	sim
Golden_18	300	27	200	995.13	sim
Golden_19	360	33	200	1365.6	sim
Golden_20	420	38	200	1817.59	sim
Li_21	560	10	1200	16212.83	não
Li_22	600	15	900	14499.04	não
Li_23	640	10	1400	18801.13	não
Li_24	720	10	1500	21389.43	não
Li_25	760	19	900	16668.51	não

Li_26	800	10	1700	23977.73	não
Li_27	840	20	900	17372.64	não
Li_28	880	10	1800	26566.03	não
Li_29	960	10	2000	29154.34	não
Li_30	1040	10	2100	31742.64	não
Li_31	1120	10	2300	34330.94	não
Li_32	1200	11	2500	37159.41	não
X-n101-k25	100	25	206	27591	sim
X-n106-k14	105	14	600	26362	sim
X-n110-k13	109	13	66	14971	sim
X-n115-k10	114	10	169	12747	sim
X-n120-k6	119	6	21	13332	sim
X-n125-k30	124	30	188	55539	sim
X-n129-k18	128	18	39	28940	sim
X-n134-k13	133	13	643	10916	sim
X-n139-k10	138	10	106	13590	sim
X-n143-k7	142	7	1190	15700	sim
X-n148-k46	147	46	18	43448	sim
X-n153-k22	152	22	144	21220	sim
X-n157-k13	156	13	12	16876	sim
X-n162-k11	161	11	1174	14138	sim
X-n167-k10	166	10	133	20557	sim
X-n172-k51	171	51	161	45607	sim
X-n176-k26	175	26	142	47812	sim
X-n181-k23	180	23	8	25569	sim
X-n186-k15	185	15	974	24145	sim
X-n190-k8	189	8	138	16980	sim
X-n195-k51	194	51	181	44225	sim
X-n200-k36	199	36	402	58578	sim
X-n204-k19	203	19	836	19565	sim
X-n209-k16	208	16	101	30656	sim
X-n214-k11	213	11	944	10856	sim
X-n219-k73	218	73	3	117595	sim
X-n223-k34	222	34	37	40437	sim
X-n228-k23	227	23	154	25742	sim
X-n233-k16	232	16	631	19230	sim
X-n237-k14	236	14	18	27042	sim
X-n242-k48	241	48	28	82751	sim
X-n247-k50	246	47	134	37274	sim

X-n251-k28	250	28	69	38684	sim
X-n256-k16	255	16	1225	18839	não
X-n261-k13	260	13	1081	26558	não
X-n266-k58	265	58	35	75478	sim
X-n270-k35	269	35	585	35291	sim
X-n275-k28	274	28	10	21245	sim
X-n280-k17	279	17	192	33503	não
X-n284-k15	283	15	109	20226	não
X-n289-k60	288	60	267	95151	sim
X-n294-k50	293	50	285	47161	não
X-n298-k31	297	31	55	34231	sim
X-n303-k21	302	21	794	21744	não
X-n308-k13	307	13	246	25859	não
X-n313-k71	312	71	248	94044	não
X-n317-k53	316	53	6	78355	sim
X-n322-k28	321	28	868	29834	não
X-n327-k20	326	20	128	27532	não
X-n331-k15	330	15	23	31102	sim
X-n336-k84	335	84	203	139135	não
X-n344-k43	343	43	61	42056	não
X-n351-k40	350	40	436	25928	não
X-n359-k29	358	29	68	51505	não
X-n367-k17	366	17	218	22814	não
X-n376-k94	375	94	4	147713	sim
X-n384-k52	383	52	564	65943	não
X-n393-k38	392	38	78	38260	não
X-n401-k29	400	29	745	66187	não
X-n411-k19	410	19	216	19718	não
X-n420-k130	419	130	18	107798	sim
X-n429-k61	428	61	536	65483	não
X-n439-k37	438	37	12	36391	sim
X-n449-k29	448	29	777	55269	não
X-n459-k26	458	26	1106	24145	não
X-n469-k138	468	138	256	221909	não
X-n480-k70	479	70	52	89458	não
X-n491-k59	490	59	428	66510	não
X-n502-k39	501	39	13	69230	não
X-n513-k21	512	21	142	24201	não
X-n524-k153	523	137	125	154593	sim

X-n536-k96	535	96	371	94988	não
X-n548-k50	547	50	11	86701	não
X-n561-k42	560	42	74	42722	não
X-n573-k30	572	30	210	50718	não
X-n586-k159	585	159	28	190423	não
X-n599-k92	598	92	487	108489	não
X-n613-k62	612	62	523	59556	não
X-n627-k43	626	43	110	62210	não
X-n641-k35	640	35	1381	63737	não
X-n655-k131	654	131	5	106780	sim
X-n670-k130	669	126	129	146477	não
X-n685-k75	684	75	408	68261	não
X-n701-k44	700	44	87	81934	não
X-n716-k35	715	35	1007	43414	não
X-n733-k159	732	159	25	136250	não
X-n749-k98	748	98	396	77365	não
X-n766-k71	765	71	166	114525	não
X-n783-k48	782	48	832	72445	não
X-n801-k40	800	40	20	73331	não
X-n819-k171	818	171	358	158265	não
X-n837-k142	836	142	44	193810	não
X-n856-k95	855	95	9	89002	não
X-n876-k59	875	59	764	99331	não
X-n895-k37	894	37	1816	53946	não
X-n916-k207	915	207	33	329247	não
X-n936-k151	935	151	138	132923	não
X-n957-k87	956	87	11	85478	não
X-n979-k58	978	58	998	119008	não
X-n1001-k43	1000	43	131	72403	não

APÊNDICE B – Códigos do analisador de instâncias do repositório CVRPlib

B.1 Arquivo project.clj

```
(defproject cvrplib_parser "1.0"
  :description "A CVRPlib instance parser written in Clojure"
  :url "https://github.com/lucasb/cvrplib_parser"
  :license {:name "Eclipse Public License"
            :url "http://www.eclipse.org/legal/epl-v10.html"}
  :dependencies [[org.clojure/clojure "1.8.0"]
                 [cheshire "5.8.0"]]
  :main ^:skip-aot cvrplib-parser.core
  :target-path "target/%s"
  :profiles {:uberjar {:aot :all}})
```

B.2 Arquivo core.clj

```
(ns cvrplib-parser.core
  (:require [cvrplib-parser.parser :as parser]
            [cvrplib-parser.distances :as distances]
            [cvrplib-parser.optimal :as optimal]
            [clojure.java.io :as io]
            [cheshire.core :as json]
            [clojure.string :as string])
  (:gen-class))

(def json-options {:pretty true, :key-fn #(string/replace (name %) #"- " "_")})

(defn- write-json
  "Write a JSON file which represents the instance 'x'"
  [x]
  (let [fout (str "./out/" (name x) ".json")]
    (io/make-parents fout)
    (spit fout (json/generate-string x json-options))))

(defn- parse-sol-file
  "Parse the sol file 'f'"
  [f sol]
  (let [fname (string/replace (str f) #".vrp" ".sol")
```

```

    [fsol] (filter #(.endsWith (str %) fname) sol)]
  (if-not fsol
    {}
    (optimal/parse-best-known (slurp fsol))))

(defn- parse-opt-file
  "Parse the opt file 'f'"
  [f opt]
  (let [fname (string/replace (str f) #".vrp" ".opt")
        [fopt] (filter #(.endsWith (str %) fname) opt)]
    (if-not fopt
      {}
      (optimal/parse-optimal (slurp fopt)))))

(defn- parse-vrp-file
  "Parse the vrp file 'f'"
  [f]
  (-> f
    slurp
    parser/parse
    distances/create-distances))

(defn- parse-file
  "Parse the instance files 'f', 'opt' and 'sol' to produce a JSON file"
  [f opt sol]
  (println "Parsing" (str f))
  (write-json (merge (parse-vrp-file f)
                    (parse-opt-file f opt)
                    (parse-sol-file f sol))))

(defn- parse-files
  "Read a specific cvrplib file or all files within a specific 'path'"
  [path]
  (println "Parsing all files on" path)
  (let [fs (file-seq (io/file path))
        vrp (filter #(.endsWith (str %) ".vrp") fs)
        opt (filter #(.endsWith (str %) ".opt") fs)
        sol (filter #(.endsWith (str %) ".sol") fs)]
    (doseq [f vrp]
      (parse-file f opt sol))))

(defn -main
  "Read the directory or file from 'args' and parse it"
  [& args]
  (parse-files (first args)))

```



```

(defn- parse-specs
  "Parse the specifications of the instance file"
  [file]
  (let [specs (re-seq #"[A-Z_\s]+:." file)]
    (apply merge (map (comp parse-spec string/trim) specs))))

;;; Parse instance data
(defn- parse-depot-coords
  "Parse the depot coords from file 'f'"
  [f]
  (when-let [depot (re-find #"DEPOT_SECTION[\n\s]+[\d.]+ [\d.]+ " f)]
    (as-> depot d
      (string/split d #"[\s\n]")
      (rest d)
      (remove empty? d)
      (mapv str->int d))))

(defn- parse-depot-section
  "If depot coordinates is given on depot section (e.g. CMT and Golden instances),
  parse it from file 'f'. Return an empty 'clojure.core/hash-map' otherwise"
  [f]
  (if-let [coords (parse-depot-coords f)]
    {:node-coord-section [(into [0] coords)],
     :demand-section [[0 0]]}
    {}))

(defn- parse-vehicles
  "Parse the min. number of vehicle from file 'f', if exists"
  [f]
  (if-let [v (re-find #"trucks: \d+|VEHICLES: \d+" f)]
    {:vehicles (-> v
      (string/replace #"\\w+:\\s" "")
      str->int)}}
    {}))

(defn- parse-section
  "Parse each line of section 's' from file 'f'"
  [f s]
  (let [re (re-pattern (str s "[\\d\\s\\n.-]+"))]
    sl (re-find re f)]
    (if-not sl
      {}
      {(str->key s) (->> sl
        string/split-lines
        rest
        (map (comp #(map str->int %))

```

```

                                #(remove empty? %)
                                #(string/split % #"\\s")
                                string/trim))))))

(defn- parse-weight-section
  "Parse each line of weight section from file 'f'"
  [f]
  (let [s (re-find #"EDGE_WEIGHT_SECTION[\\d\\s\\n]+" f)]
    (if-not s
      {}
      {:edge-weight-section (->> s
                                string/split-lines
                                (map string/trim)
                                rest
                                (map #(string/split % #"\\s"))
                                flatten
                                (remove empty?)
                                (map str->int))})))

(defn- parse-data
  "Parse the data of the instance file"
  [file]
  (merge-with concat
    (parse-depot-section file)
    (parse-vehicles file)
    (parse-section file "NODE_COORD_SECTION")
    (parse-section file "DEMAND_SECTION")
    (parse-weight-section file)))

(defn parse
  "Parse the CVRplib instance 'f' and return a 'clojure.core/hash-map'"
  [f]
  (let [file (string/replace f #"[\\t\\r]" " ")]
    (merge (parse-specs file)
      (parse-data file))))

```

B.4 Arquivo distances.clj

```

(ns cvrplib-parser.distances
  (:require [clojure.string :as string]))

;; Create distances matrix from EDGE_WEIGHT_SECTION

(defn- create-lower-triangle
  "Group each value from 'xs' to form a lower triangular sequence"
  ([xs] (create-lower-triangle [] xs 0))

```

```

([t xs n]
  (if (empty? xs)
      t
      (recur (conj t (take n xs))
              (drop n xs)
              (inc n)))))

(defn- create-upper-triangle
  "Group each value from 'xs' to form an upper triangular sequence"
  ([xs] (create-upper-triangle [nil] xs 1))
  ([t xs n]
    (if (empty? xs)
        (reverse t)
        (recur (conj t (take-last n xs))
                (drop-last n xs)
                (inc n)))))

(defn- create-distances-matrix
  "Create the distance matrix from the sequence 'xs'"
  [xs]
  (map concat (create-lower-triangle xs) (repeat [0]) (create-upper-triangle xs)))

;; Create distances matrix from NODE_COORD_SECTION

(defn- euclidean-distance
  "Calculate the euclidean distance between node 'n1' and 'n2'"
  [n1 n2]
  (let [[_ x1 y1] n1
        [_ x2 y2] n2
        xd (- x1 x2)
        yd (- y1 y2)]
    (Math/round (Math/sqrt (+ (* xd xd) (* yd yd)))))

(defn- calculate-distances-matrix
  "Calculate the distances between every node on 'coords'"
  [coords]
  (let [c (sort-by first coords)]
    (for [n1 c] (for [n2 c] (euclidean-distance n1 n2)))))

(defn create-distances
  "Create the distance matrix of 'instance'"
  [instance]
  (cond (contains? instance :node-coord-section)
        (merge instance
                { :distances (calculate-distances-matrix (:node-coord-section
                                                            instance)) })
        (contains? instance :edge-weight-section)
        ))

```

```

      (merge instance
        {:distances (create-distances-matrix (:edge-weight-section
→ instance))}))
      :else instance))

```

B.5 Arquivo optimal.clj

```

(ns cvrplib-parser.optimal
  (:require [clojure.string :as string]))

(defn- str->int
  "Convert the string 's' to a number"
  [s]
  (if (.contains s ".")
    (Double. s)
    (Integer. s)))

(defn- parse-cost
  "Parse the cost of the optimal solution from 'file'"
  [file]
  {:cost (as-> file f
    (re-find #"[C|c]ost[\d\s]+" f)
    (string/split f #"\"s")
    (second f)
    (str->int f)))})

(defn- parse-routes
  "Parse the routes of the optimal solution from 'file'"
  [file]
  {:routes (->> file
    (re-seq #"[R|r]oute #\d: [\d\s]+" )
    (map (comp #(map str->int %)
      #(remove empty? %)
      #(string/split % #"\"s")
      #(string/replace % #"Route #\d:" ""))))})

(defn parse-optimal
  "Parse the optimal solution from opt 'file'"
  [file]
  {:optimal (merge (parse-routes file)
    (parse-cost file))})

(defn parse-best-known
  "Parse the Best Known Solution from sol 'file'"
  [file]
  {:best-known (-> file

```

```
string/split-lines  
first  
str->int}})
```

APÊNDICE C – Códigos do Algoritmo Híbrido de Otimização por Colônia de Formigas

C.1 Arquivo project.clj

```
(defproject cvrplib-solver "0.1.0-SNAPSHOT"
  :description "FIXME: write description"
  :url "http://example.com/FIXME"
  :license {:name "Eclipse Public License"
            :url "http://www.eclipse.org/legal/epl-v10.html"}
  :dependencies [[org.clojure/clojure "1.8.0"]
                 [net.mikera/vectorz-clj "0.47.0"]
                 [cheshire "5.8.0"]
                 [random-seed "1.0.0"]
                 [amalloy/ring-buffer "1.2.1"]
                 [com.taoensso/timbre "4.10.0"]
                 [org.clojure/tools.cli "0.3.7"]]
  :main ^:skip-aot hybrid-aco.core
  :target-path "target/%s"
  :jvm-opts ["-Xmx1G"]
  :profiles {:uberjar {:aot :all}})
```

C.2 Arquivo core.clj

```
(ns hybrid-aco.core
  (:use [hybrid-aco.instance])
  (:require [clojure.core.matrix :as matrix]
            [hybrid-aco.solution :as solution]
            [hybrid-aco.saco :as sacco]
            [random-seed.core :as random]
            [taoensso.timbre :as log])
  (:gen-class))

;; Default configs
(matrix/set-current-implementation :vectorz)
(log/merge-config! {:appenders {:spit (log/spit-appender {:fname "solver.log"})}}})
(log/set-level! :info)
```

```

(def ^:private param-file "resources/hybrid_aco_parameters.json")
(def ^:private initial-solution (struct solution/Solution [] Integer/MAX_VALUE))

(defn- run-with-parameters-on-instance
  "Run the algorithm with the specified parameters"
  [instance params]
  (log/info "Started instance" (:name instance))
  (binding [*instance* (:name instance)
            *dimension* (:dimension instance)
            *capacity* (:capacity instance)
            *distances* (matrix/matrix (:distances instance))
            *demands* (matrix/matrix (map second (:demand-section instance)))
            *optimal* (:cost (:optimal instance) (:best-known instance 0))
            *num-ants* (:dimension instance)
            *seed* (:seed params)
            *alpha* (:alpha params)
            *beta* (:beta params)
            *evaporation-rate* (- 1 (:evaporation-rate params))
            *perturbation-ratio* (:perturbation-ratio params)
            *num-elite-ants* (:num-elite-ants params)
            *initial-trail* (:initial-trail params)
            *initial-temperature* (:initial-temperature params)
            *final-temperature* (/ (:initial-temperature params) 50)
            *sa-max-tries* (max (* 4 (:dimension instance)) 250)
            *lambda* (:lambda params)
            *tabu-size* (:tabu-size params)
            *max-iter* (:max-iter params)
            *max-best-unchanged* (:dimension instance)
            *max-best-so-far-not-improved* (:max-best-so-far-not-improved params)
            *max-runtime* (:max-runtime params)
            *evaluations* (atom 0.0)
            *date-start* (java.util.Date.)
            *start* (System/nanoTime)]
    (random/set-random-seed! *seed*)
    (write-result! (saco/solve-instance initial-solution))))

(defn- run
  "Run the SACO metaheuristic on the given instance and generate a report file"
  [instance params]
  (try (run-with-parameters-on-instance instance params)
      (catch Exception e (log/error "ERROR ON INSTANCE"
                                    (:name instance)
                                    (.toString e)))))

(defn -main
  "I don't do a whole lot ... yet."
  [& args]

```

```
(let [params (read-file param-file)
      instances (read-instances (first args))]
  (doseq [instance instances]
    (run instance params))))
```

C.3 Arquivo instance.clj

```
(ns hybrid-aco.instance
  (:require [cheshire.core :as json]
            [clojure.core.matrix :as matrix]
            [clojure.java.io :as io]
            [taoensso.timbre :as log]))

;; Instance constants

(def ^:dynamic *instance*)
(def ^:dynamic *dimension*)
(def ^:dynamic *distances*)
(def ^:dynamic *capacity*)
(def ^:dynamic *demands*)
(def ^:dynamic *optimal*)

;; Algorithm parameters

;; ACO
(def ^:dynamic *num-ants*)
(def ^:dynamic *alpha*)
(def ^:dynamic *beta*)
(def ^:dynamic *evaporation-rate*)
(def ^:dynamic *perturbation-ratio*)
(def ^:dynamic *num-elite-ants*)
(def ^:dynamic *initial-trail*)

;; Simulated Annealing
(def ^:dynamic *initial-temperature*)
(def ^:dynamic *final-temperature*)
(def ^:dynamic *sa-max-tries*)
(def ^:dynamic *lambda*)
(def ^:dynamic *tabu-size*)

;; SACO
(def ^:dynamic *max-iter*)
(def ^:dynamic *max-best-unchanged*)
(def ^:dynamic *max-best-so-far-not-improved*)

(def ^:dynamic *max-runtime*)
```

```

(def ^:dynamic *date-start*)
(def ^:dynamic *start*)
(def ^:dynamic *evaluations*)

(def ^:dynamic *seed*)
(def ^:dynamic *debug*)

(def file-dateformat (java.text.SimpleDateFormat. "ddMMyy-HH:mm:ss"))
(def result-dateformat (java.text.SimpleDateFormat. "dd-MM-yyyy HH:mm:ss"))

(defn termination-criteria-reached?
  "Stop if runtime exceeded max-runtime or solution reached optimum"
  [best-so-far]
  (or (= (:cost best-so-far) *optimal*)
      (>= (/ (- (System/nanoTime) *start*) 1e9) *max-runtime*)))

(defn- to-keyword
  ""
  [string]
  (keyword (clojure.string/replace string "_" "-")))

(defn read-file
  "Read and parse the specified JSON file"
  [filename]
  (json/parse-string (slurp filename) to-keyword))

(defn read-instances
  ""
  [path]
  (let [files (file-seq (io/file path))
        instances (filter #(.endsWith (str %) ".json") files)]
    (map read-file instances)))

(defn calculate-dev
  ""
  [best]
  (if (zero? *optimal*)
      {}
      {:optimal *optimal*
       :dev (* (/ (- (:cost best) *optimal*) *optimal*) 100)}))

(defn generate-result
  ""
  [best elapsed-time date-end]
  {:instance *instance*
   :parameters {:num-ants *num-ants*
                 :alpha *alpha*

```

```

        :beta *beta*
        :evaporation-rate *evaporation-rate*
        :perturbation-ratio *perturbation-ratio*
        :num-elite-ants *num-elite-ants*
        :initial-trail *initial-trail*
        :initial-temperature *initial-temperature*
        :sa-max-tries *sa-max-tries*
        :lambda *lambda*
        :tabu-size *tabu-size*
        :max-iter *max-iter*
        :max-best-unchanged *max-best-unchanged*
        :max-best-so-far-not-improved *max-best-so-far-not-improved*
        :max-runtime *max-runtime*}

:solution best
:start (.format result-dateformat *date-start*)
:end (.format result-dateformat date-end)
:evaluations @*evaluations*
:elapsed-time elapsed-time})

(defn write-result!
  ""
  [best-found]
  (let [end (System/nanoTime)
        date-end (java.util.Date.)
        start-date (.format file-dateformat *date-start*)
        fileoutput-name (str "./out/" *instance* "-" start-date ".out")
        elapsed-time (/ (- end *start*) 1e9)]
    (log/info "Finished instance" *instance* "in" elapsed-time "seconds")
    (spit fileoutput-name
          (json/generate-string (merge (generate-result best-found
                                                         elapsed-time
                                                         date-end)
                                       (calculate-dev best-found))
                                {:pretty true}))))

```

C.4 Arquivo solution.clj

```

(ns hybrid-aco.solution
  (:use [hybrid-aco.instance])
  (:require [clojure.core.matrix :as matrix]))

(matrix/set-current-implementation :vectorz)

(defstruct Solution :tour :cost)

(defn split-tour

```

```

"Split the tour into routes"
[tour]
(->> tour
  (partition-by zero?)
  (remove #(= % '(0)))
  (map vec)))

(defn join-routes
  "Join each route into a tour"
  [routes]
  (->> routes
    (map vec)
    (map #(conj % 0))
    (cons [0])
    flatten
    vec))

(defn construct-edge-matrix
  "Construct the edge matrix of the solution"
  [solution]
  (let [tour (:tour solution)
        size (count (distinct tour))
        edges (map vector tour (rest tour))]
    (matrix/set-indices! (matrix/new-matrix size size)
                        edges
                        (repeat (count edges) 1.0))))

(defn objective-function
  "Calculate the total distance of the tour i.e. the cost of the solution"
  [tour]
  (swap! *evaluations* inc)
  (let [dists (map #(matrix/mget *distances* %1 %2) tour (rest tour))]
    (double (matrix/esum dists))))

(defn calculate-load
  "Calculate total load of the tour for the vehicle"
  [tour]
  (matrix/esum (matrix/emap #(matrix/mget *demands* %) tour)))

(defn feasible?
  "Check if the solution is feasible"
  [solution]
  (not-any? #(> (calculate-load %) *capacity*)
    (split-tour (:tour solution))))

(defn create-solution
  ""

```

```
[tour]
(let [t (dedupe tour)] ; Remove consecutive duplicates, if any
      (struct Solution t (objective-function t))))
```

C.5 Arquivo saco.clj

```
(ns hybrid-aco.saco
  (:use [hybrid-aco.instance])
  (:require [hybrid-aco.pheromones :as pheromones]
             [hybrid-aco.aco :as aco]
             [hybrid-aco.local-search :as search]
             [hybrid-aco.sa :as sa]
             [clojure.core.matrix :as matrix]
             [taoensso.timbre :as log]))

(def operations [search/make-2opt-move search/make-swap-move])
(def heuristic (fn [dist] (/ 1.0 (+ dist 0.1))))

(defn- construct-heuristics-matrix
  "Construct the heuristic matrix"
  []
  (matrix/emap heuristic *distances*))

(defn hybrid-aco
  ""
  [best-so-far last-best iter cnt-best cnt-best-so-far heuristics pheromones]
  (if (or (= iter *max-iter*)
          (termination-criteria-reached? best-so-far))
      best-so-far
      (let [ants1 (aco/construct-solutions heuristics pheromones)
            ants2 (map #(search/local-search % operations) ants1)
            ants-ranked (sort-by :cost ants2)
            best-ant (first ants-ranked)
            elite-ants (take *num-elite-ants* ants-ranked)

            best-ant-changed? (not= best-ant last-best)
            best-so-far1 (if (< (:cost best-ant) (:cost best-so-far))
                           best-ant
                           best-so-far)
            best-so-far-improved? (not= best-so-far1 best-so-far)

            pheromones1 (pheromones/update-pheromones pheromones
                                                         elite-ants
                                                         best-so-far1)

            perturbate-pheromones? (= cnt-best *max-best-unchanged*)
```

```

    pheromones2 (if perturbate-pheromones?
                  (pheromones/perturbate-pheromones pheromones1)
                  pheromones1)

    improve-with-sa? (= cnt-best-so-far *max-best-so-far-not-improved*)
    [best-so-far2 pheromones3] (if improve-with-sa?
                                    (sa/improve-solution best-so-far1
                                                            [best-so-far1 pheromones2]))
    → pheromones2)

    (recur best-so-far2
           best-ant
           (inc iter)
           (cond best-ant-changed? 0
                 perturbate-pheromones? (- cnt-best 2)
                 :else (inc cnt-best))
           (cond best-so-far-improved? 0
                 improve-with-sa? 0
                 :else (inc cnt-best-so-far))
           heuristics
           pheromones3))))

(defn solve-instance
  "Use SACO metaheuristic to find a solution closest to optimum"
  [best-so-far]
  (let [heuristics (construct-heuristics-matrix)
        pheromones (pheromones/construct-pheromones-matrix)]
    (hybrid-aco best-so-far best-so-far 0 0 0 heuristics pheromones)))

```

C.6 Arquivo aco.clj

```

(ns hybrid-aco.aco
  (:use [hybrid-aco.instance])
  (:require [clojure.core.matrix :as matrix]
            [hybrid-aco.parameters :as params]
            [hybrid-aco.solution :as solution]
            [random-seed.core :as random]
            [taoensso.timbre :as log]))

(matrix/set-current-implementation :vectorz)

(defn build-probabilities-matrix
  "Build the probability to select each customer"
  [heuristics pheromones]
  (matrix/emul (matrix/pow pheromones *alpha*)
               (matrix/pow heuristics *beta*)))

```

```

(defn- build-customers-probabilities
  "Build the probability to select the next customer to the ant from customer i"
  [customers i probs]
  (let [cust-probs (matrix/emap #(matrix/mget probs i %) customers)
        total (matrix/esum cust-probs)]
    (map #(/ % total) cust-probs)))

(defn- build-ant-wheel
  "Calculate the accumulate probability to the ant wheel"
  [customers probabilities]
  (map #(hash-map :customer %1 :probability %2)
       customers
       (conj (vec (butlast (reductions + probabilities))) 1.0))) ; Ensuring that the
→ last accumulate probability will always be 1.0 can slow down the function but
→ avoid problems caused by rounding

(defn- select-next-customer
  "Select the next customer to be added to the ant tour"
  [last-customer customers probs]
  (let [cust-probs (build-customers-probabilities customers last-customer probs)
        wheel (build-ant-wheel customers cust-probs)
        r (random/rand)] ; Generates a random number between the interval [0, 1.0)
    (some #(cond (> (:probability %) r) (:customer %)) wheel)))

(defn- construct-ant-tour
  "Construct the tour of an ant"
  [tour acc-dem customers probs]
  (if (empty? customers)
      (conj tour 0) ; If there is no more customer to add, return the vehicle to the
→ depot and exit the function
      (let [next-cust (select-next-customer (last tour) customers probs)
            next-cust-dem (matrix/mget *demands* next-cust)
            total-dem (+ acc-dem next-cust-dem)
            return-to-depot (> total-dem *capacity*) ; If the total demand with the
→ selected customer overloads the vehicle capacity, the vehicle should return to
→ the depot first
            new-tour (if return-to-depot (conj tour 0 next-cust) (conj tour
→ next-cust))
            new-acc-dem (if return-to-depot next-cust-dem total-dem)]
        (recur new-tour
               new-acc-dem
               (remove #(= % next-cust) customers)
               probs))))

(defn- construct-ant-solution
  "Construct the solution based of the probability to select each customer"
  [probabilities]

```

```

(->> probabilities
  (construct-ant-tour [0] 0 (range 1 *dimension*))
  (solution/create-solution)))

(defn construct-solutions
  "Construct the solution for each ant of the colony"
  [heuristics pheromones]
  ;(cond *debug* (log/debug "Started ACO"))
  (let [probs (build-probabilities-matrix heuristics pheromones)]
    (map construct-ant-solution (repeat *num-ants* probs))))

```

C.7 Arquivo local_search.clj

```

(ns hybrid-aco.local-search
  (:use [hybrid-aco.instance])
  (:require [hybrid-aco.solution :as solution]
            [hybrid-aco.utils :as utils]
            [random-seed.core :as random]))

(defn make-swap-move
  "Selects two customers randomly and swaps their position"
  [tour]
  (let [customers (range 1 *dimension*)
        customer1 (random/rand-nth customers)
        customer2 (random/rand-nth (remove #{customer1} customers))]
    (replace {customer1 customer2, customer2 customer1} tour)))

(defn make-inversion
  "Selects two customers in a route randomly and inverts all customer between them"
  [route]
  ;(println route)
  (if (< (count route) 2)
    route
    (let [route-length (count route)
          n (random/rand-int (dec route-length))
          m (random/rand-int (- route-length n 1))
          subroute1 (take n route)
          subroute2 (drop n (drop-last m route))
          subroute3 (take-last m route)]
      ;(println subroute1 (reverse subroute2) subroute3)
      (vec (concat subroute1 (reverse subroute2) subroute3)))))

(defn make-2opt-move
  "Select randomly a route in the tour to apply inversion move"
  [tour]
  (let [routes (solution/split-tour tour)]

```

```

        (solution/join-routes (utils/rand-map 1 make-inversion routes))))

(defn make-insertion-move
  "Select a customer randomly and insert it in a random position"
  [tour]
  (let [customers (range 1 *dimension*)
        customer (random/rand-nth customers)
        ;_ (println customer)
        new-tour (remove #{customer} tour)
        position (random/rand-nth (range 1 (dec (count new-tour))))]
    (vec (concat (take position new-tour) [customer] (drop position new-tour)))))

(defn search-neighborhood
  "Select randomly one of the operations to make a move on the neighborhood"
  [solution operations]
  (let [make-move (random/rand-nth operations)]
    ;(println "Chose to perform" make-move)
    (solution/create-solution (make-move (:tour solution)))))

(defn local-search
  "Perform a local search on the current solution"
  [solution operations]
  ;(println "Performing local search")
  (let [new-sol (search-neighborhood solution operations)]
    (if (or (not (solution/feasible? new-sol))
            (>= (:cost new-sol) (:cost solution)))
        solution
        (recur new-sol operations))))

```

C.8 Arquivo pheromones.clj

```

(ns hybrid-aco.pheromones
  (:use [hybrid-aco.instance])
  (:require [clojure.core.matrix :as matrix]
            [hybrid-aco.solution :as solution]))

(defn construct-pheromones-matrix
  "Construct a new pheromones matrix"
  []
  (matrix/fill! (matrix/new-matrix *dimension* *dimension*) *initial-trail*))

(defn evaporate-pheromones
  "Decrease pheromone on all edges"
  [pheromones]
  (matrix/emul! pheromones *evaporation-rate*))

```

```

(defn pheromone-amount
  "Calculate the pheromone amount of an ant on an edge"
  [ant weight]
  (/ weight (:cost ant)))

(defn release-ant-pheromone
  ""
  [ant weight]
  (let [edges (solution/construct-edge-matrix ant)
        deposit-amount (pheromone-amount ant weight)]
    (matrix/emul edges deposit-amount)))

(defn update-pheromones
  ""
  [pheromones elite-ants best-so-far-ant]
  (let [elites-weight (range *num-elite-ants* 0 -1)
        elites-pheromones (map #(release-ant-pheromone %1 %2)
                                elite-ants
                                elites-weight)
        best-so-far-pheromones (release-ant-pheromone best-so-far-ant 12)]
    (matrix/add! (evaporate-pheromones pheromones)
                 (apply matrix/add elites-pheromones)
                 best-so-far-pheromones)))

(defn perturbate-pheromones
  "Apply the perturbation on the pheromones to avoid solution stagnation"
  [pheromones]
  (let [non-perturbation-ratio (- 1 *perturbation-ratio*)
        average (/ (matrix/esum pheromones) (matrix/ecount pheromones))
        perturbation (* *perturbation-ratio* average)]
    (matrix/emap! #(+ perturbation (* non-perturbation-ratio %)) pheromones)))

```

C.9 Arquivo sa.clj

```

(ns hybrid-aco.sa
  (:use [hybrid-aco.instance])
  (:require [hybrid-aco.local-search :as search]
            [hybrid-aco.solution :as solution]
            [hybrid-aco.pheromones :as pheromones]
            [clojure.core.matrix :as matrix]
            [amalloy.ring-buffer :as ring]
            [taoensso.timbre :as log]
            [random-seed.core :as random]))

(def operations [search/make-swap-move
                 search/make-2opt-move

```

```

        search/make-insertion-move])

(defn simulated-annealing
  "Perform Simulated Annealing metaheuristic to search in solution neighborhood"
  [current-solution best-so-far pheromones tabu-list tries-cnt current-temp]
  ;; If current temperature reached below final temperature
  (if (or (<= current-temp *final-temperature*)
        (termination-criteria-reached? best-so-far))
      ;; return the best-so-far solution and the pheromones updated
      [best-so-far pheromones]
      ;; otherwise perform a search in the neighborhood of current-solution
      (let [new-solution (search/search-neighborhood current-solution operations)
            max-tries-reached? (= tries-cnt *sa-max-tries*)]
        ;; If new-solution is not feasible or tabu-list contains new-solution
        (if (or (not (solution/feasible? new-solution))
                (.contains tabu-list new-solution))
            ;; update tries-cnt and current-temperature if necessary, then recur to try
            ⇒ again
            (recur current-solution
                   best-so-far
                   pheromones
                   tabu-list
                   (if max-tries-reached? 0 (inc tries-cnt))
                   (if max-tries-reached? (* *lambda* current-temp) current-temp))
            ;; otherwise recur processing the solution found and updating other
            ⇒ parameters
            (let [delta (- (:cost new-solution) (:cost current-solution))
                  accept-as-current? (or (< delta 0)
                                          (< (random/rand)
                                              (Math/pow Math/E (- (/ delta
                                                                current-temp))))))
                  accept-as-best-so-far? (< (:cost new-solution) (:cost best-so-far))]
              (recur
               (if accept-as-current?
                 new-solution ; Accepts the new solution found
                 current-solution) ; Keep the same current solution
               (if accept-as-best-so-far?
                 new-solution ; Update the new solution found as best-so-far
                 best-so-far) ; Keep the same current best-so-far solution
               (if accept-as-best-so-far?
                 (matrix/add! pheromones
                              (pheromones/release-ant-pheromone new-solution 12))
                 pheromones)
               (into tabu-list '(new-solution)) ; Add new-solution to tabu-list
               ;; If maximum number of tries in the same temperature is reached, reset
               ⇒ tries-cnt to zero and decrease temperature level:
               (if max-tries-reached? 0 (inc tries-cnt))

```

```

        (if max-tries-reached? (* *lambda* current-temp) current-temp))))))

(defn improve-solution
  "Improve the current solution"
  [solution pheromones]
  (simulated-annealing solution ; Current solution
    solution ; Best-so-far solution
    pheromones ; Current pheromone trail
    (ring/ring-buffer *tabu-size*) ; Tabu list
    0 ; Iteration counter
    *initial-temperature*)) ; Set current-temp as initial-temp

```

C.10 Arquivo utils.clj

```

(ns hybrid-aco.utils
  (:require [random-seed.core :as random]))

; Taken from github.com/bodil/unpredictable. All credits to bodil.

(defn rand-from-seq
  "Produce a set of n distinct items from seq, chosen at random. Make sure there
  ↪ _are_ n distinct items in seq, or this function will loop forever."
  [n seq]
  (let [seq (vec seq) n (int n)]
    (loop [res #{}]
      (if (= n (count res)) res
        (let [item (random/rand-nth seq)]
          (recur (conj res item))))))

(defn rand-map
  "Map n random items from s through f, leaving other items unchanged."
  [n f s]
  (let [rands (rand-from-seq n (range (count s)))
        produce (fn produce [s index]
                    (lazy-seq
                     (if (not (seq s)) ()
                       (let [item (first s)
                             mapped (if (contains? rands index)
                                         (f item) item)]
                         (cons mapped (produce (rest s) (inc index)))))))]
    (produce s 0)))

```

APÊNDICE D – Códigos da Busca Híbrida em Grande Vizinhaça

D.1 Arquivo project.clj

```
(defproject parser "0.1.0-SNAPSHOT"
  :description "FIXME: write description"
  :url "http://example.com/FIXME"
  :license {:name "Eclipse Public License"
            :url "http://www.eclipse.org/legal/epl-v10.html"}
  :dependencies [[org.clojure/clojure "1.8.0"]
                 [uncomplicate/neanderthal "0.11.0"]
                 [random-seed "1.0.0"]
                 [cheshire "5.8.0"]]
  :main ^:skip-aot hybrid-lns.core
  :target-path "target/%s"
  :jvm-opts ["-Xmx1G"]
  :profiles {:uberjar {:aot :all}})
```

D.2 Arquivo core.clj

```
(ns hybrid-lns.core
  (:require [hybrid-lns.hybrid-lns :as lns]
             [clojure.java.io :as io]
             [cheshire.core :as json])
  (:gen-class))

(defn- to-keyword
  ""
  [string]
  (keyword (clojure.string/replace string "_" "-")))

(defn create-initial-solution
  "Defines an initial solution based using ACO"
  []
  (let [h (lns/build-heuristic-matrix)
        t (lns/build-pheromone-matrix 1000.0)]
    (lns/ant-colony {:routes [] :cost Integer/MAX_VALUE} 0 t h)))

(defn read-files
```

```

"Read all output files and return its data"
[files]
(map #(json/parse-string (slurp %) to-keyword) files))

(defn run-benchmark
  "Run the algorithm on each instance files"
  [paramfile file-dir]
  (let [files (file-seq (io/file file-dir))
        instances (read-files (filter #(.endsWith (str %) ".json") files))
        params (json/parse-string (slurp paramfile) to-keyword)]
    (println "----- USING PARAMETERS FROM" paramfile " -----")
    (lns/set-parameters params)
    (doseq [instance instances]
      (println "----- INSTANCE" (:name instance) "-----")
      (lns/set-instance instance)
      (lns/lns))))

(defn -main
  "Perfrom the algorithm on the instance file or on each instance files of the
  ↪ specified folder"
  [& args]
  (println args)
  (if (= (count args) 2)
    (let [params (first args)
          files (second args)]
      (run-benchmark params files))
    (println "Wrong number of arguments")))

```

D.3 Arquivo hybrid_lns.clj

```

(ns hybrid-lns.hybrid-lns
  (:require [cheshire.core :as json]
            [clojure.java.io :as io]
            [clojure.string :as string])
  (:refer-clojure :exclude [rand rand-int rand-nth])
  (:use [uncomplicate.neanderthal core native]
        [random-seed.core]))

;; Hybrid large neighbourhood search algorithm for capacitated vehicle routing
  ↪ problem
;; Author: Akpınar [2016]

(def aco-initial-solution {:routes [] :cost Integer/MAX_VALUE})

;; Neanderthal conversions
(defn to-neanderthal-matrix

```

```

"Converts a Clojure list of lists to a neanderthal matrix"
[matrix]
(let [n (count matrix)
      values (flatten matrix)]
  (dge n n values {:order :row})))

;; Data structures
; A route consists in its matrix representation
(defstruct Route :matrix :cost :tour)
(defstruct Solution :routes :cost)

;; Set global constants
(defn set-instance
  "Set the instance to be solved"
  [instance]
  (def INSTANCE instance)
  (def N (:dimension instance)) ; Number of customers
  (def K (:vehicles instance)) ; Number of vehicles available
  (def Q (:capacity instance)) ; Max capacity of a vehicle
  (def DIST (to-neanderthal-matrix (:distances instance))) ; Distance matrix
  (def DEMS (map second (:demand-section instance))) ; Demand of each customer
  (def NUM-EVALUATIONS (atom 0))
  (def WR-Q (+ (rand-int (min 96 (* 0.4 (- N 4)))) 4))
  (def OPTIMAL (:cost (:optimal INSTANCE) (:best-known INSTANCE 0)))

(defn set-parameters
  "Set the algorithm parameters"
  [params]
  (def PARAMS params)
  (println "Maximum number of tries =" @(def LNS-MAX-ITER (:max-iter params)))
  (println "Seed =" @(def SEED (:seed params)))
  ; Worst removal parameters
  (println "Worst-removal P =" @(def WR-P (:wr-p params)))
  ; ACO parameters
  (println "r1 ACO =" @(def ACO-R1 (:aco-r1 params)))
  (println "r2 ACO =" @(def ACO-R2 (:aco-r2 params)))
  (println "r3 ACO =" @(def ACO-R3 (:aco-r3 params)))
  (def ACO-R1-R2 (+ ACO-R1 ACO-R2))
  (println "alpha ACO =" @(def ACO-ALPHA (:aco-alpha params)))
  (println "beta ACO =" @(def ACO-BETA (:aco-beta params)))
  (println "psi ACO =" @(def ACO-PSI (:aco-psi params)))
  (println "P ACO =" @(def ACO-P (- 1 (:aco-p params)))) ; Evaporation coefficient
  (println "Max iterations ACO =" @(def ACO-MAX-ITER (:aco-iter params)))
  (println "Number of ants ACO =" @(def ACO-NUM-ANTS (:aco-ants params)))
  (set-random-seed! SEED))

;; Solution utils

```

```

(defn feasible?
  "Returns if a solution is feasible or not"
  [sol]
  (let [routes (:routes sol)]
    (every? true? ; The total demand of each route does not exceed
      (for [r routes] ; the vehicle capacity
        (<= (reduce + (map #(nth DEMS %) (:tour r))) Q))))))

(defn total-cost
  "Returns the cost of a solution"
  [sol]
  (reduce + (map :cost sol)))

(defn build-solution
  "Return the solution and calculate its cost"
  [routes]
  (struct Solution routes (total-cost routes)))

;; Routes utils
(defn route-cost
  "Returns the cost of the route x"
  [x]
  (swap! NUM-EVALUATIONS inc)
  (reduce + (for [i (range (mrows x))]
    (dot (row x i)
      (row DIST i))))))

(defn empty-route
  "Returns an empty route"
  []
  (struct Route (dge N N) 0 []))

(defn build-route
  "Build the route and calculate its cost"
  [r]
  (if (empty? r)
    (empty-route)
    (let [path (map vector r (rest r))
          x (dge N N)]
      (entry! x 0 (first r) 1)
      (doseq [[i, j] path]
        (entry! x i j 1))
      (entry! x (last r) 0 1)
      (struct Route x (route-cost x) r))))

(defn copy-route
  "Create a new route x' based on a route x"

```

```

[x]
(struct Route (dge (:matrix x)) (:cost x) (:tour x)))

(defn copy-solution
  "Create a new solution s' based on a solution s"
  [s]
  (struct Solution (vec (for [x (:routes s)] (copy-route x))) (:cost s)))

(defn empty-solution
  "Returns an empty solution"
  []
  (vec (repeat K [])))

(defn finished?
  "True if there is any customer available, or false otherwise"
  [cust]
  (= (int (sum cust)) (dec (dim cust))))

(defn choose-customer
  "Chooses randomly a customer and return it"
  [cust]
  (if (finished? cust)
    -1
    (let [chosen-cust (rand-nth (range 1 (dim cust)))]
      (if (= (entry cust chosen-cust) 0.0)
        chosen-cust
        (recur cust)))))

;; Generate initial solution randomly
(defn invalid-route?
  "Check if 'r' is invalid"
  [r]
  (every? zero? r))

(defn- split-routes
  "Split the tour 't' into routes"
  [t]
  (->> t
    (partition-by zero?)
    (remove invalid-route?)
    vec))

(defn generate-random-tour
  ""
  [customers tour load-so-far]
  (if (empty? customers)
    tour

```

```

    (let [cust (rand-nth customers)
          new-load (+ (nth DEMS cust) load-so-far)]
      (if (<= new-load Q)
        (recur (remove #{cust} customers)
                (conj tour cust)
                new-load)
        (recur (remove #{cust} customers)
                (conj tour 0 cust)
                (nth DEMS cust))))))

(defn generate-random-solution
  "Generate a random solution"
  []
  (let [tour (generate-random-tour (range 1 N) [] 0)
        routes (split-routes tour)]
    (build-solution (map build-route routes))))

;; Generate initial solution using Savings Heuristic
(defn merge-routes
  "Merge two routes on vertices i,j or return nil if the two routes cant be merged"
  [route1 route2 i j]
  (let [a (:tour route1)
        b (:tour route2)
        la (:load route1)
        lb (:load route2)]
    (cond (and (= i (last a)) (= j (first b)))
          { :tour (concat a b) :load (+ la lb) }
          (and (= i (first a)) (= j (first b)))
          { :tour (concat (reverse b) a) :load (+ la lb) }
          (and (= i (last a)) (= j (last b)))
          { :tour (concat a (reverse b)) :load (+ la lb) }
          (and (= i (first a)) (= j (last b)))
          { :tour (concat b a) :load (+ la lb) })))

(defn enough-capacity?
  "Check whether the merge violates the capacity constrain of the vehicle"
  [a b]
  (<= (+ (:load a) (:load b)) Q))

(defn customer-route
  "Return the route which serves the customer i"
  [routes i]
  (let [r (first routes)
        t (:tour r)]
    (if (some #(= % i) t)
      r
      (recur (rest routes) i))))

```

```

(defn remove-routes
  "Remove the routes r1 and r2 from the list of routes"
  [routes r1 r2]
  (remove #(= % r2)
    (remove #(= % r1) routes)))

(defn calculate-savings
  "Calculate the Savings value between each pair of nodes"
  []
  (flatten
    (for [i (range 1 N)]
      (for [j (range 1 N) :when (< i j)]
        {:from i :to j :saving (- (+ (entry DIST i 0)
                                      (entry DIST 0 j))
                                   (entry DIST i j))}))))))

(defn generate-initial-solution
  ""
  [savings routes]
  (if (empty? savings)
    routes
    (let [saving (first savings)
          i (:from saving)
          j (:to saving)
          ri (customer-route routes i)
          rj (customer-route routes j)
          merge (merge-routes ri rj i j)]
      ;(println "Merging" ri "+" rj "=" merge)
      (if (and (not= ri rj) (enough-capacity? ri rj) merge)
        (recur (rest savings)
          (conj (remove-routes routes ri rj) merge))
        (recur (rest savings)
          routes)))))

(defn savings-heuristic
  ""
  []
  (let [savings (sort-by :saving > (calculate-savings))
        routes (for [i (range 1 N)] {:tour [i] :load (nth DEMS i)})]
    (build-solution
      (map build-route (map :tour (generate-initial-solution savings routes))))))

;; Removal heuristic
(defn remove-customer
  "Removes the customer i from the route x"
  [x i]

```

```

(let [nx (remove #(= % i) (:tour x))]
  (build-route nx)))

(defn routes-without-customers
  "Generates routes without each customer and calculates its cost"
  [s]
  (flatten (for [i (range (count s)) :let [x (nth s i)]]
    (for [c (:tour x)]
      (let [r (remove-customer x c)]
        { :route-id i
          :customer c
          :delta-cost (- (:cost x) (:cost r))
          :new-route r }))))))

(defn update-solution
  "Update the solution s to remove the customer chosen in worst-removal"
  [s r]
  (assoc s (:route-id r) (:new-route r)))

(defn remove-randomly
  "Removes randomly q worst costumers in the solution s to the removed bank rb"
  [s rb q]
  (if (= q 0) ; Returns the new solution (s) alongside with the
    [(build-solution s) rb] ; customers removed in the removed bank (rb)
    (let [l (sort-by :delta-cost > (routes-without-customers s))
          y (rand)
          i (int (* (Math/pow y WR-P) (count l)))
          c (nth l i)] ; Chosen customer to remove
      (recur (update-solution s c)
              (conj rb (:customer c))
              (dec q)))))

(defn worst-removal
  "Apply the Worst Removal heuristic to remove q customers from the solution"
  [s]
  (remove-randomly s [] WR-Q))

;; Insertion heuristic
(defn can-add?
  "Checks if the customer i can be added to the route x
  given q: max capacity of a vehicle and c: demands of customers"
  [i x]
  (let [i-demand (nth DEMS i)]
    (<= (+ i-demand (:cost x)) Q)))

(defn insert-at-pos
  "Insert the customer i in the position pos of the route x"

```

```

[x i pos]
(concat (take pos x) [i] (drop pos x)))

(defn insert-at-route
  "Insert the customer i on each position of the route x and returns the best pos"
  [x i]
  (let [new-routes (for [pos (range (inc (count x)))]
                      (insert-at-pos x i pos))]
    (first (sort-by :cost (map build-route new-routes)))))

(defn insert-at-best-route
  "Inserts the customer i in the first and second best possible route of the
  ⇒ solution"
  [s i]
  (let [best-pos (for [x s] ;:when (can-add? i x)
                     (insert-at-route (:tour x) i))
        best-routes (sort-by :cost best-pos)]
    [(first best-routes) (second best-routes)]))

(defn calculate-regrets
  "Calculates the reinsertion of each customers from the removed bank rc into its
  ⇒ best routes of the solution s"
  [rc s]
  (for [i rc] ; for each customer in the removed bank
    (let [[first-best second-best] (insert-at-best-route s i)
          delta (- (:cost second-best) (:cost first-best))]
      {:customer i :delta delta :route first-best})))

(defn add-to-solution
  "Adds the new route nx with the inserted customer ni to the solution s"
  [s ni nx]
  (let [old-x (remove #(= % ni) (:tour nx))]
    (cons nx (remove #(= (:tour %) old-x) s))))

(defn regret-2-insertion
  "Constructive heuristic to reinsert the customers rc removed from Worst Removal
  ⇒ into the solution s"
  [rc s]
  (if (empty? rc)
    (build-solution s)
    (let [regrets (calculate-regrets rc s)
          max-regret (first (sort-by :delta > regrets))
          chosen-cust (:customer max-regret)
          chosen-route (:route max-regret)]
      (recur (remove #(= % chosen-cust) rc)
              (add-to-solution s chosen-cust chosen-route)))))

```

```

;; Ant Colony Optimization
(defn build-heuristic-matrix
  "Build the heuristic matrix using the formula  $S_{ij} = D_{i0} + D_{0j} - D_{ij}$ "
  []
  (let [s (dge N N)]
    (doseq [i (range N)]
      (doseq [j (range N)]
        (if (= i j)
          (entry! s i j 0)
          (entry! s i j (+ (entry DIST i 0)
                           (- (entry DIST 0 j)
                              (entry DIST i j)))))))
    s))

(defn build-pheromone-matrix
  "Build the initial pheromone trail matrix"
  [total-dist]
  (let [v (repeat (* N N) total-dist)]
    (dge N N v)))

(defn cross?
  "True if any of the solutions cross the path (i, j), nil otherwise"
  [sol i j]
  (some #(= (entry % i j) 1.0) sol))

(defn release-pheromone
  "Update the pheromone matrix"
  [t-matrix sol]
  (let [cost (/ 1000 (:cost sol))
        routes (map :matrix (:routes sol))]
    (doseq [i (range N)]
      (doseq [j (range N)]
        (let [deposit-amount (if (cross? routes i j) cost 0)
              pheromone (entry t-matrix i j)]
          (entry! t-matrix i j (+ pheromone deposit-amount))))))
    t-matrix))

(defn evaporate
  "Evaporate the pheromone matrix based on the parameter P"
  [t-matrix]
  (doseq [i (range N)]
    (doseq [j (range N)]
      (let [pheromone (entry t-matrix i j)]
        (entry! t-matrix i j (* ACO-P pheromone))))))
    t-matrix)

(defn exploitation

```

```

"Defines the next customer based on exploitation, given a: available customers, i:
↪ last customer added, t: the pheromone trail matrix, h: heuristic matrix"
[a i t-matrix h-matrix]
(let [exps (for [j a] {:cust j :val (* (Math/pow (entry t-matrix i j) ACO-ALPHA)
                                         (Math/pow (entry h-matrix i j)
                                         ACO-BETA))})]
↪ ACO-BETA)))]
(:cust (apply max-key :val exps))))

(defn build-probabilities
  "Builds the probabilities of each possible customer to be inserted"
  [a i t-matrix h-matrix]
  (let [x (reduce + (for [j a] (* (Math/pow (entry t-matrix i j) ACO-ALPHA)
                                   (Math/pow (entry h-matrix i j) ACO-BETA))))
        probs (for [j a] {:cust j :prob (/ (* (Math/pow (entry t-matrix i j)
                                   ACO-ALPHA)
                                   (Math/pow (entry h-matrix i j)
                                   ACO-BETA))
                                   x))}]
↪ ACO-ALPHA)
↪ ACO-BETA))
    probs))

(defn build-roulette
  "Build a roulette based on the probabilities of each customer"
  [probs r acc]
  (if (empty? probs)
    (let []
      r)
    (let [x (first probs)
          cust (:cust x)
          prob (:prob x)
          p (+ acc prob)]
      (recur (rest probs)
              (conj r {:cust cust :val p})
              p))))

(defn spin-roulette
  "Spin the roulette and choose the next customer"
  [r drawn-num]
  (if (= (count r) 1)
    (:cust (first r))
    (let [cur (first r)]
      (if (> (:val cur) drawn-num)
        (:cust cur)
        (recur (rest r) drawn-num)))))

(defn biased-exploration
  "Defines the next customer based on biased exploration, given a: available
↪ customers, "

```

```

[a i t-matrix h-matrix]
(let [probs (build-probabilities a i t-matrix h-matrix)
      roulette (build-roulette probs [] 0)
      p (rand)]
  (spin-roulette roulette p)))

(defn random-selection
  "Defines the next customer randomly"
  [a]
  (rand-nth a))

(defn last-customer
  "Returns the last customer of a route"
  [x]
  (if (empty? x) 0 (last x)))

(defn add-next-customer
  "Select the next customer to be added to the route x"
  [x cust t-matrix h-matrix]
  (let [i (last-customer x) ; Last customer added to the route
        r (rand)
        j (cond (<= r ACO-R1)
                  (exploitation cust i t-matrix h-matrix)
                  (and (< ACO-R1 r) (<= r ACO-R1-R2) (not= i 0))
                  (biased-exploration cust i t-matrix h-matrix)
                  :else
                  (random-selection cust))]
    (conj x j)))

(defn generate-solution
  "Inserts the customers into the route i based on ACO"
  [routes cust i t-matrix h-matrix]
  (if (empty? cust)
      (build-solution (map build-route routes))
      (let [x (nth routes i)
            new-x (add-next-customer x cust t-matrix h-matrix)]
        (recur (assoc routes i new-x)
                (remove #(= % (last-customer new-x)) cust)
                (mod (inc i) K)
                t-matrix
                h-matrix)))))

(defn- good-solution?
  "Compare the tour length of the incumbent solution with the best solution so far"
  [s sbest]
  (let [max-length (* (apply max (mapv (comp count :tour) (:routes sbest)))
                      ACO-PSI)]
    <→ ACO-PSI)])

```

```

    (every? #(< % max-length) (map (comp count :tour) (:routes s))))))

(defn ants-walk
  "Create a solution for each ant on the colony"
  [best-s i t-matrix t-matrix1 h-matrix]
  (if (== i ACO-NUM-ANTS)
    [best-s t-matrix1]
    (let [s (generate-solution (empty-solution) (range 1 N) 0 t-matrix h-matrix)
          best (if (and (feasible? s)
                        (< (:cost s) (:cost best-s))
                        (good-solution? s best-s))
                  s
                  best-s)
          t (release-pheromone t-matrix1 s)]
      (recur best (inc i) t-matrix t h-matrix))))

(defn ant-colony
  "Perform Ant Colony Optimization to construct a new solution"
  [best-s i t-matrix h-matrix]
  (if (== i ACO-MAX-ITER)
    best-s
    (let [t-matrix1 (dge t-matrix) ; Copy the pheromone matrix
          [best t] (ants-walk best-s 0 t-matrix t-matrix1 h-matrix)
          new-t (evaporate t)]
      (recur best (inc i) new-t h-matrix))))

;; Large Neighborhood Search
(defn start
  "Performs a Large Neighborhood Search on the solution s"
  [sbest i h]
  (if (or (== i LNS-MAX-ITER)
          (== (:cost sbest) OPTIMAL))
    sbest
    (let [s1 (copy-solution sbest)
          [s2 rb] (worst-removal (:routes s1))
          s3 (regret-2-insertion rb (:routes s2))
          new-best (if (and (feasible? s3) (< (:cost s3) (:cost sbest)))
                      s3
                      (let [t (build-pheromone-matrix (:cost sbest))
                            s4 (ant-colony sbest 0 t h)]
                        (if (< (:cost s4) (:cost sbest))
                          s4
                          sbest)))))]
      (println "Iteration" i "Best" (map :tour (:routes sbest)) "Cost" (:cost
→ sbest))
      (recur new-best (inc i) h))))

```

```

(defn lns
  "Search for an optimal solution using Hybrid Large Neighborhood Search"
  []
  (let [fmt-file (java.text.SimpleDateFormat. "ddMMyy-HHmmss")
        fmt-date (java.text.SimpleDateFormat. "dd-MM-yyyy HH:mm:ss")
        date-start (java.util.Date.)
        init (System/nanoTime)
        s (generate-random-solution)
        h (build-heuristic-matrix)
        sol (start s 0 h)
        end (System/nanoTime)
        date-end (java.util.Date.)
        file-output-name (str "./out/" (:name INSTANCE) "-" (.format fmt-file
⇒  date-start) ".out")]
    (io/make-parents file-output-name)
    (spit file-output-name
      (json/generate-string {:instance (:name INSTANCE)
                             :parameters (assoc PARAMS :wr-q WR-Q)
                             :start (.format fmt-date date-start)
                             :end (.format fmt-date date-end)
                             :initial-solution {:routes (map :tour (:routes s))
                                                  :cost (:cost s)}
                             :solution {:routes (map :tour (:routes sol))
                                          :cost (:cost sol)}
                             :elapsed-time (/ (- end init) 1e9)
                             :evaluations @NUM-EVALUATIONS
                             :optimal OPTIMAL
                             :dev (* (/ (- (:cost sol) OPTIMAL) OPTIMAL) 100)}
                             {:pretty true
                              :key-fn #(string/replace (name %) #"- " "_")}}))))

```