Ministério da Educação
Secretaria de Educação Profissional e Tecnológica
Instituto Federal Catarinense
Campus Rio do Sul

RAFAEL JOSÉ SCHMIDT DOS SANTOS

USO DE ALGORITMOS DE BUSCA HEURÍSTICA E ALGORITMOS GENÉTICOS PARA DINAMIZAR A LOGÍSTICA NO EMPACOTAMENTO DE CARGAS

Rio do Sul
2019

RAFAEL JOSÉ SCHMIDT DOS SANTOS

USO DE ALGORITMOS DE BUSCA HEURÍSTICA E ALGORITMOS GENÉTICOS PARA DINAMIZAR A LOGÍSTICA NO EMPACOTAMENTO DE CARGAS

Trabalho de Conclusão de Curso apresentado ao Curso de graduação em Ciência da Computação do Instituto Federal Catarinense – Campus Rio do Sul para obtenção do título de bacharel em Ciência da Computação.
Orientador: Juliano Tonizetti Brignoli, Dr.

Rio do Sul
2019

RAFAEL JOSÉ SCHMIDT DOS SANTOS

USO DE ALGORITMOS DE BUSCA HEURÍSTICA E ALGORITMOS GENÉTICOS PARA DINAMIZAR A LOGÍSTICA NO EMPACOTAMENTO DE CARGAS

Este Trabalho de Curso foi julgado adequado para a obtenção do título de Bacharel em Ciência da Computação e aprovado em sua forma final pelo curso de graduação em Ciência da Computação do Instituto Federal Catarinense – Campus Rio do Sul.

Rio do Sul (SC), 21 de fevereiro de 2019

Juliano Tonizetti Brignoli, Dr.
Instituto Federal Catarinense – Campus Rio do Sul

BANCA EXAMINADORA

Daniel Gomes Soares, Msc.
Instituto Federal Catarinense – Campus Rio do Sul

Cristhian Heck, M.Eng.
Instituto Federal Catarinense – Campus Rio do Sul

AGRADECIMENTOS

Agradeço primeiramente a Deus por ter concedido esta oportunidade. Aos meus familiares e amigos que me incentivaram e apoiaram. A meu orientador Dr. Juliano Tonizetti Brignoli pelos conselhos fundamentais prestados durante o desenvolvimento deste trabalho. E por fim, aos colegas e a todos os diletos professores pela colaboração, dedicação e amizade durante todo o curso.

RESUMO

Problemas de empacotamento possuem uma relação forte com a área de logística, quando trata da alocação de um conjunto de itens em um recipiente, considerando variáveis como a otimização de espaço e ordem de entrega.
Este trabalho apresenta o desenvolvimento de uma aplicação para dinamizar a logística no empacotamento de cargas utilizando algoritmos de Busca em Espaços de Estados com Heurística A* e Algoritmos Genéticos. Seu principal objetivo é apresentar a melhor forma de alocar um conjunto de pacotes de variados tamanhos em um confinamento também de tamanho variável, de forma com que se aproveite ao máximo o espaço, deixando a maior área útil livre possível no confinamento, como também deixar os pacotes com maior prioridade de retirada mais próximos da saída, e então comparar os resultados obtidos com ambas as técnicas.
Os resultados mostram que é possível utilizar tanto o algoritmo de Busca em Espaços de Estados quanto Algoritmos Genéticos para solucionar o problema do empacotamento, verificando que para itens de grandes dimensões ou com maior largura, o Algoritmo de Busca em Espaço de Estados obteve melhor resultado, e para itens pequenos ou com maior comprimento, o Algoritmo Genético obteve o melhor resultado.

Palavras-chave: Empacotamento. Inteligencia Artificial. Algoritmos Genéticos. Busca Heurística

ABSTRACT

The Packaging problems have a strong relationship with the logistics area when dealing with the allocation of a set of items in a container, considering variables such as space optimization and delivery order.
This work presents the development of an application to dynamize the logistics in the packing of loads using algorithms of Search in Spaces of States with A * Heuristics and Genetic Algorithms. Its main objective is to present the best way to allocate a set of packages of varied sizes in a confinement also of variable size, so that the space is maximized, leaving the largest free area possible in the confinement, and comparing the results obtained with both techniques.
The results show that it is possible to use both the State Spacing and Genetic Algorithm algorithms to solve the problem of packaging, verifying that for large or large items, the State Space Search Algorithm obtained better results, and for small items or with longer length, the Genetic Algorithm obtained the best result.

Key-words: Packing. Artificial intelligence. Genetic Algorithms. Heuristic Search

LISTA DE ILUSTRAÇÕES

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

IA	Inteligência Artificial
RF	Requisito Funcional
RNF	Requisito Não Funcional
UC	Caso de Uso
PLP	Pallet Load Problem
AG	Algoritmo Genético
API	Aplication Program Interface

SUMÁRIO

1 INTRODUÇÃO

Encontrar o melhor arranjo na alocação de cargas em um espaço confinado é uma prioridade em diversos segmentos de negócios como portos, empresas de transporte, cargueiros, entre outros.

Em sua origem, a logística estava essencialmente ligada às operações militares, no deslocamento de munição, víveres, equipamentos e socorro médico para o campo de batalha (NOVAES, 2015). A logística foi aplicada às indústrias ao se perceber o valor da mão de obra e tempo desperdiçados, como o transporte de matéria prima ou mesmo do produto final até um revendedor.

A logística possui várias vertentes, e o empacotamento é uma delas, sendo este o objeto dessa pesquisa. Por definição, este problema consiste preencher um recipiente com a maior quantidade de itens possíveis da melhor forma possível (WASCHER, 2007). Exemplo desse tipo de problema é o carregamento de cargas em um baú de caminhão, que equivale à organizar as caixas de produtos, visando otimizar a ocupação de área disponível, reduzindo então custos com transporte, descarga, higiene, produtos frágeis, entre outros. Este contratempo pode ser resolvido computacionalmente através de técnicas de Inteligência Artificial.

A Inteligência Artificial é uma área estudada a décadas. Seu termo nasceu oficialmente em 1956 no famoso encontro de Dartmouth e ao longo do tempo o estudo da Inteligência Artificial proporcionou grande progresso na resolução de problemas e diversos métodos foram desenvolvidos (RUSSEL; NORVIG, 2013).

Diante deste contexto, percebe-se o potencial no estudo do empacotamento. Sendo assim, foi desenvolvido um software que utilize um algoritmo de busca em espaços de estados com heurística A*, e da mesma forma algoritmos genéticos, podendo comparar ambas as técnicas aplicadas no problema de empacotamento, permitindo edição das variáveis do problema, retornando possíveis combinações com os melhores resultados.

1.1 PROBLEMATIZAÇÃO

Parafraseando Ballou (2006), o transporte é essencial, pois nenhuma empresa moderna consegue operar sem providenciar a movimentação da sua matéria-prima ou de seus produtos acabados. Ainda para o autor, o transporte também representa um dos elementos mais importantes em termos de custos logísticos para as empresas, visto que a movimentação de

cargas pode absorver de um a dois terços dos custos logísticos totais, além de impactar diretamente no atendimento dos pedidos dos clientes. Logo, dimensionar um bom empacotamento acarreta em uma economia significativa de recursos, pois, em uma mesma carga, acomoda-se uma quantidade maior de objetos, auxiliando na rapidez de entrega e diminuindo a jornada de viagens.

Por isso surge o questionamento: Como encontrar o melhor arranjo na alocação de cargas em um espaço confinado?

1.1.1 Solução Proposta

A alocação das cargas em um espaço confinado se torna complexa devido a não linearidade dos objetos a serem alocados. Diante deste problema enfrentado por empresas dos mais diversos setores, surge a ideia de uma ferramenta que possa auxiliar na atribuição de artigos em um recinto delimitado, desta maneira, auxiliando o profissional pertinente a acomodar da melhor forma possível os produtos em um ambiente.

1.1.2 Delimitação do escopo

O trabalho contempla o desenvolvimento de uma ferramenta, que visa encontrar o melhor arranjo na alocação de cargas em um espaço confinado, de forma com que o usuário entre com as dimensões de cada objeto, do espaço que vai abrigar os mesmos. Então o sistema busca a melhor forma de alocá-los, utilizando técnicas de Inteligencia Artificial, mais especificamente, a busca em espaços de estados com heurística A* e também algoritmos genéticos. Para alcançar o objetivo proposto, será desenvolvida uma aplicação Desktop implementada na linguagem Java.

1.1.3 Justificativa

Bowersox e Closs (2001) afirmam que a logística empresarial se tornou singular e está presente em qualquer atividade de produção ou de marketing. A maioria dos consumidores em nações industriais altamente desenvolvidas, quando vão as lojas, esperam encontrar os produtos recém-fabricados, hortifrutigranjeiro de boa qualidade e de grande variedade disponíveis. Diante deste cenário, a implementação das melhores práticas logísticas tornou-se uma das áreas operacionais mais desafiadoras e interessantes da administração, podendo contar com o auxílio de aplicações que fazem uso de inteligência artificial.

A técnica de Inteligência Artificial, abordada no presente trabalho, simula o esforço cognitivo de um humano, assim, uma atividade muitas vezes empírica, onde o trabalhador aloca as cargas no confinamento por "tentativa e erro" pode ser substituída por uma ferramenta, sendo utilizada para otimizar a logística no cenário geral, auxiliando na alocação de cargas onde o conhecimento prévio da disposição dos elementos poupa tempo no carregamento, e maximiza a quantidade de produtos que podem ser levados em um mesmo frete.

Tal conhecimento na maioria das aplicações já existentes provém da utilização de Algoritmos Genéticos como método de solução. Porém, o problema do empacotamento também pode ser representado em uma Busca em Espaços de Estados. Isso sugere a oportunidade de verificação da utilização de algoritmos de busca para resolver o mesmo problema do empacotamento, mostrando a eficiência computacional, considerando aspectos positivos e/ou negativos das técnicas comparadas.

Segundo Marques (2008), o problema do empacotamento é considerado complexo devido as suas muitas variáveis. Portanto é válido sair do conhecimento comum e buscar soluções alternativas para a resolução do mesmo problema.

1.2 OBJETIVOS

1.2.1 Objetivo Geral

Otimizar a resolução do problema de empacotamento bidimensional utilizando algoritmos de busca em espaços de estado com heurística A* e também algoritmos genéticos, traçando comparativos entre as duas técnicas.

1.2.2 Objetivos Específicos

a) Analisar a melhor forma de representação do espaço de busca;

b) Disponibilizar edição de pacotes e confinamento com tamanhos variáveis e prioridade variável;

c) Verificar a eficácia da modelagem do algoritmo, determinando se o mesmo pode ou não resolver o problema do empacotamento, e sua eficiência, determinando se as soluções são de boa qualidade e em um tempo computacional aceitável;

d) Comparar a busca em espaços de estado com heurística A* com algoritmos genéticos na aplicação.

1.3 METODOLOGIA

O trabalho foi desenvolvido através das seguintes etapas:

1 Revisão Bibliográfica: O trabalho será fundamentado através de pesquisa em livros, artigos publicados, trabalhos correlatos e monografias sobre logística e alocação de cargas.

2 Estudo sobre algoritmos de otimização do problema de empacotamento, com foco na heurística que poderá ser utilizada, a fim de atender o primeiro objetivo específico.

3 Elaboração da definição de estado: Elaborar como será a representação dos estados que representam a alocação dos elementos na busca em espaço de estados.

4 Modelagem: Modelagem do sistema através de Diagrama de Caso de Uso, tem por finalidade completar o primeiro objetivo específico.

5 Desenvolvimento: Para o desenvolvimento do projeto será utilizado a linguagem de programação Java.

6 Fase de testes: Esta fase do trabalho, consiste em aplicar testes com diferentes arranjos de cargas, simulando e medindo o desempenho de condições em que o sistema deve operar.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados aspectos teóricos relacionados ao trabalho. A seção 2.1 expõe uma visão geral sobre a área da logística e o problema de empacotamento dos mais diversos tipos de carga. Já a seção 2.2 aborda o conceito de busca heurística.

2.1 LOGÍSTICA

Marques (2008) afirma que a utilização da logística existe desde o início da civilização. Sua origem e desenvolvimento está diretamente ligada às atividades militares e necessidades resultantes das guerras. Parafraseando Coelis (2008), a logística relacionada a estratégia militar caracterizava-se pela movimentação e coordenação das tropas, armamentos e munições para os locais necessários. Todo este processo logístico foi construído com o objetivo de abastecer, transportar e alojar tropas, proporcionando os recursos necessários no local certo e na hora certa.

Atualmente o conceito da logística foi expandido para ser aplicado à gestão empresarial. Novaes (2015) descreve que as indústrias passaram a observar um fator importante na cadeia produtiva, o valor do tempo. Este conceito antes não era considerado como fator competitivo na produção de um produto e, atualmente, é um dos elementos mais críticos do processo logístico.

Bowersox e Closs (2001) afirmam que a logística empresarial se tornou singular e está presente em qualquer atividade de produção ou de marketing. A maioria dos consumidores em nações industriais altamente desenvolvidas, quando vão as lojas, esperam encontrar os produtos recém-fabricados disponíveis. Diante deste cenário, a implementação das melhores práticas logísticas tornou-se uma das áreas operacionais mais desafiadoras e interessantes da administração. Basicamente existem três atividades primárias que caracterizam a logística: transportes, manutenção de estoques e processamento de pedidos. Tendo cada etapa do processo considerada como importantíssima em um ramo extremamente concorrente, o problema de empacotamento torna-se um fator decisivo no valor do tempo e desperdícios de viagem.

Nas palavras de Ballou (2006), o transporte é essencial, pois nenhuma empresa moderna consegue operar sem providenciar a movimentação da sua matéria-prima ou de seus produtos acabados. O transporte também representa um dos elementos mais importantes em termos de custos logísticos para as empresas, visto que a movimentação de cargas pode absorver

de um a dois terços dos custos logísticos totais, além de impactar diretamente no atendimento dos pedidos dos clientes.

A manutenção de estoque, assim como o transporte é de extrema importância, pois é destinado a gerenciar os recursos designados a atender futuras demandas (MENDONÇA, 2013). Enquanto o transporte é responsável por garantir o produto no lugar disponível para o cliente, os estoques são responsáveis por diminuir o tempo do transporte, agindo como amortecedores entre a oferta e a demanda (MENDONÇA, 2013).

O processamento de pedidos tem o seu custo considerado pequeno quando comparados aos custos do transporte ou de manutenção de estoque. Contudo, este processo é considerado uma atividade primária por ser um elemento crítico em termos de tempo necessário para levar os produtos até o cliente (BALLOU, 2006).

2.1.1 Problema de empacotamento

Para Falkenauer (1996), existem diversos tipos de problemas de empacotamento, desde o unidimensional até o tridimensional. O problema de empacotamento unidimensional é o mais antigo de todos e consiste em empacotar n pedaços, onde cada pedaço é um número racional entre 0 ou 1, em um número mínimo de caixas considerando que a soma dos tamanhos dos pedaços deve ser menor ou igual a 1.

Parafraseando Martello e Vigo (1998), o problema de e empacotamento tridimensional, é definido por empacotar um número máximo de caixas dentro de um número mínimo de contêineres.

De acordo com Wascher (2007), o problema de empacotamento é classificado por: dimensão, tipo de tarefa, variedade dos objetos de entrada e variedade dos objetos de saída, onde:

a) dimensão: existem 4 classificações diferentes, sendo elas: unidimensional, bidimensional, tridimensional e n-dimensional onde $n > 3$;

b) tipo de tarefa: indica qual é a tarefa principal do algoritmo, podendo ser elas: entrada grande e uma saída menor ou uma entrada selecionada e uma saída ampla;

c) variedade dos objetos de entrada: indica qual a variedade dos objetos de entrada, podendo ser apenas um objeto, objetos idênticos ou objetos variados;

d) variedade de objetos de saída: indica qual a variedade de objetos de saída, podendo ser objetos diferentes de pouca variedade, muitos objetos de múltiplas variedades, muitos

objetos de relativamente poucas variedades não congruentes e objetos totalmente congruentes.

Segundo Park (1996), o problema de empacotamento pode ainda gerar subproblemas com complexidades ainda maiores, tais como:

a) agendamento de entrega, onde além de considerar o número máximo de empacotamento, deve-se considerar a ordem de saída das caixas;

b) caixas frágeis, que devem ficar sempre acima das demais caixas;

c) transporte através de prateleiras para determinados tipos de mercadoria;

d) entre outros.

Faz-se necessário o uso da tecnologia da informação para resolver problemas complexos deste tipo.

2.1.2 Uso Tecnologia da Informação na logística

Segundo Martello e Vigo (1998), com a globalização e o aumento no número de fornecedores disputando o mesmo nicho, o mercado inflou, forçando mudanças em múltiplos domínios, por exemplo, nos sistemas de transportes, circuitos de distribuição, embalagens, entre outros.

Ainda para o autor, nas últimas décadas intensificou-se a utilização da Tecnologia da Informação em muitas áreas da logística, o que facilitou seu desempenho, principalmente nas ligações no interior das organizações e com os parceiros externos. Foi a Tecnologia da Informação que facilitou a desintermediação, ou seja, reduziu ou até em alguns casos eliminou passos dentro de uma empresa ou organizações externas terceirizadas para fazerem determinado serviço.

Logo, como uma das várias técnicas da Inteligencia Artificial, a Busca em Espaços de Estados, como também os Algoritmos Genéticos, podem ser utilizados para automatizar a procura pelo melhor arranjo de cargas.

2.2 BUSCA EM ESPAÇOS DE ESTADOS

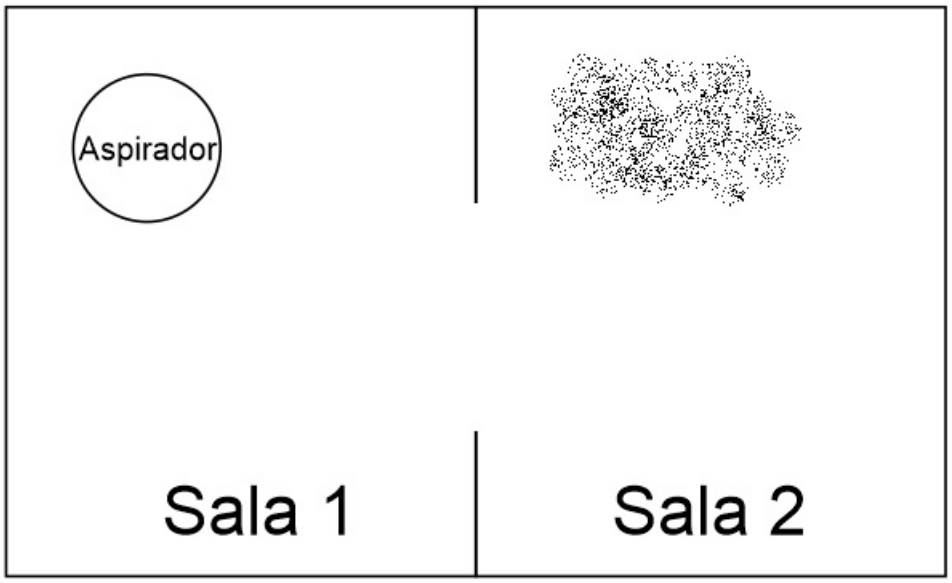
Parafraseando Rich e Knight (2009), um espaço de estados é definido por um conjunto S de estados possíveis e por um conjunto A de ações, denominadas funções de transição, que levam um estado à outro.

Assim, dados um estado inicial representando a configuração corrente do mundo do agente, um conjunto de ações que o agente é capaz de executar e uma descrição do estado alvo

que se deseja atingir, a solução do problema consiste numa sequência de ações que, quando executada pelo agente, transforma o estado inicial num estado meta (RICH; KNIGHT, 2009).

Russel e Norvig (2013) trazem como exemplo didático o "Mundo do Aspirador". Nesse mundo, o agente é um aspirador autômato, cuja função é limpar as salas de um edifício. Em uma versão simplificada, o espaço do aspirador é composto por duas salas, que podem estar limpas ou sujas, como exemplificado na Figura 1. O aspirador pode executar três ações, sendo: EntrarSala1, EntrarSala2 e Aspirar.

Figura 1 – Exemplificação do Mundo do Aspirador



Fonte: Simplificado de Russel e Norvig (2013).

2.2.1 Representação de estados

Estados são representados por estruturas, onde cada componente denota um atributo do estado representado (RICH; KNIGHT, 2009).

No exemplo de Russel e Norvig (2013), cada estado pode ser representado por uma estrutura da forma [X, Y, Z], onde:

- X indica a posição do aspirador e pode assumir 1 ou 2, representando sala 1(1) ou sala 2(2);

- Y indica a condição da sala 1 e pode assumir 0 ou 1, representando se a sala está limpa(0) ou suja(1);

- Z indica a condição da sala 2 e pode assumir 0 ou 1, representando se a sala está limpa(0) ou suja(1);

Logo, se por exemplo, o aspirador se encontrar na sala 1, e ambas as salas estiverem sujas, o estado será representado por [1, 0, 0].

O conjunto de todos os estados possíveis é dado por: S = [1, 0, 0], [1, 0, 1], [1, 1, 0], [1, 1, 1], [2, 0, 0], [2, 0, 1], [2, 1, 0], [2, 1, 1].

2.2.2 Funções de Transição

Para Russel e Norvig (2013), as funções de transição são representações de ações que transformam um estado *s* em um estado *s'*.

As funções de transição podem ser representadas por $(\alpha, s, s') \leftarrow \beta$, onde α é uma ação que transforma o estado *s* no estado *s'*, dado que a condição β esteja satisfeita (RUSSEL; NORVIG, 2013).

Por exemplo, a ação aspirar pode ser representada pelos seguintes operadores:

- aspirar, [1, Y, Z], [1, 0, Z]]Y = 1 (a sala 1 precisa estar suja para ser aspirada);

- aspirar, [2, Y, Z], [2, Y, 0]]Z = 1 (a sala 2 precisa estar suja para ser aspirada).

Geralmente, condições que envolvem apenas teste de igualdade podem ser estabelecidas de forma implícita. Por exemplo, os operadores acima também podem ser codificados como

- (Aspirar, [1, 1, Z], [1, 0, Z])

- (Aspirar, [2, Y, 1], [2, Y, 0])

Assim, o conjunto de ações do exemplo é:

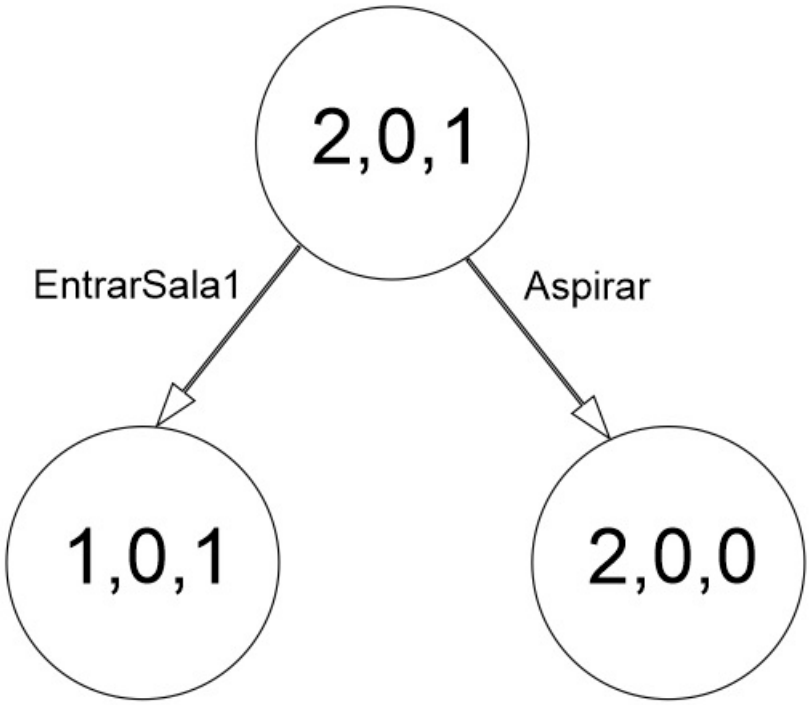
A = (EntrarSala1, [2, Y, Z], [1, Y, Z]), (EntrarSala2, [1, Y, Z], [2, Y, Z]), (Aspirar, [1, 1, Z], [1, 0, Z]), (Aspirar, [2, Y, 1], [2, Y, 0])

2.2.3 Estados sucessores

Segundo Russel e Norvig (2013), dado um estado *s*, seus estados sucessores são todos aqueles que podem ser atingidos, a partir de *s*, pela aplicação de uma das funções de transferência.

Por exemplo, expandindo o estado [2, 0, 1], obtemos como estados sucessores [1, 0, 1] e [2, 0, 0]. Esses estados são gerados pela aplicação dos operadores entrarSala1 e aspirar, como ilustrado na Figura 2.

Figura 2 – Expansão do estado [2, 0, 1]



Fonte: Elaborado pelo autor, 2018.

O operador entrarSala2 não pode ser usado na expansão do estado [2, 0, 1], já que o aspirador já se encontra na sala 2.

2.3 MODELAGEM DO PROBLEMA DE BUSCA

Para Bratko (1990), um problema de busca é especificado através de três componentes:

– um espaço de estados (denotado pelos conjuntos S e A);

– um estado inicial (denotado por um estado particular que seja permitido);

– um conjunto de estados alvos (denotado por um conjunto que contenha um estado alvo).

No exemplo proposto por Russel e Norvig (2013), O Mundo do Aspirador, um possível problema de busca seria o seguinte: dado que inicialmente o aspirador esteja na primeira sala e que ambas as salas estejam sujas, encontre um estado onde ambas as salas estejam limpas.

Têm-se a descrição do problema:

– espaço de estados:

S = [1, 0, 0], [1, 0, 1], [1, 1, 0], [1, 1, 1], [2, 0, 0], [2, 0, 1], [2, 1, 0], [2, 1, 1];

A = (EntrarSala1, [2, Y,Z], [1, Y,Z]),

(EntrarSala2, [1, Y,Z], [2, Y,Z]),

(Aspirar, [1, 1,Z], [1, 0,Z]),

(Aspirar, [2, Y, 1], [2, Y, 0])

– estado inicial: [1, 1, 1];

– estados meta: G = [1, 0, 0], [2, 0, 0].

A solução para um problema de busca consiste numa sequencia de ações que rotulam o caminho que leva do estado inicial a um dos estados meta no espaço de estados do problema (RUSSEL; NORVIG, 2013).

2.4 BUSCA CEGA

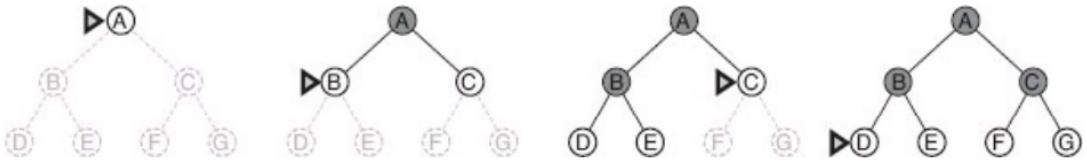
As estratégias de busca cega ou busca não-informada consistem em métodos que sistematizam o comportamento de um algoritmo de busca, sem levar em conta a qualidade da solução encontrada.

Há duas estratégias de busca cega que são bastante utilizadas: busca em largura e busca em profundidade.

2.4.1 Busca em largura

Parafraseando Rich e Knight (2009), na busca em largura, o estado inicial (níıvel 0) é expandido primeiro, sendo seus sucessores posicionados no nível 1 da arvore de busca. Em seguida, cada um dos estados do nível 1 são expandidos, sendo seus sucessores posicionados no nível 2, e assim por diante, de tal forma que todos os estados de um nível *n* sejam expandidos antes daqueles no nível *n* + 1. Em geral, todos os nós em dada profundidade na árvore de busca são expandidos, antes que todos os nós no nível seguinte sejam expandidos, como ilustrado na Figura 3, onde em cada fase, o próximo nó a ser expandido é indicado por um marcador (RUSSEL; NORVIG, 2013).

Figura 3 – Busca em largura em uma árvore binária simples



Fonte: Russel e Norvig (2013).

Porém, Russel e Norvig (2013) apontam o desempenho da busca em largura como

um fator crítico. Uma árvore que gera b nós no primeiro nível, o qual gerará b outros nós, totalizando b ao expoente 2 nós no segundo nível, e assim por diante. No pior caso, o nó alvo se encontrará na profundidade d. Logo, o número total de nós gerados é expresso por:

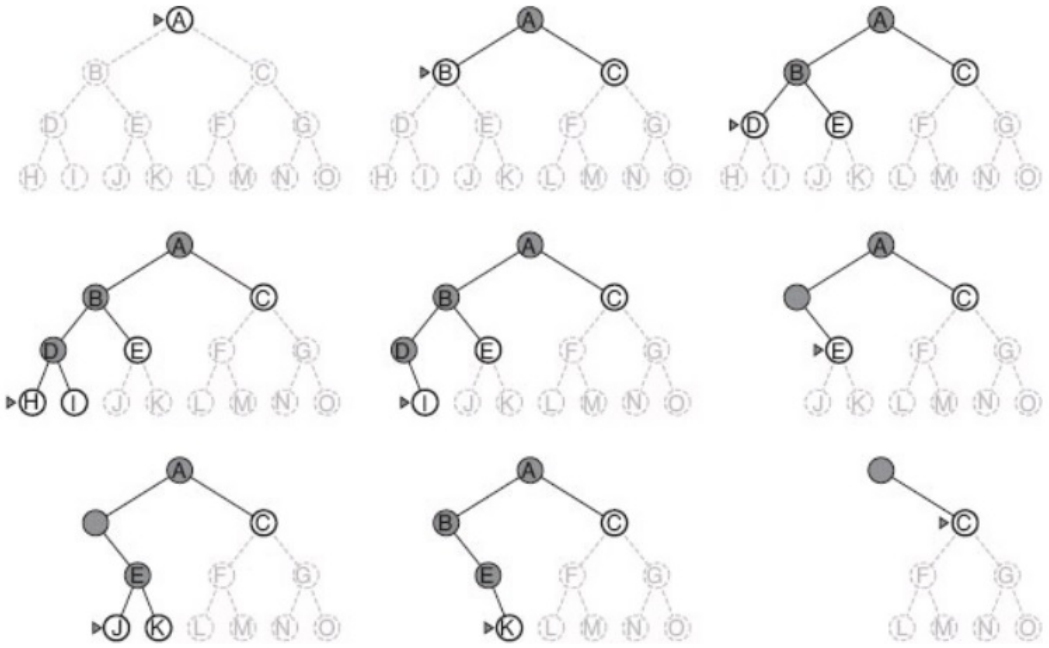
$$b + b^2 + b^3 + ... + b^d = o(b^d)$$

(1)

2.4.2 Busca em Profundidade

Segundo Rich e Knight (2009), na busca em profundidade, expandimos sempre o estado mais à esquerda, no nível mais profundo da árvore de busca até que uma seja encontrada, ou até que um beco seja atingido. Nesse último caso, retrocedemos e reiniciamos a busca no próximo estado ainda não expandido, posicionado mais a esquerda, no nível mais profundo da árvore, como Exemplificado pela Figura 4, onde a região inexplorada é mostrada em cinza-claro. Os nós na profundidade 3 não têm sucessores e M é o único nó alvo.

Figura 4 – Busca em profundidade em uma árvore binária



Fonte: Russel e Norvig (2013).

Aparentemente, a Busca em Profundidade não tem vantagem sobre a Busca em Largura, porém, Cormen (2009) aponta que em uma Busca em Profundidade em árvore, precisa-se armazenar apenas um único caminho da raiz até um nó folha, juntamente com os nós irmãos remanescentes não expandidos para cada nó no caminho. Uma vez que um nó é expandido, ele

pode ser removido da memória, tão logo todos os seus descendentes tenham sido completamente explorados

2.5 ESTRATÉGIAS DE BUSCA HEURÍSTICA

Heurística é uma palavra derivada do grego heureka, comumente conhecida por ser exclamada por Arquimedes de Siracusa ao descobrir como calcular o volume de ouro da coroa do rei Hierão. Parafraseando Cruz, Souza e Mine (2012), as heuríısticas são técnicas que visam a obtenção de soluções de boa qualidade em um tempo computacional aceitável, utilizando-se de um conhecimento prévio sobre o problema.

As estratégias de busca cega encontram soluções testando e expandindo estados, sistematicamente. Infelizmente, além de serem ineficientes, essas estratégias não garantem encontrar soluções de custo míinimo

Esta seção mostra como uma estratégias de busca informada, ou seja: que utilizam um conhecimento específico além da definição do problema, podendo encontrar soluções de forma mais eficiente do que uma estratégia de busca sem informação.

2.5.1 Função Heurística

Uma função heurística para Rich e Knight (2009) é uma função que estima o custo mínimo de um caminho desconhecido que leva de um determinado estado a um estado alvo.

Uma função heurística pode ser qualquer função h que apresente as seguintes propriedades:

$$h(s) = 0 \; se \; a \; somente \; se \; s \; \in G \tag{2.1}$$

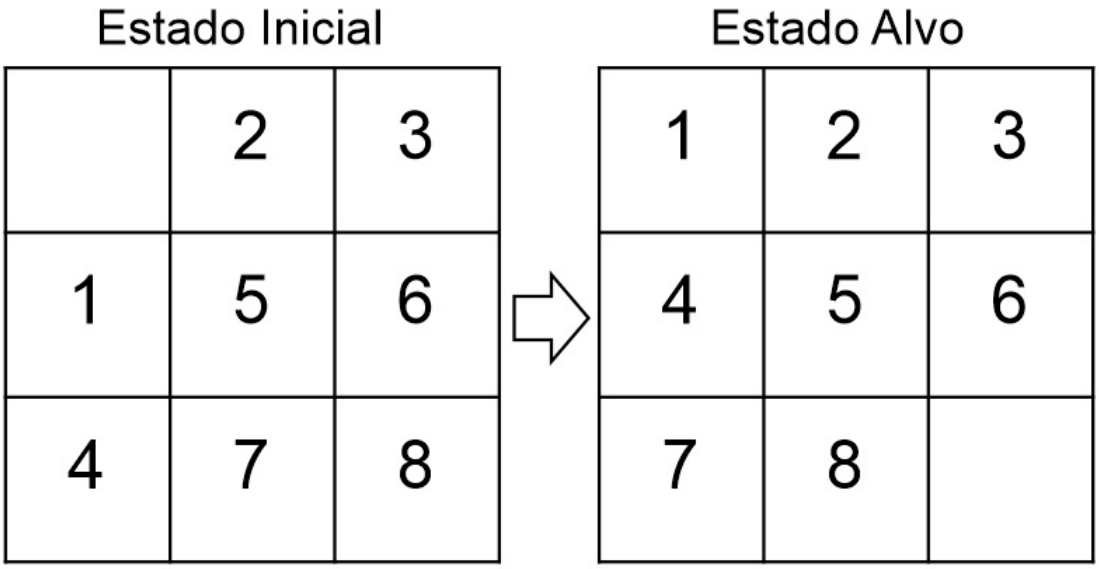
$$para \; todo \; s \; \in S, \; h(s) \; \leq h*(s) \tag{2.2}$$

Essas propriedades garantem que a estimativa dada pela função heurística seja admissível, ou seja, que nunca superestime o custo real de uma solução. As funções heurísticas dependem do domínio de aplicação, bem como da criatividade e experiência de quem a projeta (RICH; KNIGHT, 2009).

Como exemplificação, podemos utilizar o Puzzle 8, que consiste em movimentar as peças do quebra-cabeça horizontal ou verticalmente, para ocupar a posição vazia adjacente à peça, de modo que a configuração final seja alcançada, conforme representado na Figura 5.

Expandindo o estado inicial, como representado na Figura 6, temos: usando uma

Figura 5 – Representação do estado inicial e estado alvo do Puzzle de 8 peças



Fonte: Elaborado pelo autor, 2018.

função heurística, o algoritmo de busca deveria expandir o melhor entre esses dois estados sucessores. Mas como decidir qual deles é o melhor? Uma possibilidade é verificar o quão longe cada peça encontra-se de sua posição na configuração final e apontar como melhor estado aquele cuja soma das distâncias é menor. Por exemplo, adotada a função heurística como sendo a soma das distancias dos elementos até sua posição final temos:

Para s1, as peças 1, 2, 3, 5 e 6 já se encontram em suas posições finais. Para as peças 4, 7 e 8, a distância é 1. Portanto, h(s1) = 3.

Para s2, seguindo o mesmo procedimento, constatamos que para as peças 2, 4, 7 e 8, a distância é 1. Portanto, h(s2) = 4.

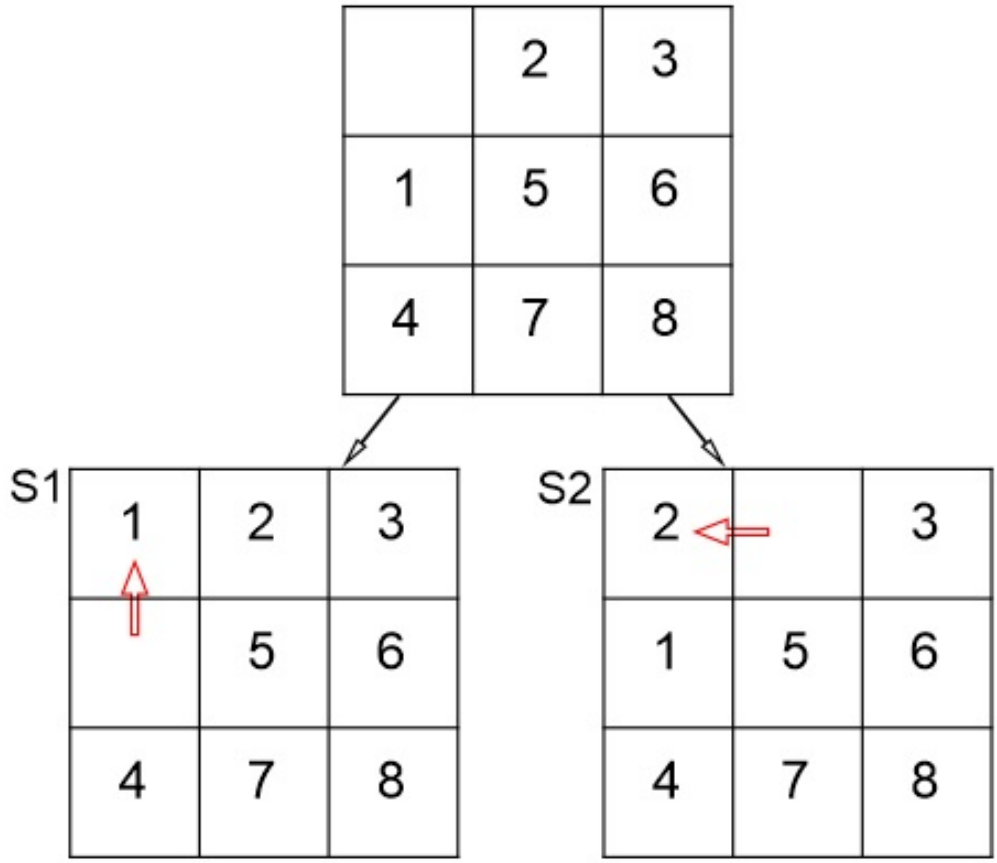
Esses valores indicam que uma solução a partir do estado s1 pode ser obtida com no mínimo mais três expansões, enquanto que uma solução a partir de s2 requer no mínimo mais quatro expansões. Evidentemente, o algoritmo de busca deve expandir o estado s1.

2.5.2 Busca A*

Segundo Russel e Norvig (2013), a forma de solução mais amplamente conhecida da busca de melhor escolha é a busca A* (pronuncia-se “busca A estrela”). Ela avalia os nós através da combinação de g(n), o custo para alcançar o nó, e h(n), o custo para ir do nó ao objetivo:

$$f(n) = g(n) + h(n) \tag{2.3}$$

Figura 6 – Expansão do estado inicial no Puzzle de 8 peças



Fonte: Acervo do autor, 2018.

Uma vez que g(n) dá o custo do caminho desde o nó inicial até o nó n e h(n) é o custo estimado do caminho de menor custo de n até o objetivo, temos que f(n) representa o custo estimado da solução de menor custo através de n.

Logo, a busca A estrela é uma busca heurística, diferenciando-se apenas pela função que prioriza o nó a ser expandido.

Como exemplo, pode-se retomar o Puzzle de 8 peças, vide Figura 6, onde uma possível função heurística A* seria a soma das peças fora da posição final com a distância de cada peça até a sua posição final.

2.6 ALGORITMOS GENÉTICOS

Segundo Linden (2008), Algoritmos Genéticos (AG) são um ramo de algoritmos evolucionários e podem ser definidos como técnicas de busca baseadas na metáfora do processo biológico de evolução natural. Em muitos problemas o caminho até o objetivo torna-se irrele-

vante, sendo o estado final do algoritmo o mais importante (RUSSEL; NORVIG, 2013). Para estes problemas é introduzido o conceito de busca local, que operam baseados em um único estado atual, onde em geral se movem apenas para os vizinhos desse estado.

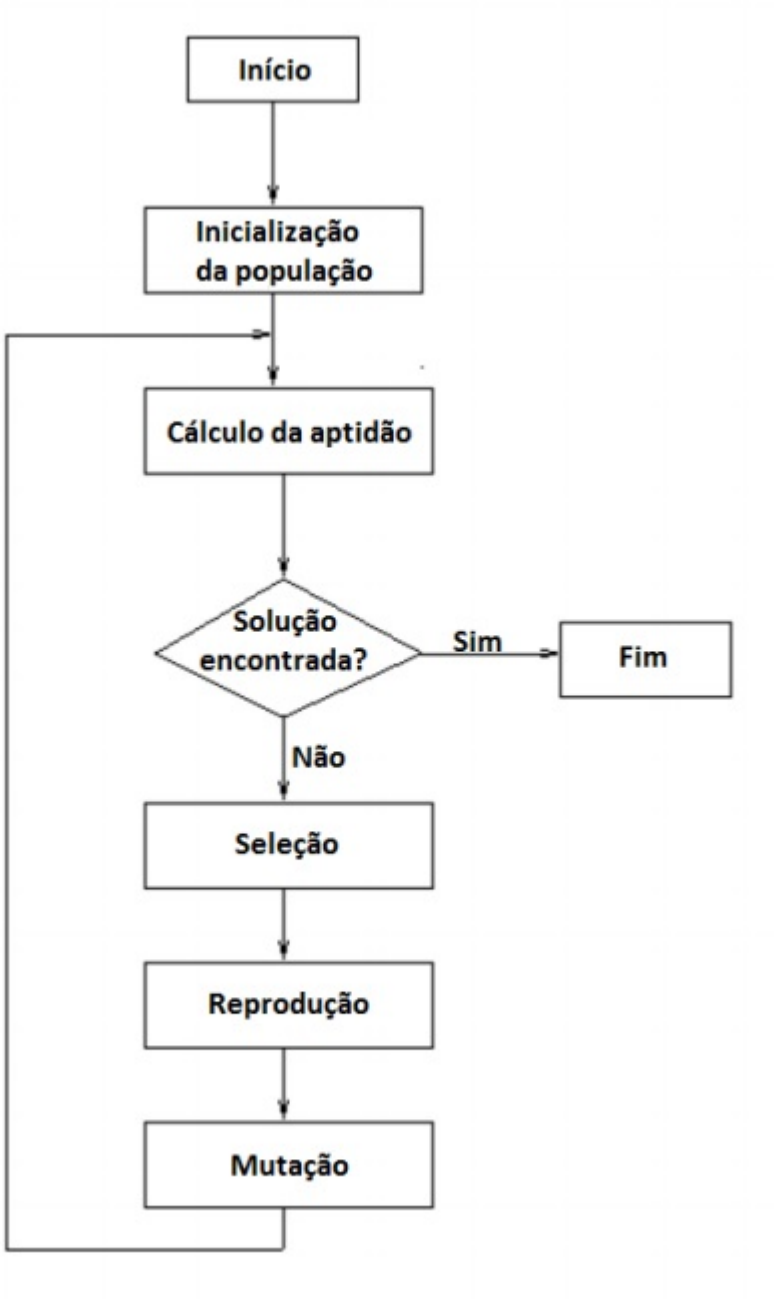
Além de encontrar objetivos, os algoritmos de busca local são indicados para problemas de otimização, onde o objetivo é encontrar o melhor estado de acordo com a função objetivo (RUSSEL; NORVIG, 2013). O Algoritmo Genético utiliza técnicas heurísticas de forma a encontrar o máximo de uma função, ou seja, a solução o mais próximo possível da solução ótima, mesmo em casos em que a solução ótima não seja possível de ser alcançada, como por exemplo problemas de agendamento de horário, otimizações de metodologias, problemas de paletização e o próprio problema de empacotamento.

As técnicas heurísticas são regras gerais de influência utilizadas para julgamentos em decisões de incertezas (TONETTO, 2006). Linden (2008) afirma também que os AG utilizam tais técnicas para tomada de decisão que não visam necessariamente encontrar a solução ótima de um problema e, quando conseguem, nem sempre podem repetir o feito.

A Figura 7 representa um fluxo de uma solução AG simples. Na imagem é possível verificar que o algoritmo inicia adicionando os indivíduos na população e, daí em diante, entra num *loop* para verificar se a solução foi encontrada. O *loop* funciona nos seguintes passos: verifica se a solução já foi encontrada (através do cálculo de aptidão); enquanto a solução não for encontrada o algoritmo executa a seleção, reprodução e mutação para então executar o cálculo de aptidão e assim sucessivamente até encontrar a solução Miranda (2007).

As etapas descritas na Figura 7 serão apresentadas nas subseções seguintes. Segundo Poli et al. (2008), estas etapas são separadas em indivíduo, população, função objetivo, seleção, reprodução e mutação.

Figura 7 – Exemplo de AG

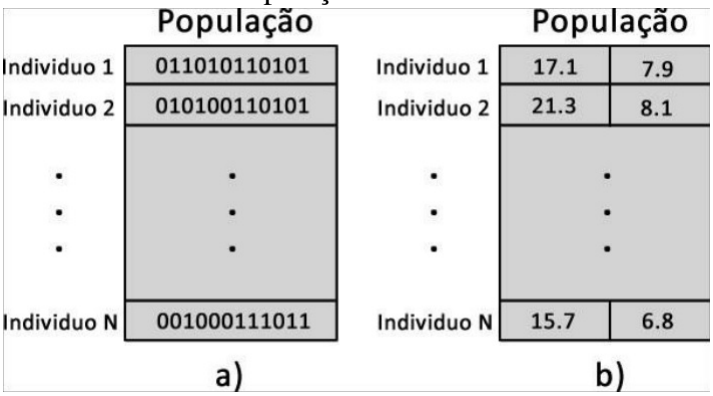


Fonte: adaptado de Miranda (2007).

2.6.1 Indivíduo e População

A população é definida por um agrupamento de indivíduos. A Figura 8 ilustra duas populações de diferentes indivíduos. É possível perceber que cada indivíduo possui seus próprios genes e uma população pode ser composta por n indivíduos. Segundo Russel e Norvig (2013), o indivíduo é representado por uma cadeia sobre um alfabeto finito, normalmente de valores binários.

Figura 8 – Exemplo de Indivíduo e População



Fonte: adaptado de Pacheco (2016).

2.6.2 Função objetivo

Identifica a natureza do problema, como por exemplo, se deseja maximizar ou minimizar o resultado de um determinado algoritmo. A função objetivo tem por objetivo encontrar o melhor estado dentro de uma população (RUSSEL; NORVIG, 2013).

Russel e Norvig (2013) identificam um exemplo de função objetivo na própria natureza, denominada adaptação reprodutiva, onde a natureza otimiza-se do ponto de vista da evolução de Darwin. Esta função objetivo pode ser chamada também de decisão, função avaliadora, função de adequação entre outras. Um exemplo de função objetivo seria a minimização de materiais utilizados para determinado processo.

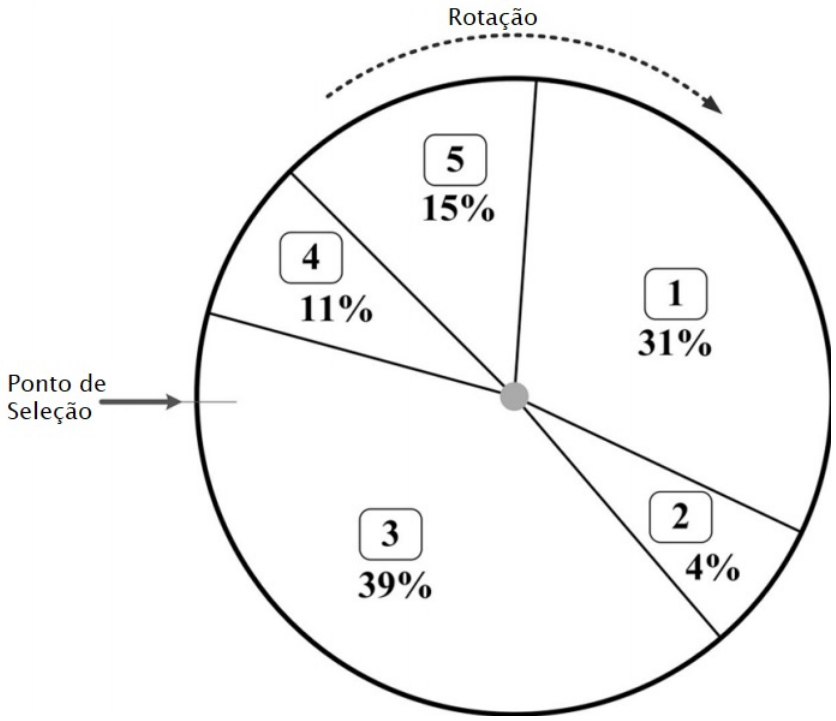
2.6.3 Seleção

Seleciona amostragem de indivíduos para comporem um conjunto de pais para a próxima geração. Segundo Silva e Soma (2003a), existem duas formas comuns de executar o método de seleção. A primeira delas é através do método de roleta viciada ou tendenciosa, que possibilita que os indivíduos melhor avaliados tenham mais probabilidade de escolha, através da seleção baseada em sua função de objetivo. Outro método comum é a seleção por critérios

elitistas, onde os melhores indivíduos são sempre escolhidos, sem a necessidade da roleta para randomizar.

Poli et al. (2008) defende que existem inúmeros algoritmos de seleção diferentes, sendo o mais comum denominado seleção por torneio, que escolhe um número de indivíduos randomicamente da população, para então serem comparados entre si, onde o melhor deles é escolhido para ser o pai da próxima geração, desta maneira, não sendo necessário roleta ou classificação por resultado da função de objetivo. A Figura 9 representa graficamente a seleção por roleta. Os indivíduos são divididos em 5 grupos onde os grupos que têm maior aptidão possuem mais chances de serem selecionados. Feito esta divisão, é girada a roleta aleatoriamente e onde o ponto de seleção indica será selecionado.

Figura 9 – Exemplo de seleção por roleta viciada



Fonte: adaptado de Pacheco (2016).

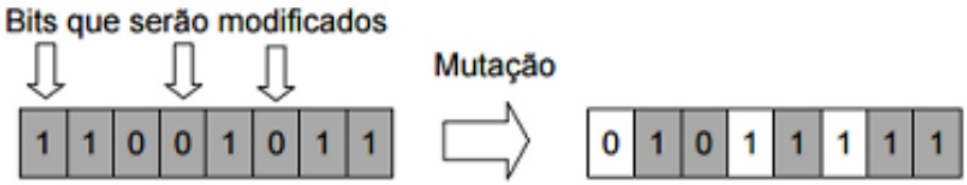
2.6.4 Mutação

No processo de mutação é modificado aleatoriamente alguma característica do cromossomo sobre o qual é aplicada, conforme pode ser visualizado na Figura 10. Esta troca é um importante fenômeno para a diversidade e evolução, e acaba por criar novas características que não existiam ou existiam em pequena quantidade em uma população (POZO et al., 2005).

O operador de mutação é aplicado aos indivíduos com uma taxa de mutação ge-

ralmente pequena, pois a mutação excessiva em uma população pode gerar várias soluções irregulares, impedindo a evolução da população (POZO et al., 2005). Para Linden (2008), taxas de mutações muito altas, fazem com que o AG se comporte de forma estranha, agindo como uma técnica aleatória, na qual a solução pouco depende do mecanismo do algoritmo em si.

Figura 10 – Exemplo do processo de mutação

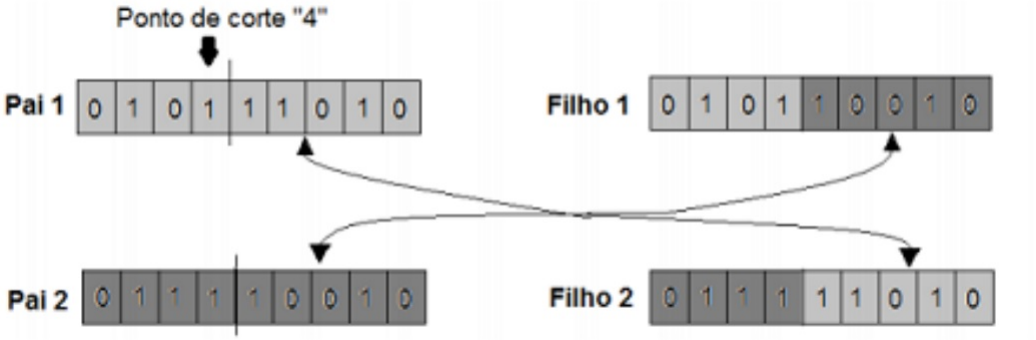


Fonte: adaptado de Linden (2008).

2.6.5 Reprodução ou Cruzamento

É responsável pela cópia de características da geração atual para servir de base para a próxima geração que poderá ser alterada através da mutação. O operador de cruzamento é responsável pela reprodução dos cromossomos e troca de genes, garantindo a diversidade de constante evolução da população. Esta etapa basicamente consiste em selecionar pontos de cruzamento dos cromossomos pai, separar estes cromossomos, e trocar as partes destes cromossomos para posteriormente realizar a mutação (BUENO, 2009). A Figura 11 representa o modelo de cruzamento.

Figura 11 – Exemplo de cruzamento



Fonte: adaptado de Linden (2008).

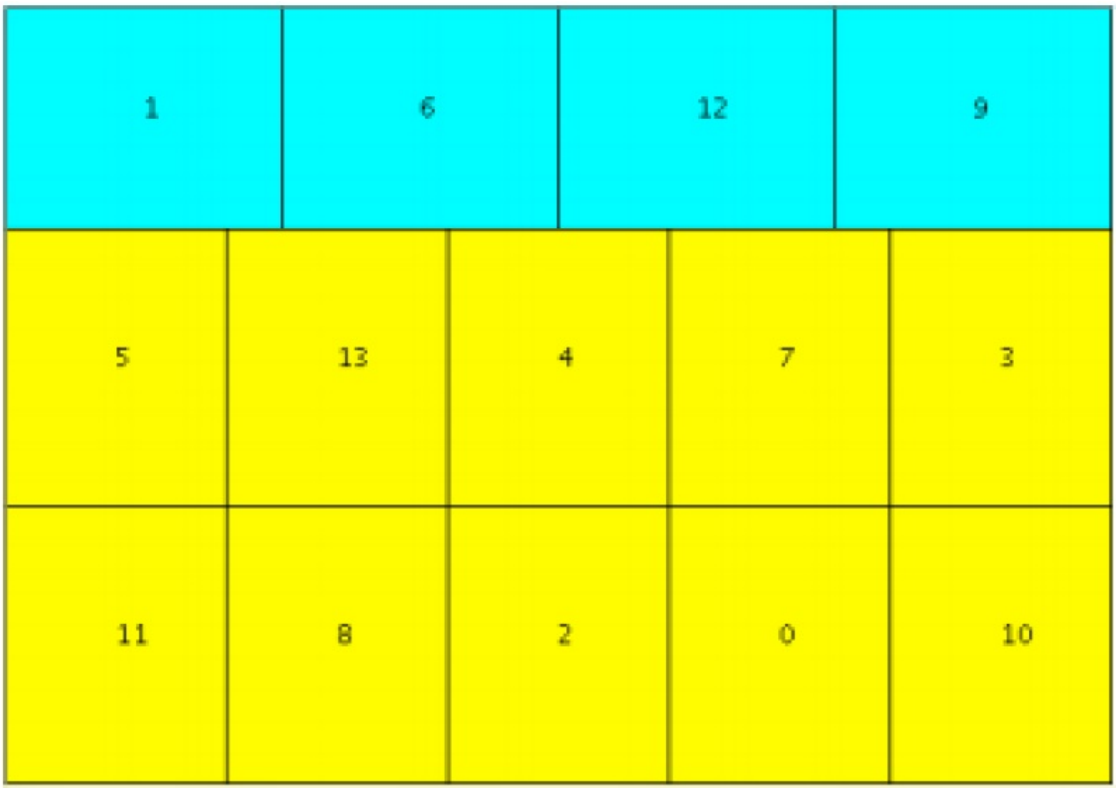
3 TRABALHOS CORRELATOS

3.1 SIP – SISTEMA INTELIGENTE DE CARREGAMENTO DE PALETES

Cavalcanti (2009) desenvolveu um sistema para mostrar o posicionamento das caixas sobre um palete, otimizando a sua ocupação. Este problema é conhecido como problema de carregamento de paletes. Diferentemente da implementação alvo da presente implementação, que utiliza Algoritmos de Busca Heurística, Cavalcanti (2009) optou por uma abordagem utilizando Algoritmos Genéticos.

No algoritmo desenvolvido por Cavalcanti (2009) os indivíduos são representados pelo conjunto de caixas sobre o palete. Uma caixa constitui um gene com dois atributos representando as suas coordenadas (x,y) e uma representando a sua orientação (“0” = horizontal e “1” = vertical). A função objetivo do algoritmo é calculada pela quantidade de sobreposição de caixas e o percentual da área do palete ocupada pelas caixas. A Figura 12 apresenta um resultado obtido através do sistema desenvolvido para o PLP, Package Load Problem, onde estão dispostas 14 caixas, ocupando 100% da área de um palete.

Figura 12 – Exemplo de resultado do SIP



Fonte: Cavalcanti (2009).

Inicialmente foram realizados pequenos experimentos para determinar o melhor

método de cruzamento para o algoritmo. Após determinado o melhor método de cruzamento foram realizados experimentos utilizando três tamanhos de população (20, 40 e 60 indivíduos), limitados a 2000 gerações. Os resultados foram obtidos através de uma média do resultado de 30 instâncias de testes.

Comparando os resultados o autor concluiu que, apesar de ter um tempo maior de execução o conjunto de testes utilizando uma população de 60 indivíduos convergiu mais rapidamente para a solução ótima. Este também foi o único considerado totalmente válido, pois atingiu 0 de média de sobreposição nas simulações realizadas.

3.2 A GENETIC ALGORITHM FOR THE THREE-DIMENSIONAL BIN PACKING PROBLEM WITH HETEROGENOUS BINS

Li, Zhao e Zhang (2008) implementam um algoritmo genético para o problema de empacotamento considerando caixas heterogêneas, ou seja, caixas de tamanhos diferentes. Novamente, o autor optou pela implementação de um Algoritmo Genético, no caso, apresentado com uma nova heurística de embalagem, denominada Best Match Heuristic Packing Strategy.

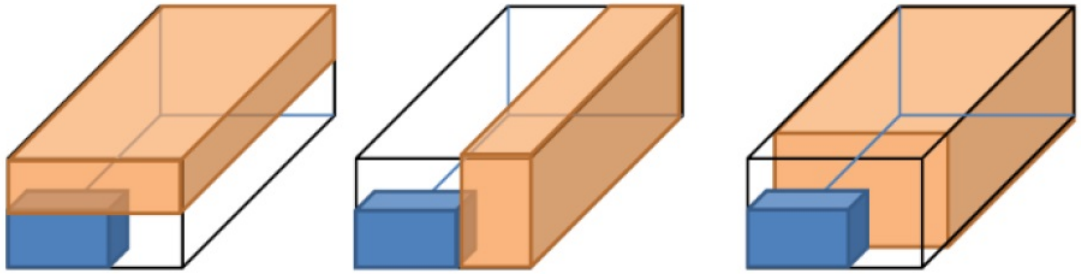
Foram analisados os pontos fortes e fracos do algoritmo e suas variações, para então ser proposto a solução final do algoritmo. Os principais pontos fracos encontrados no algoritmo foram que estas heurísticas empregadas nele sempre observam pelo espaço mínimo contra maior profundidade para trocar o item no estado atual. Sendo assim, a coordenada x sempre fica predominante sobre as demais coordenadas; o item ou o espaço é primeiro determinando e somente depois é selecionado baseado em certas prioridades (LI; ZHAO; ZHANG, 2008).

Para a nova abordagem, foi utilizado o conceito do máximo espaço disponível EMS (Empty Maximal Spaces) para representar os espaços vazios nos contêineres. Na Figura 13 é possível verificar o tratamento implementado, onde ao colocar a caixa menor no canto esquerdo de um container, uma lista é gerada ordenada pelo maior espaço vazio contida no container.

Depois do processo de EMS é escolhido a prioridade dentre os espaços previamente selecionados. Primeiro são comparadas as menores coordenadas e a coordenada com as menores ganha prioridade no algoritmo.

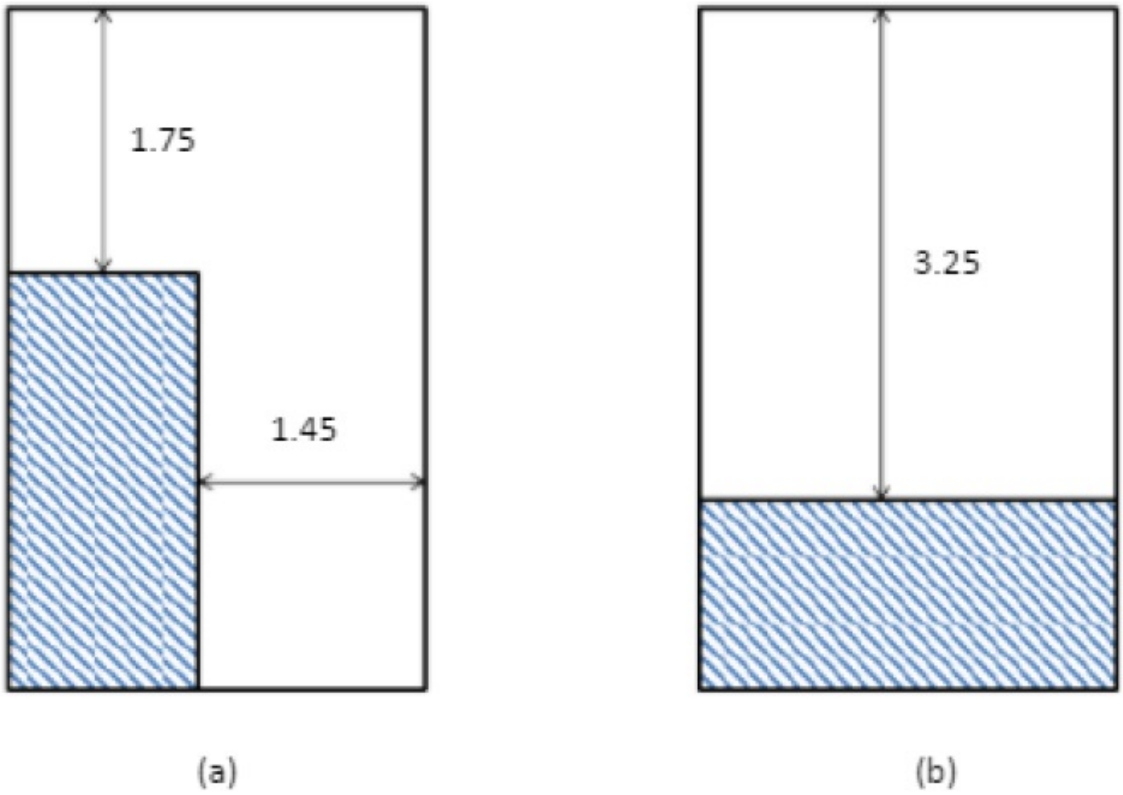
Selecionado o espaço, deve-se fazer a seleção de onde alocar a caixa. A posição que ganha prioridade de seleção é a que gera a menor margem em uma direção. A Figura 14 mostra um exemplo, onde a alocação a possui uma margem de 1.45 - 1.75 e a alocação b possui uma margem de 3.45 apenas em uma direção. Neste caso, o algoritmo escolhe a alocação b.

Figura 13 – Representação do EMS



Fonte: Li, Zhao e Zhang (2008).

Figura 14 – Exemplo de seleção de alocação



Fonte: Li, Zhao e Zhang (2008).

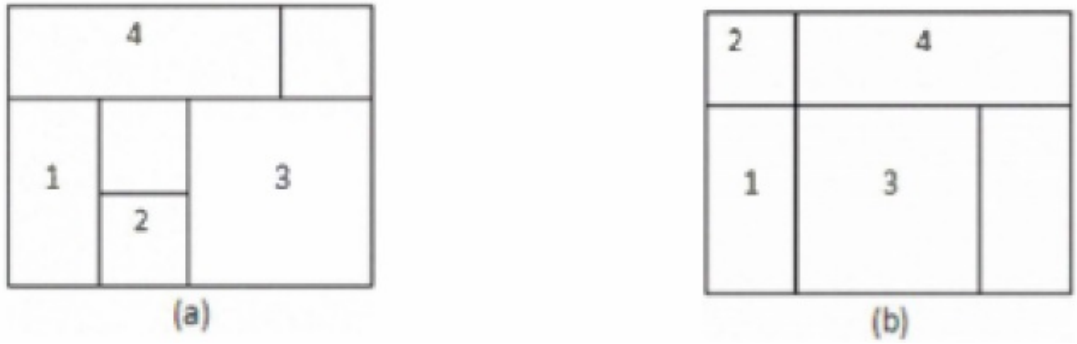
O algoritmo foi testado baseado primeiramente em 12 instâncias industriais e depois testado em instâncias geradas aleatoriamente. Os resultados demonstram que podem ser encontradas ótimas soluções em tempos razoáveis, onde o método proposto tem uma melhor performance do que o modelo convencional, porém possui algumas falhas em relação a encontrar uma solução ótima para instâncias com tamanhos moderados. Foi observado também que utilizando o método EMS para gestão de espaços vazios é mais eficiente e efetivo do que usar o método corner points (LI; ZHAO; ZHANG, 2008).

3.3 A HYBRID GENETIC ALGORITHM FOR 3D BIN PACKING PROBLEMS

Wang e Chen (2010) implementam um algoritmo híbrido genético focado no problema de empacotamento, em que os pacotes fossem reposicionados dentro do container mesclando seus espaços vazios. Foi utilizado uma extensão do algoritmo Deepest Bottom Left With Fill (DBLF), para otimizar o espaço dentro do container, tendo em vista que qualquer mesclagem de espaços vizinhos dentro de um recipiente é benéfico para o encaixe do próximo pacote (WANG; CHEN, 2010).

Na Figura 15 ilustra-se uma aplicação simples e estendida do algoritmo DBLF, onde o container a possui 4 pacotes alocados de forma randômica, restando 2 espaços vazios distintos e, no container b, é representado o resultado do algoritmo DBLF estendido, onde os mesmos pacotes de a são reposicionados, otimizando-o, de modo que os espaços vazios distintos fiquem mesclados

Figura 15 – Exemplo de mesclagem de espaços vazios



Fonte:Wang e Chen (2010).

Os comportamentos dos algoritmos foram simulados em uma série de problemas de testes, baseados no método de geração de Bischoff/Ratcliff, onde os tamanhos dos pacotes foram definidos com largura=233, comprimento=235 e altura=1200, e o número total de pacotes foi considerado 60. Foram definidos então três cenários de teste, o primeiro onde os itens não rotacionam, o segundo teste em que os itens podem ser rotacionados horizontalmente (2 rotações permitidas) e o terceiro teste em que os itens podem ser rotacionados como quiserem (até 6 rotações permitidas). A Figura 16 exibe o resultado dos testes, onde a coluna HGA (Hybrid Genetic Algorithm) significa os percentuais de acerto do trabalho anterior a este (considerando margem de erro) e a coluna HGAI (Hybrid Genetic Algorithm Improved) representa os resultados deste trabalho. É possível observar que nos três ambientes de testes (rotações) este trabalho tornou-se mais eficiente e com menor margem de erro, dando destaque para o teste com 2 rota-

ções, onde o percentual de acerto foi de 81,30% para 91,16% e a margem de erro caiu de 3,33% para 0,71%.

Figura 16 – Exemplo de mesclagem de espaços vazios

Test Instances	Algorithms		
	HGA	HGAI	*t*-test result
non-rotation	80.43%±1.95%	86.05%±0.85%	*s*+
2-ways rotation	81.30%±3.33%	91.16%±0.71%	*s*+
6-ways rotation	81.06%±3.84%	92.88%±1.38%	*s*+

Fonte: Wang e Chen (2010).

Diante desses trabalhos, é possível verificar que todos utilizam Algoritmos Genéticos. Isso sugere a oportunidade de verificação da utilização da Busca em Espaços de Estados para resolver o mesmo problema do empacotamento.

4 DESENVOLVIMENTO

Neste capítulo são descritas as especificações formais da aplicação proposta. São descritos os requisitos funcionais e não funcionais do programa, além das especificações do projeto, através de diagramas de caso de uso, diagramas de classe, e Modelo de Entidade Relacional.

4.1 ESPECIFICAÇÕES FORMAIS

4.1.1 Requisitos Funcionais

Nesta sessão será apresentado os Requisitos Funcionais que deverão ser atendidos pela aplicação, descritos no Quadro 1.

Quadro 1 – Requisitos Funcionais

RF01	O software deve permitir a seleção de tamanho do confinamento.
RF02	O software deve permitir a seleção de tamanho e prioridade dos pacotes.
RF03	O software deve otimizar a alocação de pacotes dentro do confinamento, alocando a maior quantidade de pacotes possíveis.
RF04	O software deve permitir a seleção da prioridade dos pacotes, e manter os com prioridade maior, mais próximos da saída do confinamento.
RF05	O software deve disponibilizar uma tela para exibir a disposição de pacotes encontrada pela busca.

4.1.2 Requisitos Não Funcionais

Nesta sessão será apresentado os Requisitos Não Funcionais que deverão ser atendidos pela aplicação, descritos no Quadro 2.

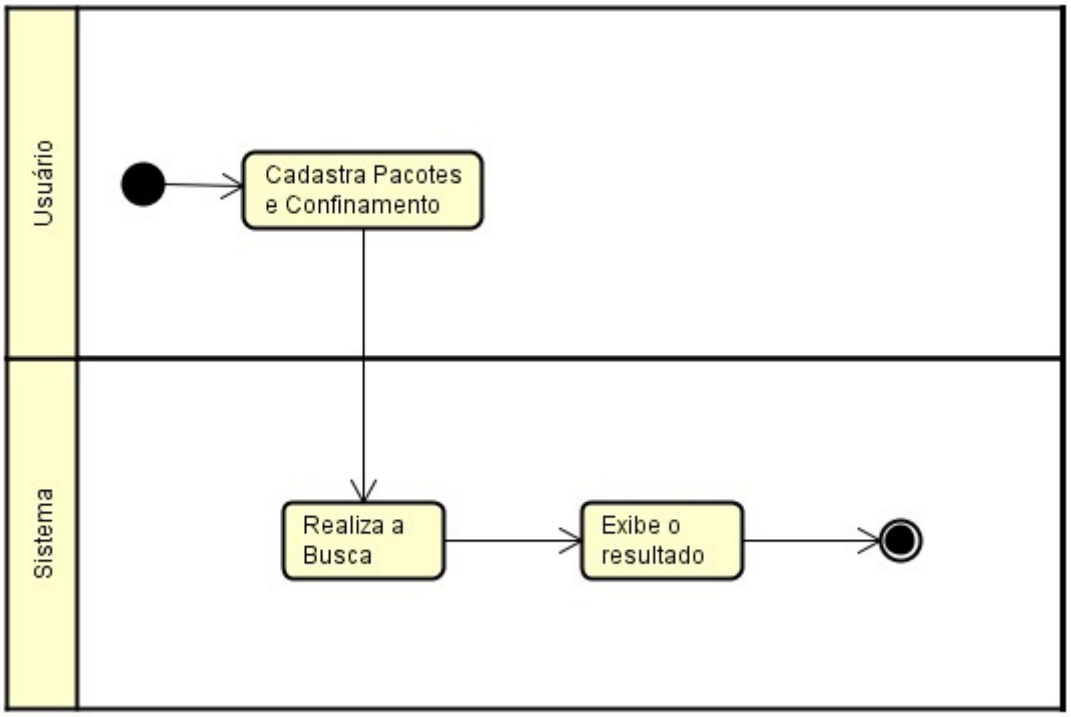
Quadro 2 – Requisitos Não Funcionais

RNF01	O software deve ser implementado utilizando busca heurística A$*$ e também Algoritmos Genéticos.
RNF02	O software deve utilizar a linguagem de programação Java.

4.1.3 Diagrama de Atividade

O diagrama de atividade tem como função demonstrar o fluxo das atividades empregados para fazer a modelagem de aspectos dinâmicos do sistema. Uma atividade é uma execução não atômica em andamento em uma máquina de estados tais quais acabam resultando em alguma ação, formada pelas computações atômicas executáveis que resultam em uma mudança de estado do sistema ou o retorno de um valor. (FERNANDES, 2016). A Figura 17 mostra as atividades do software. Inicialmente o usuário cadastra as dimensões dos pacotes e do confinamento, quando pronto, o sistema realiza a busca e exibe o resultado ao usuário.

Figura 17 – Diagrama de Atividade



Fonte: Acervo do Autor, 2018.

4.1.4 Diagrama de casos de uso

Os casos de uso que são apresentados nesta seção, estão relacionados aos requisitos funcionais descritos no Quadro 8, onde o usuário pode cadastrar o confinamento e os pacotes, realizar a busca e ter o retorno da melhor disposição graficamente, sendo que o detalhamento desses requisitos é relatado do Quadro 3 ao Quadro 4. Tendo como finalidade descrever um cenário que mostram as funcionalidades da aplicação.

Quadro 3 – UC01 – Cadastro Confinamento e Pacotes

Atores	Administrador
Pré-condições	Acessar a página principal da aplicação
Fluxo Principal	
1	O usuário seleciona as dimensões em “Cadastro Confinamento”
2	O usuário insere as dimensões e prioridade de cada pacote em "Cadastro Pacote"
3	O usuário seleciona a opção "Inserir Pacote"

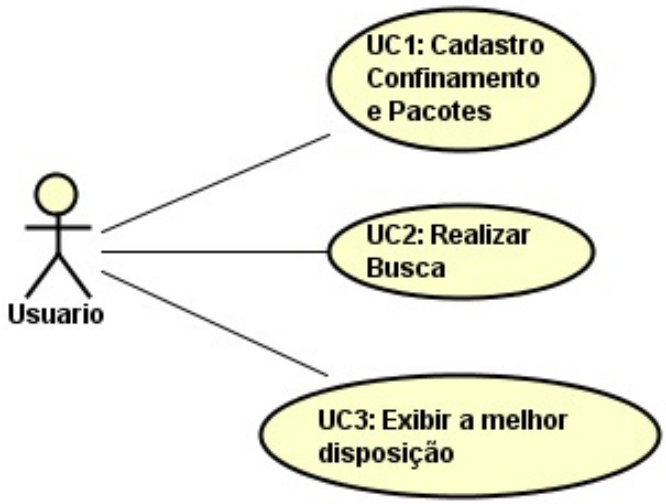
Quadro 4 – UC02 – Realizar Busca

Atores	Administrador
Pré-condições	Pacotes e confinamento devidamente cadastrados
Fluxo Principal	
1	O usuário altera os parâmetros da busca a ser realizada, sendo genética ou heurística
2	O usuário seleciona a opção “Busca” no campo de Busca Heurística ou no campo Algoritmo Genético
3	O sistema faz a busca pelo melhor arranjo
4	O sistema retorna ao usuário o resultado encontrado

No Quadro 3 é representado o cadastro do confinamento e dos pacotes. No quadro 4 é representado o diagrama para a busca.

A representação gráfica é apresentada na Figura 18.

Figura 18 – Diagrama de Caso de Uso

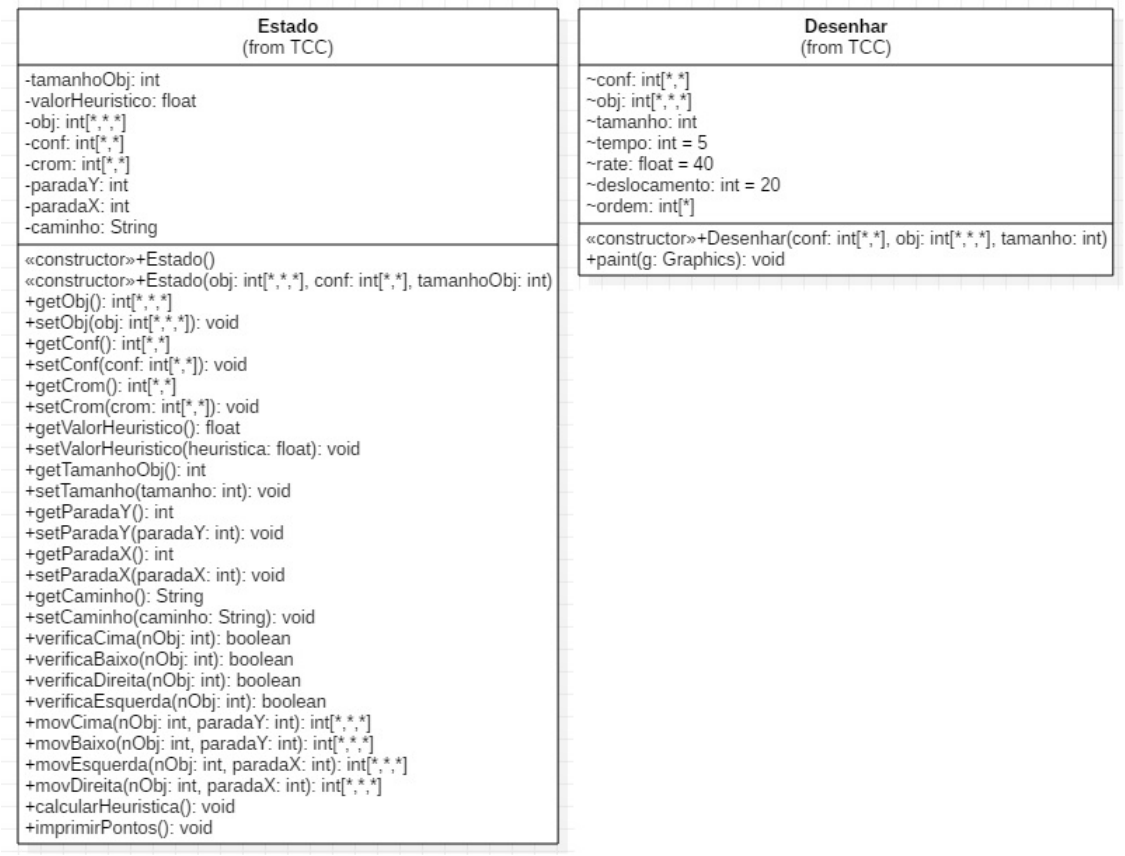


Fonte: Acervo do Autor, 2018.

4.1.5 Diagrama de Classes

As classes que representam a estrutura e relações das classes são ilustradas na Figura 19.

Figura 19 – Diagrama de Classe

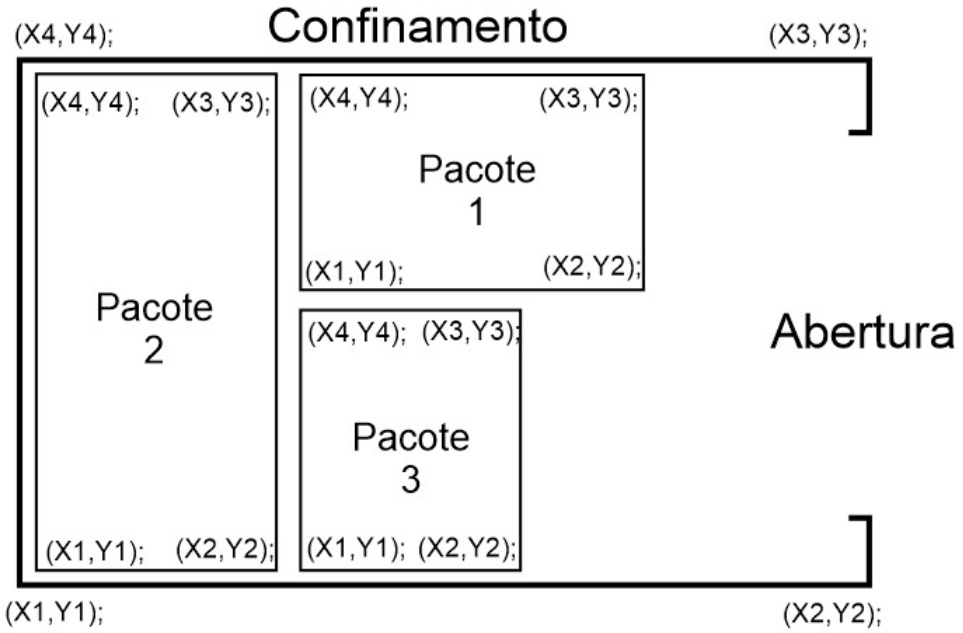


Fonte: Acervo do Autor, 2018.

4.2 IMPLEMENTAÇÃO

Nesta seção será descrito as técnicas e ferramentas utilizadas no desenvolvimento, afim de detalhar a implementação do projeto. Inicia-se por assuntos da estrutura do projeto, seguidos pela implementação do Busca em Espaços de Estados e Algoritmos Genéticos.

Figura 20 – Exemplo de confinamento contendo pacotes



Fonte: Acervo do Autor, 2018.

O projeto foi desenvolvido na linguagem Java (versão 1.8 do JDK) com auxílio do ambiente de desenvolvimento Netbeans 8.2 e sistema operacional Windows 10.

4.2.1 Representação do Estado

Contextualizando com a realidade, o propósito deste trabalho é encontrar a melhor forma de arranjo de pacotes dentro de um container, como ilustrado na Figura 20, de forma a maximizar o espaço útil de carga.

A primeira etapa do software foi desenvolver um mecanismo onde fosse possível representar as formas geométricas referentes aos pacotes e ao confinamento, e assim possibilitar a alocação de caixas dentro de um contêiner.

A solução adotada, foi armazenar os vértices de todos os elementos em um único vetor de 3 dimensões, representado por: `obj[A][B][C]` ;

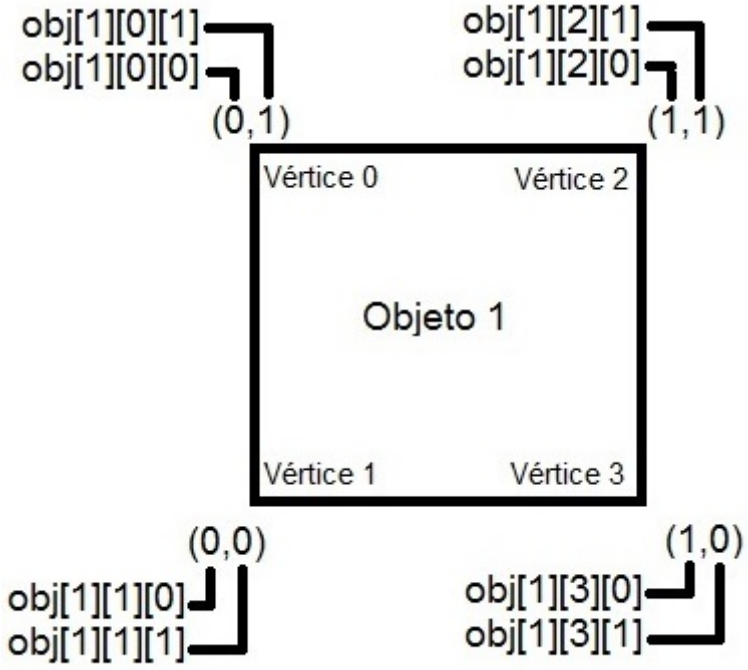
→ **A** Indica o número do objeto em que o vértice pertence;

→ **B** Como os objetos são retangulares ou quadrados, Indica de 0 a 3 o vértice ao qual a coordenada pertence;

→ **C** Indica a qual coordenada o valor armazenado pertence: 0 para X, 1 para Y.

A Figura 21 representa a utilização do vetor `obj`.

Figura 21 – Armazenamento das coordenadas dos pacotes



Acervo do Autor, 2018.

O confinamento, por se tratar também de um objeto retangular ou quadrado, porém único, foi representado por: `conf[B][C]`, seguindo a mesma descrição anterior. A Figura 22 representa a utilização do vetor `conf`.

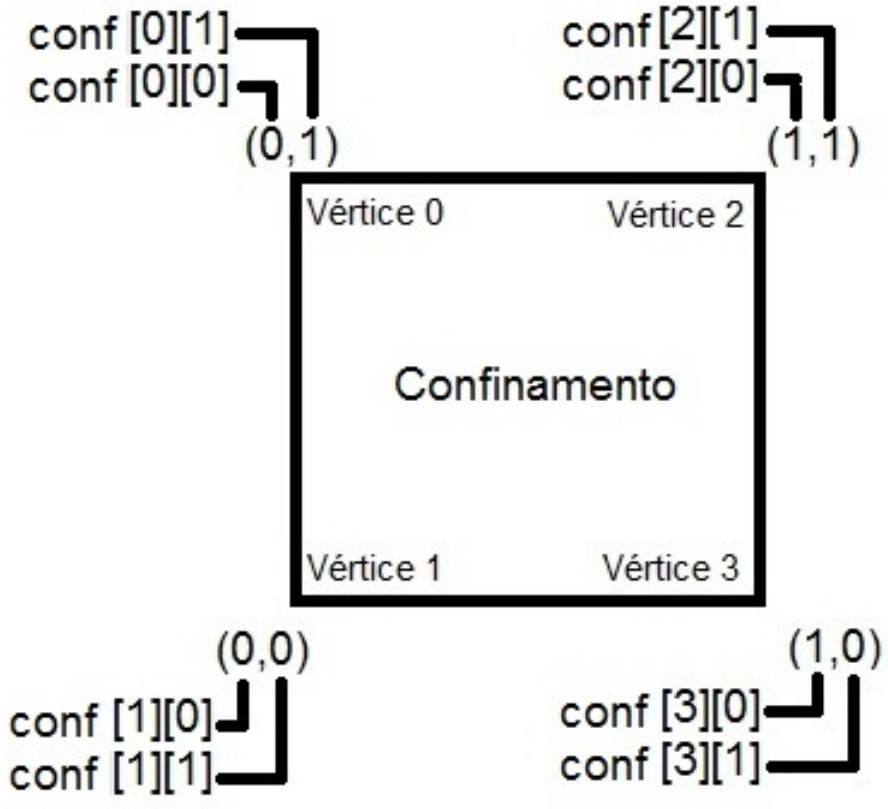
Com o intuito de facilitar a compreensão e implementação, optou-se por definir o espaço de representação do confinamento e de todos os objetos dentro do primeiro quadrante, ou seja, sendo a origem o ponto (0,0),e assumindo valores positivos para X e Y.

4.2.2 Funções de Transição

Em seguida, foi necessário definir quais os movimentos possíveis no contexto dos objetos, sendo eles 4 movimentos possíveis para cada objeto: mover para cima, mover para baixo, mover para direita e mover para esquerda. Cada movimento possível para cada objeto futuramente será um novo estado na busca.

Porém, antes disso, é necessário saber se dentro do cenário de cada estado, o objeto pode fazer tal movimento, e se sim, até que ponto pode-se mover. A Figura 23 descreve uma situação onde o Objeto 1 só pode se mover para cima: esquerda e baixo estão trancadas com o confinamento, e à direita com o Objeto 2. Para isso, na classe Estado, foram elaborados quatro métodos que retornam um valor booleano, informando se é possível tal movimento, e já alteram

Figura 22 – Representação das coordenadas do confinamento



Fonte: Acervo do Autor, 2018.

as variáveis de classe `paradaY` e `paradaX`, que informam até qual posição de parada no eixo X ou Y o objeto pode se deslocar.

São esses métodos descritos a seguir:

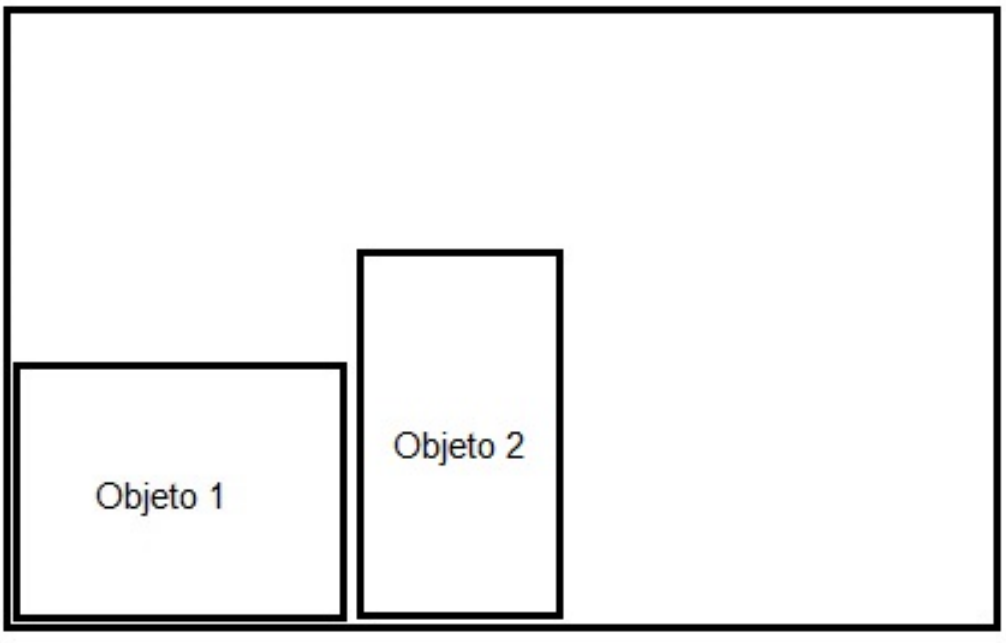
→ **`verificaCima(nObj)`** Retorna verdadeiro caso tenha possibilidade de movimentar o objeto para cima, e atualiza a variável `paradaX` com o valor máximo de deslocamento;

→ **`verificaBaixo(nObj)`** Retorna verdadeiro caso tenha possibilidade de movimentar o objeto para baixo, e atualiza a variável `paradaX` com o valor máximo de deslocamento;

→ **`verificaEsquerda(nObj)`** Retorna verdadeiro caso tenha possibilidade de movimentar o objeto para direita, e atualiza a variável `paradaY` com o valor máximo de deslocamento;

→ **`verificaDireita(nObj)`** Retorna verdadeiro caso tenha possibilidade de movimentar o objeto para esquerda, e atualiza a variável `paradaY` com o valor máximo de deslocamento.

Figura 23 – Exemplo de confinamento



Fonte: Acervo do Autor, 2018.

No Quadro 6 consta a implementação do método `verificaCima`. A linha 4 verifica se o objeto está com o lado de cima já encostado no confinamento, caso verdadeiro, o movimento é inválido. O laço de repetição da linha 7 passa por todos os objetos do estado verificando se estão no caminho do objeto em que se visa movimentar, caso exista, atualiza a variável `paradaX` com o valor do eixo X do objeto no caminho.

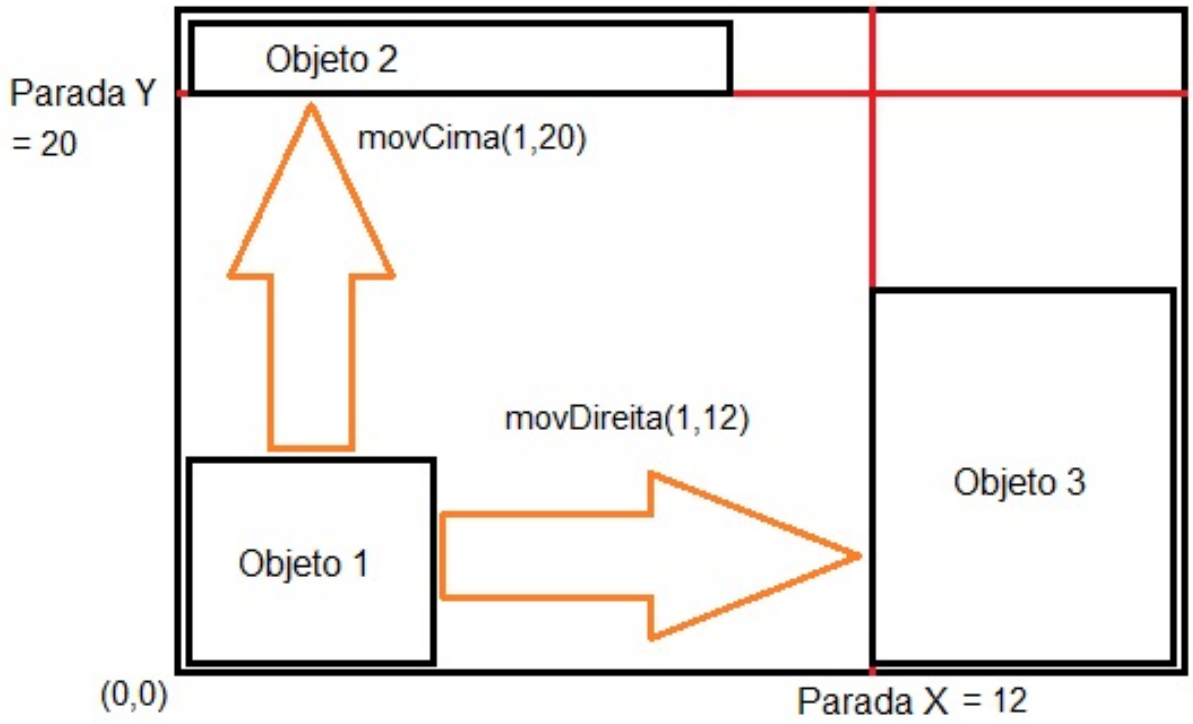
O laço da linha 7 passa por todos os objetos, logo, se por exemplo, for encontrado o objeto mais próximo e atualizada a variável de parada, e em seguida encontrado outro objeto, porém mais distante, a variável será atualizada novamente, e consequentemente ao deslocar o objeto ele irá sobrepor o anterior. Para contornar isso, entre a linha 18 e 34 é verificado se o objeto em questão é mesmo o mais próximo, evitando a atualização da variável de parada para um valor não correspondente.

Quadro 5 – Método verificaCima

```
1  public boolean verificaCima(int nObj) {
2          paradaY = conf[0][1];
3          boolean primeiro = true;
4          if (obj[nObj][0][1] == conf[0][1]) {
5              return false;
6          }
7          for (int i = 0; i <= tamanhoObj; i++) {
8              if ((((obj[nObj][0][0] <= obj[i][1][0] &&
9              obj[i][1][0] <= obj[nObj][2][0])
10                         || (obj[nObj][0][0] <= obj[i][3][0] &&
11                         obj[i][3][0] <= obj[nObj][2][0])
12                         || (obj[nObj][0][0] >= obj[i][1][0] &&
13                         obj[i][3][0] >= obj[nObj][2][0])
14                         || (obj[nObj][0][0] <= obj[i][1][0] &&
15                         obj[i][3][0] <= obj[nObj][2][0]))
16                         && (obj[nObj][0][1] <= obj[i][1][1])) &&
17                         nObj != i) {
18                     if (primeiro) {
19                         paradaY = obj[i][1][1];
20                         primeiro = false;
21                     }
22                     if (paradaY > obj[i][1][1]) {
23                         paradaY = obj[i][1][1];
24                     }
25             }
26         }
27         return true;
28     }
```

Como o exemplo do funcionamento, temos Figura 24, onde se avalia os movimentos possíveis do Objeto 1. Inicialmente os métodos `verificaBaixo(1)` e `verificaEsquerda(1)` retornam falso pois o objeto não pode ser movimentado para esquerda ou para baixo devido ao confinamento. O método `verificaDireita(1)` verifica que, imediatamente, não há nenhum outro objeto ou confinamento no lado direito do Objeto 1, então procura qualquer elemento o mais próximo do objeto em questão para ser o ponto de parada, nesse caso, encontra o Objeto 3, logo o método atualiza a variável `paradaX` para o valor X do vértice presente no Objeto 3 mais próximo do Objeto 1, no caso 12. O mesmo ocorre em relação ao Objeto 2, porém atualizando a variável `ParadaY`, pois movimentos para cima e baixo alteram o valor apenas no eixo Y, mantendo X, e movimentos para esquerda e direita alteram o eixo X, mantendo Y.

Figura 24 – Exemplo de possíveis movimentos



Fonte: Acervo do Autor, 2018.

Ainda na Figura 24, cada verificação que retorna verdadeiro, é armazenada, e o movimento do objeto é feito também por quatro métodos, um para cada direção possível. O método para movimento faz uma cópia do vetor `obj[][][]`, em seguida move o objeto `nObj` até o limite de parada(`paradaX` para os movimentos direita e esquerda, ou `paradaY` para os movimentos cima e baixo), e retorna o vetor com o objeto já em sua nova posição. São esses movimentos:

Quadro 6 – Método movCima

```
1  public int[][][] movCima(int nObj, int paradaY) {
2          int[][][] objeto = new int[200][4][2];
3          for (int a = 0; a <= tamanhoObj; a++) {
4              for (int b = 0; b < 4; b++) {
5                  for (int c = 0; c < 2; c++) {
6                      objeto[a][b][c] = getObj()[a][b][c];
7                  }
8              }
9          }
10         int dist = objeto[nObj][0][1] - objeto[nObj][1][1];
11         objeto[nObj][0][1] = paradaY;
12         objeto[nObj][2][1] = paradaY;
13         objeto[nObj][1][1] = (paradaY - dist);
14         objeto[nObj][3][1] = (paradaY - dist);
15         return objeto;
16     }
```

→ **movCima(nObj,paradaY)** Move o objeto `nObj` para cima, até o ponto superior do objeto alcançar o eixo Y na posição `paradaY`;

→ **movBaixo(nObj,paradaY)** Move o objeto `nObj` para baixo, até o ponto inferior do objeto alcançar o eixo Y na posição `paradaY`;

→ **movDireita(nObj,paradaX)** Move o objeto `nObj` para direita, até o ponto direito do objeto alcançar o eixo X na posição `paradaX`;

→ **movEsquerda(nObj,paradaX)** Move o objeto `nObj` para esquerda, até o ponto esquerdo do objeto alcançar o eixo X na posição `paradaX`.

No Quadro 7 consta a implementação do método `movCima`.

4.2.3 Cálculo da Heurística

As técnicas de heurística, segundo Tonetto (2006), são regras gerais de influência utilizadas para julgamentos em decisões de incertezas, logo, no contexto desse projeto, essenciais para determinar o quão bom é determinado estado, possibilitando tanto um critério de parada quanto a seleção do melhor arranjo de cargas.

No decorrer do projeto, com o intuito de facilitar a compreensão e implementação, optou-se por definir o melhor estado como o que possui a maior quantidade e maior proximidade

Quadro 7 – Método calcularHeuristica

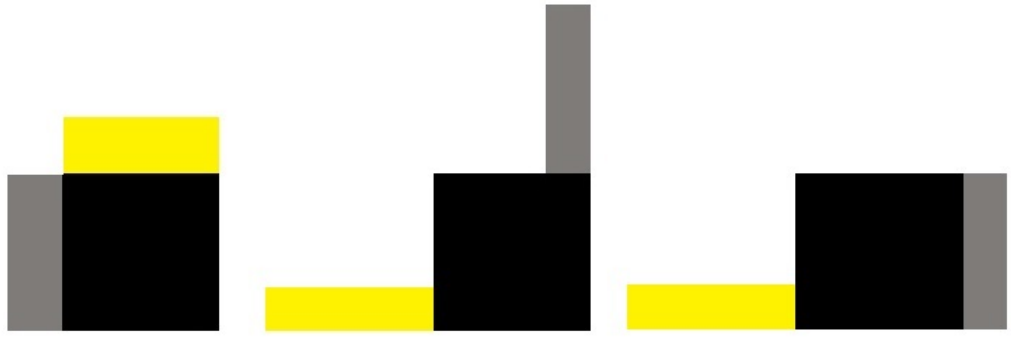
```
1       public void calcularHeuristica() {
2         float somaX = 0, somaY = 0;
3         for (int y = 0; y <= tamanhoObj; y++) {
4             for (int x = 0; x < 4; x++) {
5                 somaX += (obj[y][x][0]*obj[y][2][2]);
6                 somaY += obj[y][x][1];
7             }
8         }
9         setValorHeuristico(somaX + somaY + caminho.length());
10    }
```

de caixas no vértice 2 do confinamento,e o menor número de movimentos até o estado. Ou seja, quanto maior a soma dos valores X e Y dos objetos, e menor é o caminho, melhor é o estado. A variável `obj[y][2][2]` representa a prioridade do objeto, ela multiplica o valor da coordenada do eixo X para que quanto maior o valor da variável, mais distante o objeto fique longe do fundo do confinamento, agilizando sua retirada. O cálculo do valor heurístico é apresentado no Quadro 8.

4.2.4 Representação do Estado Alvo e Critério de Parada

Como estado alvo não temos um estado propriamente dito, mas sim um conjunto de *n* estados com um valor heurístico relevante, pois, com *n* disposições distintas podemos obter o mesmo volume aproveitável, não existindo uma combinação ideal, como representado na Figura 25, onde em todos os casos a área contígua é a mesma.

Figura 25 – Exemplo de possíveis alocações



Fonte:Elaborado pelo Autor, 2018.

Com isso, o critério de parada será a chegada a um valor heurístico considerado

aceitável, ou o limite de iterações.

A solução encontrada foi, durante a checagem do valor heurístico, armazenar o melhor valor até o momento, e o respectivo estado. Assim foi definida uma "chance"de encontrar um estado melhor, definida por um número máximo de iterações sem avanço na heurística.

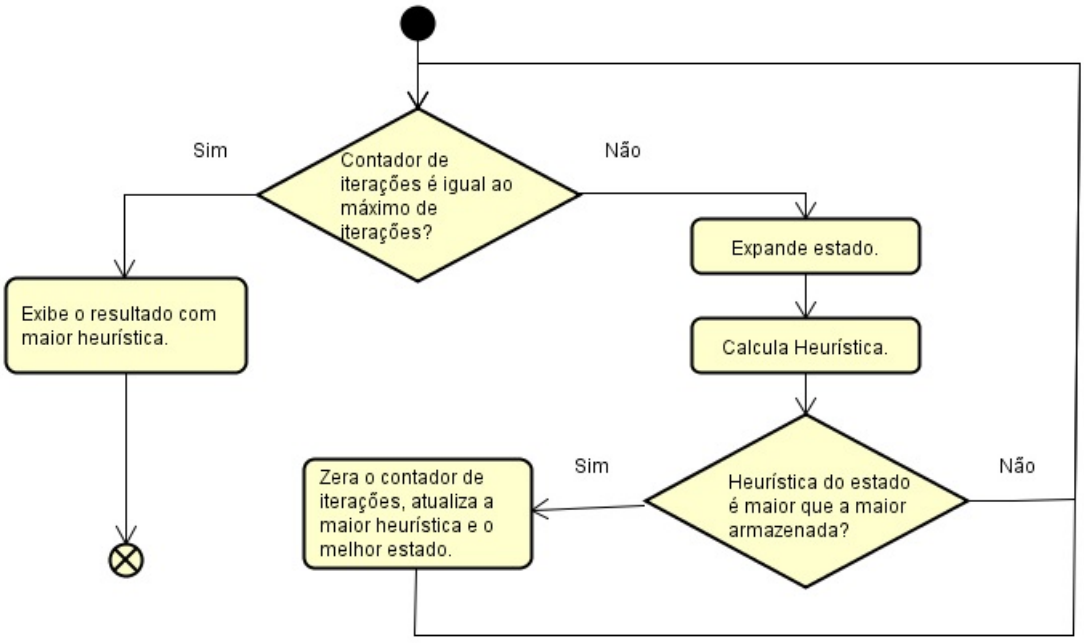
Funciona da seguinte forma: para cada nó expandido, avalia-se a heurística. Se essa for melhor do que a anterior, atualiza-se a variável de comparação, armazena o estado, e zera o contador de iterações. Caso contrário, o contador é incrementado.

Então, a cada estado melhor que o algoritmo encontrar, ele tem mais n chances de encontrar um outro estado melhor ainda antes da execução ser finalizada, como mostrado no diagrama da Figura 26.

A escolha do número de chances permite optar por fugir de eventuais máximos locais, ou priorizar uma execução mais breve.

Também foi implementado um contador de iterações globais, para se caso o algoritmo não pare de encontrar resultados melhores, tenha um limite devido ao tempo ou à memória do sistema.

Figura 26 – Diagrama cálculo da heurística



Fonte: Acervo do Autor, 2018.

4.2.5 Entrada de Dados

Ao iniciar o programa, a tela inicial, ilustrada na Figura 27, aparecerá para o usuário. Ele deve definir as dimensões do confinamento, e cadastrar os pacotes inserindo suas dimensões e em seguida clicando em "Inserir Pacote".

Ao inserir o pacote, o contador de objetos é incrementado, e o objeto é primeiramente armazenado no vetor `entrada`, e o confinamento já armazenado no vetor `conf`, que será passado para o mecanismo de busca.

Figura 27 – Interface da aplicação

Fonte: Acervo do Autor, 2018.

4.2.6 Disposição Inicial dos Objetos

Para que o algoritmo de busca possa funcionar, é necessário que todo os elementos estejam dispostos na área de resolução para que o método então possa começar a fazer os movimentos a fim de buscar o melhor estado possível.

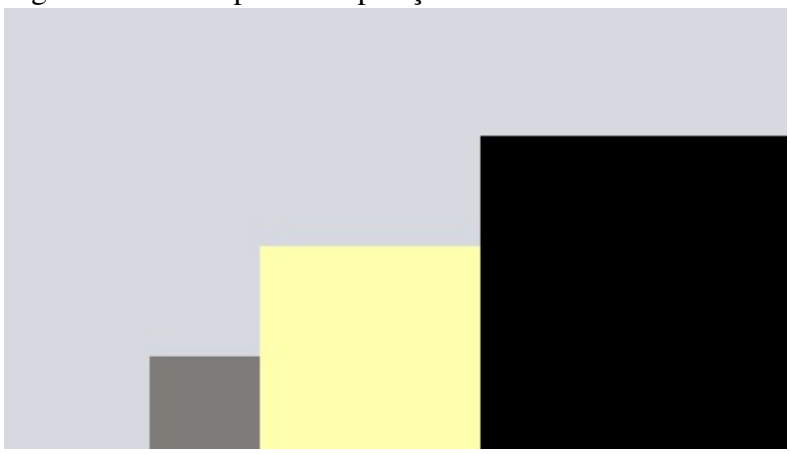
Então, se é obrigatório já dispor todos os pacotes, é conveniente já alocar na posição de suposta melhor heurística a fim de poupar trabalho do algoritmo de busca.

No presente trabalho, ao clicar em "Busca", é feita uma ordenação por área do pacote, ordenando-o de forma com que a disposição final dentro do vetor seja o pacote com maior área no índice 0, e o de menor área no maior índice utilizado.

Em seguida, os pacotes são pré alocados no confinamento, de modo ao pacote de índice 0 ficar no ponto de maior heurística, e os demais serem alocados no mesmo eixo Y, alterando apenas o valor do eixo X, como ilustrado no exemplo de disposição da Figura 28.

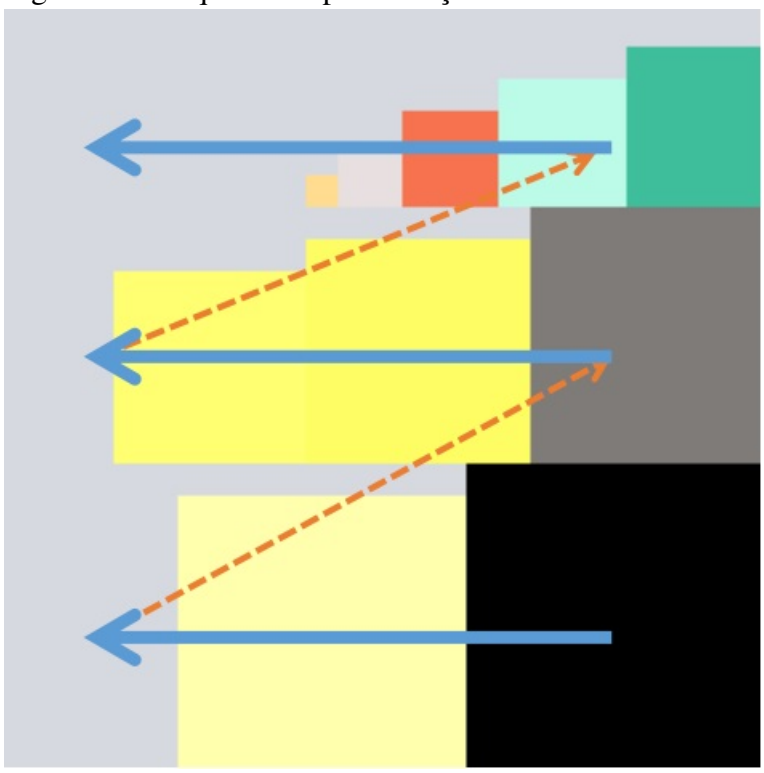
Caso a primeira fileira fique lotada, é pego o valor Y do maior pacote da fileira, e criado uma nova mantendo esse valor, como representado na Figura 29.

Figura 28 – Exemplo de disposição



Acervo do Autor, 2018.

Figura 29 – Esquema de pré alocação



Acervo do Autor, 2018.

Com todos os elementos já definidos e dispostos dentro do confinamento, têm-se um estado inicial, logo, é possível iniciar a busca. Com a pré alocação, é possível que a busca não encontre um estado melhor que o inicial, exibindo o mesmo.

4.2.7 Busca Horizontal com Heurística

Com um estado inicial já preparado, caso o usuário opte pela Busca Horizontal, é limpo o *array* A, que armazena os estados em aberto, que ainda não foram verificados, e o F, que armazena os nós fechados, que já foram verificados. Atribui-se o estado inicial como sendo o melhor estado e sua heurística como sendo a melhor. O laço *for* da linha 15 passa por todos os objetos dentro do estado atual, sendo que dentro de cada um deles, são definidos quatro novos estados, que correspondem aos quatro movimentos possíveis.

O Quadro 8 apresenta o método principal da busca, abreviado apenas com o movimento cima, sendo os demais movimentos implementados da mesma forma.

Temos que para cada expansão de estado, se os quatro movimentos forem possíveis, todo objeto cria quatro novos possíveis estados. Exemplificando, se tivermos dois objetos em um estado, serão criados quatro novos estados movendo o objeto 0, e mais quatro novos estados movendo o objeto 1. A cada nível de profundidade alcançado, o número de nós quadruplica, reforçando a necessidade de um artificio de heurística para uma expansão mais específica, concentrando apenas nos nós que teoricamente trarão melhores resultados.

Na linha 18 é verificado se o objeto em questão pode ser movido para cima, caso verdadeiro, o estado referente já é modificado com o movimento, e seu identificador é atualizado, faltando apenas o estado ser posto no *array* A. Porém, para poupar esforço computacional, após todos os estados serem modificados com os movimentos, o estado gerador é movido do *array* de nós abertos para o de nós fechados, e é feita a verificação se o estado não pertence a nenhum dos dois *arays*. Só depois da certeza de ser inédito o nó é adicionado aos abertos na linha 26.

A linha 29 verifica se o laço da linha 15 passou por todos os objetos dentro do estado, caso sim, irá remover o nó gerador, deixando apenas os filhos inéditos, em seguida é calculado qual objeto do *array* A possui a melhor heurística, este então será o próximo a ser expandido.

Esse processo segue até que o número máximo de chances ou de iterações totais seja alcançado, então o melhor estado é exibido ao usuário.

Quadro 8 – Busca Horizontal

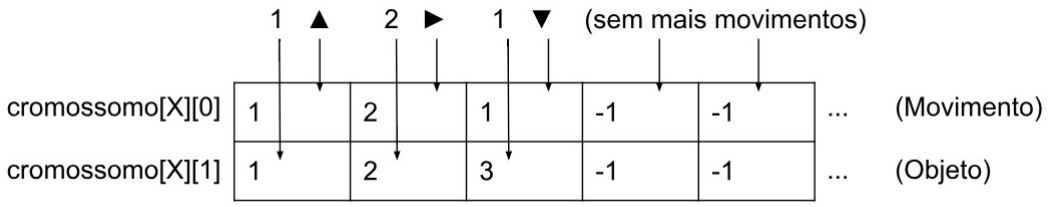
```
1   private void buscaHorizontal(int[][][] obj, int[][] conf, JTextPane
2         A.clear();
3         F.clear();
4         Estado gerador = new Estado(obj, conf, tamanho);
5         Estado melhor = new Estado(obj, conf, tamanho);
6         gerador.setCaminho("");
7         gerador.calcularHeuristica();
8         heuristica = gerador.getValorHeuristico();
9         A.add(gerador);
10        int posMenorHeuristica = 0;
11        int chance = 0;
12        boolean fim = false;
13        int cont = 0;
14        while (!fim) {
15            for (int nObj = 0; nObj <= tamanho; nObj++) {
16                boolean suc1 = false,;
17                Estado sucessor1 = new Estado(A.get(posMenorHeuristica)
18                if (sucessor1.verificaCima(nObj)) {
19                    sucessor1.setObj(sucessor1.movCima(nObj, sucessor1.
20                    suc1 = true;
21                }
22                F.add(A.get(posMenorHeuristica));
23                if (suc1) {
24                    if (!F.contains(sucessor1) && !A.contains(sucessor1
25                        sucessor1.calcularHeuristica();
26                        A.add(sucessor1);//Cima
27                    }
28                }
29                if (nObj == tamanho) {
30                    A.remove(posMenorHeuristica);
31                    if (chance == maxchance) {
32                        fim = true;
33                        Desenhar();
34                    } else {
35                        cont++;
36                        chance++;
37                    }
38                    atualizarMelhorEstado(posMenorHeuristica, heuristic
39                    if (cont > maxit) {
40                        fim = true;
41                        Desenhar();
42                    }
43                }
44            }
45        }
46    }
```

4.2.8 Algoritmo Genético

Caso o usuário opte pelo Algoritmo Genético, é utilizado o mesmo estado inicial descrito anteriormente. A partir daí, ocorrerão reproduções e mutações nos estados para buscar a melhor disposição.

A representação cromossomial adotada foi a própria sequência de movimentos para cada objeto, sendo representada por um vetor de 40 linhas e 2 colunas, `int[][] cromossomo = new int[40][2];`, onde a primeira coluna representa o movimento, e segunda o número do pacote. Cada coluna representa uma movimentação possível de um pacote. A Figura 30 ilustra um exemplo de representação de um conjunto de movimentos no cromossomo. O valor -1 indica que não há movimento ou objeto.

Figura 30 – Exemplo de da representação cromossomial



Fonte: Elaborado pelo Autor, 2018.

Com isso, foi possível reutilizar a mesma representação de estado, pré alocação dos objetos como estado inicial, as funções de transição, o cálculo de heurística como *fitness*, e o método de apresentação ao usuário, descrito em seguida.

Inicialmente a população é preenchida apenas com um movimento, que é aleatório, e então ordenada de acordo com o *fitness* para em seguida partir para a reprodução.

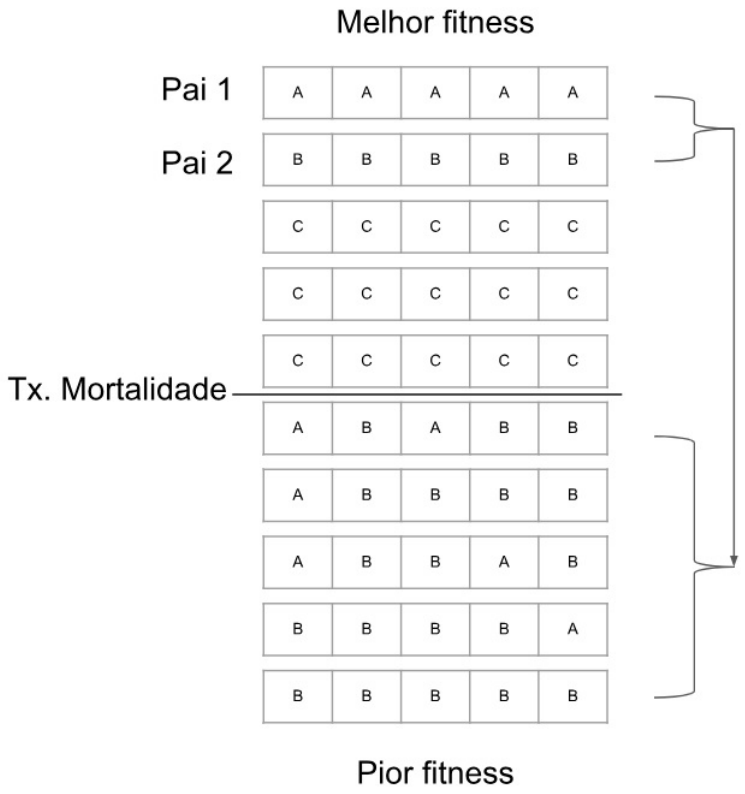
No contexto da reprodução dos cromossomos, um *crossover* não é tão eficiente, pois, supondo um cromossomo contendo muitos movimentos, onde seu estado já esteja quase ótimo, a troca dos primeiros movimentos pode redefinir todo o contexto da alocação, prejudicando a convergência do estado, que em uma futura iteração, terá seu *fitness* baixo, sendo eliminado. Então, o algoritmo poderia deixar de encontrar uma solução talvez até ótima, "estragando"futuros máximos locais.

Por isso, adotou-se a hereditariedade no algoritmo, onde, apenas as características positivas são passadas aos descendentes. Funciona da seguinte forma: é informado uma taxa de mortalidade, por exemplo, 50%. Então, se a população for de 10 cromossomos, inicialmente

ordenados de acordo com o *fitness*, os 5 cromossomos com pior desempenho vão receber características aleatórias dos dois cromossomos com melhor desempenho, denominados pais.

A Figura 31 representa o processo de reprodução.

Figura 31 – Representação do processo de reprodução



Fonte: Elaborado pelo Autor, 2018.

Como inicialmente os cromossomos são populados apenas com um movimento, a cada 5 reproduções é adicionado um movimento a todos os cromossomos. Esse intervalo é necessário para que o algoritmo possa trabalhar mais possibilidades com a população antes que seja imposto um movimento a mais.

A implementação da reprodução é apresentada no Quadro 9.

O cálculo do *fitness* é o mesmo utilizado pela heurística na Busca em Espaços de Estados. Logo, é necessário criar um estado com os passos descritos no cromossomo, para então utilizar o método `calcularHeuristica()`.

O quadro 10 representa o método que calcula o *Fitness* de cada cromossomo da população. Inicialmente é criado um estado auxiliar contendo os passos do cromossomo, em seguida, o método `realizarCromossomo()`, descrito no Quadro 11, efetiva os movimentos do cromossomo no estado, utilizando também os mesmos métodos da Busca em Espaços de Estados, verificando se o movimento é possível, e caso verdadeiro, realizando-o. Assim, não há

Quadro 9 – Reprodução cromossimial

```
1       private void renovarCromossomos() {
2         int inicioExcluidos = (int) (taxaMortalidade * nPopulacao);
3         for (int i = inicioExcluidos; i < nPopulacao; i++) {
4             int pai1, pai2;
5             pai1 = new Random().nextInt(inicioExcluidos + 1);
6
7             do {
8                 pai2 = new Random().nextInt(inicioExcluidos + 1);
9             } while (pai1 == pai2);
10
11            for (int j = 0; j < 20; j++) {
12                int pos = new Random().nextInt(40);
13                cromossomos[i][pos][0] = cromossomos[pai1][pos][0];
14                cromossomos[i][pos][1] = cromossomos[pai1][pos][1];
15            }
16            for (int j = 20; j < 40; j++) {
17                int pos = new Random().nextInt(40);
18                cromossomos[i][pos][0] = cromossomos[pai2][pos][0];
19                cromossomos[i][pos][1] = cromossomos[pai2][pos][1];
20            }
21        }
22    }
```

a preocupação de haver apenas movimentos válidos no cromossomo, pois o método irá ignorá-los, pulando para o próximo passo até encontrar um válido ou o fim do cromossomo. Também não há a preocupação com estouro de pilha ao instanciar os estados, pois o coletor de lixo da linguagem Java limpa os objetos inativos da memória automaticamente.

No decorrer de todo o processo, é mantida sempre uma cópia do cromossomo com melhor *fitness*, independente da geração ou iteração, e ao final, é retornado ao usuário a disposição do estado com base nesse cromossomo.

O método principal do Algoritmo Genético é apresentado no Quadro 12.

4.2.9 Apresentação do Resultado ao Usuário

Após o algoritmo encontrar o resultado, é necessário apresenta-lo ao usuário, para isso foi implementada uma interface gráfica utilizando a API Java 2D que fornece funcionalidades básicas para o desenho de objetos gráficos 2D.

Ao instanciar a classe Desenhar são passados como parâmetros o confinamento, os

Quadro 10 – Cálculo de *Fitness*

```
1     private float[] calcularFit() {
2         float[] fit = new float[nPopulacao];
3         int[][] crom = new int[40][2];
4         for (int i = 0; i < nPopulacao; i++) {
5             Estado estado = new Estado(obj, conf, tamanho);
6             for (int a = 0; a < 40; a++) {
7                 crom[a][0] = cromossomos[i][a][0];
8                 crom[a][1] = cromossomos[i][a][1];
9             }
10            estado.setCrom(crom);
11            realizarCromossomo(estado);
12            estado.calcularHeuristica();
13            fit[i] = estado.getValorHeuristico();
14        }
15        return fit;
16    }
```

Quadro 11 – Método `realizarCromossomo()`

```
1     private void realizarCromossomo(Estado estado) {
2         for (int i = 0; i < 40; i++) {
3             if (estado.getCrom()[i][0] == 0 &&
4             estado.verificaCima(estado.getCrom()[i][1])) {
5                 estado.setObj(estado.movCima(estado.getCrom()[i][1],
6                 estado.getParadaY()));
7             }
8             if (estado.getCrom()[i][0] == 1 &&
9             estado.verificaBaixo(estado.getCrom()[i][1])) {
10                estado.setObj(estado.movBaixo(estado.getCrom()[i][1],
11                estado.getParadaY()));
12            }
13            if (estado.getCrom()[i][0] == 2 &&
14            estado.verificaDireita(estado.getCrom()[i][1])) {
15                estado.setObj(estado.movDireita(estado.getCrom()[i][1],
16                estado.getParadaX()));
17            }
18            if (estado.getCrom()[i][0] == 3 &&
19            estado.verificaEsquerda(estado.getCrom()[i][1])) {
20                estado.setObj(estado.movEsquerda(estado.getCrom()[i][1]
21                estado.getParadaX()));
22            }
23        }
24    }
```

Quadro 12 – Método principal do Algoritmo Genético

```
1     private void buscaGenetica(int[][][] cromossomos, JTextPane painel)
2         fitness = calcularFit();
3         int cont = 0;
4         int[][] melhorCrom = new int[40][2];
5         float melhorFit=0;
6         int melhorGeracao=0;
7         for (int i = 0; i <= numeroEvolucoes; i++) {
8             popular(passo);
9             if (cont == 5 && passo < 40) {
10                cont = 0;
11                passo++;
12            }
13            ordenarCromossomos();
14            renovarCromossomos();
15            fitness = calcularFit();
16            for(int j=0; j< nPopulacao;j++){
17                if(melhorFit==0 || fitness[j] > melhorFit){
18                    melhorFit=fitness[j];
19                    for (int k = 0; k < 40; k++) {
20                        melhorCrom[k][0] = cromossomos[j][k][0];
21                        melhorCrom[k][1] = cromossomos[j][k][1];
22                        melhorGeracao=i;
23                    }
24                }
25            }
26            System.out.println("Geração: " + (i));
27            escreverCromossomo(cromossomos);
28            cont++;
29        }
30        Estado resultado = new Estado(obj, conf, tamanho);
31        resultado.setCrom(melhorCrom);
32        realizarCromossomo(resultado);
33        painel.setText(painel.getText() + "\n" + escreverResultado(resu
34        Desenhar desenhar = new Desenhar(conf, resultado.getObj(), tama
35    }
```

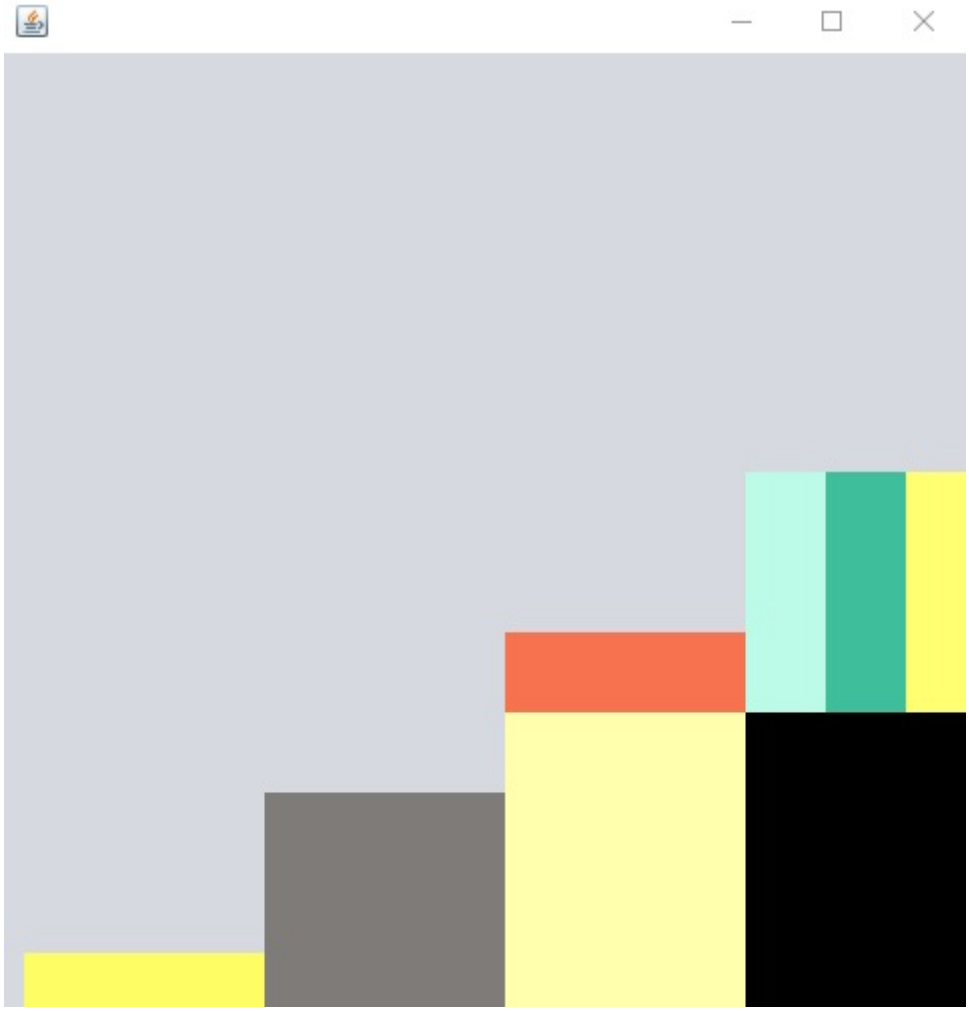
objetos e a quantidade de objetos.

A Figura 32 apresenta a interface gráfica.

Cada unidade de medida corresponde a um pixel, logo, se o tamanho do confinamento exceder a resolução da tela, a área de exibição será cortada. Então, de acordo com a maior dimensão do confinamento, é atualizada a variável `rate` que multiplica todas as dimensões do confinamento e dos objetos antes de serem exibidos, alterando assim a proporção para que tudo caiba na tela. A variável `deslocamento` serve para compensar a barra superior da janela, ela não altera a proporção, apenas move os objetos e o confinamento para baixo.

O método `paint(Graphics g)` que faz o gerenciamento do conteúdo a ser exibido. Ele deve ser sobre-escrito pelo programador, pois não pode ser chamado diretamente, sendo chamado automaticamente quando o computador exibe o componente na tela, ou também se acontecer um evento que exija uma atualização do conteúdo exibido.

Figura 32 – Exemplo da interface gráfica



Fonte: Acervo do Autor, 2018.

5 RESULTADOS

Os testes deste trabalho foram adaptados dos testes de Silva e Soma (2003b), onde os cenários são divididos em classes. Para este software, os testes foram divididos nas seguintes classes:

1 a maioria dos itens (70%) têm grandes dimensões (L=(25 15), C=(25 15));

2 a maioria dos itens (70%) são muito largos (L=(25 15), C=(5 7));

3 a maioria dos itens (70%) são muito compridos (L=(5 7), C=(25 15));

4 a maioria dos itens (70%) têm pequenas dimensões (L=(5 7), C=(5 7)).

Por classe foram efetuados testes alternando as configurações de número de população e percentual de ponto de corte (taxa de mortalidade) para o Algoritmo Genético, onde, devido aleatoriedade presente no mecanismo, foi executado o algoritmo 5 vezes e extraído sua média de execução para comparação. Para a Busca em Espaços de Estados, foi alternado o número de chances. Os testes prosseguiram em 4 conjuntos distintos de pacotes, respeitando a definição da classe. O número de pacotes foi fixado em 5 em todos os testes, com prioridade de 100%. Para cada teste as dimensões do contêiner foram definidas em: largura=50 e comprimento=50, onde todos os contêineres possuem as mesmas dimensões. Os testes foram executados em um computador notebook com processador Intel® Core™ i5-3317 CPU @ 1.70GHz, memória RAM de 8GB e sistema operacional Windows 10 64 bits. A unidade apresentada nos testes é o valor heurístico(*Fitness*) do estado final, pois ambos os mecanismos de busca utilizam o mesmo método para calcula-lo.

5.0.1 Análise dos resultados – Classe 1

A classe 1 é definida por 70% dos itens muito altos e compridos e os demais aleatórios. O Quadro 14 demonstra as médias de execução deste cenário com o Algoritmo Genético, variando o parâmetro de número de população, com taxa de mortalidade em 80% e 30 gerações.

É possível observar na classe 1, que ao aumentar o número da população ocorreu também um pequeno aumento no percentual de aptidão e no tempo de execução.

Ainda no Algoritmo Genético, outro fator testado foi o percentual de ponto de corte, apresentado no Quadro 15, onde para termos de comparação foi utilizado de base a média

Quadro 13 – Médias de aptidão por população no Algoritmo Genético Classe 1

População	Aptidão do Conjunto 1	Aptidão do Conjunto 2	Aptidão do Conjunto 3	Aptidão do Conjunto 4	Tempo total(ms)
20	12.02	12.14	12.08	11.58	1346
25	12.14	**12.34**	**12.28**	11.82	**1807**
30	**12.54**	12.34	12.28	**12.1**	2111

Quadro 14 – Médias de aptidão por população no Algoritmo Genético Classe 1

Tx. Mortalidade	Aptidão do Conjunto 1	Aptidão do Conjunto 2	Aptidão do Conjunto 3	Aptidão do Conjunto 4	Tempo total
70%	12.14	12.14	11.88	12.02	1373
80%	12.02	12.14	12.08	11.58	1346
90%	**12.54**	**12.34**	**12.08**	**12.2**	**1101**

Quadro 15 – Médias de aptidão por chances Classe 1

Chance	Aptidão do Conjunto 1	Aptidão do Conjunto 2	Aptidão do Conjunto 3	Aptidão do Conjunto 4	Tempo total
15	12.54	12.44	12.44	**12.5**	**559**
30	**12.7**	12.44	12.44	12.5	980
60	12.7	12.44	12.44	12.5	1535
160	12.7	**12.54**	**12.54**	12.5	3735

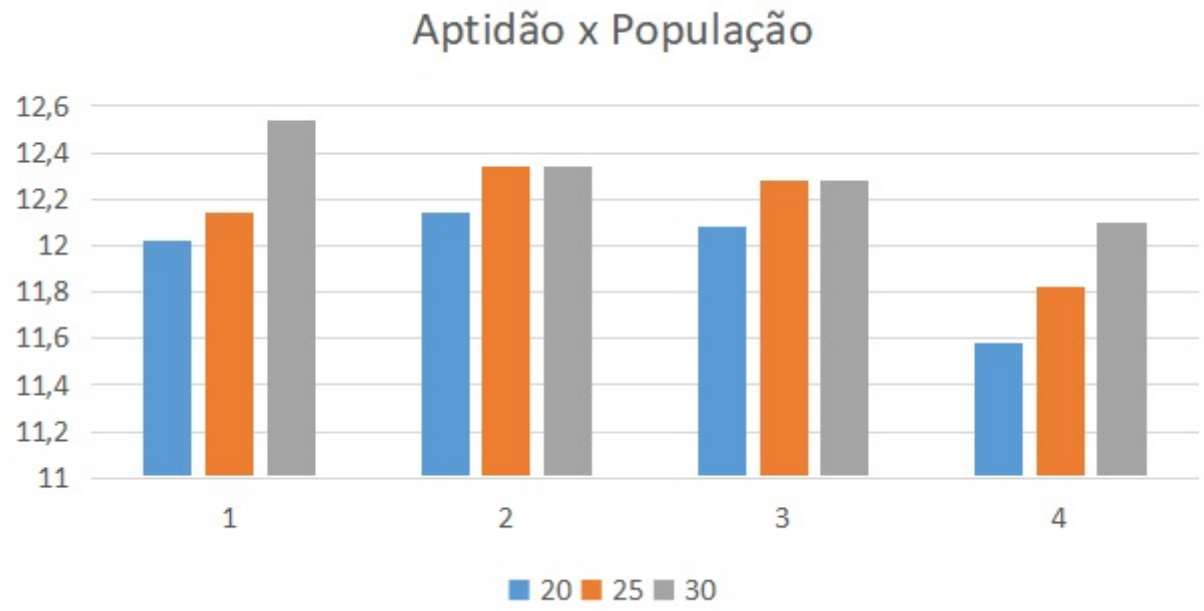
da geração de população 20, representada na primeira linha do Quadro 14, efetuada no teste anterior.

Nota-se que variando o ponto de corte para 90% houve um ganho razoável na aptidão dos estados, e uma queda no tempo de execução se comparado ao corte em 80%, sendo muito similar ao de 70%.

Como o cálculo da aptidão do estado no Algoritmo de Busca em Espaços de Estados (heurística) é exatamente o mesmo do Algoritmo Genético (*fitness*), no Algoritmo de Busca em Espaços de Estados o teste foi conduzido da mesma forma, porém alterando a quantidade de chances, para verificar a saída de máximos locais. Os dados de teste constam na Tabela 16.

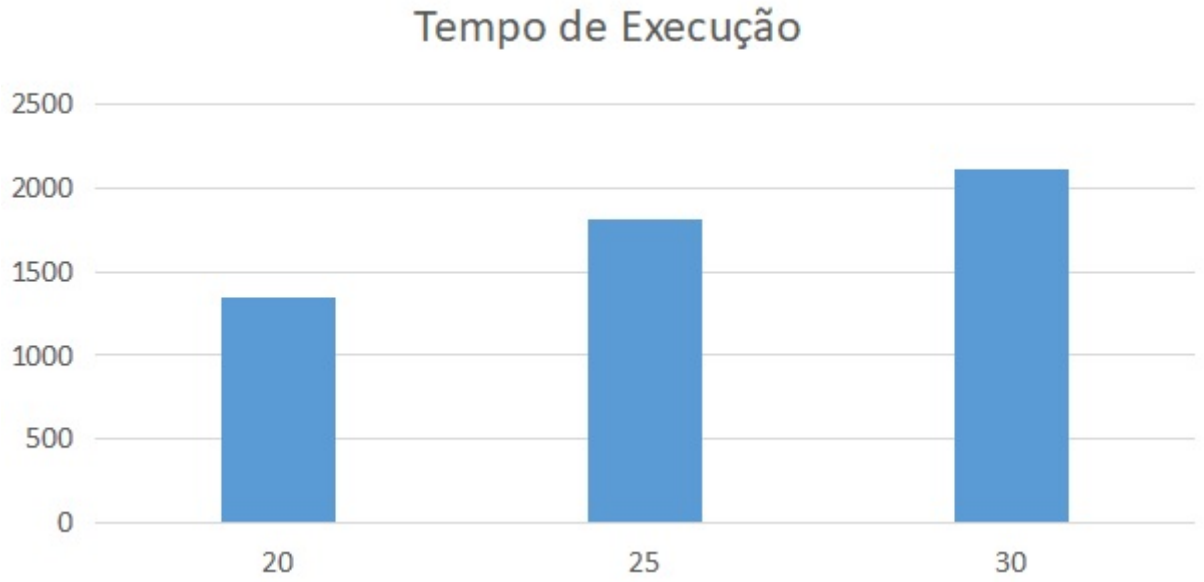
Na Figura 33 consta o gráfico demonstrando a relação da variação da aptidão dos conjuntos de acordo com a população, e na Figura 34 a soma do tempo de execução em todos os conjuntos. Na Figura 35 consta a relação da variação da aptidão dos conjuntos de acordo com a taxa de mortalidade, e na Figura 36 a soma do tempo de execução em todos os conjuntos, ambas as variações presentes no Algoritmo Genético. Na Figura 37 conta o gráfico demonstrando a relação da variação da aptidão dos conjuntos de acordo com a chance, e na Figura 38 a soma do tempo de execução em todos os conjuntos, referentes ao algoritmo de Busca em Espaços de Estados.

Figura 33 – Aptidão X População



Fonte: Elaborado pelo Autor, 2019.

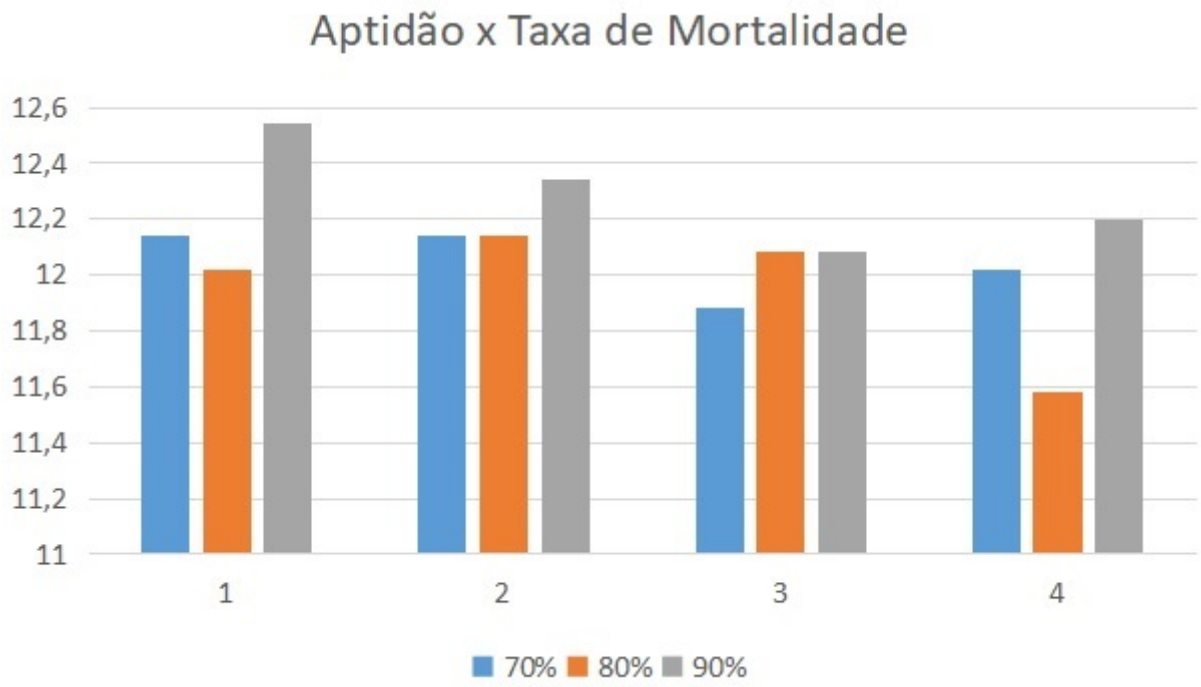
Figura 34 – Tempo de Execução variando a População



Fonte: Elaborado pelo Autor, 2019.

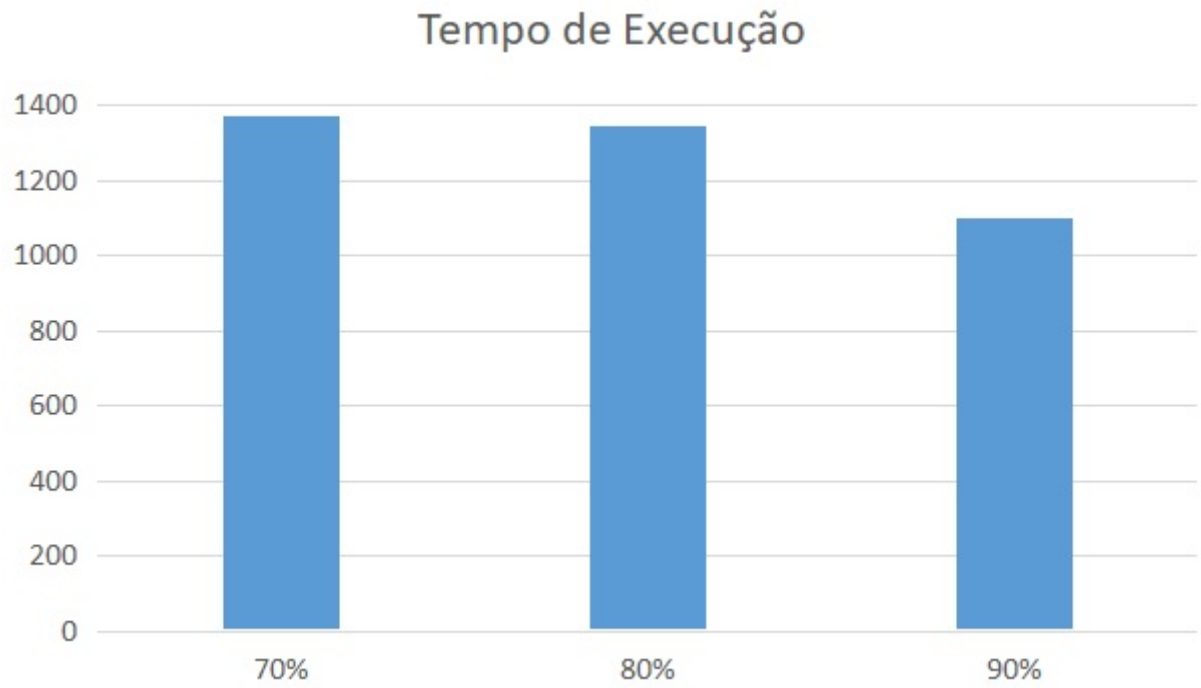
Nota-se que desde as primeiras execuções a Busca em Espaços de Estados encontrou resultados melhores que Algoritmo Genético.

Figura 35 – Aptidão X Taxa de Mortalidade



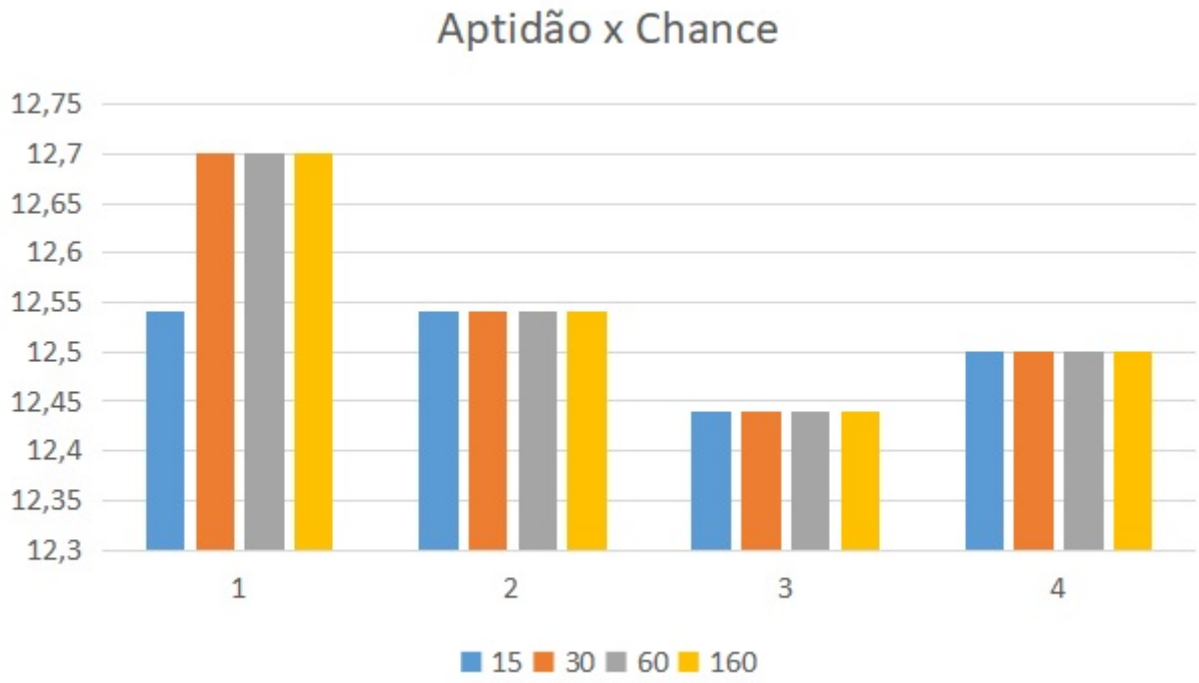
Fonte: Elaborado pelo Autor, 2019.

Figura 36 – Tempo de Execução Variando a Taxa de Mortalidade



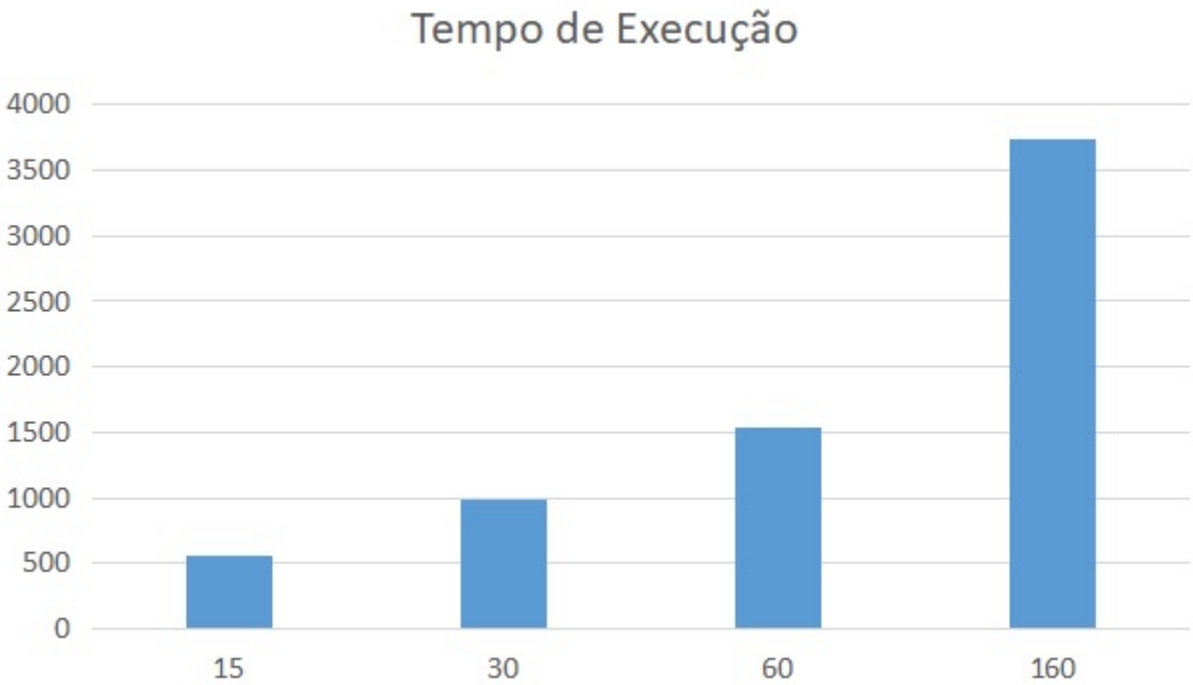
Fonte: Elaborado pelo Autor, 2019.

Figura 37 – Aptidão X Chance



Fonte: Elaborado pelo Autor, 2019.

Figura 38 – Tempo de Execução Variando a Chance



Fonte: Elaborado pelo Autor, 2019.

Quadro 16 – Médias de aptidão por população no Algoritmo Genético Classe 2

População	Aptidão do Conjunto 1	Aptidão do Conjunto 2	Aptidão do Conjunto 3	Aptidão do Conjunto 4	Tempo total
20	13.9	14.32	13.94	14.08	**1446**
25	13.9	14.4	13.9	14.28	1892
30	**14.3**	**14.32**	**14.74**	**15.2**	2179

Quadro 17 – Médias de aptidão por população no Algoritmo Genético Classe 2

Tx. Mortalidade	Aptidão do Conjunto 1	Aptidão do Conjunto 2	Aptidão do Conjunto 3	Aptidão do Conjunto 4	Tempo total
70%	13.9	13.72	13.82	14.08	1142
80%	13.9	14.32	13.94	14.08	1446
90%	**14.46**	**14.32**	**14.66**	**14.64**	**1129**

Quadro 18 – Médias de aptidão por chances Classe 2

Chance	Aptidão do Conjunto 1	Aptidão do Conjunto 2	Aptidão do Conjunto 3	Aptidão do Conjunto 4	Tempo total
15	13.9	14.32	13.82	14.08	**390**
30	13.9	14.32	13.82	14.08	636
60	13.9	14.32	**14.78**	15.52	1936
160	**14.52**	**14.84**	14.78	**15.72**	14331

5.0.2 Análise dos resultados – Classe 2

A classe 2 é definida por 70% dos itens muito largos e os demais aleatórios. O Quadro 17 demonstra as médias de execução deste cenário com o Algoritmo Genético, variando o parâmetro de número de população, com taxa de mortalidade em 80% e 30 gerações.

É possível observar que na classe 2, que aumentando o número da população ocorreu um aumento significativo no percentual de aptidão e no tempo de execução.

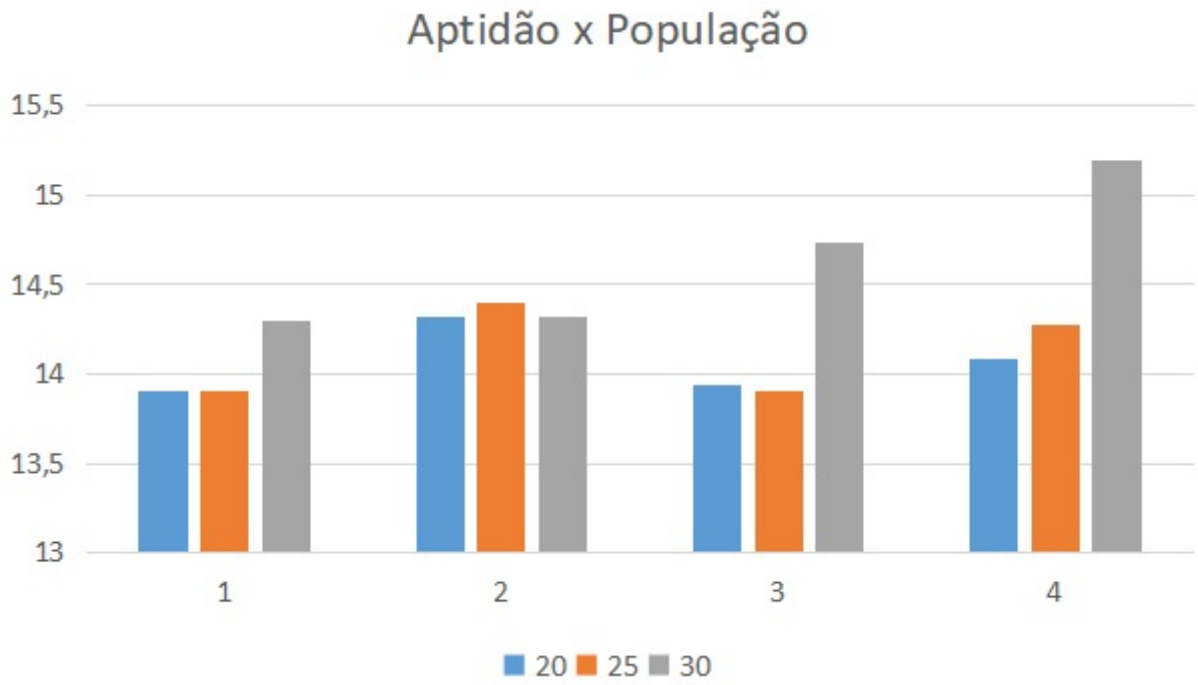
Ainda no Algoritmo Genético, outro fator testado foi o percentual de ponto de corte, conforme apresentado no Quadro 18, onde para termos de comparação foi utilizado de base a média da geração de população 20, representada na primeira linha do Quadro 17, efetuada no teste anterior.

Nota-se que variando o ponto de corte para 90% houve um ganho razoável na aptidão dos estados, e uma queda no tempo de execução se comparado ao corte em 80%, sendo muito similar ao de 70%.

No Algoritmo de Busca em Espaços de Estados o teste foi conduzido da mesma forma, porém alterando a quantidade de chances, para verificar a saída de máximos locais. Os dados de teste constam na Tabela 19.

Na Figura 39 consta o gráfico demonstrando a relação da variação da aptidão dos conjuntos de acordo com a população, e na Figura 40 a soma do tempo de execução em todos os conjuntos. Na Figura 41 consta a relação da variação da aptidão dos conjuntos de acordo com a taxa de mortalidade, e na Figura 42 a soma do tempo de execução em todos os conjuntos, ambas as variações presentes no Algoritmo Genético. Na Figura 43 conta o gráfico demonstrando a relação da variação da aptidão dos conjuntos de acordo com a chance, e na Figura 44 a soma do tempo de execução em todos os conjuntos, referentes ao algoritmo de Busca em Espaços de Estados.

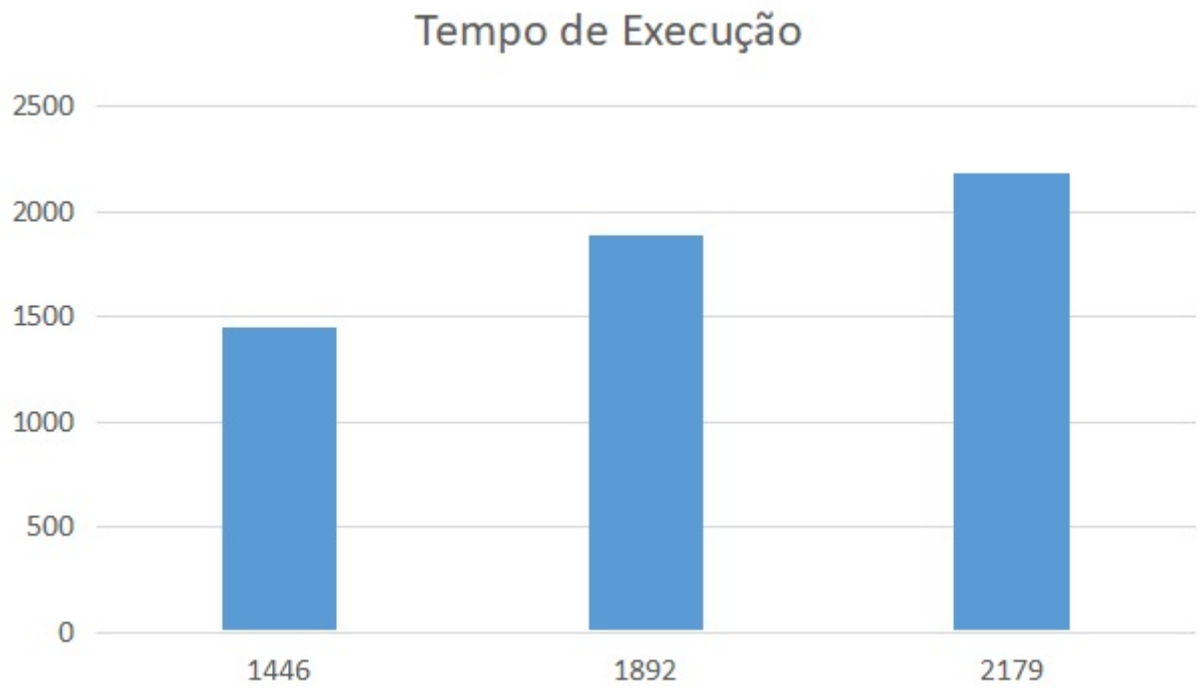
Figura 39 – Aptidão X População



Fonte: Elaborado pelo Autor, 2019.

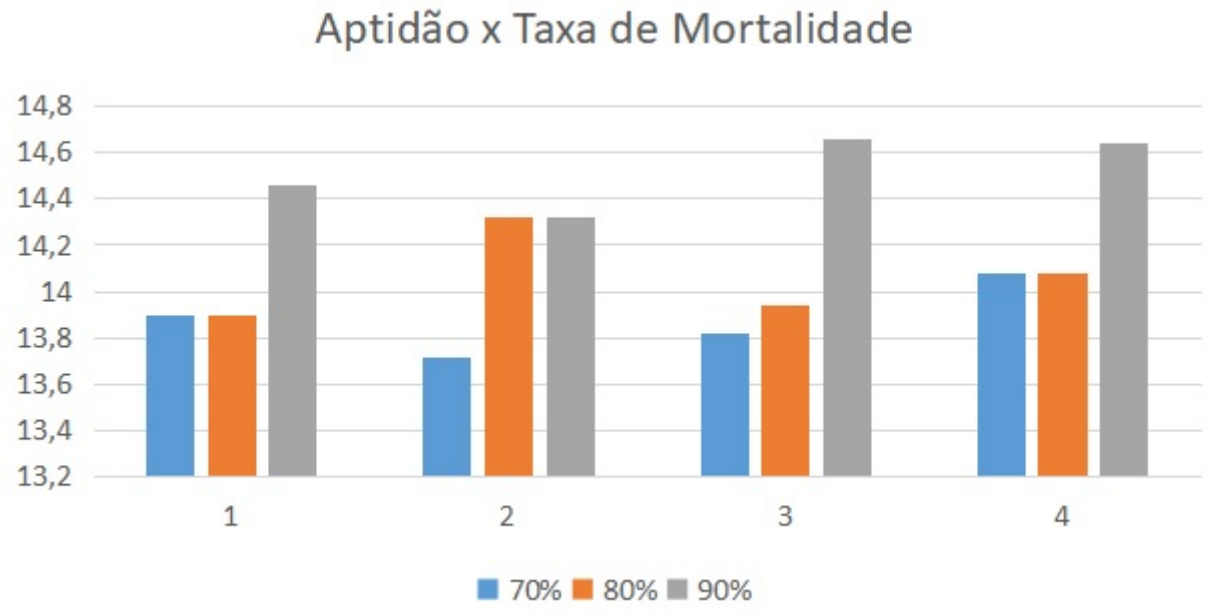
Nota-se que a busca só encontrou resultados melhores que Algoritmo Genético com mais de 60 iterações.

Figura 40 – Tempo de Execução variando a População



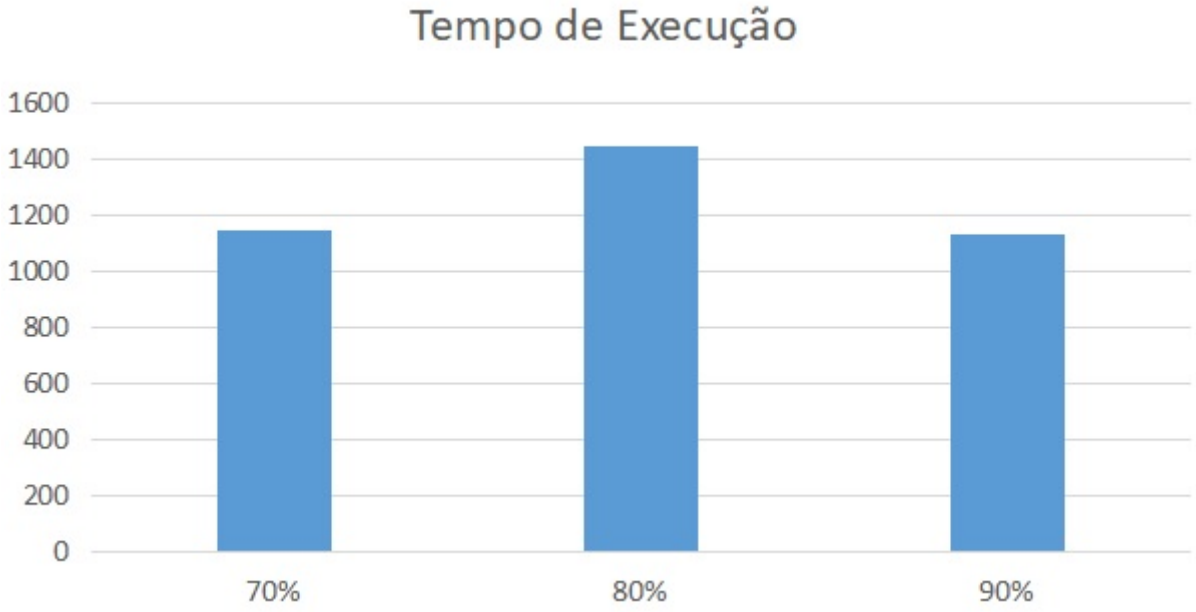
Fonte: Elaborado pelo Autor, 2019.

Figura 41 – Aptidão X Taxa de Mortalidade



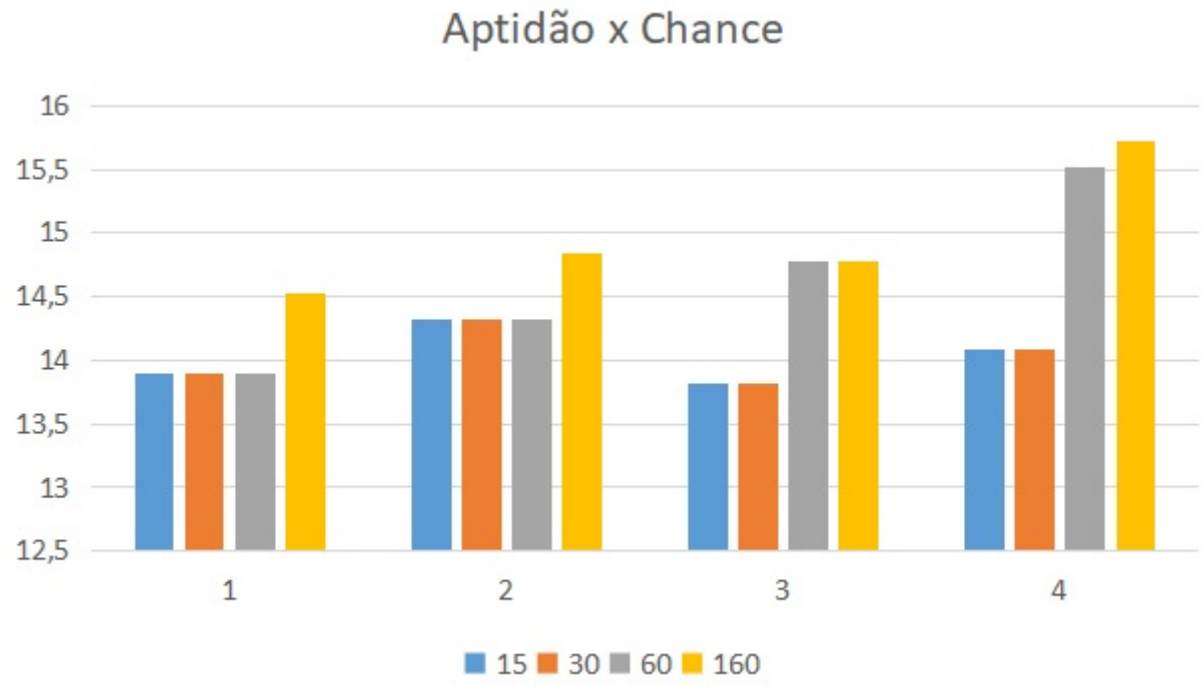
Fonte: Elaborado pelo Autor, 2019.

Figura 42 – Tempo de Execução Variando a Taxa de Mortalidade



Fonte: Elaborado pelo Autor, 2019.

Figura 43 – Aptidão X Chance



Fonte: Elaborado pelo Autor, 2019.

Figura 44 – Tempo de Execução Variando a Chance

Fonte: Elaborado pelo Autor, 2019.

Quadro 19 – Médias de aptidão por população no Algoritmo Genético Classe 3

População	Aptidão do Conjunto 1	Aptidão do Conjunto 2	Aptidão do Conjunto 3	Aptidão do Conjunto 4	Tempo total
20	15.54	**15.9**	14.92	**16.58**	**1553**
25	15.54	15.9	**15.16**	16.58	2161
30	**15.66**	15.9	15.16	16.58	2295

Quadro 20 – Médias de aptidão por população no Algoritmo Genético Classe 3

Tx. Mortalidade	Aptidão do Conjunto 1	Aptidão do Conjunto 2	Aptidão do Conjunto 3	Aptidão do Conjunto 4	Tempo total
70%	15.54	15.9	14.92	16.58	1591
80%	15.54	15.9	14.92	16.58	1553
90%	**15.66**	**15.9**	**14.92**	**16.58**	**1414**

5.0.3 Análise dos resultados – Classe 3

A classe 2 é definida por 70% dos itens muito compridos e os demais aleatórios. O Quadro 20 demonstra as médias de execução deste cenário com o Algoritmo Genético, variando o parâmetro de número de população, com taxa de mortalidade em 80% e 30 gerações.

É possível observar que na classe 3, que aumentando o número da população ocorreu um aumento mínimo no percentual de aptidão do conjunto 1 e 3 e no tempo de execução em geral.

Ainda no Algoritmo Genético, outro fator testado foi o percentual de ponto de corte, conforme apresentado no quadro 21, onde para termos de comparação foi utilizado de base a média da geração de população 20, representada na primeira linha do Quadro 20, efetuada no teste anterior.

Nota-se que variando o ponto de corte para 90% houve um ganho razoável na aptidão dos estados, e uma queda no tempo de execução se comparado ao corte em 80%, sendo muito similar ao de 70%.

No Algoritmo de Busca em Espaços de Estados o teste foi conduzido da mesma forma, porém alterando a quantidade de chances, para verificar a saída de máximos locais. Os dados de teste constam na Tabela 22.

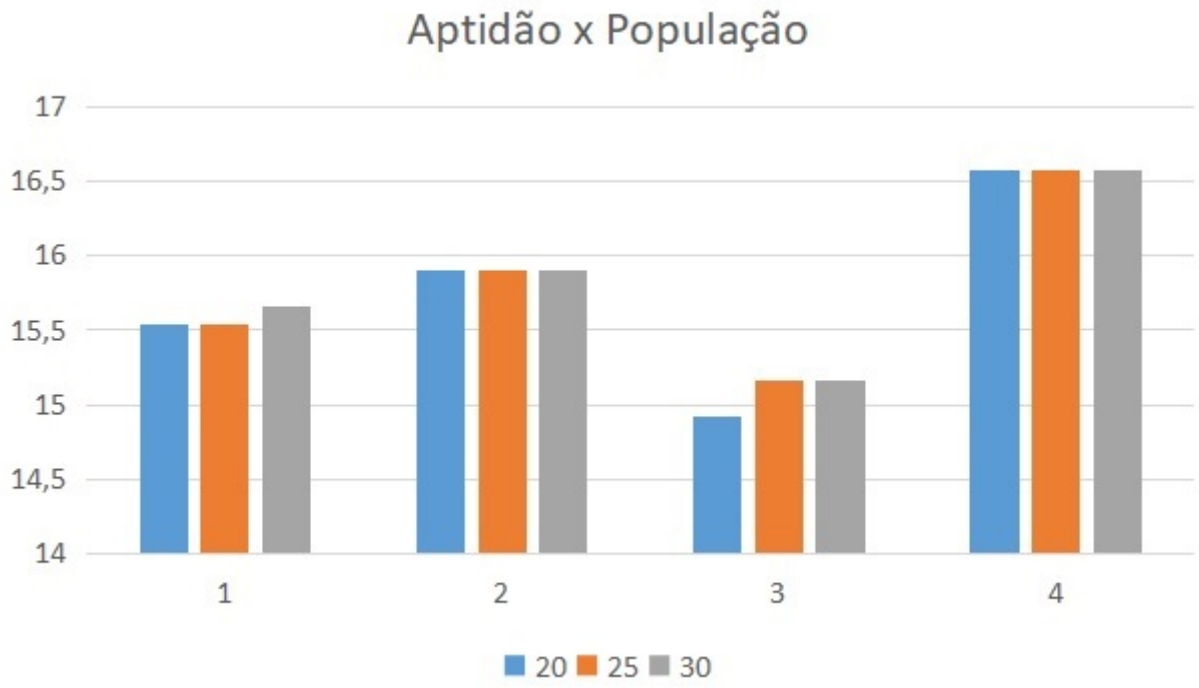
Na Figura 45 consta o gráfico demonstrando a relação da variação da aptidão dos conjuntos de acordo com a população, e na Figura 46 a soma do tempo de execução em todos os conjuntos. Na Figura 47 consta a relação da variação da aptidão dos conjuntos de acordo com a taxa de mortalidade, e na Figura 48 a soma do tempo de execução em todos os conjuntos, ambas as variações presentes no Algoritmo Genético. Na Figura 49 conta o gráfico demonstrando a

Quadro 21 – Médias de aptidão por chances Classe 3

Chance	Aptidão do Conjunto 1	Aptidão do Conjunto 2	Aptidão do Conjunto 3	Aptidão do Conjunto 4	Tempo total
15	**15.54**	15.9	14.92	**16.58**	**357**
30	15.54	15.9	14.92	16.58	648
60	15.54	15.9	14.92	16.58	1349
160	15.54	**15.98**	**15.04**	16.58	6952

relação da variação da aptidão dos conjuntos de acordo com a chance, e na Figura 50 a soma do tempo de execução em todos os conjuntos, referentes ao algoritmo de Busca em Espaços de Estados.

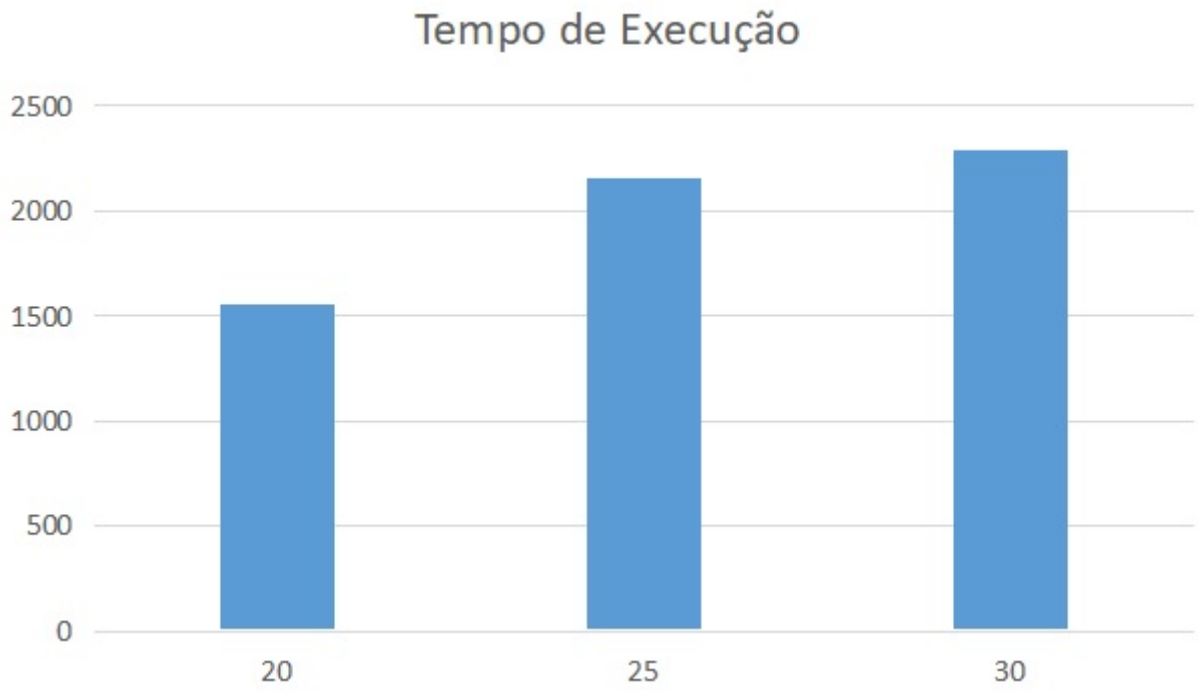
Figura 45 – Aptidão X População



Fonte: Elaborado pelo Autor, 2019.

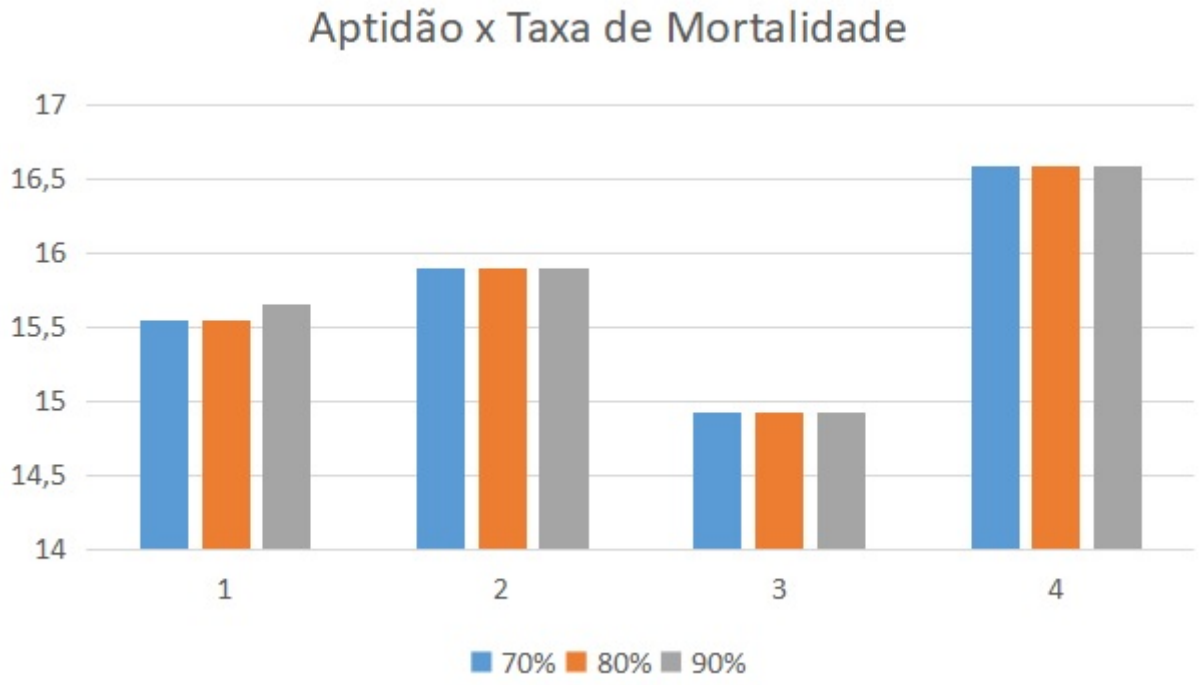
Nota-se que nos conjunto 1 e 3 o Algoritmo Genético encontrou uma alocação melhor que a busca. Esse resultado se deve ao fato de que o caminho para o estado ótimo requer que se passe por estados onde a heurística é reduzida, como exemplificado na Figura 51, fazendo com que a Busca em Espaços de Estados não expanda o estado alvo até 160 chances, tendo seu tempo de execução superior a 3x o do Algoritmo Genético.

Figura 46 – Tempo de Execução variando a População



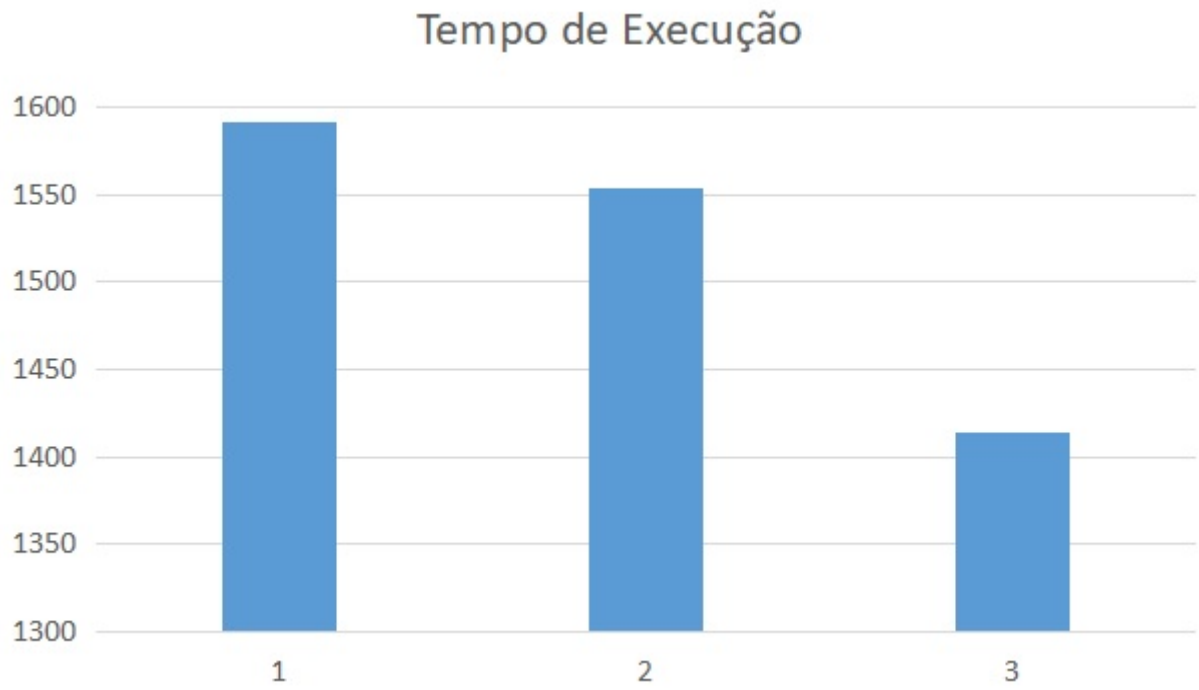
Fonte: Elaborado pelo Autor, 2019.

Figura 47 – Aptidão X Taxa de Mortalidade



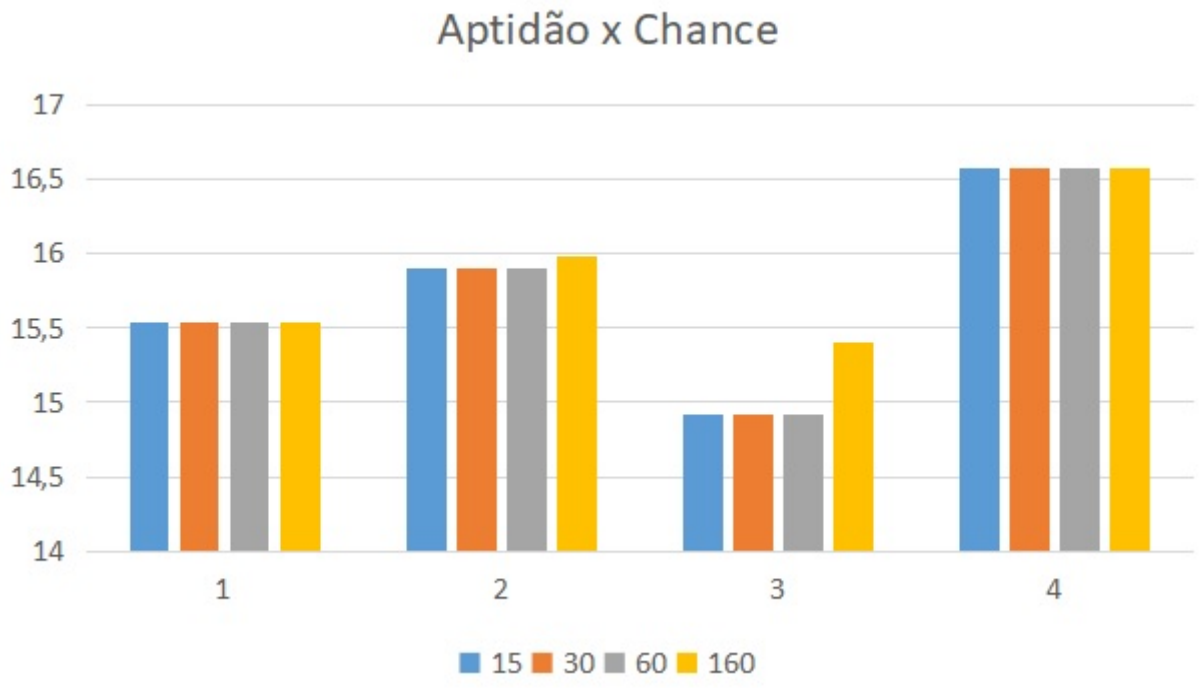
Fonte: Elaborado pelo Autor, 2019.

Figura 48 – Tempo de Execução Variando a Taxa de Mortalidade



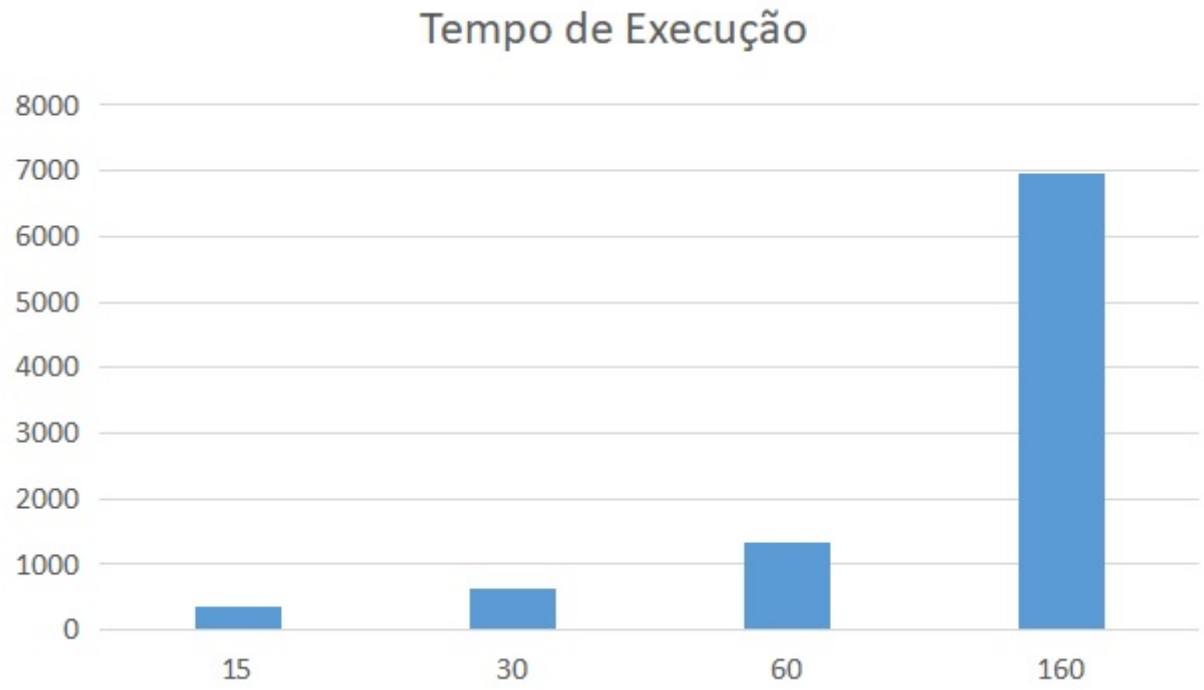
Fonte: Elaborado pelo Autor, 2019.

Figura 49 – Aptidão X Chance



Fonte: Elaborado pelo Autor, 2019.

Figura 50 – Tempo de Execução Variando a Chance



Fonte: Elaborado pelo Autor, 2019.

Figura 51 – Caminho do objeto para chegar ao máximo global.

Fonte: Acervo pelo Autor, 2019.

Quadro 22 – Médias de aptidão por população no Algoritmo Genético Classe 4

População	Aptidão do Conjunto 1	Aptidão do Conjunto 2	Aptidão do Conjunto 3	Aptidão do Conjunto 4	Tempo total
20	15.02	15.62	14.78	12.72	**1385**
25	15.34	15.62	**15.34**	12.72	1941
30	**15.74**	**16.22**	15.34	**13.05**	2179

Quadro 23 – Médias de aptidão por população no Algoritmo Genético Classe 4

Tx. Mortalidade	Aptidão do Conjunto 1	Aptidão do Conjunto 2	Aptidão do Conjunto 3	Aptidão do Conjunto 4	Tempo total
70%	15.02	15.5	15.34	12.34	1393
80%	15.02	15.62	15.78	12.78	1518
90%	**16.06**	**16.22**	**15.34**	**12.9**	**1321**

Quadro 24 – Médias de aptidão por chances Classe 4

Chance	Aptidão do Conjunto 1	Aptidão do Conjunto 2	Aptidão do Conjunto 3	Aptidão do Conjunto 4	Tempo total
15	15.62	**15.5**	15.34	12.34	**338**
30	16.1	15.5	**15.78**	**12.9**	748
60	**16.78**	15.5	15.78	12.9	1846
160	16.78	15.5	15.78	12.9	9634

5.0.4 Análise dos resultados – Classe 4

A classe 4 é definida por 70% dos itens são pequenos e os demais aleatórios. O Quadro 23 demonstra as médias de execução deste cenário com o Algoritmo Genético, variando o parâmetro de número de população, com taxa de mortalidade em 80% e 30 gerações.

É possível observar que na classe 4, que aumentando o número da população ocorreu um aumento significativo no percentual de aptidão e no tempo de execução, conforme o Quadro 24. Ainda no Algoritmo Genético, outro fator testado foi o percentual de ponto de corte, onde para termos de comparação foi utilizado de base a média da geração de população 20, representada na primeira linha do Quadro 23, efetuada no teste anterior.

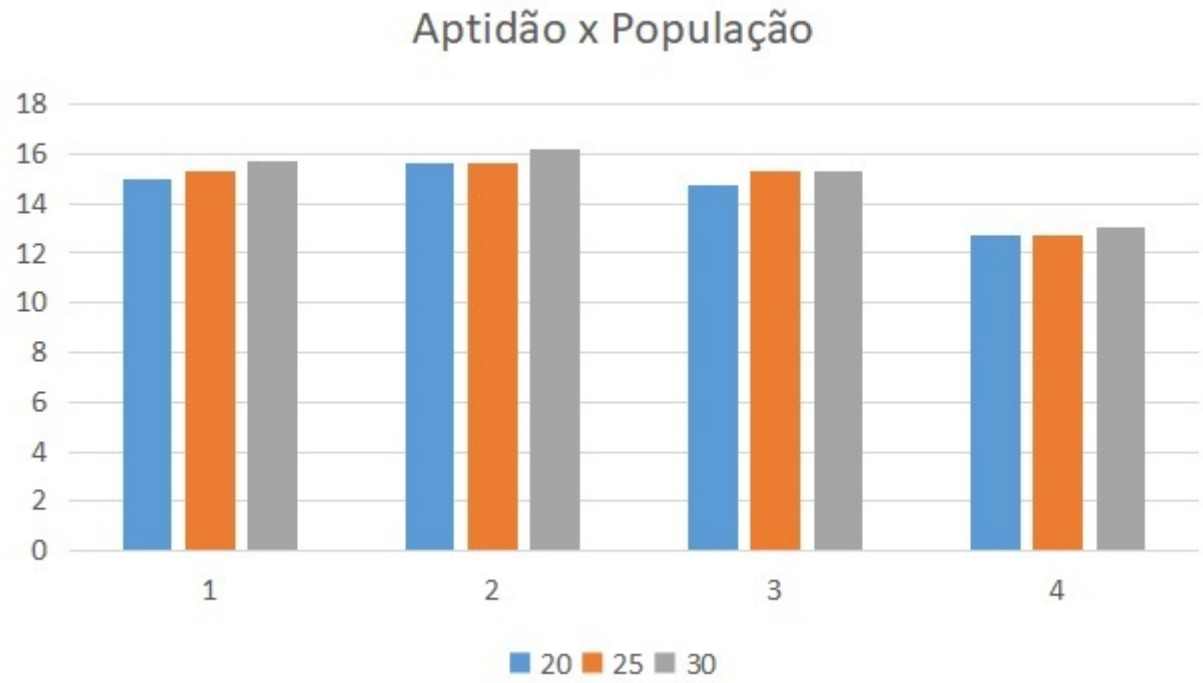
Nota-se que variando o ponto de corte para 90% houve um ganho razoável na aptidão dos estados, e uma queda no tempo de execução se comparado ao corte em 80%, sendo muito similar ao de 70%.

No Algoritmo de Busca em Espaços de Estados o teste foi conduzido da mesma forma, porém alterando a quantidade de chances, para verificar a saída de máximos locais. Os dados de teste constam na Tabela 25.

Na Figura 52 consta o gráfico demonstrando a relação da variação da aptidão dos

conjuntos de acordo com a população, e na Figura 53 a soma do tempo de execução em todos os conjuntos. Na Figura 54 consta a relação da variação da aptidão dos conjuntos de acordo com a taxa de mortalidade, e na Figura 55 a soma do tempo de execução em todos os conjuntos, ambas as variações presentes no Algoritmo Genético. Na Figura 56 conta o gráfico demonstrando a relação da variação da aptidão dos conjuntos de acordo com a chance, e na Figura 57 a soma do tempo de execução em todos os conjuntos, referentes ao algoritmo de Busca em Espaços de Estados.

Figura 52 – Aptidão X População

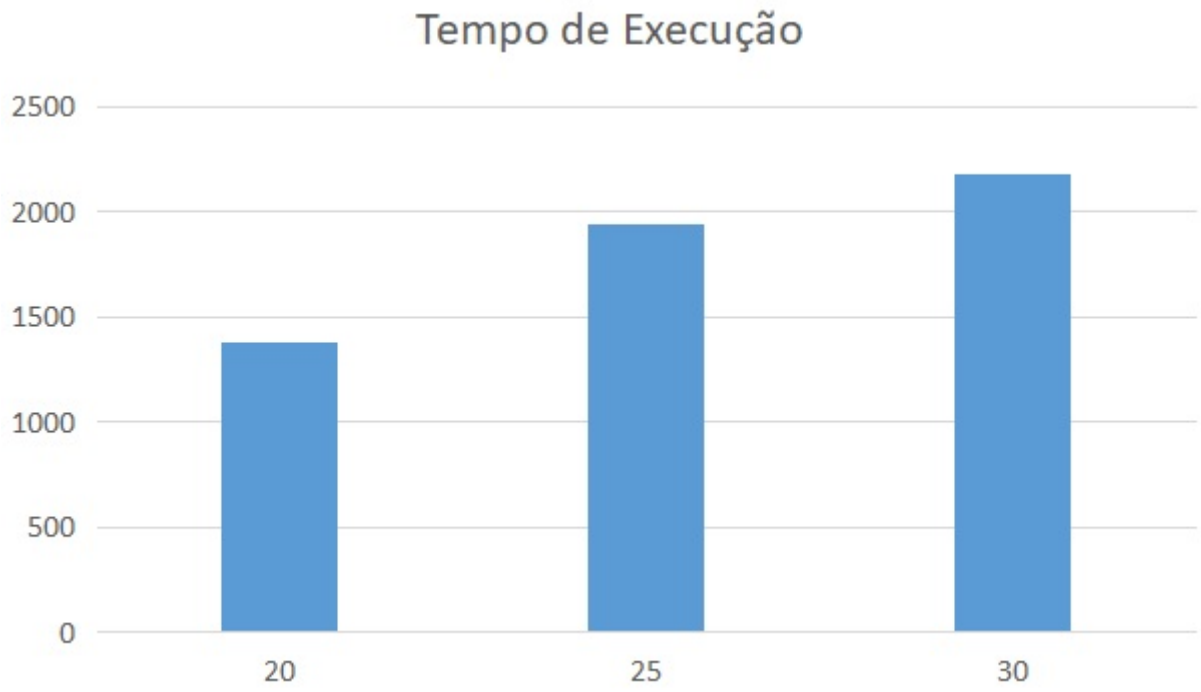


Fonte: Elaborado pelo Autor, 2019.

Nota-se que na classe 4 ocorre o mesmo caso da classe 3, onde o Algoritmo Genético expande nós que o algoritmo de Busca em Espaços de Estados descarta, chegando a um máximo global.

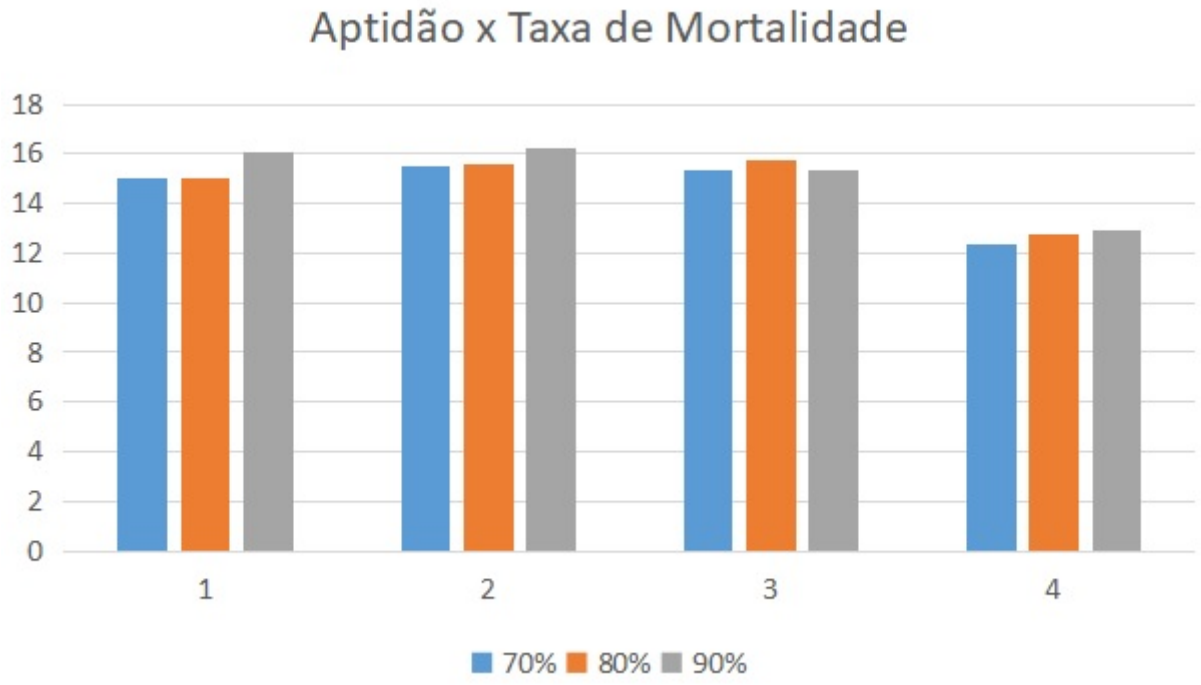
Durante todos os testes, a maioria dos resultados já apresenta uma aptidão razoável, provando que o sistema de pré alocação, utilizado em ambos os algoritmos é eficiente.

Figura 53 – Tempo de Execução variando a População



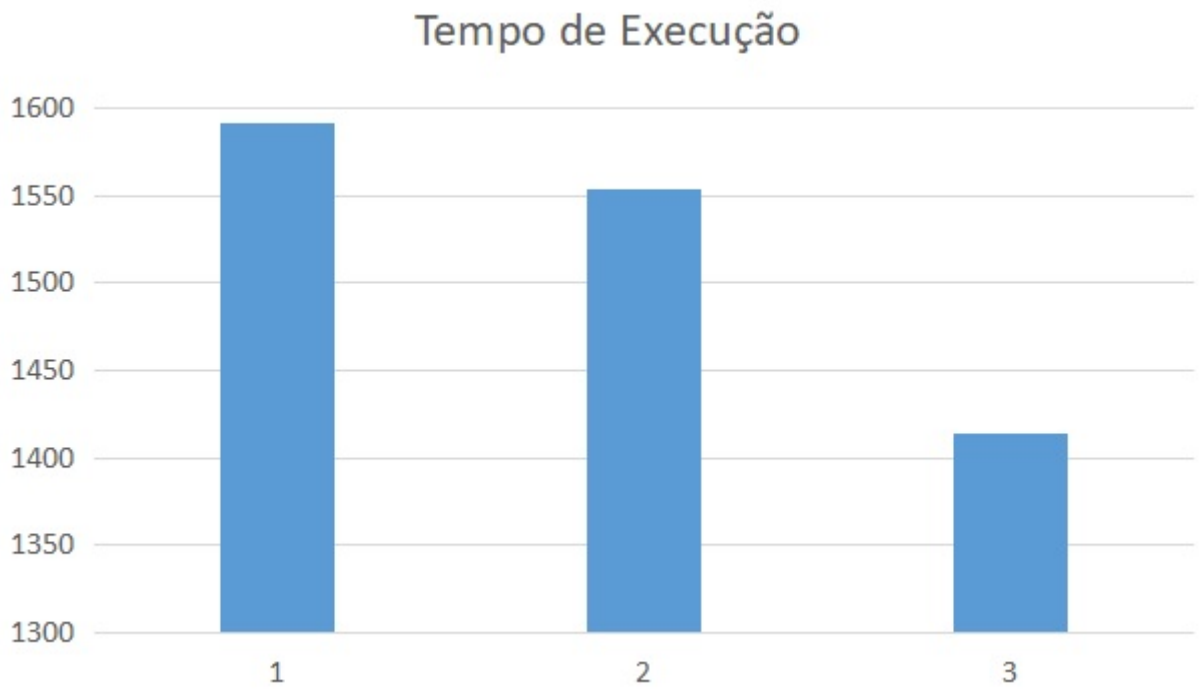
Fonte: Elaborado pelo Autor, 2019.

Figura 54 – Aptidão X Taxa de Mortalidade



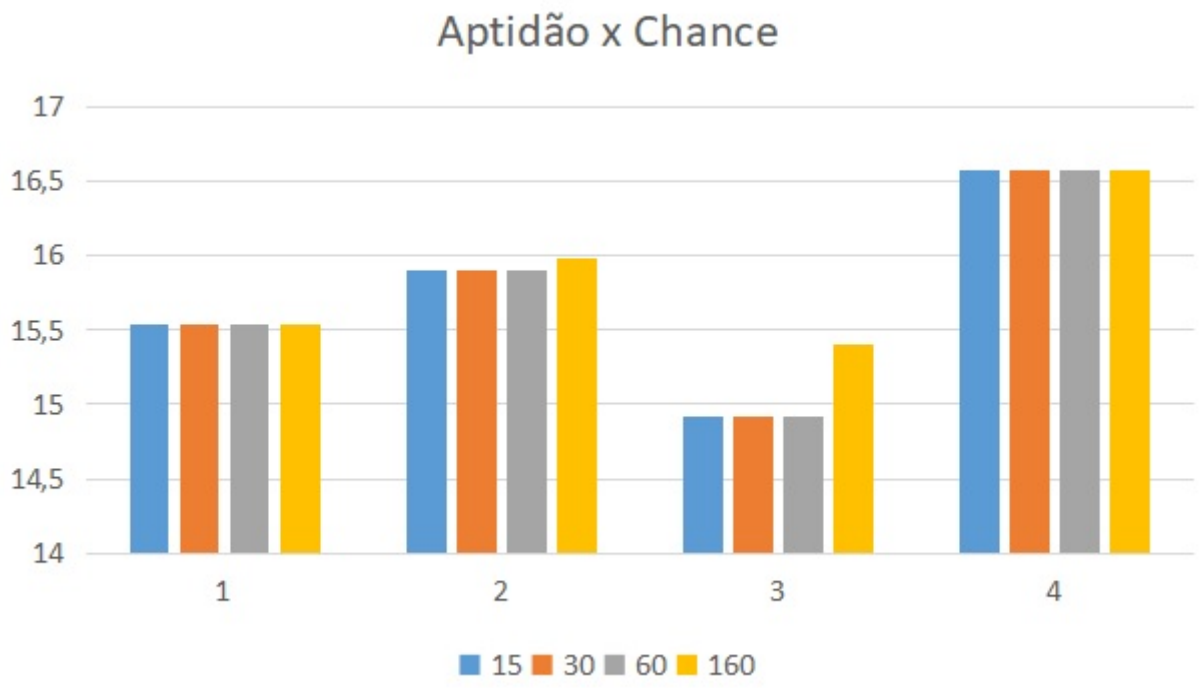
Fonte: Elaborado pelo Autor, 2019.

Figura 55 – Tempo de Execução Variando a Taxa de Mortalidade



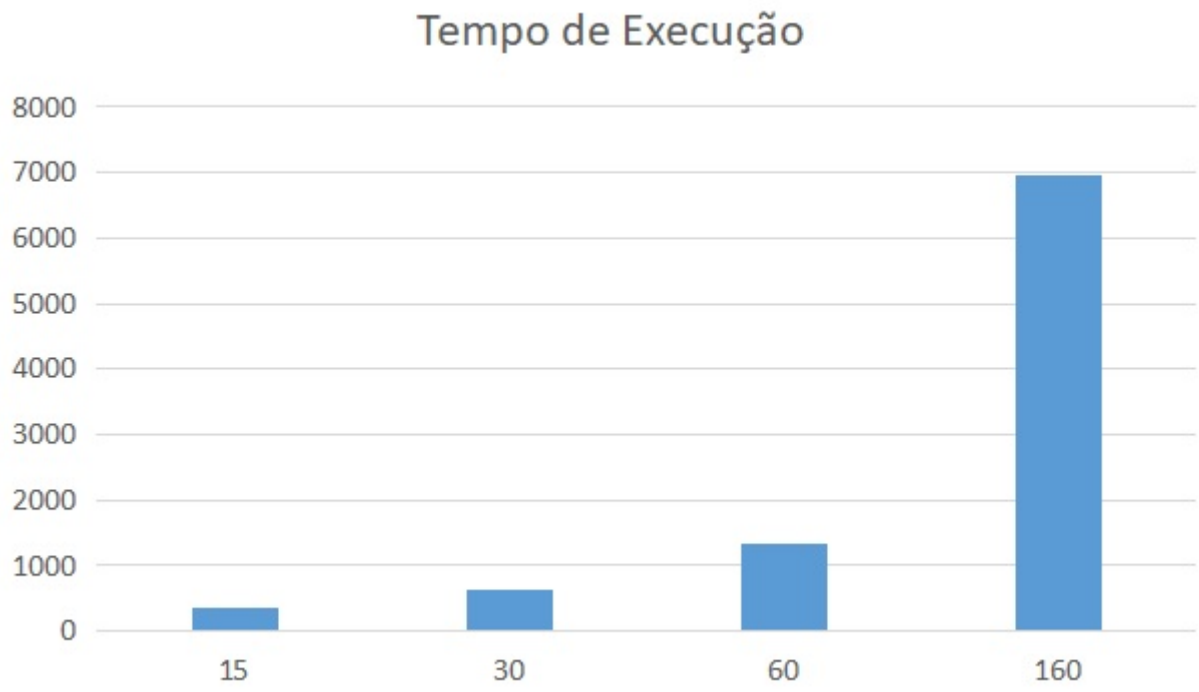
Fonte: Elaborado pelo Autor, 2019.

Figura 56 – Aptidão X Chance



Fonte: Elaborado pelo Autor, 2019.

Figura 57 – Tempo de Execução Variando a Chance



Fonte: Elaborado pelo Autor, 2019.

6 CONCLUSÃO

A boa alocação de objetos em confinamentos é um fator crucial no setor de logística, acarretando uma economia significativa às empresas do ramo se bem empregada. Tal tarefa pode ser delegada a um sistema que utilize algum mecanismo de busca para encontrar a melhor disposição dos objetos.

Dentre as diversas técnicas de Inteligência Artificial existem os algoritmos de busca, como a Busca em Espaços de Estados, que propõe expandir os espaços de estados em si até encontrar um bom caminho, como também o Algoritmo Genético, que utiliza conceitos da biologia como mutação, reprodução, entre outros, para buscar um estado alvo. A partir disto, este trabalho se propôs a modelar e implementar um sistema de alocação de cargas utilizando ambas as técnicas.

No decorrer do trabalho, foi realizada uma extensa pesquisa bibliográfica na literatura a fim de identificar, nos trabalhos correlatos, os modelos que mais têm sido usados para estes tipos de busca, bem como aqueles que obtiveram maior destaque. Após realizar tal coleta de dados, foi modelada a estrutura de representação do estado, onde os pacotes estão alocados no contêiner, feito uma métrica para avaliar a qualidade do estado referente ao espaço contíguo disponível,formulou-se funções para a movimentação dos pacotes dentro do confinamento sem que aconteça a sobreposição dos mesmos,e foram implementados uma representação gráfica do estado e os algoritmos de busca em si, que em conjunto com os elementos anteriores compõem o sistema.

Durante o desenvolvimento do trabalho, as abordagens sofreram mudanças por diversas vezes até atingir o resultado final. Um dos pontos interessantes do trabalho foi a forma de mapeamento das caixas dentro de um contêiner, através de coordenadas `X, Y`, que se mostraram indispensáveis para este tipo de solução, como também a abordagem de limites de paradas nas funções de transição, que impedem as caixas de se sobreporem.

O Algoritmo de Busca em Espaços de Estados implementado nesse trabalho foi feito com base na heurística A* que utiliza uma função matemática como forma de calcular uma nota para cada estado, nota essa que é utilizada tanto para expandir estados com maior prioridade, quanto para definir qual o melhor estado a ser apresentado para o usuário.

Para fins de comparação de técnicas, também foi implementado um Algoritmo Genético com características de hereditariedade, onde a cada geração uma parte da população com

baixo *fitness*, definida pela taxa de mortalidade, dá lugar a novos elementos com características herdadas dos cromossomos com melhor *fitness*.

Para avaliar o desempenho de ambas as técnicas, foram elaborados quatro cenários variando a proporção dos pacotes, com quatro conjuntos de elementos distintos em cada cenário. Diante de cada conjunto, foram testadas diversas configurações alterando a taxa de mortalidade e população no Algoritmo Genético e a quantidade de chances na Busca em Espaços de Estados.

Ao comparar os resultados obtidos pela Busca em Espaço de Estados com os resultados do Algoritmo Genético, percebe-se certa vantagem em relação a Busca em Espaços de Estados, isso devido ao fato de que o Algoritmo Genético apenas busca a sequência de movimentos executados sobre o estado inicial. Porém, em um caso específico observou-se uma superioridade do Algoritmo Genético, pois o estado alvo necessitava uma sequencia de passos onde a heurística diminuía, sendo esses estados não expandidos pela Busca em Espaços de Estados, mas considerados pelo Algoritmo Genético devido à mutação e recombinação.

Apesar dos resultados obtidos pelo Algorítimo Genético, destaca-se que essa técnica utiliza aleatoriedade, logo, é possível que não se encontre um estado alvo todas as vezes em que este algoritmo for executado.

6.1 TRABALHOS FUTUROS

O projeto apresentado nesta monografia atingiu todos os objetivos propostos, mas além do que foi alcançado, existem pontos que podem ser melhorados e novas funcionalidades que podem ser implementadas. São eles:

a) Implementar a possibilidade de operar com objetos tridimensionais.

b) Desenvolver uma nova heurística para garantir a estabilidade dos produtos dentro do confinamento.

c) Formalizar um método de busca que faça uso das características de ambos os algoritmos.

d) Considerar mais restrições para o problema, como por exemplo fragilidade das caixas, outros tipos de confinamentos, entre outros.

e) Tornar a execução do algoritmo *multi-thread* para melhorias de performance.

REFERÊNCIAS

BALLOU, Ronald H. **Gerenciamento da cadeia de suprimentos: logística empresarial**. Rio de Janeiro - RJ: Bookman, 2006.

BOWERSOX, Donald J.; CLOSS, David J. **ogística empresarial: o processo de integração da cadeia de suprimentos**. São Paulo: Atlas, 2001.

BRATKO, Ivan. **Prolog Programing For Artificial Inteligence**. 1. ed. Yugoslavia: Addison-Wesley, 1990.

BUENO, Fabrício. Métodos heurísticos: Teoria e implementações. **Instituto Federal de Santa Catarina, Araranguá**, 2009.

CAVALCANTI, George Moraes. **SIP – Sistema Inteligente de Carregamento de Paletes**. 56 p. Monografia (Graduação) — Escola Politécnica de Pernambuco, Recife, 2009. Disponível em: <http://tcc.ecomp.poli.br/20091/TCC_George_Moraes_SIP.pdf>.

COELIS, Elenilce Lopes. **Logística empresarial**. [S.l.], 2008. Disponível em: <http://www.techoje.com.br/site/techoje/categoria/detalhe_artigo/507>.

CORMEN, Thomas H. **Introduction to algorithms**. [S.l.], 2009. Disponível em: <http://ressources.unisciel.fr/algoprog/s00aaroot/aa00module1/res/BCormen-AL20115DIntroduction_To_Algorithms-A3.pdf>.

CRUZ, RAPHAEL CARLOS; SOUZA, MJF; MINE, MT. Roteamento de veículos com coleta e entrega simultânea: uma abordagem heurística-parte ii. **Relatório técnico, Universidade Federal de Ouro Preto, Relatório Técnico do Programa Institucional de Bolsas de Iniciação Científica do CNPq-PIBIC/CNPq**, 2012.

FALKENAUER, Emanuel. **A hybrid grouping genetic algorithm for bin packing**. [S.l.], 1996. Disponível em: <https://link.springer.com/article/10.1007/BF00226291>.

FERNANDES, HÉLTON JOSÉ OLIVEIRA. SOFTWARE PARA DETERMINAÇÃO DA CIRCULARIDADE DA CÓRNEA EM TEMPO DE EXECUÇÃO NO PÓS-OPERATÓRIO EM CERATOPLASTIA. 2016.

LI, Xueping; ZHAO, Zhaoxia; ZHANG, Zhang. **A genetic algorithm for the three-dimensional bin packing problem with heterogeneous bins**. 1. ed. [S.l.], 2008. Disponível em: <https://www.researchgate.net/publication/273121476_A_genetic_algorithm_for_the_three-dimensional_bin_packing_problem_with_heterogeneous_bins>.

LINDEN, Ricardo. Algoritmos Genéticos. Rio de Janeiro, 2008.

MARQUES, Robert. **História da Logística**. [S.l.], 2008. Disponível em: <http://www.administradores.com.br/artigos/marketing/historia-da-logistica/24829/>.

MARTELLO, Silvano; VIGO, Daniele. **Exact Solution of the Two-Dimensional Finite Bin Packing Problem**. [S.l.], 1998. Disponível em: <http://pubsonline.informs.org/doi/abs/10.1287/mnsc.44.3.388>.

MENDONÇA, Juliana Karim. **Inteligência nos Negócios: Logística faz a diferença**. [S.l.], 2013. Disponível em: <http://docplayer.com.br/107848-Inteligencia-nos-negocios-logistica-faz-a-diferenca.html>.

MIRANDA, Márcio Nunes de. **Algoritmos Genéticos: Fundamentos e Aplicações**. 2007.

NOVAES, Antonio G. **Logística e Gerenciamento da Cadeia de Distribuição**. 4. ed. [S.l.]: Elsevier, 2015.

PACHECO, André. Computação Inteligente: Máquinas aprendendo a solucionar problemas complexos. 2016.

PARK, Kyungchul. **Modeling and solving the spatial block scheduling problem in a shipbuilding company**. [S.l.], 1996. Disponível em: <http://www.sciencedirect.com/science/article/pii/0360835296000046>.

POLI, Riccardo et al. **A field guide to genetic programming**. [S.l.]: Lulu. com, 2008.

POZO, Aurora et al. Computação evolutiva. **Universidade Federal do Paraná, 61p.(Grupo de Pesquisas em Computação Evolutiva, Departamento de Informática-Universidade Federal do Paraná)**, 2005.

RICH, Elaine; KNIGHT, Kevin. **Inteligência Artificial**. 3. ed. New Delhi: McGraw, 2009.

RUSSEL, Stuart; NORVIG, Peter. **Inteligência artificial**. 3. ed. New Jersey: Pearson Education, 2013.

SILVA, José Lassance de Castro; SOMA, Nei Yoshihiro. Um algoritmo polinomial para o problema de empacotamento de contêineres com estabillidade estática da carga. **Pesquisa Operacional**, SciELO Brasil, v. 23, n. 1, p. 79–98, 2003.

____. ____. **Pesquisa Operacional**, SciELO Brasil, v. 23, n. 1, p. 79–98, 2003.

TONETTO, Leandro. **O Papel das heurísticas no julgamento e na tomada de decisão sob incerteza**. 1. ed. São Paulo: [s.n.], 2006.

WANG, Hongfeng; CHEN, Yanjie. **A hybrid genetic algorithm for 3D bin packing problems**. [S.l.], 2010. Disponível em: <http://ieeexplore.ieee.org/document/5645211/>.

WASCHER, Gerhard. **An improved typology of cutting and packing problems**. [S.l.], 2007.

APÊNDICE A – CLASSE DESENHAR

```
1  public class Desenhar extends JFrame {
2
3      int[][] conf;
4      int[][][] obj;
5      int tamanho;
6      float rate = 40;
7      int deslocamento = 20;
8      int[] ordem = {0, 1, 3, 2};
9
10     public Desenhar(int[][] conf, int[][][] obj, int tamanho) {//
11         if (conf[2][0] > 20 || conf[2][1] > 20) {
12             rate = 20;
13         }
14         if (conf[2][0] > 200 || conf[2][1] > 200) {
15             rate = 2;
16         }
17         if (conf[2][0] > 500 || conf[2][1] > 500) {
18             rate = (float) 0.5;
19         }
20         setSize((int) (conf[2][0] * rate), (int) (conf[2][1] * rate + d
21         setVisible(true);
22         this.conf = conf;
23         this.obj = obj;
24         this.tamanho = tamanho;
25     }
26     public void paint(Graphics g) {
27         int[] x = new int[4];
28         int[] y = new int[4];
29         int passo = conf[2][0];
30         for (int i = 0; i <= tamanho; i++) {
```

```
31                for (int cont = 0; cont <= passo; cont++) {
32                    for (int j = 0; j <= 3; j++) {
33                        x[ordem[j]] = (int) (obj[i][j][0] * rate);
34                        y[ordem[j]] = (int) (obj[i][j][1] * rate + deslocam
35                    }
36                    g.setColor(Color.getHSBColor(i * 50, i * 100, i * 100))
37                    g.fillPolygon(x, y, 4);
38                }
39            }
40        }
41    }
```

APÊNDICE B – DESCRIÇÃO DOS PACOTES TESTADOS

Quadro 25 – Pacotes testados na Classe 1

Conjunto 1	Conjunto 2	Conjunto 3	Conjunto 4
Pacote nº 1	Pacote nº 1	Pacote nº 1	Pacote nº 1
Largura = 15	Largura = 15	Largura = 18	Largura = 15
Comprimento = 20	Comprimento = 16	Comprimento = 21	Comprimento = 22
Pacote nº 2	Pacote nº 2	Pacote nº 2	Pacote nº 2
Largura = 17	Largura = 20	Largura = 17	Largura = 19
Comprimento = 16	Comprimento = 20	Comprimento = 15	Comprimento = 16
Pacote nº 3	Pacote nº 3	Pacote nº 3	Pacote nº 3
Largura = 20	Largura = 17	Largura = 20	Largura = 21
Comprimento = 17	Comprimento = 15	Comprimento = 15	Comprimento = 17
Pacote nº 4	Pacote nº 4	Pacote nº 4	Pacote nº 4
Largura = 15	Largura = 20	Largura = 19	Largura = 15
Comprimento = 15	Comprimento = 17	Comprimento = 16	Comprimento = 18
Pacote nº 5	Pacote nº 5	Pacote nº 5	Pacote nº 5
Largura = 5	Largura = 5	Largura = 5	Largura = 5
Comprimento = 15	Comprimento = 4	Comprimento = 10	Comprimento = 3

Quadro 26 – Pacotes testados na Classe 2

Conjunto 1	Conjunto 2	Conjunto 3	Conjunto 4
Pacote nº 1	Pacote nº 1	Pacote nº 1	Pacote nº 1
Largura = 15	Largura = 15	Largura = 18	Largura = 15
Comprimento = 5	Comprimento = 7	Comprimento = 5	Comprimento = 8
Pacote nº 2	Pacote nº 2	Pacote nº 2	Pacote nº 2
Largura = 17	Largura = 20	Largura = 17	Largura = 19
Comprimento = 9	Comprimento = 12	Comprimento = 10	Comprimento = 5
Pacote nº 3	Pacote nº 3	Pacote nº 3	Pacote nº 3
Largura = 20	Largura = 17	Largura = 20	Largura = 21
Comprimento = 13	Comprimento = 4	Comprimento = 7	Comprimento = 7
Pacote nº 4	Pacote nº 4	Pacote nº 4	Pacote nº 4
Largura = 15	Largura = 20	Largura = 19	Largura = 15
Comprimento = 3	Comprimento = 5	Comprimento = 8	Comprimento = 4
Pacote nº 5	Pacote nº 5	Pacote nº 5	Pacote nº 5
Largura = 5	Largura = 5	Largura = 5	Largura = 5
Comprimento = 15	Comprimento = 4	Comprimento = 10	Comprimento = 3

Quadro 27 – Pacotes testados na Classe 3

Conjunto 1	Conjunto 2	Conjunto 3	Conjunto 4
Pacote nº 1	Pacote nº 1	Pacote nº 1	Pacote nº 1
Largura = 5	Largura = 7	Largura = 8	Largura = 5
Comprimento = 15	Comprimento = 17	Comprimento = 15	Comprimento = 18
Pacote nº 2	Pacote nº 2	Pacote nº 2	Pacote nº 2
Largura = 7	Largura = 5	Largura = 6	Largura = 5
Comprimento = 17	Comprimento = 12	Comprimento = 10	Comprimento = 15
Pacote nº 3	Pacote nº 3	Pacote nº 3	Pacote nº 3
Largura = 6	Largura = 5	Largura = 6	Largura = 3
Comprimento = 20	Comprimento = 21	Comprimento = 15	Comprimento = 15
Pacote nº 4	Pacote nº 4	Pacote nº 4	Pacote nº 4
Largura = 5	Largura = 2	Largura = 9	Largura = 5
Comprimento = 18	Comprimento = 17	Comprimento = 17	Comprimento = 14
Pacote nº 5	Pacote nº 5	Pacote nº 5	Pacote nº 5
Largura = 5	Largura = 5	Largura = 5	Largura = 5
Comprimento = 5	Comprimento = 4	Comprimento = 7	Comprimento = 3

Quadro 28 – Pacotes testados na Classe 4

Conjunto 1	Conjunto 2	Conjunto 3	Conjunto 4
Pacote nº 1	Pacote nº 1	Pacote nº 1	Pacote nº 1
Largura = 5	Largura = 7	Largura = 8	Largura = 5
Comprimento = 5	Comprimento = 9	Comprimento = 5	Comprimento = 8
Pacote nº 2	Pacote nº 2	Pacote nº 2	Pacote nº 2
Largura = 7	Largura = 5	Largura = 6	Largura = 5
Comprimento = 7	Comprimento = 12	Comprimento = 8	Comprimento = 10
Pacote nº 3	Pacote nº 3	Pacote nº 3	Pacote nº 3
Largura = 6	Largura = 5	Largura = 6	Largura = 3
Comprimento = 8	Comprimento = 8	Comprimento = 5	Comprimento = 5
Pacote nº 4	Pacote nº 4	Pacote nº 4	Pacote nº 4
Largura = 5	Largura = 2	Largura = 9	Largura = 5
Comprimento = 8	Comprimento = 2	Comprimento = 17	Comprimento = 9
Pacote nº 5	Pacote nº 5	Pacote nº 5	Pacote nº 5
Largura = 13	Largura = 15	Largura = 5	Largura = 5
Comprimento = 5	Comprimento = 4	Comprimento = 22	Comprimento = 24