UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Hugo Gustavo Valin Oliveira da Cunha

Algoritmo Genético e Algoritmo de Vaga-lumes aplicados ao Problema do Caixeiro Viajante

Uberlândia, Brasil

2019

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Hugo Gustavo Valin Oliveira da Cunha

Algoritmo Genético e Algoritmo de Vaga-lumes aplicados ao Problema do Caixeiro Viajante

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

Orientador: Dr^{a}. Christiane Regina Soares Brasil

Universidade Federal de Uberlândia – UFU

Faculdade de Computação

Bacharelado em Ciência da Computação

Uberlândia, Brasil

2019

Hugo Gustavo Valin Oliveira da Cunha

Algoritmo Genético e Algoritmo de Vaga-lumes aplicados ao Problema do Caixeiro Viajante

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

Drª. Christiane Regina Soares Brasil

Dr. Lásaro Jonas Camargos

Dr. Wendel Alexandre Xavier de Melo

Uberlândia, Brasil
2019

Dedico esse trabalho ao Supremo Criador, O Rei da Eternidade, Jeová Deus.
(Isaías 40:26; 42:8)

Agradecimentos

Agradeço a primeiramente a Jeová Deus, por estar ao meu lado, ter me apoiado e consolado, e em especial, nos momentos mais difíceis da minha vida. Se consegui chegar até aqui, e ser o que sou, nunca foi por força própria, mas pela força ativa de Jeová Deus, seu espírito santo.

Agradeço a minha mãe, Cleusdete Maria de Oliveira, que sempre me amou, me apoiou, e se esforçou tanto para me dar uma vida digna e a oportunidade de estudar que ela não pode ter. Espero que possa lhe retribuir todo carinho e amor que ela tem por mim.

À Profª. Drª. Christiane Regina Soares Brasil pela orientação na condução deste trabalho, pela confiança, conselhos, motivação, compreensão e parceria.

Aos professores do Curso de Bacharelado em Ciência da Computação da Universidade Federal de Uberlândia (UFU) por partilharem experiências e conhecimentos.

A Universidade Federal de Uberlândia, seu corpo docente, direção e administração que proporcionaram um novo caminho, baseado em confiança, mérito e ética, qualidades estas presentes na instituição.

"Eu, Jeová, sou o seu Deus, Aquele que ensina o que é melhor para você, Aquele que o guia no caminho em que deve andar." - Isaías 48:17

Resumo

Os algoritmos de otimização computacional são amplamente aplicados em diversas áreas, computacionais e do mundo real, para encontrar soluções ou aproximações das mesmas de problemas complexos. Os algoritmos de otimização estudados neste trabalho são: o Algoritmo Genético (AG) e o Algoritmo de Vaga-lumes (AVL). Esses algoritmos são bio-inspirados, isto é, se baseiam em processos que ocorrem na natureza. Ambos os métodos foram utilizados para encontrar uma solução ótima, ou aproximação, para o Problema do Caixeiro Viajante (PCV), caracterizado como um problema NP por sua alta complexidade. O PCV é definido como um problema de busca de um percurso em um grafo, partindo-se de um vértice inicial e visitando cada um dos outros vértices uma única vez e retornando ao ponto de partida. Portanto, este trabalho teve como objetivo principal um estudo do AG e AVL usando o PCV como problema alvo por sua complexidade e sua facilidade em relacioná-lo com diversos problemas do mundo real. O estudo foi realizado por meio do desenvolvimento dos métodos e da execução de experimentos com instâncias obtidas da biblioteca TSPLIB.

Palavras-chave: Otimização, Algoritmo Genético, Algoritmo de Vaga-lumes, Problema do Caixeiro Viajante.

Lista de ilustrações

Figura 1 – Exemplo de *Crossover* de 1-ponto. 16
Figura 2 – Exemplo de Mutação. 16
Figura 3 – Fluxograma do Algoritmo Genético. 17
Figura 4 – Fluxograma do Algoritmo AVL. 20
Figura 5 – Um exemplo de circuito hamiltoniano no grafo. 21
Figura 6 – Um exemplo de rota no mapa. 22
Figura 7 – Relação entre melhor brilho e média do brilho ao longo das iterações para a variação do número de iterações. 50
Figura 8 – Relação entre melhor brilho e média do brilho ao longo das iterações para a variação do valor de alfa. 51
Figura 9 – Relação entre melhor brilho e média do brilho ao longo das iterações da melhor execução de ajuste para o parâmetro gama. 52
Figura 10 – Relação entre melhor brilho e média do brilho ao longo das iterações da melhor execução de ajuste para o passo alfa. 53

Lista de tabelas

Tabela 1 – Teste do Algoritmo Genético para a biblioteca TSPLIB. 48
Tabela 2 – Teste do Algoritmo de Vaga-lumes para a biblioteca TSPLIB. 49
Tabela 3 – Teste da instância eil51 variando o número de iterações. 50
Tabela 4 – Teste da instância eil51 variando o valor de alfa. 51
Tabela 5 – Teste da instância eil51 variando o valor de gama. 51
Tabela 6 – Teste do Algoritmo de Vaga-lumes com passo alfa alternado. 52

Lista de abreviaturas e siglas

ABC	*Artificial Bee Colony*
ACO	*Ant Colony Optimization*
AG	Algoritmos Genéticos
AVL	Algoritmo de Vaga-lumes
GRASP	*Greedy Randomized Adaptive Search Procedure*
PCV	Problema do Caixeiro Viajante
PCVA	Problema do Caixeiro Viajante Aleatório
PMCV	Problema de Múltiplos Caixeiros Viajantes
PSO	*Particle Swarm Optimization*
SA	*Simulated Annealing*

Sumário

1 INTRODUÇÃO . . . 11
1.1 Objetivo . . . 12
1.2 Justificativa . . . 12
1.3 Organização da monografia . . . 12

2 ALGORITMOS DE OTIMIZAÇÃO . . . 14
2.1 Algoritmos Genéticos . . . 14
2.2 Algoritmo de Vaga-lumes (AVL) . . . 17

3 PROBLEMA DO CAIXEIRO VIAJANTE . . . 21
3.1 Algoritmo Genético para o Problema do Caixeiro Viajante . . . 22
3.2 Algoritmo de Vaga-lumes para o Problema do Caixeiro Viajante . . . 23
3.3 Trabalhos Correlatos . . . 24

4 IMPLEMENTAÇÃO DO TRABALHO . . . 27
4.1 Implementação do método AG para o PCV . . . 27
4.1.1 Descrição do método . . . 27
4.1.2 Representação do indivíduo . . . 27
4.1.3 Visão geral do algoritmo . . . 27
4.1.4 Pseudocódigo . . . 28
4.2 Implementação do método AVL para o PCV . . . 36
4.2.1 Descrição do método . . . 37
4.2.2 Representação do vaga-lume . . . 37
4.2.3 Visão geral do algoritmo . . . 37
4.2.4 Pseudocódigo . . . 39

5 RESULTADO E DISCUSSÕES . . . 48
5.1 Algoritmo Genético . . . 48
5.2 Algoritmo de Vaga-lumes . . . 49
5.2.1 Reajuste de parâmetros para Algoritmo de Vaga-lumes . . . 49

6 CONCLUSÃO . . . 54

REFERÊNCIAS . . . 55

1 Introdução

A Computação Bioinspirada é uma área de pesquisa em que elementos da natureza são utilizados como fonte de inspiração para a criação e o desenvolvimento de diversos algoritmos que buscam soluções ou aproximações para inúmeros problemas complexos, computacionais ou do mundo real. Com o recente entusiasmo e seu avanço imensurável, a Computação Bioinspirada está presente nas formas mais variadas da atualidade, como máquinas de lavar roupa, brinquedos, dispositivos de filmes, até sistemas financeiros e ferroviários, por exemplo (NGUYEN; VU; DAI, 2018). As razões para esse crescimento deve-se: (1) ao desenvolvimento de algoritmos de otimização computacional capazes de resolver problemas complexos, que dificilmente são resolvidos por técnicas determinísticas clássicas; (2) à relativa simplicidade de seus métodos baseados na biologia; (3) à sua adequação e à aplicação nas mais diversas áreas do conhecimento (DELBEM; GABRIEL, 2008).

Um dos estudos mais importantes da área são os Algoritmos Genéticos, que foram desenvolvidos durante as décadas de 60 e 70 (HOLLAND, 1975). Fundamentado nas pesquisas de Darwin sobre sistemas naturais e sua capacidade de adaptação (DARWIN, 1859; GOLDBERG, 1989), David E. Goldberg verificou que tal processo poderia ser incorporado e simulado matematicamente em um algoritmo de busca, que posteriormente foi chamado de Algoritmo Genético (RAMOS, 2001). Mesmo com sua simplicidade, fortes aproximações numéricas têm sido encontradas em problemas difíceis de serem resolvidos.

Além dos Algoritmos Genéticos, existe uma diversidade de outras meta-heurísticas, dentre as quais pode-se citar o ABC (*Artificial Bee Colony* - Colônia de Abelhas Artificial)[1], o ACO (*Ant Colony Optimization* - Otimização por Colônia de Formigas)[2] e, entre tantas outras, pode-se destacar o Algoritmo de Vaga-lumes (AVL), introduzido por Yang (2009). O AVL é um algoritmo estocástico que se baseia nos mecanismos de comunicação entre insetos, mais especificamente, vaga-lumes. Por sua característica randômica, o processo permite que a busca evite problemas de soluções locais e platôs (uma área plana da topologia de espaço de estados) (ARBOLEDA, 2012). Os blocos de construção são agrupados com diferentes soluções por meio de cruzamento, permitindo que soluções promissoras se mantenham em cada etapa do processo. Ademais, o ajuste de parâmetros permitidos pelo algoritmo é a oportunidade de se reter boas soluções e equilibrar a exploração, bem como a direção no espaço de busca (FISTER et al., 2013).

[1] *Artificial Bee Colony* (ABC) é uma meta-heurística de otimização bioinspirada no comportamento social de colônias de abelhas durante a coleta de alimento (ARBOLEDA, 2012).

[2] *Ant Colony Optimization* (ACO) é um algoritmo de otimização cuja a ideia básica é simular o comportamento de certas espécies de formigas, que depositam nos caminhos por elas trilhados um feromônio, possibilitando o reforço nos caminhos que provavelmente são os melhores(DORIGO; STüZLE, 2004).

O PCV é uma questão matemática muito importante, não só para o meio acadêmico, mas também para problemas do mundo real que tem sua representação em grafos, como manufatura, sistemas de transportes e comunicação de dados, dentre outros (RAMOS, 2001). O problema, apesar de simples, tem perdurado por longo tempo e, portanto, parmance sendo um alvo de estudos. O PCV é definido como um problema de busca de um percurso em um grafo, partindo-se de vértice inicial e visitando cada um dos outros vértices e retornando ao ponto de partida, sem que nenhum vértice se repita (NILSSON, 1982). Por ser um problema de otimização combinatória, o PCV é classificado como um problema NP-Completo. Deste modo, não se pode determinar sua solução ótima em um tempo razoável (LINDEN, 2008).

1.1 Objetivo

Este trabalho de Conclusão de Curso tem como objetivo principal realizar um estudo de dois métodos de otimização computacional, Algoritmo Genético e Algoritmo de Vaga-lumes, aplicados ao clássico Problema do Caixeiro Viajante, especialmente o segundo que foi criando originalmente para problemas contínuos. Para tal, ambos os métodos foram desenvolvidos e executados para validação e análise de resultados.

1.2 Justificativa

O Problema do Caixeiro Viajante é um dos principais problemas de otimização computacional que pode ser facilmente adaptado para problemas combinatórios do mundo real, como fabricação de placas de circuitos eletrônicos, sequenciamento de tarefas, serialização em arqueologia, roteamento IP, tráfego urbano, distribuição de rede elétrica e entre tantos outros. Diante de sua importância e sua complexidade computacional, tal problema é uma excelente aplicação para os algoritmos genéticos e o algoritmo de vaga-lumes em busca de uma solução ótima ou aproximação da mesma. Tais algoritmos representam exemplos importantes da área de otimização computacional, sendo o Algoritmo Genético uma abordagem bastante conhecida na literatura, enquanto que o Algoritmo de Vaga-lume é um método mais recente em que há uma quantidade de trabalhos significativamente menor que o AG, mas que vem apresentando bons resultados nos artigos encontrados.

1.3 Organização da monografia

No Capítulo 2 há a descrição dos principais conceitos dos dois algoritmos de otimização: Algoritmo Genético e Algoritmo de Vaga-lume. No Capítulo 3 está a explicação do Problema do Caixeiro Viajante e a apresentação de alguns trabalhos relacionados. No

Capítulo 4 tem-se a descrição da implementação dos métodos de estudo. No Capítulo 5 há a apresentação dos resultados obtidos e breves análises dos mesmos. No Capítulo 6, é apresentada a conclusão do estudo dos algoritmos desenvolvidos durante este trabalho.

2 Algoritmos de Otimização

2.1 Algoritmos Genéticos

Os Algoritmos Genéticos (AGs) são métodos de busca e otimização inspirados na biologia evolucionária da população de seres vivos (DARWIN, 1859; HOLLAND, 1975; GOLDBERG, 1989). Como uma técnica de otimização e busca, os algoritmos genéticos apresentam uma função objetivo (também chamada aptidão ou *fitness*), utilizada para avaliar cada uma de uma soluções obtidas, uma vez que o método gera um conjunto de soluções, e não apenas uma.

A primeira etapa de um típico AG é a geração de uma população inicial de indivíduos, cada qual representando uma possível solução do problema em questão. Durante o processo de evolução, cada indivíduo é avaliado de acordo com sua aptidão, ou seja, existem indivíduos mais ou menos aptos. Dentre estes, são selecionados alguns para sofrerem operações de reprodução, como cruzamento ou *crossover* gerando, deste modo, os possíveis descendentes para a nova população. Todos esses passos são repetidos até que o número máximo de iterações seja atingido ou até que a solução ótima seja encontrada quando esta é conhecida.

A inspiração dos algoritmos genéticos na Biologia fez com que certas expressões fossem herdadas desta ciência. Para um melhor entendimento, uma lista de termos específicos é apresentada com os seus respectivos significados(DELBEM; GABRIEL, 2008):

- População: é um conjunto de indivíduos, que são soluções codificadas para um determinado problema.
- Indivíduo: é um elemento da população formado por um conjunto de genes e pela aptidão, isto é, a qualidade da solução que ele representa.
- Gene: é um elemento do cromossomo do indivíduo.
- Cruzamento (*crossover*): é o processo de troca ou de combinação de genes entre dois ou mais indivíduos, obtendo novos indivíduos.
- Mutação: é a operação que modifica um ou mais genes de um cromossomo.
- Geração: é a população em um determinado instante da iteração do algoritmo genético durante a evolução.

Tendo conhecimento desses termos, pode-se discutir cada um dos passos deste algoritmo.

O Algoritmo 1 apresenta um típico Algoritmo Genético. Um AG processa sua população de indivíduos, onde cada um é codificado de uma forma que possa ser manipulado, em que, geralmente, é usado a representação binária ou real dependendo do problema. Para cada indivíduo é atribuída uma aptidão, isto é, uma informação de quão boa é sua solução.

A partir de sua população, o AG seleciona alguns indivíduos, para gerar novos filhos (ou variantes dos pais) por mutação e/ou *crossover*.

Algoritmo 1: Algoritmo Genético

```
Entrada: Tamanho da população e número de gerações
Saída: População de soluções
início
    Criação da população inicial;
    Avaliação de cada indivíduo da população inicial;
    repita
        Seleção dos pais;
        Cruzamento entre os pais;
        Mutação dos descentes gerados;
        Avaliação de novos candidatos;
        Seleção de indivíduos para nova geração;
    até Condição seja satisfeita;
fim
```

O processo de reprodução que é composto pelos processos de cruzamento e mutação é de extrema importância em problema de permutações (EIBEN; SMITH, 2003). No entanto, não basta apenas combinarmos soluções, mas também garantir que novas soluções sejam válidas no espaço de busca.

A operação de cruzamento é feita pela troca ou combinação de dois ou mais indivíduos no intuito de se obter novos indivíduos (Figura 1). Um tipo bastante comum é a combinação n-pontos, no qual se divide cromossomos em n partições que serão utilizadas para composição de novos indivíduos. Outra forma possível é a recombinação uniforme, que leva em consideração quais genes individualmente serão passados para os filhos. Para isso, foram desenvolvidos vários operadores conforme descrito a seguir (DELBEM; GABRIEL, 2008):

- *Crossover* OX (Baseado em ordem): selecionam-se dois pontos de corte que definem uma subsequência de genes que os filhos herdarão dos pais. O restante dos genes é obtido de maneira aleatória.
- *Crossover* PBX (Baseado em Posição): seleciona-se um conjunto de posições aleatórias de um dos pais que serão copiados para o filho, exatamente na mesma posição,

os outros genes que estarão presente no filho são herdados do outro pai, compondo assim o cromossomo.

- *Crossover* OBX (Baseado em Ordem): similar ao PBX, seleciona de forma aleatória os genes do pai 1, herdando genes do pai 2 de mesma posição. Depois disso, o processo seleciona os genes que estão faltando do pai 1.
- *Crossover* CX (Cíclico): o processo começa copiando o primeiro elemento do pai 1 para o filho 1 evitando repetições. Na etapa final, as posições que ficaram em branco são obtidas por troca de elementos entre os pais.
- *Crossover* PMX (Parcialmente Mapeado): semelhante ao OX, inicia-se com dois pontos de cortes que definem subsequências. Em seguida, realizam-se trocas no sentido do pai 1 ao pai 2, e logo após, em sentido inverso evitando repetições de sequências.

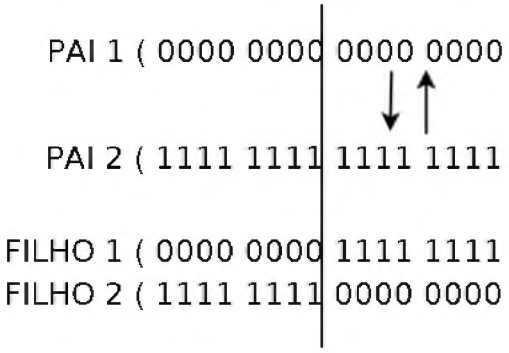

Figura 1 Exemplo de *Crossover* de 1-ponto.

Na mutação, um ou mais genes de um determinado indivíduo sofrem alterações. A desvantagem desses processos é que podem fazer com que os melhores cromossomos sejam perdidos (Figura 2). Neste sentido, o elitismo é responsável por manter a melhor solução (ou uma porcentagem das melhores soluções). Deste modo, acelera-se o processo de evolução em busca do ótimo global.

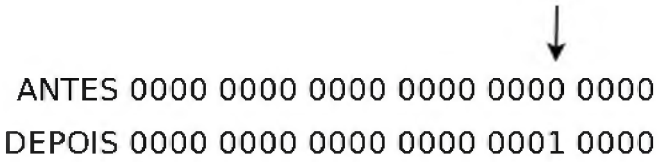

Figura 2 – Exemplo de Mutação.

O fluxograma do algoritmo genético é ilustrado na Figura 3:

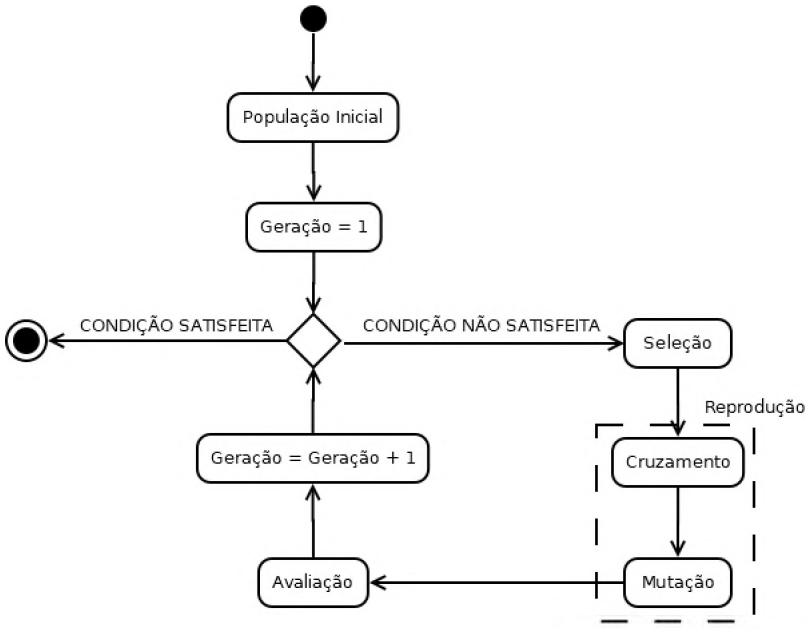

Figura 3 Fluxograma do Algoritmo Genético.

Vale ressaltar que em uma algoritmo genético o *crossover* e a mutação são dois mecanismos de busca que podem levar à exploração de pontos inteiramente novos do espaço de busca (*exploration*), enquanto a seleção dirige a busca em direção aos pontos do espaço de busca em profundidade (*exploitation*).

Portanto, o AG pode convergir para um mínimo ou máximo local no espaço comum. Pode-se destacar algumas vantagens na aplicação dos AGs, que são:

- Realizam buscas simultâneas em várias regiões do espaço de busca.
- Funcionam tanto com parâmetros contínuos como discretos.
- São simples de serem implementados em computadores.
- São flexíveis para otimizar e trabalhar com funções objetivos conflitantes (problemas multiobjetivos).

2.2 Algoritmo de Vaga-lumes (AVL)

Os vaga-lumes são insetos lampirídeos, isto é, insetos que produzem padrões de luzes únicos em cada espécie. A característica do lampejo como duração, ritmo e velocidade é determinante para a sua sobrevivência, uma vez que pode ser utilizado para o acasalamento ou para atrair uma possível presa (ARBOLEDA, 2012).

O matemático Xin-She Yang (YANG, 2009; YANG, 2010) introduziu um algoritmo de otimização bioinspirado nos vaga-lumes, baseado na comunicação entre os lampirídeos.

O algoritmo associa a função objetivo a ser otimizada com a intensidade dos lampejos produzidos (ARBOLEDA, 2012). A concepção do algoritmo assume algumas simplificações como: (a) os vaga-lumes são unissex, de modo que a atração entre eles independe do sexo; (b) o brilho de um vaga-lume é determinado pela função objetivo a ser otimizada; (c) a atratividade é inversamente proporcional ao quadrado da distância dos comunicantes e proporcional ao maior brilho produzido por um deles (KOIDE, 2016).

O funcionamento básico está ilustrado conforme o Algoritmo 2. A intensidade dos flashes de cada lampirídeo está associado a função objetivo F(X). O coeficiente I_0 representa a luminosidade inicial, γ representa absorção de luz no ambiente e α um parâmetro de busca global, β representa a atratividade dos vaga-lumes que é relativo a cada um e que varia com a distância $R_{i,j}$.

A seguir, serão apresentados mais detalhes sobre cada um dos parâmetros citados anteriormente (KOIDE, 2016). Com relação à intensidade luminosa I, sua forma de cálculo em relação à distância r pode ser dado pela Equação (2.1).

$$I(r) = I_0 e^{-\gamma r^2} \tag{2.1}$$

Como a atratividade é proporcional à intensidade dos vaga-lumes comunicantes, e ao tamanho da distância entre eles, sua equação pode ser dada por (2.2).

$$\beta(r) = \beta_0 e^{-\gamma r^2} \tag{2.2}$$

Algoritmo 2: Algoritmo de Vaga-lumes

```
Entrada: F(X), X=(X_1, X_2, ..., X_n), I_0, γ, α
Saída: O melhor vaga-lume X_g
início
    para i de 1 até n faça
        X_i ← soluçãoInicial();
    fim
    determinação do coeficiente de absorção γ;
    enquanto critério de fim não terminar faça
        para i de 1 até n faça
            para j de 1 até n faça
                se F(X_i) < F(X_j) então
                    R_{i,j} ← distância(X_i, X_j);
                    β ← atratividade(I_0, γ, R_{i,j});
                    X_j ← (1 − β)X_j + βX_i + α(rand − (0.5));
                fim
            fim
        fim
        X_min ← X_min + (rand − (0.5));
    fim
    retorna X_min
fim
```

Se a atratividade β for igual a 0, pode-se concluir que os vaga-lumes não vêem uns aos outros. Além disso, se β for igual a β_0 os vaga-lumes também veem com dificuldade.

A distância cartesiana $R_{i,j}$ a partir do vetor de posição X_i e X_j que representa as posições dos lampirídeos, pode ser definida pela Equação (2.3).

$$R_{i,j} = \|X_i - X_j\| = \sqrt{\sum_{k=1}^{t} (X_{i,k} - X_{j,k})^2} \tag{2.3}$$

O movimento do vaga-lume j é dado em função do brilho de outro vaga-lume i de maior brilho, se for um problema de minimização, é determinado pela Equação (2.4), em que o primeiro termo é a probabilidade da nova solução ser composta pelo vaga-lume X_i, segundo termo similarmente é a probabilidade da nova solução ser composta pelo vaga-lume X_j. O terceiro termo é formado por um número randômico α formando a parte aleatória do algoritmo.

$$X_j = (1 - \beta)X_j + \beta X_i + \alpha(rand - (0.5)) \tag{2.4}$$

O fluxograma do algoritmo de vaga-lumes é ilustrado na Figura 4.

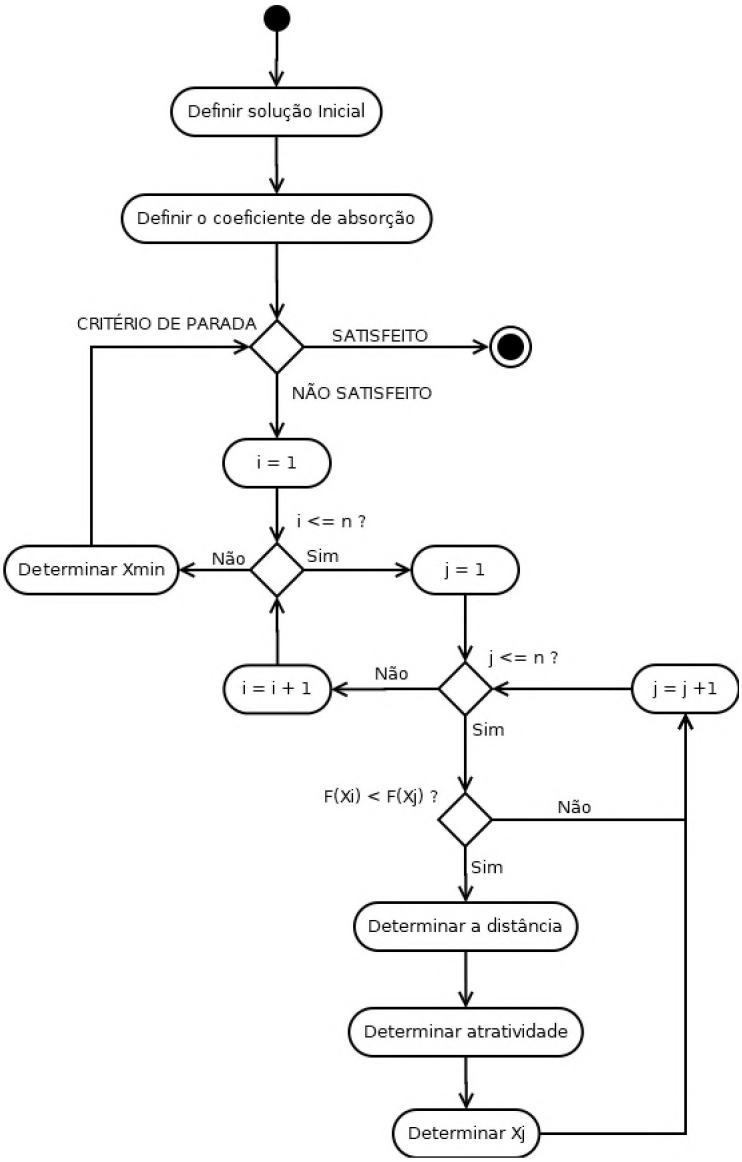

Figura 4 Fluxograma do Algoritmo AVL.

3 Problema do Caixeiro Viajante

O Problema do Caixeiro Viajante (PCV) é um problema combinatório fundamentalmente matemático que tem perdurado por longo tempo. Em meados de 1800, problemas relacionados ao PCV começaram a ser desenvolvidos, mas somente em 1950 ficou conhecido globalmente em sua forma geral (APPLEGATE et al., 2007).

Define-se o Problema do Caixeiro Viajante como a busca de um circuito em um grafo, em que se partindo de um vértice inicial visita-se cada um dos outros vértices uma única vez, voltando ao ponto de partida, de forma que a distância do circuito percorrido seja mínimo (NILSSON, 1982). Denomina-se este circuito como ciclo ou circuito hamiltoniano. A Figura 5 mostra um grafo com um possível circuito hamiltoniano.

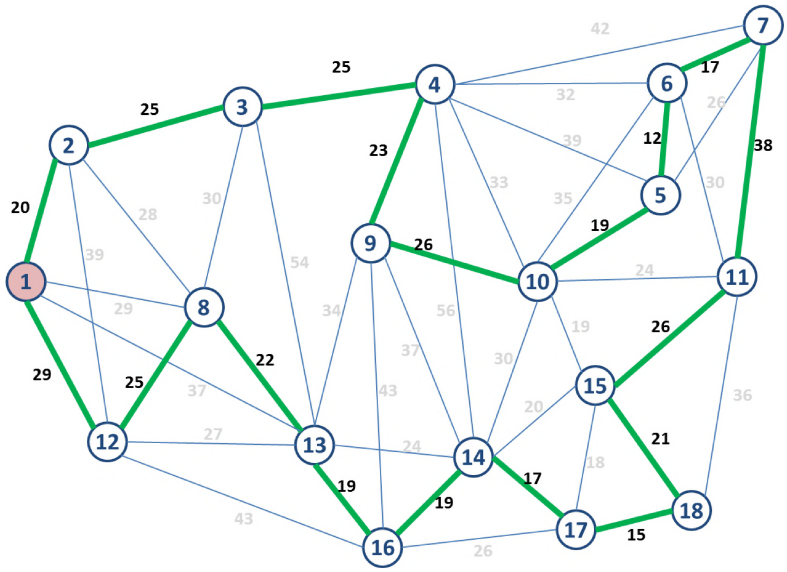

Figura 5 – Um exemplo de circuito hamiltoniano no grafo.

Considere o circuito a ser formado a partir de um grafo com n vértices. Partindo-se de determinado vértice, o próximo vértice a ser escolhido deve ser um dos n-1 vértices restantes do conjunto, dado que não haja repetição de vértice, exceto o primeiro e o último vértice caracterizando o circuito hamiltoniano. O seguinte a ser escolhido será um dos n-2 vértices ainda não escolhidos. Dessa forma, por um raciocínio indutivo, conclui-se que o conjunto de todas as rotas possíveis é descrita pela relação (3.1).

$$(n-1)(n-2)...(2)(1) = (n-1)! \tag{3.1}$$

Por exemplo, para um grafo de 30 vértices, o número de rotas será dado por 29!, isto é, 8.841.761.993.739.701.954.543.616.000.000 caminhos possíveis, o que caracteriza o pro-

blema como intratável computacionalmente. Por se tratar de um problema NP-Completo (SIPSER, 2010), o tempo de se encontrar a solução ótima pode ser inviável computacionalmente com métodos determinísticos (RAMOS, 2001). Portanto, existem diversas abordagens de otimização computacional para auxiliar na solução deste problema. A seguir, serão descritas duas abordagens aplicadas ao problema PCV: Algoritmo Genético e Algoritmo de Vaga-lumes.

3.1 Algoritmo Genético para o Problema do Caixeiro Viajante

Considere um grafo com n vértices, representado em uma matriz de distâncias ligando cada par de vértice. O objetivo do PCV é, partindo de um vértice de origem, visitar cada um dos outros vértices uma única vez e retornar ao vértice original, minimizando a distância total (RAMOS, 2001).

Dados os **n** vértices do grafo, as rotas possíveis podem ser representadas como uma permutação dos **n** vértices, isto é, um circuito ligando cada um desses pontos considerando que há uma ligação entre o último vértice e o primeiro de tal permutação. Por exemplo, a permutação [7,2,5,6,1,4,3] corresponde ao circuito da Figura 6.

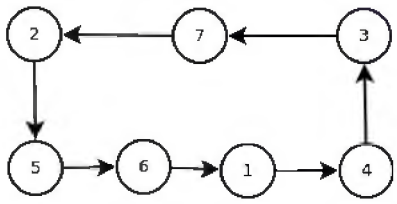

Figura 6 – Um exemplo de rota no mapa.

A representação de uma rota no algoritmo evolutivo pode ser representado de forma similar (RAMOS, 2001). O cromossomo é denotado por um vetor idêntico à permutação, ou seja, dada a rota ou permutação descrita anteriormente, seu cromossomo respectivo será [7,2,5,6,1,3,4].

Deste modo, utilizando essa representação, o PCV tem como espaço de busca o conjunto de todas as permutações possíveis, ou seja, para uma coleção de rotas, determina-se a melhor solução como o caminho de menor distância.

O processo de geração da população inicial de indivíduos no AG, para o PCV, pode ser abordado de várias formas, utilizando até mesmo um procedimento aleatório. Por exemplo, para a população inicial com **m** indivíduos, **m** vetores distintos de tamanho **n** são gerados por uma sequência de vértices que também são distintos entre si (RAMOS, 2001).

A função de avaliação a ser otimizada será a soma das distâncias $D_{i,j}$, ou seja, deve-se minimizar custos do percurso ou descobrir a menor rota possível. Logo, a função de avaliação f será dada por (3.2):

$$F = \sum_{i=1}^{n-1} D_{i,i+1} + D_{n,1} \tag{3.2}$$

O processo de reprodução consiste na seleção de rotas da população atual como melhor *fitness*, para a formação de novos indivíduos. Os operadores de reprodução são aqueles comuns aos AG, como *crossover* e mutação. Um exemplo do operador de *crossover* é o OX criado por Davis (1985) citado na seção 2.1, que impede que rotas inválidas sejam criadas no processo de geração, além de manter subsequência dos pais geradores, enquanto que o processo de mutação altera um ou mais genes do cromossomo gerado de acordo com algum operador. Por exemplo, atribui-se uma probabilidade a cada um dos genes presente no cromossomo, e utilizando uma função de probabilidades, selecionam-se dois genes para que seus alelos seja invertidos (RAMOS, 2001), como descrito na seção 2.1.

3.2 Algoritmo de Vaga-lumes para o Problema do Caixeiro Viajante

O algoritmo AVL foi concebido para lidar com variáveis contínuas. Contudo, pode-se estender para soluções discretas, aplicando em alguns problemas como o problema do caixeiro viajante (KOIDE, 2016).

No problema em questão é necessário redefinir algumas funções. Segundo Durkota (2011), considera-se o espaço de busca como todas as permutações possíveis de (1, 2, ..., n). Para as equações do movimento é definido pela Equação (3.3).

$$X_i = (1 - \beta)X_i + \beta X_j + (rand - 0.5)) \tag{3.3}$$

A solução inicial, isto é, a permutação (1, 2, ..., n) pode ser eleita de forma randômica. O cálculo da distância para o caso de variáveis discretas pode ser realizada com o cálculo da distância de Hamming, que é definida como o número de elementos que não se correspondem entre duas permutações de mesmo tamanho (KOIDE, 2016). Por exemplo, dado as permutações P_1 = [1, 2, 3, 4, 5] e P_2 = [1, 2, 3, 5, 4], a distância de Hamming (P_1, P_2) = 2.

No decorrer das iterações do algoritmo, as distâncias dos vaga-lumes em relação a outros vaga-lumes tendem a diminuir. Utilizando a distância de Hamming, as quantidades de elementos comuns devem aumentar para obterem-se as menores distâncias. Deste modo, o passo-β inicia com a análise dos elementos comuns entre as permutações (KOIDE, 2016). Para explicação, considere as seguintes permutações P_1 = [4, 9, 3, 7, 6, 8, 2, 1, 5] e P_2 =

[4, 1, 3, 2, 6, 5, 9, 7, 8]. Verifique que P_1 é a semelhante a P_2 nas seguintes posições: [4, _, 3, _, 6, _, _, _, _].

Os espaços vazios são completados em relação às permutações P_1 e P_2. Utilizando a probabilidade de $\beta(r)$, sendo r a distância de Hamming, complementa-se a permutação $P_1 \rightarrow P_2$) inserindo elementos de P_1 ou P_2, evitando a repetição de elementos. Contudo, se ainda assim houver espaços em branco, eles são preenchidos de maneira aleatória. Pode-se observar que o passo-β se assemelha ao processo de *crossover* nos algoritmos genéticos.

Para o passo-α utiliza-se um processo semelhante à mutação, que consiste em permutar dois elementos vizinhos. Essa permutação pode ser tanto pela escolha aleatória de alguns elementos, quanto pela troca de dois elementos adjacentes.

3.3 Trabalhos Correlatos

O Problema do Caixeiro Viajante é um dos problemas de otimização combinatória mais largamente estudado na área de Computação. Sua definição é simples, mas ainda continua sendo um dos problemas mais desafiadores e aplicáveis dos últimos anos, motivando a importantes desenvolvimentos teóricos. Lawler (1985) fornece uma pesquisa abrangente de todos os principais resultados de PCV com diversos métodos de otimização até aquele momento.

Laporte (1991) apresenta uma visão geral do PCV juntamente com aplicações utilizando alguns dos melhores algoritmos desenvolvidos até a década de 90, incluindo algoritmos exatos e heurísticos, como os algoritmos genéticos, por exemplo.

Ramos (2001) apresenta uma nova proposta de um método de aproximação baseado na combinação dos Algoritmos Genéticos (AG) e o *Simulated Annealing (SA - Recozimento Simulado)*, observando seu comportamento no problema combinatorial do caixeiro viajante.

Nilsson (2003) desenvolve uma série de heurísticas para o PCV simétrico, mas que podem ser facilmente adaptadas para lidar com caso assimétrico. Além disso, o trabalho engloba questões sobre tempo de execução e a qualidade das soluções.

Ribeiro, Mine e Silva (2009) apresenta um método heurístico, baseado na heurística *GRASP (Greedy Randomized Adaptive Search Procedure)*[1] e Busca Tabu[2] para resolver o

[1] GRASP (*Greedy Randomized Adaptive Search Procedure*) - É um método iterativo proposto por Feo e Resende (1995), que consiste de duas fases: uma fase de construção, na qual uma solução é gerada elemento a elemento e de uma fase de busca local, na qual um ótimo local na vizinhança da solução construída é pesquisado.

[2] Busca Tabu - É um procedimento adaptativo que utiliza uma estrutura de memória para guiar um método de descida a continuar a exploração do espaço de soluções mesmo na ausência de movimentos de melhora, evitando o retorno a um ótimo local previamente visitado.

clássico Problema de Roteamento de Veículos. Uma solução inicial é gerada por meio de heurísticas seguida de um refinamento pela meta-heurística Busca Tabu.

Lukasik e Żak (2010) explicam como a maioria dos algoritmos evolutivos, e também o algoritmo de vaga-lumes, enfrenta o problema de configuração de parâmetros. Deste modo, sugere definições sobre o tamanho da população e o máximo de coeficiente de absorção que poderiam ser utilizados. Alguns recursos adicionais, como redução aleatória do tamanho da população e a hibridação de outras técnicas aumentaria sua eficiência.

Yang (2010) fornece um estudo comparativo do AVL com PSO (*Particle Swarm Optimization* – Otimização Partícula de Enxame)[3] e Algoritmo Genético, descrevendo como o AVL parece mais promissor ao lidar com algumas funções multimodais específicas.

Kumbharana e Pandey (2012) implementa o Algoritmo de Vaga-lumes básico com alguns modificações em parâmetros para adaptá-lo para o Problema do Caixeiro Viajante. Essas modificações consistem em construir uma conversão adequada das funções contínuas (como atratividade, distância e movimento) em novas funções discretas. Por fim, o trabalho também mostra como o AVL fornece melhores resultados em relação a outras heurísticas.

Sureja e Chawda (2012) apresenta o Problema do Caixeiro Viajante Aleatório (PCVA) por meio do modelo do algoritmo genético. O PCVA é basicamente um PCV com todas as instâncias geradas aleatoriamente. Ao resolver esse problema usando o algoritmo genético (AG), é utilizado um esquema de codificação normal juntamente com o mecanismo de cruzamento básico.

Fister et al. (2013) apresenta um revisão abrangente do algoritmo de vaga-lumes e seus variantes, aplicado a área de otimização, bem como de engenharia. Além disso, o trabalho apresenta modificações e hibridização do AVL para resolver diversos problemas.

Gupta (2013) utiliza oito algoritmos bioinspirados para comparação de resultados, dentre estes estão o AVL e o AG, mostrando uma base de dados para experimentos com seis casos. Este artigo foi utilizado como base de experimentos devido ao *benchmark*.

Saraei e Mansouri (2015) apresenta o Problema de Múltiplos Caixeiros Viajantes (PMCV), que é uma generalização do PCV. Nesta nova abordagem tem uma restrição na tarefa de otimização, onde cada caixeiro viaja um conjunto único de cidades. Este trabalho avalia como o algoritmo genético pode ser aplicado para resolver esse problema e propõe uma solução eficiente.

Umbarkar, Balande e Seth (2017) apresenta um estudo detalhado do algoritmo AVL utilizando dois algoritmos de ordenação (a saber, ordenação bolha - *Bubble Sort* e

[3] *Particle Swarm Optimization* (PSO) é uma técnica de otimização estocástica, bio-inspirada no comportamento do bando de aves e de cardumes de peixes na procura de alimento. Neste algoritmo cada solução é considerado uma partícula, daí seu nome (ARBOLEDA, 2012).

a ordenação rápida - *Quicksort*), como um meio facilitador na comparação entre os vaga-lumes, uma vez que o trabalho original apresentado por Yang (2009) utiliza o algoritmo bolha para ordenação. Ao final é apresentado o melhor e o pior algoritmo de ordenação, bem como a média e o desvio padrão em relação ao número de comparações e o tempo de execução.

Nguyen, Vu e Dai (2018) apresenta uma melhoria do algoritmo de vaga-lumes para lidar com o problema da operação ótima de geração térmica com o objetivo de reduzir o custo total de geração de energia elétrica. O efeito de sua melhoria é investigado pela execução de cinco funções de *benchmark*. Os resultados obtidos indicam as melhorias propostas em termos de alta qualidade da solução ótima, capacidade de busca e convergência rápida em comparação ao algoritmo original. Além disso, as comparações com outros métodos indicam que o método proposto é uma ferramenta de otimização muito promissora para sistemas com função de custo de combustível com restrições complicadas.

4 Implementação do Trabalho

4.1 Implementação do método AG para o PCV

O algoritmo genético foi implementado em Java baseado no paradigma orientado a objetos, em uma máquina com sistema operacional Windows 10 Home Single Language, e um processador Intel(R) Core(TM) i5-7200U CPU @ 2.50 GHz com 8GB de memória RAM. O algoritmo genético foi desenvolvido sem a utilização de nenhuma API ou *framework* de terceiros para composição do projeto. Além disso, todo o código fonte está hospedado na plataforma GitHub.

4.1.1 Descrição do método

O AG é iniciado com a criação de uma população de indivíduos diferentes entre si (não somente o *fitness* diferente), tal como no processo biológico e evoluem para atingir a solução ótima ou aproximar-se da mesma.

4.1.2 Representação do indivíduo

Um indivíduo é representado por uma classe, de mesmo nome, composto dos seguintes atributos: um cromossomo, representado por um vetor de inteiros; uma aptidão, no formato de ponto flutuante, que é calculado a partir do cromossomo; e um inteiro representado a geração em que o indivíduo foi produzido. A população, também representada por uma classe, contém uma lista de indivíduos possibilitando a ordenação de seus membros pela aptidão.

Considerando o problema do caixeiro viajante, o cromossomo representa o vetor idêntico à rota correspondente, por exemplo, um cromossomo do tipo [9,8,7,6,5,4,3,2,1,0], em que representa uma rota ou instância do grafo de 10 vértices. Uma vez representado tal cromossomo, a ordem natural dos elementos dispostos conduz a função de avaliação, de forma a percorrer tal cromossomo da esquerda para direita, calculando a soma das distâncias de posições consecutivas, e por fim, a distância do última para o primeira posição do cromossomo, formando todo o circuito presente no problema do caixeiro viajante.

4.1.3 Visão geral do algoritmo

O algoritmo prossegue obedecendo ao critério de parada, que é atingir a quantidade de gerações pré-determinada. Inicialmente, gera-se a população inicial e selecionam-se os indivíduos pais para operação de reprodução. Vários métodos são possíveis, como roda da

roleta, seleção por torneio, seleção aleatória e entre outros, vistos no Capítulo 2. Para o presente trabalho, foi utilizado o método da roleta, sendo feito do seguinte modo: calcula-se uma soma de aptidões a partir da população atual. Em seguida, um número aleatório entre zero e a soma anteriormente calculada é gerado. Por fim, o indivíduo selecionado é o primeiro da lista ordenada por aptidão da população em questão, que apresenta aptidão maior ou igual à soma.

Após a seleção dos pais, os mesmos são utilizados para realizar o *crossover*. Neste trabalho foi utilizado o operador OX, conforme descrito na Seção 2.1. Para gerar outro filho, basta seguir os mesmo passos e pontos de corte invertendo os pais. Para a aplicação do processo de *crossover* no algoritmo genético foi utilizado uma taxa de 0.95.

A mutação foi inicialmente testada para dois tipos: a mutação de Holland e a mutação de Troca Dois Genes. Para a mutação de Troca de Dois Genes, selecionado o indivíduo, escolhe-se duas posições distintas aleatoriamente, e troca os genes em questão, recalculando toda aptidão. Para a mutação de Holland, o processo seleciona duas posições também distintas, e deste modo, o conteúdo da subsequência delimitada por essas posições é invertido. Conforme analisado posteriormente nos testes, a mutação de Holland obteve melhores resultados. Além disso, o processo ocorre sobre um ou mais indivíduos gerados, dado a probabilidade de 0.03 de cada um ser escolhido.

Para evitar a perda dos melhores indivíduos ao longo das gerações, duas abordagens foram utilizadas. Primeiro, ao utilizar mutação, verifica-se se o processo em questão melhora ou não o indivíduo de forma a validá-lo. Além disso, ocorrendo sua mutação ou não, o processo de reprodução deve ser capaz de gerar indivíduos totalmente diferentes e cada vez melhores para compor a população de filhos, ou seja, só é adicionado na população indivíduos com *fitness* melhores. Na segunda abordagem, um elitismo é empregado, isto é, identificam-se os melhores indivíduos de uma geração, que serão transferidos para a geração seguinte mantendo uma taxa de 0.8 das melhores soluções.

4.1.4 Pseudocódigo

O AG é iniciado com a criação de cada indivíduo (X_i) por meio da função $soluçãoInicial()$, compondo toda a população inicial. O algoritmo prossegue obedecendo ao critério de parada, que é atingir a quantidade de iterações pré-determinada, neste trabalho específico. Todo processo descrito é representado pelo Algoritmo 3.

Inicialmente uma população de novos indivíduos é gerada ($populaçãoNovoIndividuos$) representando os que vão compor a população da geração seguinte. Enquanto esta nova população de indivíduos não estiver completa, isto é, não tiver o tamanho da população inicial, pais que são selecionados para reprodução formando novos indivíduos.

Durante o processo de reprodução, são feitas dez tentativas até que se gere um

novo indivíduo a partir dos pais pré-selecionados (*gerouNovoIndivíduo*), ou até que se atinja o número máximo de tentativas (*tentativasMaxima*). Este processo é utilizado com o intuito de tentar gerar sempre indivíduos melhores dos que já existem, buscando garantir a variabilidade genética e aumentando, simultaneamente, a qualidade das soluções.

Para cada um dos filhos gerados, utiliza-se o mutador genético (*mutadorGenético*) para que novas soluções sejam descobertas. Além disso, a fim de que sempre novas soluções sejam encontradas explorando cada vez mais possibilidades, um teste é feito para que cada indivíduo ($filhos_i$) seja uma solução diferente, antes de ser colocado na população que fará parte na geração seguinte, garantindo a diversidade da população. Após a composição dos novos indivíduos (*populacaoNovosIndivíduos*) é feita a composição da população da geração anterior (X) selecionando os melhores indivíduos de ambos.

Algoritmo 3: Pseudocódigo do Algoritmo Genético.

```
Entrada: F(X), X=(X_1, X_2, ..., X_n), reprodutorGenético, mutadorGenético,
         elitismo
Saída: O melhor indivíduo X_i
início
    para i de 1 até n faça
        X_i ← soluçãoInicial();
    fim
    enquanto criterioDeParada == falso faça
        populaçãoNovoIndividuos ← novaPopulação();
        enquanto tamanho(populaçãoNovosIndividuos) <> n faça
            pais ← selecionarPais(X);
            tentativas ← 0;
            gerouNovoIndivíduo ← falso;
            enquanto não gerouNovoIndivíduo e tentativas <
             tentativasMaxima faça
                filhos ← reprodutorGenético(pais_0, pais_1);
                para i de 1 até tamanho(filhos) faça
                    mutadorGenético(filhos_i);
                    se não contem(X, filhos_i) e não
                     contem(populacaoNovosIndivíduos, filhos_i) então
                        adicionar(populacaoNovosIndivíduos, filhos_i);
                        gerouNovoIndivíduo ← verdadeiro;
                    fim
                    se contem(X, filhos_i) ou
                     contem(populacaoNovosIndiviudos,filhos_i) então
                        tentativas ← tentativas + 1;
                    fim
                fim
            fim
        fim
        elitismo(população, populacaoNovosIndivíduos);
    fim
fim
```

O processo de seleção dos pais é descrito pelo Algoritmo 4. A população de pais (Y) é gerada por um processo repetitivo, até que se encontre dois pais distintos por meio do método da roleta.

Algoritmo 4: Pseudocódigo do Algoritmo Genético - Selecionar de Pais

```
Entrada: X=(X_1, X_2, ..., X_n)
Saída: População de pais Y=(Y_1, Y_2)
início
    população ← novaPopulação();
    enquanto tamanho(pais) <> 2 faça
        pai ← roleta(X);
        se não contem(pais, pai) então
            adicionar(pais, pai);
        fim
    fim
    retorna pais
fim
```

O método da roleta utilizado para a seleção de pais é descrito no Algoritmo 5. Primeiramente, calcula-se a soma as aptidões de todos os indivíduos representado por *total*. Após isso é selecionado um valor entre 0 e total que represente um limite da soma parcial que será calculada (*totalLimite*). Por fim, para seleção dos pais, é feita uma varredura da população previamente ordenada por aptidão, de forma a selecionar o primeiro indivíduo cuja aptidão acumulada seja menor ou igual a aptidão calculada previamente.

Algoritmo 5: Pseudocódigo do Algoritmo Genético - Selecionar de Pais - Roleta

```
Entrada: F(X), X=(X_1, X_2, ..., X_n)
Saída: X_i selecionado aleatoriamente
início
    total ← calcularSomaAptidões(X);
    totalLimite ← random() * total;
    totalParcial ← 0.0;
    i ← 0;
    enquanto totalParcial < totalLimite e totalParcial <= total faça
        totalParcial ← totalParcial + F(X_i);
        i ← i + 1;
    fim
    retorna X_{i-1}
fim
```

O método de reprodução dos pais é descrito no Algoritmo 6. Primeiramente, gera-se um população vazia representando os filhos ($filhos$). Após isso é gerada uma probabilidade aleatória que determina se haverá ou não o *crossover* entre os pais. Em caso de sucesso, dois novos indivíduos são gerados montando seus cromossomos (*montarCromossomo*) e calculando suas aptidões (*calculadoraAptidão*).

Algoritmo 6: Pseudocódigo do Algoritmo Genético - Reprodutor Indivíduos - Crossover OX

Entrada: X_1, X_2
Saída: Y=(Y_1, Y_2) representando os filhos
início
  $filhos \leftarrow novaPopulação();$
  se *random() < probabilidadeCrossover* **então**
    $Y1_{novo} \leftarrow montarCromossomo(X_1, X_2);$
    $F(Y1_{novo}) \leftarrow calculadoraAptidão(Y1_{novo});$
    $adicionar(filhos, Y1_{novo});$
    $Y2_{novo} \leftarrow montarCromossomo(X_1, X_2);$
    $F(Y2_{novo}) \leftarrow calculadoraAptidão(Y2_{novo});$
    $adicionar(filhos, Y2_{novo});$
  fim
  retorna $filhos$
fim

O método de montagem do cromossomo é descrito no Algoritmo 7. A seleção das posições definem uma subsequência que é copiada do Pai 1(X_1) para o Filho(Y) na mesma posição em que os genes estão no pai. Para preencher as posições restantes do filho, copia-se do Pai 2 (X_2) o primeiro gene posicionado a partir do segundo ponto de corte e que ainda não está presente no filho para a primeira posição não preenchida no filho depois do segundo ponto de corte. Esse processo de cópia continua com o próximo gene de Pai 2 (X_2) utilizando este pai como uma lista circular até completar os genes do Filho.

Algoritmo 7: Pseudocódigo do Algoritmo Genético - Reprodutor Indivíduos - Crossover OX - Montar Cromossomo

```
Entrada: Cromossomo X1, X2
Saída: Y novo cromossomo
início
    posicao1 ← random(1, tamanho(X1));
    posicao2 ← posicao1;
    repita
        posicao2 ← random(1, tamanho(X1));
    até posicao2 <> posicao1;
    i ← posicao1;
    enquanto i <= posicao2 faça
        Y[i] ← X1[i];
        i ← i + 1;
    fim
    i ← 0;
    para j de 1 até tamanho(X2) faça
        se i == posicao1 então
            i ← posicao2 + 1;
        fim
        se não repetido(Y, X2[j]) então
            Y[i] ← X2[j];
            i ← i + 1;
        fim
    fim
    retorna Y
fim
```

O método de mutação genética é representado pelo Algoritmo 8. Primeiro é gerada uma probabilidade aleatória que determina se haverá ou não a mutação. Em caso de sucesso, selecionam-se duas posições distintas (*posicao1, prosicao2*) que representa os limites da subsequência do cromossomo que será invertido. Uma vez selecionadas, e invertida a subsequência, calcula-se a nova aptidão, que formará o novo indivíduo (X_{novo}).

Algoritmo 8: Pseudocódigo do Algoritmo Genético - Mutador Genético - Holland

Entrada: X
Saída: (X_{novo}) representando o individuo X
início
  se *random() < probabilidadeMutação* **então**
    $posicao1 \leftarrow random(1, tamanho(X))$;
    $posicao2 \leftarrow posicao1$;
    repita
      $posicao2 \leftarrow random(1, tamanho(X))$;
    até *posicao2<> posicao1*;
    $X_{novo} \leftarrow inverterSubsequência(X, posicao1, posicao2)$;
    $F(X_{novo}) \leftarrow calculadoraAptidão(X_{novo})$;
    $X \leftarrow X_{novo}$;
    $F(X) \leftarrow F(X_{novo})$;
  fim
  retorna X_{novo}
fim

O método de inverter um subsequência é representado pelo Algoritmo 9. Primeiro é verificado se as posições estão em ordem correta. Caso não esteja, é ajustado para que posicao1 < posicao2. Uma vez verificado, toda subsequência delimitada por essas posições são invertidas, similarmente à criação de um palíndromo dado um vetor de caracteres.

Algoritmo 9: Pseudocódigo do Algoritmo Genético - Mutador Genético - Holland - Inverter Subsequencia

Entrada: X, posicao1, posicao2
Saída: (X_{novo}) representando o individuo X
início
  $X_{novo} \leftarrow copia(X)$;
  se *posicao1* > *posicao2* **então**
    $aux \leftarrow posicao1$;
    $posicao1 \leftarrow posicao2$;
    $posicao2 \leftarrow aux$;
  fim
  $i \leftarrow posicao1$;
  $j \leftarrow posicao2$;
  enquanto $i < j$ **faça**
    $aux \leftarrow X_{novo}[i]$;
    $X_{novo}[i] \leftarrow X_{novo}[j]$;
    $X_{novo}[j] \leftarrow aux$;
    $i \leftarrow i + 1$;
    $j \leftarrow j - 1$;
  fim
  retorna X_{novo}
fim

O método de elitismo é representado pelo Algoritmo 10. Primeiro são adicionados à população da geração atual (*População*) os novos indivíduos gerados (*PopulaçãoNovosIndivíduos*). Segundo, a população da geração atual é ordenada por aptidão, e por fim, removem-se os n piores indivíduos de forma que a população mantenha o tamanho original

(m).

Algoritmo 10: Pseudocódigo do Algoritmo Genético - Elitismo.

Entrada: População=$(X_1, X_2, ..., X_m)$,
PopulaçãoNovosIndivíduos=$(Y_1, Y_2, ..., Y_n)$
Saída: População=$(X_1, X_2, ..., X_m)$
início
  $adicionar(População, PopulaçãoNovosIndivíduos)$;
  $ordernarPorAptidão(População)$;
  $removerPioresIndivíduos(População, n)$;
fim

A calculadora de aptidão, detalhada pelo Algoritmo 11, tem como intuito computar a aptidão de um indivíduo, com base na distância entre dois elementos adjacentes na permutação. Como a permutação representa um caminho no problema do caixeiro viajante, é necessário o cálculo da distância entre a última posição ($X_i[tamanho(X_i)]$) e a primeira posição ($X_i[1]$), que posteriormente é adicionado ao cálculo final.

Algoritmo 11: Pseudocódigo do Algoritmo Genético - Mutador Genético - Calculadora de Aptidão

Entrada: X_i
Saída: aptidão
início
  $aptidão \leftarrow 0.0$;
  para *j de 1 até tamanho*$(X_{novo}) - 1$ **faça**
    $aptidão \leftarrow aptidão + distancia(X_i[j], X_i[j+1])$;
  fim
  $aptidão \leftarrow aptidão + distancia(X_i[tamanho(X_i)], X_i[1])$;
  retorna *aptidão*
fim

4.2 Implementação do método AVL para o PCV

O algoritmo de vaga-lumes foi implementado em Java baseado no paradigma orientado a objetos, em uma máquina com sistema operacional Windows 10 Home Single Language, e um processador Intel(R) Core(TM) i5-7200U CPU @ 2.50 GHz com 8GB de memória RAM.

O algoritmo de vaga-lumes foi desenvolvido sem a utilização de nenhuma API ou framework de terceiros para composição do projeto. Além disso, todo o código fonte está hospedado na plataforma GitHub.

4.2.1 Descrição do método

O AVL (Algoritmo de Vaga-lumes) é iniciado com a criação, de modo aleatório, de uma população de vaga-lumes diferentes entre si; determinando o brilho correspondente de cada vaga-lume que compõem a população, de acordo com uma função objetivo específica, que depende do problema a ser tratado. Tal como no processo biológico, a convergência do algoritmo ocorre pela movimentação dos vaga-lumes induzidos pela bioluminescência de outros vaga-lumes.

4.2.2 Representação do vaga-lume

Um vaga-lume é representado por uma classe, de mesmo nome, composto dos seguintes atributos de classe:

- uma permutação, que é representada por um vetor de inteiros;
- um brilho, no formato de ponto flutuante, que é calculado a partir da permutação;
- um inteiro representando a última vez em que o vaga-lume foi atraído.

A população, também representada por uma classe, contém uma lista de indivíduos possibilitando a ordenação de seus membros pelo brilho.

Considerando o problema do caixeiro viajante, a permutação representa o vetor idêntico à rota correspondente, isto é, dada uma permutação do tipo [9,8,7,6,5,4,3,2,1,0] esta representará um rota ou instância do grafo de 10 vértices, neste exemplo.

4.2.3 Visão geral do algoritmo

O algoritmo prossegue obedecendo ao critério de parada, que é atingir a quantidade de iterações pré-determinada, neste trabalho em questão.

Inicialmente dois laços aninhados percorrem toda a população de vaga-lumes de forma a comparar o brilho entre eles.

A comparação feita entre os dois vaga-lumes serve como base para análise de atração e o movimento entre os vaga-lumes. Conforme sua definição, o vaga-lume de menor brilho (no problema do caixeiro viajante, de maior rota) é atraído pelo vaga-lume de maior brilho (isto é, de menor rota), uma vez que o problema em questão visa a minimização de rota.

Caso o algoritmo encontre um vaga-lume de menor brilho, seu deslocamento funcionará em três passos. Primeiro, calcula-se a distância entre os vaga-lumes utilizando a distância de Hamming, ou seja, o número de elementos entre as duas permutações que

não se correspondem. Segundo, a atratividade é calculada em função de um beta inicial (a probabilidade de combinação de soluções), um valor gama (coeficiente de absorção do meio) e a distância entre os vaga-lumes. Por último, é realizada a movimentação do vaga-lume de menor brilho em direção ao de maior brilho, e a intensidade de seu brilho é atualizada.

Caso o algoritmo não encontre um vaga-lume de menor brilho de forma a haver uma atração, o vaga-lume selecionado previamente do laço externo terá seu deslocamento de forma de aleatória. Considerando a equação do movimento conforme o Capítulo 3, esse processo ocorre por dois parâmetros: alfa e um randômico, chamado tal processo de passo alfa, que será explicado em breve.

O Algoritmo de Vaga-lumes foi inicialmente desenvolvido para variáveis contínuas. Entretanto, pode-se discretizar a solução de modo apropriado para muitos problemas.

A movimentação para variáveis contínuas, definida no Capítulo 3, mostra a dependência do movimento em termos de alfa e beta, contudo sua reformulação para variáveis discretas pode ser feita em função de beta, e somente então em função de alfa.

No decorrer das iterações do algoritmo, os vaga-lumes tendem a uma aproximação, diminuindo as distâncias entre si. Esse processo é dado pelo passo beta, que se inicia com a checagem e a permanência dos elementos comuns entre as duas permutações consideradas em questão. Uma vez feito tal procedimento, os elementos que não se correspondem são preenchidos aleatoriamente com elementos da permutação dos dois vaga-lumes, similarmente ao processo de *crossover* do algoritmo genético.

Note que no Algoritmo de Vaga-lumes original não existe uma comparação do valor do brilho atual do vaga-lumes com o seu brilho após atratividade, de forma a verificar uma melhora ou não, sempre se movimentando em direção ao vaga-lume de maior brilho. Ou seja, os vaga-lumes não retornam à sua posição original quando seu brilho não é melhorado (ARBOLEDA, 2012). Contudo, para o presente trabalho um teste a mais é feito, isto é, a movimentação apenas ocorre quando há uma melhora no vaga-lume. O mesmo teste é feito para movimentação randômica, preservando assim a melhor solução ao longo das iterações.

O passo alfa pode ser feito de duas formas. A primeira, consiste em trocar dois elementos da permutação, sendo repetido o processo) $\alpha * random(2, dist\hat{a}nciaHamming)$ vezes. A segunda forma, consiste em embaralhar $\alpha * random(2, dist\hat{a}nciaHamming)$ posições distintas. Esse processo pode se assemelhar ao processo de mutação do algoritmo genético.

4.2.4 Pseudocódigo

O AVL é iniciado com a criação de cada vaga-lume (X_i) por meio da função $solu\tilde{c}aoInicial()$, compondo assim toda a população inicial. O algoritmo prossegue obedecendo ao critério de parada, que é atingir a quantidade de iterações pré-determinada. Todo esse processo é representado pelo Algoritmo 12.

Inicialmente dois laços aninhados percorrem toda a população de vaga-lumes de forma a comparar o brilho de ambos, servindo de base para análise de atração e o movimento entre eles, com já fora mencionado. Caso o algoritmo encontre um vaga-lume de menor brilho, seu deslocamento funcionará em três passos. Primeiro, calcula-se a distância entre os vaga-lumes ($R_{i,j}$) utilizando a distância de Hamming ($dist\hat{a}ncia(X_i, X_j)$). Segundo, a atratividade é calculada em função de um beta inicial (β_0), um valor gama (γ) e a distância entre os vaga-lumes ($R_{i,j}$). Por último, é realizada a movimentação do vaga-lume de menor brilho em direção ao de maior brilho, e a intensidade de seu brilho é atualizada utilizando os parâmetros α, β, $R_{i,j}$.

Além disso, caso nenhum vaga-lume seja atraído por X_i uma movimentação randômica é feito no vaga-lume X_i. Essa etapa do movimento corresponde à uma diversificação por melhores resultados na busca global.

Algoritmo 12: Pseudocódigo do Algoritmo de Vaga-lumes

```
Entrada: F(X), X=(X_1, X_2, ..., X_n), α, β_0, γ,
Saída: O melhor vaga-lume X_i
início
    para i de 1 até n faça
        X_i ← soluçãoInicial();
    fim
    enquanto criterioDeParada == falso faça
        para i de 1 até n faça
            houveMovimentacao ← falso;
            para j de 1 até n faça
                se F(X_i) < F(X_j) então
                    houveMovimentacao ← verdadeiro;
                    R_{i,j} ← distância(X_i, X_j);
                    β ← atratividade(β_0, γ, R_{i,j});
                    houveMovimentacao ← movimentacao(X_j, X_i, α, β, R_{i,j});
                fim
            fim
            se houveMovimentacao == falso então
                movimentacaoRandomica(X_i);
            fim
        fim
    fim
fim
```

O cálculo da distância Hamming entre os vaga-lumes é representado pelo Algoritmo 13, que é definido como o número de elementos que não se correspondem entre duas permutações de mesmo tamanho.

Algoritmo 13: Pseudocódigo do Algoritmo de Vaga-lumes - Distância da Hamming

```
Entrada: X_i, X_j
Saída: Distância entre os vaga-lumes
início
    diferentes ← 0;
    para k de 1 até tamanho(X_i) faça
        se X_i[k] <> X_j[k] então
            diferentes ← diferentes + 1;
        fim
    fim
    retorna diferentes
fim
```

O cálculo da atratividade entre os vaga-lumes é representado pelo Algoritmo 14, que é definido pela Equação (2.2).

Algoritmo 14: Pseudocódigo do Algoritmo de Vaga-lumes - Atratividade

```
Entrada: β_0, γ, R_{i,j}
Saída: Atratividade entre os vaga-lumes
início
    atratividade ← β_0/(1 + (γ * R^2_{i,j});
    retorna atratividade
fim
```

A movimentação representado pelo Algoritmo 15, inicia-se com um laço de tentativas de movimentação até que se encontre uma solução melhor, ou até o número de tentativas se encerre. Para cada tentativa inicia-se com a criação de novo vaga-lume (X_{novo}), com baseado na cópia de um dos vaga-lumes de entrada, no caso X_j. A movimentação segue juntamente na ordem de dois passos importantes: passo-alfa e passo-beta; o processo é descrito justamente pelas próximas duas funções: $passoAlfa()$ e $passoBeta()$ respectivamente. Uma vez construída a nova permutação, utiliza-se uma calculadora de brilho para cálculo do brilho correspondente desta nova permutação. Caso a nova solução seja melhor que anterior o vaga-lume de menor brilho (X_j) deve ser representado por X_{novo}, e seu brilho $F(X_j)$ por $F(X_{novo})$, formando a nova solução após a movimentação.

Algoritmo 15: Pseudocódigo Movimentação dos Vaga-lumes

```
Entrada: X_j, X_i, α, β, R_{i,j}
Saída: Verdadeiro ou falso, indicando se houve movimentação ou não
início
    houveMovimentacao ← falso;
    tentativas ← 0;
    enquanto houveMovimentacao == falso e tentativas <
     tentativasMaximo faça
        X_novo ← copia(X_j);
        X_novo ← passoBeta(X_j, X_i, X_novo, β);
        X_novo ← passoAlfa(X_j, X_i, X_novo, R_{i,j}, α);
        F(X_novo) ← calculadoraDeBrilho(X_novo);
        se F(X_novo) < F(X_j) então
            X_j ← X_novo;
            F(X_j) ← F(X_novo);
            houveMovimentacao ← verdadeiro;
        fim
        tentativas ← tentativas + 1;
    fim
    retorna houveMovimentacao
fim
```

A movimentação randômica para o vaga-lume X_i é representado pelo Algoritmo 16. Primeiro é verificado se as posições estão em ordem correta. Caso não esteja, é ajustado para que posicao1 $<$ posicao2. Uma vez verificado, toda subsequência delimitada por essas posições são invertidas, similarmente a criação de um palíndromo dado um vetor de caracteres. Uma vez construída a nova permutação, utiliza-se uma calculadora de brilho para cálculo do brilho correspondente desta nova permutação. Caso a nova solução seja melhor que anterior o vaga-lume (X_i) deve ser representado por X_{novo}, e seu brilho $F(X_i)$ por $F(X_{novo})$, formando a nova solução após a movimentação.

Algoritmo 16: Pseudocódigo do Algoritmo de Vaga-lumes - Movimentação Randômica

Entrada: X_i
início
  $X_{novo} \leftarrow copia(X_i)$;
  $posicao1 \leftarrow random(1, tamanho(X_{novo}))$;
  $posicao2 \leftarrow posicao1$;
  enquanto *posicao1 == posicao2* **faça**
    $posicao2 \leftarrow random(1, tamanho(X_{novo}))$;
  fim
  se *posicao1 > posicao2* **então**
    $aux \leftarrow posicao1$;
    $posicao1 \leftarrow posicao2$;
    $posicao2 \leftarrow aux$;
  fim
  $k \leftarrow posicao1$;
  $l \leftarrow posicao2$;
  enquanto $k < l$ **faça**
    $aux \leftarrow X_{novo}[k]$;
    $X_{novo}[k] \leftarrow X_{novo}[l]$;
    $X_{novo}[l] \leftarrow aux$;
    $k \leftarrow k + 1$;
    $l \leftarrow l - 1$;
  fim
  $F(X_{novo}) \leftarrow calculadoraDeBrilho(X_{novo})$;
  se $F(X_{novo}) < F(X_i)$ **então**
    $X_i \leftarrow X_{novo}$;
    $F(X_i) \leftarrow F(X_{novo})$;
  fim
fim

O passo beta representado pelo Algoritmo 17, inicia da seguinte forma: preenche-se a nova solução (X_{novo}) com elementos comuns das permutações entre os dois vaga-lumes X_i e X_j. Quando o processo de preenchimento de elementos comuns terminar, pode acontecer de haver posições vazias dentro da na nova solução. Logo, o espaços vazios são preenchidos pela função $completarEspacosVazios()$, tomando novamente os vaga-lumes X_i, X_j e X_{novo}, utilizando uma probabilidade determinada por β. Se ainda acontecer de conter os espaços vazios, um preenchimento randômico de tais espaços é feito utilizando o procedimento $preenchimentoRandomico()$.

Algoritmo 17: Pseudocódigo Movimentação dos Vaga-lumes - Passo Beta

Entrada: $X_j, X_i, X_{novo}, \beta$
Saída: X_j
início
  $X_{novo} \leftarrow preenchimentoElementosComuns(X_j, X_i, X_{novo})$;
  $X_{novo} \leftarrow completarEspacosVazios(X_j, X_i, X_{novo}, \beta)$;
  se $contemEspacosVazios(X_{novo})$ **então**
    $X_{novo} \leftarrow preenchimentoRandomico(X_j, X_i, X_{novo})$;
  fim
  retorna X_j
fim

O preenchimento de elementos comuns, representado pelo Algoritmo 18, inicia-se pela marcação da nova solução (X_{novo}) como totalmente vazia, isto é, marcando cada posição k da permutação com -1. Logo em seguida, os elementos em comum entre a permutação X_i e X_j são utilizados para compor a nova solução X_{novo}.

Algoritmo 18: Pseudocódigo Movimentação dos Vaga-lumes - Passo Beta - Preenchimento de Elementos Comuns

Entrada: X_j, X_i, X_{novo}
Saída: X_{novo}
início
  para *k de 1 até tamanho*(X_j) **faça**
    $X_{novo}[k] \leftarrow -1$;
  fim
  para *k de 1 até tamanho*(X_j) **faça**
    se $X_i[k] == X_j[k]$ **então**
      $X_{novo} \leftarrow X_i[k]$;
    fim
  fim
  retorna X_{novo}
fim

Para completar os passos vazios, varre-se a permutação X_{novo} encontrando posições marcadas previamente com valor -1. Uma vez encontrado, gera-se um número randômico de forma a determinar qual permutação será utilizada para o preenchimento do espaço vazio. Com probabilidade β, e $1 - \beta$ seleciona-se a permutação X_j e X_i respectivamente para o preenchimento do espaço vazio. Com a permutação selecionada para preenchimento, verifica-se antes se a nova solução (X_{novo}) não contém já o novo valor, uma vez que não poderá haver posições repetidas. Este passo pode ser representado conforme pelo Algoritmo 19.

Algoritmo 19: Pseudocódigo Movimentação dos Vaga-lumes - Passo Beta - Completar Espaços Vazios

```
Entrada: X_j, X_i, X_novo, β
Saída: X_novo
início
    para k de 1 até tamanho(X_novo) faça
        se X_novo[k] == −1 então
            random ← random();
            se random < β então
                X ← X_j;
            fim
            se random >= β então
                X ← X_i;
            fim
        fim
        se contem(X_novo, X[k]) == falso então
            X_novo[k] ← X[k];
        fim
    fim
    retorna X_novo
fim
```

A verificação de espaços vazios representado pelo Algoritmo 20, varre a permutação X_{novo} em busca de posições, demarcados com o valor -1.

Algoritmo 20: Pseudocódigo Movimentação dos Vaga-lumes - Passo Beta - Contém Espaços Vazios

```
Entrada: X_novo
Saída: verdadeiro ou falso
início
    para i de 1 até tamanho(X_novo) faça
        se X_novo[i] == −1 então
            retorna verdadeiro
        fim
    fim
    retorna falso
fim
```

O preenchimento randômico representado pelo Algoritmo 21, inicia-se varrendo a permutação X_{novo} encontrando espaços vazios marcados com -1. Uma vez encontrado,

gera-se um número randômico de forma a determinar qual permutação será utilizada para o preenchimento do espaço vazio. Com probabilidade de 50% seleciona-se a permutação X_i ou X_j. Com a permutação selecionada para preenchimento, verifica-se antes se a nova solução não contém já o novo valor, uma vez que não poderá haver posições repetidas.

Algoritmo 21: Pseudocódigo Movimentação dos Vaga-lumes - Passo Beta - Preenchimento Randômico

```
Entrada: X_j, X_i, X_novo, β
Saída: X_novo
início
    para k de 1 até tamanho(X_novo) faça
        se X_novo == −1 então
            random ← random();
            se random < 0.5 então
                X ← X_j;
            fim
            se random >= 0.5 então
                X ← X_i;
            fim
        fim
        se contem(X_novo, X[k]) == falso então
            X_novo[k] ← X[k];
        fim
    fim
    retorna X_novo
fim
```

O passo alfa representado pelo Algoritmo 22, inicia-se com o cálculo de quantidade de trocas que devem ser feitas. Para isso, calcula-se um número randômico entre 2 e $R_{i,j}$ que representa a distância entre os vaga-lumes geradores. Para cada uma das trocas, selecionam-se duas posições distintas e trocam-se os valores das posições correspondentes na permutação X_i.

Algoritmo 22: Pseudocódigo Movimentação dos Vaga-lumes - Passo Alfa - Troca Dois Elementos

Entrada: $X_j, X_i, X_{novo}, R_{i,j}, \alpha$
Saída: X_j
início
  $quantidadeDeTrocas \leftarrow \alpha * random(2, R_{i,j})$;
  para *k de 1 até quantidadeDeTrocas* **faça**
    $posicao1 \leftarrow random(1, tamanho(X_j))$;
    $posicao2 \leftarrow posicao1$;
    repita
      $posicao2 \leftarrow random(1, tamanho(X_j))$;
    até *posicao2 <> posicao1*;
    $trocar(X_j[posicao1], X_j[posicao2])$;
  fim
  retorna X_j
fim

A calculadora de brilho explanado pelo Algoritmo 23, tem como intuito computar o brilho de um vaga-lume, com base na distância entre dois elementos adjacentes na permutação. Como a permutação representa um caminho no problema do caixeiro viajante, o cálculo da distância entre a última posição ($X_i[tamanho(X_i)]$) e a primeira posição ($X_i[1]$) é adicionado ao brilho.

Algoritmo 23: Pseudocódigo Movimentação dos Vaga-lumes - Calculadora de Brilho

Entrada: X_j
Saída: brilho
início
  $brilho \leftarrow 0.0$;
  para *k de 1 até tamanho(X_{novo}) − 1* **faça**
    $brilho \leftarrow brilho + distancia(X_j[k], X_j[k+1])$;
  fim
  $brilho \leftarrow brilho + distancia(X_j[tamanho(X_j)], X_j[1])$;
  retorna *brilho*
fim

5 Resultado e discussões

Para os experimentos realizados neste trabalho, foi utilizada a TSPLIB (*Travelling Salesman Problem Library*) [1] que é uma biblioteca de instâncias online e gratuita para o Problema do Caixeiro Viajante. As instâncias disponíveis estão em arquivos, sendo cada instância identificada por um nome e uma numeração, sendo que a última indica a quantidade de nós no grafo que deve ser visitado. Cada grafo representa o conjunto de cidades onde se deseja encontrar o ciclo hamiltoniano. Estas instâncias foram usadas para validar os métodos implementados neste trabalho: Algoritmo Genético e Algoritmo de Vaga-lumes.

5.1 Algoritmo Genético

Os testes para as instâncias presentes no TSPLIB utilizando Algoritmo Genético são apresentados pela Tabela 1. Para os parâmetros de porcentagem de elitismo, probabilidade de *crossover* e probabilidade de mutação, tem-se os valores de 0.8, 0.95 e 0.03, respectivamente. Para o processo de *crossover* foi utilizado o operador genético baseado em ordem (*Crossover* OX). Para o processo de mutação o operador de utilizado foi o de Holland. Tais instâncias foram baseadas nos experimentos apresentados no artigo de Kumbharana e Pandey (2012).

Tabela 1 – Teste do Algoritmo Genético para a biblioteca TSPLIB.

Instância	**Tamanho da População**	**Iterações**	**Tempo (ms)**	**Média Aptidão**	**Desvio Aptidão**	**Solução Encontrada**	**Solução Ótima**
burma14	65	140	234	3589,935	81,481	3323	3323
ulysses16	65	140	309	6999,282	42,974	6859	6859
bays29	135	385	380	2034,728	3,715	2020	2020
oliver30	135	425	709	430,485	1,931	423,741	423,741
eil51	155	890	10047	435,313	1,289	426	426

Note que em todas as instâncias do experimento foram alcançadas as soluções ótimas usando o Algoritmo Genético, sendo que para cada instância foram realizadas 10 execuções. Em cada linha é mostrada a melhor execução de 10, perceba que para instâncias maiores (bays29, oliver30 e eil51), foi necessário ajuste do tamanho da população e o número de iterações. Vale ressaltar que a coluna Iterações da Tabela 1 mostra em qual iteração foi encontrada a melhor solução, onde para instância pequena o limite máximo de iteração foi 3000, e para as maiores foi 10000. Logo, para instâncias maiores foi necessária maior quantidade de iterações, justamente por aumentar a complexidade nestas instâncias.

[1] https://wwwproxy.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/

Deste modo, pode-se perceber que já nos primeiros experimentos com AG foram alcançados os resultados esperados.

5.2 Algoritmo de Vaga-lumes

Os testes para as instâncias presentes no TSPLIB utilizando o método de Algoritmo de Vaga-lumes são apresentados pela Tabela 2. Para os parâmetros α, β e γ os valores utilizados foram 0.5, 1.0 e 0.0, respectivamente. Para o passo alfa, o método de trocar dois elementos foi aplicado, bem como para a função distância foi utilizada a função de distância de Hamming. Estes parâmetros foram obtidos do artigo de Koide et al. (2015).

Tabela 2 – Teste do Algoritmo de Vaga-lumes para a biblioteca TSPLIB.

Instância	**Número de Vaga-lumes**	**Iterações**	**Tempo (ms)**	**Média Aptidão**	**Desvio Aptidão**	**Solução Encon-trada**	**Solução Ótima**
burma14	65	95	63	4488,938	816,581	3323	3323
ulysses16	65	95	120	8325,84	1279,043	6859	6859
bays29	120	315	1216	3273,066	881,049	2020	2020
oliver30	120	375	7668	708,849	170,007	423,741	423,741
eil51	240	985	9266	446,000	0,000	446	426

Em cada linha é mostrada a melhor execução de 10, observe que nestes primeiros experimentos, as soluções ótimas foram encontradas pelo AVL para instâncias menores. No entanto, para instância maior(eil51) o AVL não foi capaz de alcançar o resultado esperado.

Deste modo, percebeu-se a necessidade de melhoria no ajuste de parâmetros do algoritmo AVL, bem como na estratégia evolutiva dos vaga-lumes. A seguir, será apresentada uma série de experimentos com a finalidade de refinamento dos resultados para instância eil51.

5.2.1 Reajuste de parâmetros para Algoritmo de Vaga-lumes

Por ser a maior instância de teste, e devido o AVL não ter encontrado ainda a solução ótima, decidiu-se efetuar reajuste dos parâmetros, bem como na estratégia evolutiva dos vaga-lumes para a instância eil51.

Para a estratégia evolutiva, foi realizada a seguinte modificação: a movimentação apenas ocorre quando há uma melhora no vaga-lume. O mesmo teste é feito para o caso de movimentação randômica, preservando assim a melhor solução ao longo das iterações.

Em relação à quantidade de vaga-lumes, manteve-se 50, aumentando consideravelmente o número de iterações.

Na Tabela 3, pode-se observar que a melhor configuração foi 50 e 10000 para o tamanho da população e o número de iterações, respectivamente. Nos testes, foi mantido o tamanho da população de 50, aumentando o número de iterações, de 3000 até 10000. É possível observar uma melhora nos resultados, sendo que já se obteve uma solução de 430, lembrando que a meta é 426. No entanto, o desvio do brilho ser zero, assim como na Tabela 3, indica que o conjunto final de soluções está homogêneo, ou seja, as soluções convergiram todas para um ótimo local, conforme indicado pela Figura 7.

A Figura 7 mostra a média do brilho e o valor do melhor brilho durante a execução do AVL. Pode-se verificar que há a convergência muito prematura para um melhor resultado com essa configuração, coincidindo o valor do melhor brilho com a média do brilho.

Tabela 3 – Teste da instância eil51 variando o número de iterações.

Iterações	**Média Brilho**	**Desvio Brilho**	**Melhor Brilho**
3000	446,000	0,000	446
4000	446,000	0,000	446
5000	442,000	0,000	442
6000	440,000	0,000	440
7000	437,000	0,000	437
8000	437,000	0,000	437
9000	437,000	0,000	437
10000	430,000	0,000	430

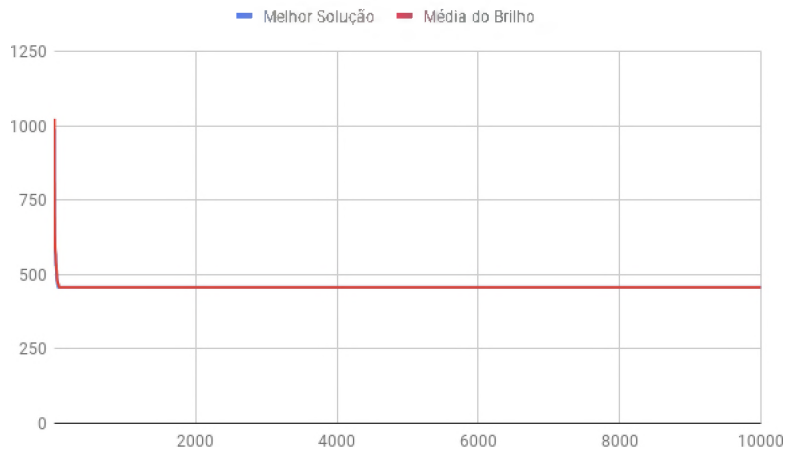

Figura 7 – Relação entre melhor brilho e média do brilho ao longo das iterações para a variação do número de iterações.

Logo, ainda há necessidade de mais ajustes, mantendo a melhor configuração encontrada até o momento, mas ajustando o parâmetro alfa e gama. Na Tabela 4, é mostrado o ajuste para o parâmetro alfa. Igualmente, na Tabela 5 é mostrado o ajuste para o parâmetro gama. A Figura 8 e Figura 9 mostram novamente a média do brilho e o valor

do melhor brilho durante a execução do AVL. Pode-se perceber uma notável melhoria na evolução da média do brilho e do melhor valor com esse novo ajuste de parâmetros, onde a convergência não se mostrou tão precoce quanto na Figura 7.

Tabela 4 – Teste da instância eil51 variando o valor de alfa.

Alfa	**Média Brilho**	**Desvio Brilho**	**Melhor Brilho**
0.10	459.50	9.714	436.0
0.50	452.72	9.734	431.0
0.75	455.86	10.413	438.0
1.00	451.74	8.543	434.0

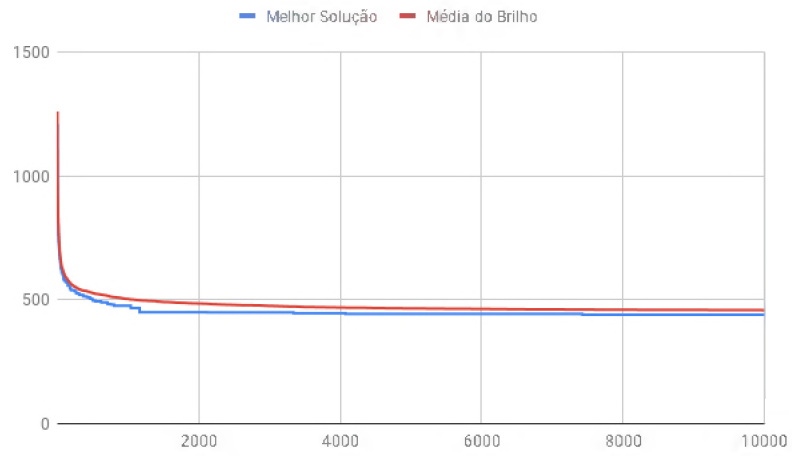

Figura 8 – Relação entre melhor brilho e média do brilho ao longo das iterações para a variação do valor de alfa.

Tabela 5 – Teste da instância eil51 variando o valor de gama.

Gama	**Média Brilho**	**Desvio Brilho**	**Melhor Brilho**
0.0	446,000	0,000	446
0.2	454,360	12,399	430
0.4	454,400	9,325	432
0.6	452,960	9,921	430
0.8	450,160	9,825	428
1.0	453,820	12,393	430
2.0	455,780	11,496	430
4.0	454,040	10,125	430
6.0	449,520	8,978	430
8.0	451,680	9,934	432
10.0	452,000	9,814	431

A partir desses testes, pode-se obter a melhor configuração sendo 0.5 para alfa e 0.8 para gama. Observe que a melhor solução da Tabela 5 é um valor menor que na Tabela 3, por isso, os resultados são mais satisfatórios. Outro detalhe que se pode verificar é que o desvio de brilho é diferente de zero (exceto para o caso onde gama é zero), isto é, o conjunto de soluções finais apresentou uma variabilidade. No entanto, ainda houve a necessidade

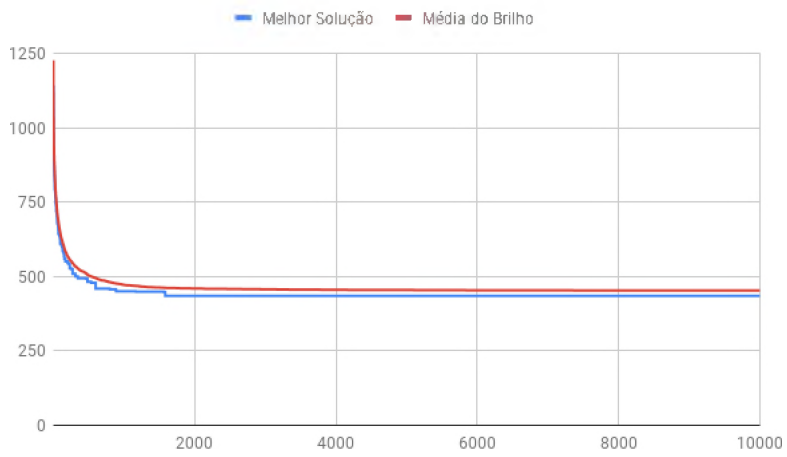

Figura 9 – Relação entre melhor brilho e média do brilho ao longo das iterações da melhor execução de ajuste para o parâmetro gama.

Tabela 6 – Teste do Algoritmo de Vaga-lumes com passo alfa alternado.

Troca Dois Elementos	**Embaralha Posições**	**Média Brilho**	**Desvio Brilho**	**Melhor Brilho**
0.0	1.0	452,40	10,186	433
0.1	0.9	449,56	8,899	430
0.2	0.8	454,48	8,852	432
0.3	0.7	453,38	11,283	430
0.4	0.6	453,00	10,103	432
0.5	0.5	452,24	9,422	429
0.6	0.4	452,88	10,124	431
0.7	0.3	453,02	11,707	431
0.8	0.2	451,42	9,134	430
0.9	0.1	451,52	10,042	430
1.0	0.0	452,04	9,370	427

de ajuste para o passo alfa, onde pode ser feita usando a troca de dois elementos distintos da permutação e o embaralhamento de várias posições da permutação, selecionando de forma aleatória entre estes. Na Tabela 6 e na Figura 10 pode-se verificar os resultados obtidos.

Com a melhor configuração obtida, pode-se observar que a melhor solução encontrada (427) está bem mais próxima da ótima (426). Na Figura 10 pode-se perceber que a convergência do melhor valor de brilho e a média do brilho também não foi tão prematura, mantendo-se a mesma apenas após 2000 iterações, o que não tinha acontecido nos testes anteriores.

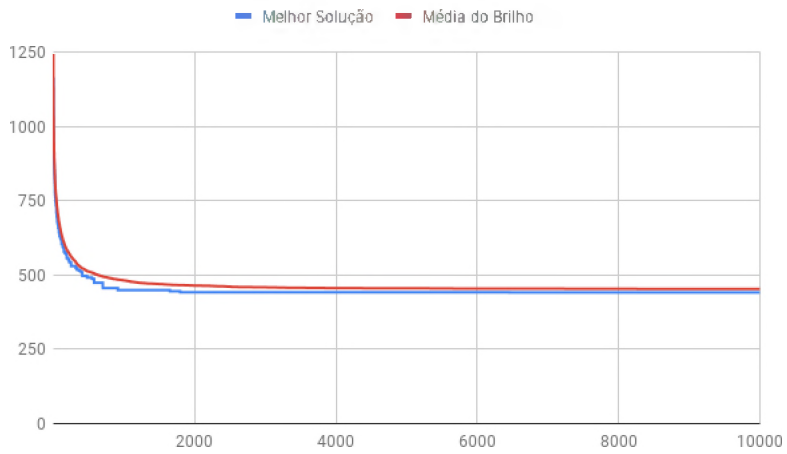

Figura 10 – Relação entre melhor brilho e média do brilho ao longo das iterações da melhor execução de ajuste para o passo alfa.

6 Conclusão

Com este Trabalho de Conclusão de Curso, foi possível realizar um estudo de dois algoritmos de otimização computacional, e por meio de experimentos, pode-se validá-los com uma base da biblioteca TSPLIB. A partir dos testes executados, pode-se confirmar a eficiência do Algoritmo Genético para o clássico Problema do Caixeiro Viajante. Para o AG, não houve dificuldade no desenvolvimento nem na etapa de ajustes de parâmetro, apresentando facilmente os resultados esperados.

Para o Algoritmo de Vaga-lumes, o desenvolvimento apresentou uma primeira dificuldade por terem várias versões de pseudocódigo na literatura, que apresentavam instruções imprecisas. Superada essa limitação, percebeu-se que a etapa de ajuste de parâmetros foi mais custosa, em termos de tempo de desenvolvimento, posto que haviam mais parâmetros que o AG para serem tratados (número de vaga-lumes, quantidade de iterações, alfa, beta, gama, passo alfa). Novamente, a dificuldade encontrada no estudo do AVL foi a limitação de artigos com boas informações, especialmente para ajuste de parâmetros. Logo, essa etapa de refinamento de resultados mostrou-se bem mais demorada que o esperado por conta desse limitante, tendo que ser realizada de modo empírico. Para trabalhos futuros, existem outros métodos para calcular a distância entre as soluções (função de distância) que podem ser experimentados, uma vez que neste trabalho manteve-se a distãncia de Hamming.

Referências

APPLEGATE, D. L. et al. *The Traveling Salesman Problem: A Computational Study*. Princeton, Estados Unidos da América: Princeton University Press, 2007. Citado na página 21.

ARBOLEDA, D. M. M. *Otimização por inteligência de enxames usando arquiteturas paralelas para aplicações embarcadas*. Tese (Doutorado) — Faculdade de Tecnologia Departamento de Engenharia Mecânica, Universidade de Brasília, Brasília, Brasil, 2012. Disponível em: <http://repositorio.unb.br/bitstream/10482/13055/1/2012_DanielMauricioMunozArboleda.pdf>. Acesso em: 27.11.2017. Citado 5 vezes nas páginas 11, 17, 18, 25 e 38.

DARWIN, C. *On the origin of species by means of natural selection, or, the preservation of favoured races in the struggle for life*. Londres, Reino Unido: [s.n.], 1859. Citado 2 vezes nas páginas 11 e 14.

DAVIS, L. Applying adaptive algorithms to epistatic domains. 1985. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.76.4906&rep=rep1&type=pdf>. Citado na página 23.

DELBEM, A. C. B.; GABRIEL, P. H. R. Fundamentos de algoritmos evolutivos. 2008. Disponível em: <http://conteudo.icmc.usp.br/CMS/Arquivos/arquivos_enviados/BIBLIOTECA_113_ND_75.pdf>. Citado 3 vezes nas páginas 11, 14 e 15.

DORIGO, M.; STüZLE, T. *Ant Colony Optimization*. Cambridge, Estados Unidos da América: The MIT Press, 2004. Citado na página 11.

DURKOTA, K. *Implementation of a Discrete Firefly Algorithm Within the SEAGE*. 2011. Disponível em: <http://agents.fel.cvut.cz/~durkota/files/BachelorThesisDurkota.pdf>. Citado na página 23.

EIBEN, A.; SMITH, J. *Introduction to evolutionary computing*. Berlim, Alemanha: Springer, 2003. Citado na página 15.

FISTER, I. et al. A comprehensive review of firefly algorithms. 2013. Disponível em: <https://www.researchgate.net/publication/259170624_A_comprehensive_review_of_firefly_algorithms>. Citado 2 vezes nas páginas 11 e 25.

GOLDBERG, D. E. *Genetic Algorithms in Search, Optimization and Machine Learning*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc, 1989. Citado 2 vezes nas páginas 11 e 14.

GUPTA, D. Solving tsp using various meta-heuristic algorithms. 2013. Disponível em: <http://online-journals.org/index.php/i-jes/article/download/3233/2930>. Citado na página 25.

HOLLAND, J. H. *Adaptation in Natural and Artificial Systems*. Cambridge, MA, USA: John Murray, 1975. Citado 2 vezes nas páginas 11 e 14.

KOIDE, R. M. *Aplicacação de regressão de vetores de suporte na otimização em flambagem e pós-flambagem de estruturas compósitas*. Tese (Doutorado) — Programa de Pós-graduação em Engenharia Mecânica e de Materiais, Universidade Tecnológica Federal do Paraná, Curitiba, Brasil, 2016. Disponível em: <http://www.utfpr.edu.br/curitiba/estrutura-universitaria/diretorias/dirppg/programas/ppgem/banco-teses/teses/Tese_Rubem_Koide.pdf>. Acesso em: 27.11.2017. Citado 2 vezes nas páginas 18 e 23.

KOIDE, R. M. et al. Algoritmo de vaga-lumes aplicado à otimização da carga de flambagem de placas compósitas laminadas. Curitiba, Brasil, 2015. Disponível em: <https://ssl4799.websiteseguro.com/swge5/PROCEEDINGS/PDF/CILAMCE2015-0135.pdf>. Acesso em: 30.06.2019. Citado na página 49.

KUMBHARANA, S. N.; PANDEY, G. M. Solving travelling salesman problem using firefly algorithm. 2012. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=A0C8279078FD51227F808A3AFFFD52B4?doi=10.1.1.668.3119&rep=rep1&type=pdf>. Citado 2 vezes nas páginas 25 e 48.

LAPORTE, G. The traveling salesman problem: An overview of exact and approximate algorithms. 1991. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.461.189&rep=rep1&type=pdf>. Citado na página 24.

LAWLER, E. L. *The Traveling salesman problem : a guided tour of combinatorial optimization*. Nova Iorque, Estados Unidos da América: Chichester [West Sussex], 1985. Citado na página 24.

LINDEN, R. *Algoritmos Genéticos*. São Paulo, Brasil: Brasport, 2008. Citado na página 12.

LUKASIK, S.; ŻAK, S. Firefly algorithm for continuous constrained optimization tasks. 2010. Disponível em: <http://home.agh.edu.pl/~slukasik/pub/016_Lukasik_Zak_LNAI_ICCCI2009.pdf>. Citado na página 25.

NGUYEN, T. T.; VU, Q. N.; DAI, L. V. Improved firefly algorithm: A novel method for optimal operation of thermal generating units. 2018. Disponível em: <http://downloads.hindawi.com/journals/complexity/2018/7267593.pdf>. Citado 2 vezes nas páginas 11 e 26.

NILSSON, C. Heuristics for the traveling salesman problem. 2003. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.104.9895&rep=rep1&type=pdf>. Citado na página 24.

NILSSON, N. J. *Principles of Artificial Intelligence*. Nova Iorque, Estados Unidos da América: Springer-Verlag Berlin Heidelberg, 1982. Citado 2 vezes nas páginas 12 e 21.

RAMOS, J. M. B. Implementação e análise do problema do caixeiro viajante usando uma nova abordagem através dos algoritmos genético e simulated annealing. 2001. Disponível em: <https://repositorio.ufsc.br/xmlui/bitstream/handle/123456789/79640/250657.pdf?sequence=1&isAllowed=y>. Citado 5 vezes nas páginas 11, 12, 22, 23 e 24.

RIBEIRO, F. N.; MINE, M. T.; SILVA, M. d. S. A. Problema de roteamento de veículos. 2009. Disponível em: <http://www.decom.ufop.br/marcone/Disciplinas/

InteligenciaComputacional/ProblemadeRoteamentodeVeículos.doc>. Citado na página 24.

SARAEI, M.; MANSOURI, P. Solving of travelling salesman problem using firefly algorithm with greedy approach. 2015. Disponível em: <https://www.researchgate.net/profile/Mohammad_Saraei4/publication/304625935_Solving_of_Travelling_Salesman_Problem_using_Firefly_Algorithm_with_Greedy_Approach/links/5775302e08ae1b18a7dfdaac/Solving-of-Travelling-Salesman-Problem-using-Firefly-Algorithm-with-Greedy-Approach.pdf>. Citado na página 25.

SIPSER, M. *Introdução à Teoria da Computação*. Cengage Learning Edições Ltda., 2010. ISBN 9788522108862. Disponível em: <https://books.google.com.br/books?id=dLLqnQAACAAJ>. Citado na página 22.

SUREJA, N. M.; CHAWDA, B. V. Random travelling salesman problem using genetic algorithms. 2012. Disponível em: <https://www.researchgate.net/publication/291697285_Random_Travelling_Salesman_Problem_using_Genetic_Algorithms>. Citado na página 25.

UMBARKAR, A. J.; BALANDE, U. T.; SETH, P. D. Performance evaluation of firefly algorithm with variation in sorting for non-linear benchmark problems. 2017. Disponível em: <https://aip.scitation.org/doi/pdf/10.1063/1.4981972>. Citado na página 25.

YANG, X.-S. Firefly algorithms for multimodal optimization. 2009. Disponível em: <https://arxiv.org/pdf/1003.1466.pdf>. Citado 3 vezes nas páginas 11, 17 e 26.

YANG, X.-S. *Nature-Inspired Metaheuristic Algorithms*. Frome, Reino Unido: Luniver Press, 2010. Citado 2 vezes nas páginas 17 e 25.