UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

JPACKING: PROGRAMA PARA DISTRIBUIÇÃO OTIMIZADA DE POLÍGONOS EM UM PLANO BIDIMENSIONAL UTILIZANDO ALGORITMOS GENÉTICOS

RODRIGO D'AVILA

BLUMENAU
2017

RODRIGO D'AVILA

JPACKING: PROGRAMA PARA DISTRIBUIÇÃO OTIMIZADA DE POLÍGONOS EM UM PLANO BIDIMENSIONAL UTILIZANDO ALGORITMOS GENÉTICOS

Trabalho de Conclusão de Curso apresentado ao curso de graduação em Ciência da Computação do Centro de Ciências Exatas e Naturais da Universidade Regional de Blumenau como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação.

Prof. Daniel Theisges dos Santos, Mestre - Orientador

BLUMENAU
2017

JPACKING: PROGRAMA PARA DISTRIBUIÇÃO OTIMIZADA DE POLÍGONOS EM UM PLANO BIDIMENSIONAL UTILIZANDO ALGORITMOS GENÉTICOS

Por

RODRIGO D'AVILA

Trabalho de Conclusão de Curso aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II pela banca examinadora formada por:

Presidente:	Prof. Daniel Theisges dos Santos, Mestre – Orientador, FURB
Membro:	Prof. Dalton Solano dos Reis, Mestre – FURB
Membro:	Prof.(a). Andreza Sartori, Doutora – FURB

Blumenau, 05 de Julho de 2017

Dedico este trabalho aos meus familiares e amigos.

AGRADECIMENTOS

Dedico este trabalho à minha família, meus amigos, meu orientador e todos os professores que durante a minha formação me ajudaram a enriquecer meus conhecimentos e me despertar a curiosidade.

“A melhor maneira de prever o futuro é inventa-lo. ”

Alan Kay, 1971

RESUMO

Este trabalho apresenta o desenvolvimento do programa JPacking, capaz de realizar a distribuição otimizada de polígonos em um plano bidimensional utilizando Algoritmos genéticos. Programas como O JPacking são fundamentais para a área de corte de tecido, metal, madeira, e entre outros pois, ajudam a maximizar à utilização de matéria-prima afim de evitar o desperdício. O arranje otimizado trata-se de um problema NP-Difícil não existindo uma resposta determinística em um tempo polinomial. Sendo assim, optou-se pela implementação do JPacking utilizando Algoritmos Genéticos para a geração da ordem de inserção dos polígonos e dos algoritmos de No-Fit-Polygon e Bottom-left fill para arranje dos polígonos. Também foi implementado a possibilidade de fazer importação e a exportação dos resultados através de arquivos SVG. Para avaliar os resultados foram criados testes comparando parâmetros como número de gerações e população, rotações, fator de Crossover e Mutação, comparação de ocupação com os algoritmos de Hill Climbing e Tabu Search, variação da altura da matéria-prima e comportamento do empacotamento. Os resultados mostraram que o JPacking é capaz de executar o empacotamento otimizado e de polígonos irregulares, sendo capaz de alcançar mesmos resultados de algoritmos como Hill Climbing e Tabu Search e certos datasets, e apresentar um comportamento sem erros do empacotamento na maioria dos testes. Foi concluído que o JPacking pode ser otimizado através da execução de testes para calibração dos parâmetros de execução para determinados datasets, e que através da implementação de suas extensões e a aplicação de melhoramentos podem tornar o JPacking utilizável em ambientes de produção.

Palavras-chave: Distribuição otimizada. Empacotamento. Polígonos irregulares. Algoritmos Genéticos. No-Fit-Polygon. NFP. Botton-left fill. Scalable Vector Graphics. SVG.

ABSTRACT

This work presents the development of the JPacking program, capable of performing the optimized distribution of polygons in a two-dimensional plane using Genetic Algorithms. Programs such as JPacking are fundamental for the cutting of fabric, metal, wood, among others. They help to maximize the use of raw materials to avoid waste. The optimized arrangement is an NP-Difficult problem with no deterministic response in a polynomial time. Therefore, the implementation of JPacking uses Genetic Algorithms to generate the order of insertion of the polygons and the algorithms of No-fit polygon and Bottom-left fill to arrange the polygons. The possibility of importing and exporting the results through SVG files was also implemented. To evaluate the results were created tests comparing parameters such as number of generations and population, rotations, Crossover and Mutation factors, occupation comparison with the Hill Climbing and Tabu Search algorithms, variation of the height of the raw material and behavior of the packaging. The results showed that JPacking is capable to perform the optimized packaging of irregular polygons, also being able to achieve the same results of algorithms like Hill Climbing and Tabu Search on certain datasets, and to perform error-free behavior in most of the tests. It was concluded that JPacking can be optimized by running tests for calibration of the execution parameters for certain datasets, and by implementing its extensions and applying improvements can make JPacking usable in production environments.

Key-words: Optimized distribution. 2D Nesting. Packing. Irregular polygons. Genetic Algorithms. No-Fit-Polygon. NFP. Botton-left fill. Scalable Vector Graphics. SVG.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CSV - Comma-Separated Values

NFP – No-Fit Polygon

NFPAB – No-Fit Polygon AB

PMX – Partial Matched Crossover

SVG – Scalable Vector Graphics

XML – eXtensible Markup Language

SUMÁRIO

1 INTRODUÇÃO

Um problema frequente na indústria de manufatura de peças de roupa, tecido, couro, até metal e madeira é o empacotamento de peças para corte. Também conhecido como distribuição otimizada de polígonos em um plano bidimensional, esse problema tem por objetivo agrupar polígonos visando maximizar a utilização do material de maneia a ocupar o menor espaço possível, evitando desperdício de matéria-prima (BRANDT, 2011; HAIMING, 2006; JUNIOR, 2013).

A distribuição otimizada trata-se de uma tarefa de natureza NP-Difícil, sendo assim, acredita-se não haver uma resposta determinística em um tempo polinomial (JUNIOR; PINHEIRO; SARAIVA, 2013). Contudo, este problema pode ser separado em dois passos distintos: o primeiro se refere a estratégia de encaixe, onde é utilizado um algoritmo geométrico para posicionar as peças no plano; o segundo trata a ordem de disposição, onde procura-se uma ordem que permita maximizar a utilização do material (HAIMING; JIONG; XINSHENG, 2006).

Para realizar o encaixe dos polígonos, dado a sua forma irregular, uma estratégia comumente utilizada na literatura é a aplicação do algoritmo de Botton-left fill com No-Fit-Polygon (NFP). Esses algoritmos utilizam a natureza geométrica dos polígonos para realizar a rotações e translações dos polígonos, de forma a evitar a sobreposição e respeitar as dimensões definidas por recipiente (MUNDIM; QUEIROZ, 2012).

Além da realização do encaixe, torna-se necessário a aplicação de um método heurístico para determinar a ordem de disposição das peças (HAIMING; JIONG; XINSHENG, 2006). Existem implementações que aplicam técnicas de heurística local, como Hill Climbing e Tabu Search. Essas técnicas visam buscar soluções através da seleção de uma peça e a manipulação das suas peças vizinhas, a diferença entre as duas é que o Hill Climbing aplica somente a solução corrente e o Tabu Search procura manter um histórico de soluções geradas (BRANDT, 2011).

Além da aplicação destas técnicas para determinar a disposição das peças, técnicas que utilizam Algoritmos Genéticos são amplamente utilizadas (HAIMING; JIONG; XINSHENG, 2006). Diferente das técnicas utilizadas por Brandt (2011), onde a aplicação ocorre em uma peça e seus vizinhos, os operadores do Algoritmo Genético permitem aplicar em todo o conjunto de peças, possibilitando uma melhor convergência, resultando na maximização da ocupação de matéria-prima (HAIMING; JIONG; XINSHENG, 2006).

Por esse motivo, o objetivo desse trabalho é implementar um programa capaz retornar uma solução completa para o problema de disposição de formas irregulares em uma superfície plana bidimensional, utilizando a técnica de Bottom-left fill com NFP. Além de prever a disposição das peças utilizando Algoritmo Genético.

1.1 OBJETIVOS

O objetivo deste trabalho é implementar um software capaz retornar uma solução completa para o problema de disposição de polígonos em uma superfície plana bidimensional.

Os objetivos específicos do trabalho são:

a) implementar o algoritmo de Bottom-left fill com NFP para o encaixe dos polígonos;
b) implementar o Algoritmo Genético com operador de Crossover, Mutação, Seleção e Fitness para gerar a ordem de disposição dos polígonos;
c) integrar os algoritmos (a) e (b) em uma aplicação que tenha como entrada um arquivo SVG contendo os vértices dos polígonos e como saída outro arquivo SVG com uma solução completa.

1.2 ESTRUTURA

Este trabalho está dividido em quatro capítulos. O primeiro capítulo apresenta a introdução do trabalho e os objetivos. O segundo capítulo apresenta a fundamentação teórica sobre corte e empacotamento e Algoritmos Genéticos. No terceiro capítulo é demonstrado o desenvolvimento do sistema com requisitos, especificação, implementação e resultado. Por fim, no quarto capítulo são relatadas a conclusões e também as possíveis extensões.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados aspectos teóricos relacionados ao trabalho. Na Seção 2.1 são apresentados os problemas referentes ao corte e empacotamento. A Seção 2.2 expõe a geometria do problema de corte e empacotamento, seguido das subseções onde é abordado a utilização do NFP com polígonos convexos e não convexos. Por fim, a Seção 2.3 trata da otimização do problema de corte e empacotamento abordando o funcionamento dos algoritmos de Hill Climbin, Tabu Search e mais profundamente o Algoritmo Genético, com sua fundamentação, seguido de seus operadores de Crossover, Mutação, Seleção e Fitness.

2.1 PROBLEMA DE CORTE E EMPACOTAMENTO

Seres humanos são capazes de resolver problemas de empacotamento no seu dia-a-dia, seja para organizar itens em um refrigerador até roupas em uma mala, tudo devido a intuição humana de percepção espacial. Contudo em um ambiente industrial onde ocorrem numerosos problemas de empacotamento, a resolução manual humana para esse problema não é viável, pois o tempo que uma pessoa leva para fazer um arranje considerado ótimo é aproximadamente de vinte minutos (WHITWELL, 2004, p. 9).

Para sua resolução, foi proposta a automatização do processo de empacotamento através de programas de computador, que são responsáveis por maximizar a utilização do material e com isso evitar desperdícios que acabam tendo custo financeiro alto para a indústria de corte (WHITWELL, 2004, p. 10).

Apesar de aparentar ser um problema simples, grande esforço da comunidade acadêmica vem sendo gasto nesse problema, pois se trata de um problema NP-difícil no qual não existe uma solução ótima em um tempo polinomial (WHITWELL, 2004, p. 10).

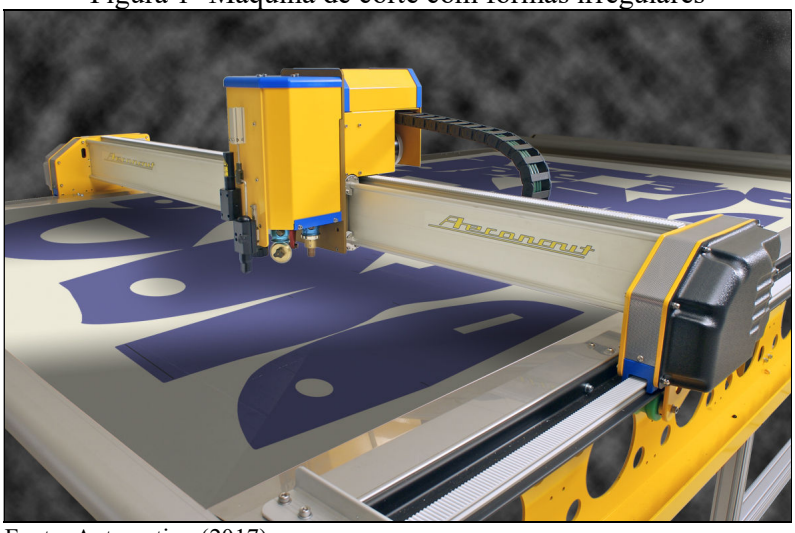
Segundo Whitwell (2004), o termo corte e empacotamento remete a uma infinidade de diferentes problemas dentro da área acadêmica como Bin Packing, Knapsack Problem, Space Allocation, Nesting, etc. todos esses problemas remetem as mesmas características que são (a) dados um número finito de recurso, (b) um número N de itens deve ser atribuído a esse recurso.

Sendo assim esse trabalho foca no problema conhecido como Nesting ou 2d Irregular Bin Packing, no qual se trata de realizar uma distribuição otimizada de polígonos em uma superfície bidimensional.

Muitos segmentos da indústria se beneficiam de soluções para o problema de Nesting, uma delas é a indústria de tecidos no qual são comuns formas irregulares como partes de calças e de camisetas, o arranje dessas formas requer manipulações geométricas complexas

(WHITWELL, 2004, p. 57). Um exemplo de uma máquina de corte com o arranje de peças irregulares pode ser vista na Figura 1.

Figura 1- Máquina de corte com formas irregulares



Fonte: Automation (2017).

2.2 GEOMETRIA DO PROBLEMA DE CORTE E EMPACOTAMENTO

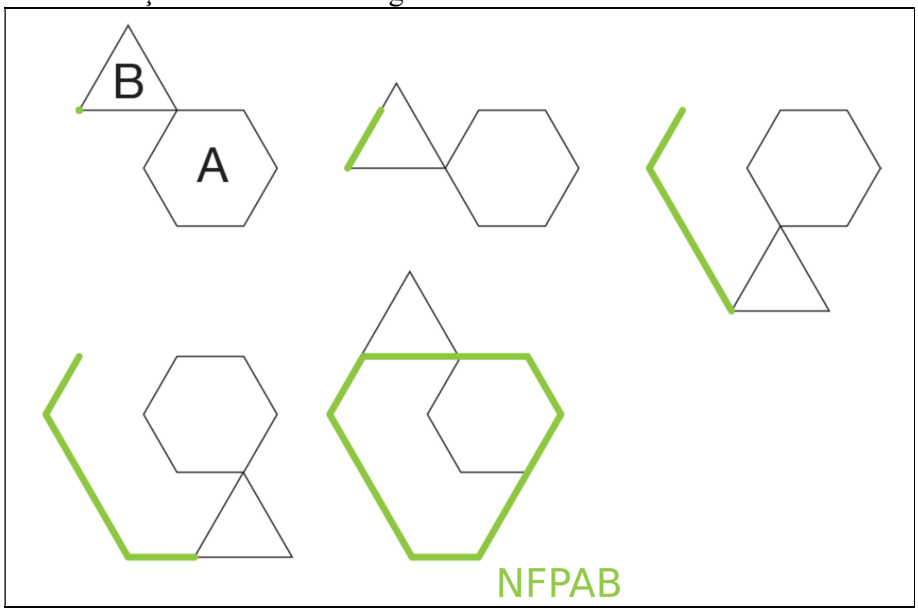
As primeiras abordagens para realização do arranje de peças era através do cálculo da caixa delimitadora mínima (*minimum bounding box*) de um polígono, no qual não era necessária aplicação de cálculos muito complexos, pois na década de 50 cálculos geométricos envolvendo polígonos irregulares eram custosos (WHITWELL, 2004, p. 33).

Contudo, como problemas envolvendo peças irregulares acontecem em vários segmentos da indústria como tecido e metal, e com o aumento da capacidade de processamento dos computadores, rapidamente inúmeras estratégias envolvendo polígonos irregulares foram estudadas no meio acadêmico. Uma das técnicas mais utilizadas é a de No-Fit-Polygon (WHITWELL, 2004, p. 134).

2.2.1 Algoritmo de No-Fit Polygon

O algoritmo de No-Fit Polygon (NFP) é um algoritmo geométrico no qual o objetivo é arranjar dois polígonos A e B de forma que fiquem próximos, mas que não se sobreponham (KENDALL, 2000). Esse arranjo é obtido após a escolha de um ponto de referência em B e da órbita do polígono B em torno dos vértices do polígono A no qual permanece estacionário. O resultado destas rotações gera um polígono que é chamado de No-Fit Polygon AB (NFPAB) ou polígono de não encaixe (BURKE, 2007). Na Figura 2 é possível ver o processo de órbita do polígono B em A e o polígono NFPAB resultante.

Figura 2 - Execução e resultado do algoritmo de NFP com a técnica de Orbital Sliding



Fonte: Qiao (2017).

Segundo Burke (2007), após a obtenção do polígono NFPAB ocorrem três situações para verificar se o polígono A e B se sobrepõem, tocam ou não tocam um no outro:

a) se o ponto de referência de B está dentro do polígono NFPAB então B se sobrepõe em A (Figura 4a);
b) se o ponto de referência de B está nos limites do NFPAB então o polígono B toca o polígono A em alguma parte (Figura 4b);
c) se o ponto de referência está fora do NFPAB então A e B não se sobrepõem e não se tocam (Figura 4c).

Segundo Burke (2007), o algoritmo de NFP pode ser construído de duas formas distintas:

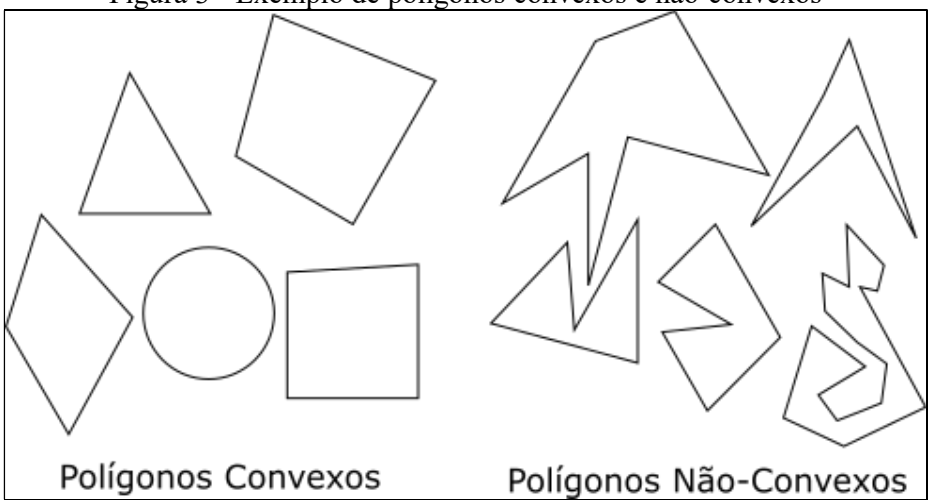
a) para polígonos convexos no qual se trata da forma mais simples do algoritmo;
b) também para polígonos não convexos no qual o algoritmo mais utilizado é o de Orbital Sliding.

2.2.1.1 Descrição da técnica de NFP para polígonos convexos

A forma básica do algoritmo de NFP é aplicada quando os dois polígonos A e B são convexos. Na Figura 3 é possível visualizar exemplos de polígonos convexos e não convexos. Ao utilizar polígonos convexos torna-se possível a obtenção do polígono NFPAB com a

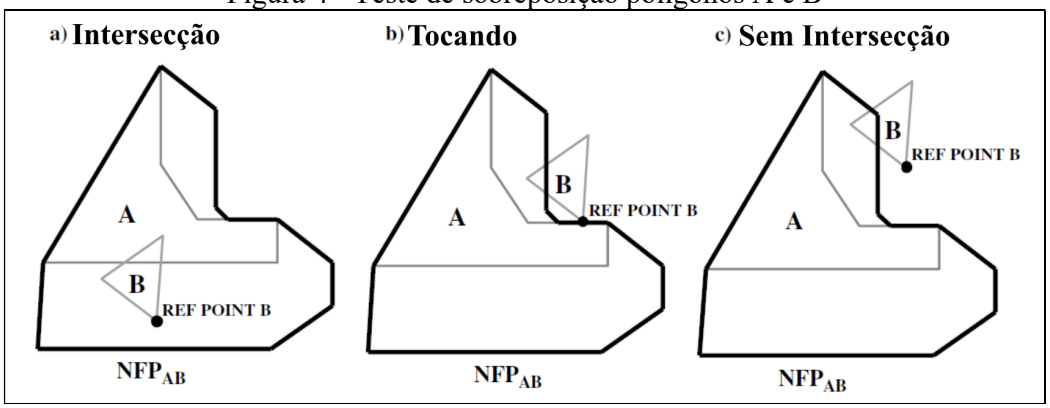
execução de três passos (BURKE, 2007). A Figura 5 apresenta os três passos realizados para a obtenção do polígono NFPAB utilizando a técnica para polígonos convexos:

Figura 3 - Exemplo de polígonos convexos e não-convexos



Fonte: Elaborado pelo autor.

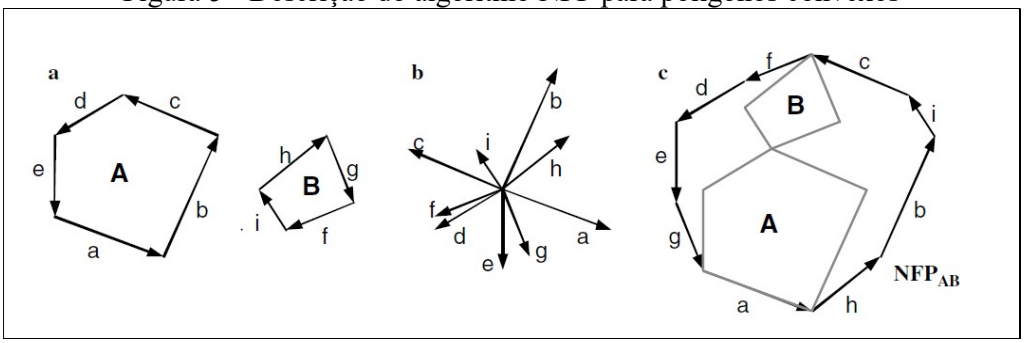
Figura 4 - Teste de sobreposição polígonos A e B



Fonte: Burke (2007)

a) realizar a orientação dos vértices do polígono A no sentido anti-horário e do polígono B em sentido horário (Figura 5a);
b) fazer a translação de todos os vértices de A e B para um único ponto (Figura 5b);
c) concatenar todos os vértices em sentido anti-horário para obtenção o polígono NFPAB (Figura 5c).

Figura 5 - Descrição do algoritmo NFP para polígonos convexos



Fonte: Burke (2007).

2.2.1.2 Descrição da técnica de NFP para polígonos não convexos

Uma das técnicas de NFP utilizada para polígonos não convexos é a técnica de Orbital Sliding (Figura 2), que envolve a utilização de funções trigonométricas para a realização da órbita do polígono B em volta do polígono A (BURKE, 2007). Conforme Burke (2007) essa técnica pode ser dividida em dois estágios:

a) encontrar um vetor de translação e para isso é identificado as arestas que se tocam através de todas as rotações do polígono B em A;
b) procurar posições iniciais nas quais permitem que um polígono toque no outro, mas não ocorra intersecção.

2.2.2 Algoritmo de Bottom-left fill

O Bottom-left fill é um algoritmo utilizado para gerar o empacotamento, ele utiliza o algoritmo de NFP para resolução de sobreposição (BRANDT, 2011). Para o funcionamento do Bottom-left fill são necessários uma ordem de empacotamento e a quantidade de rotações permitidas. (BRANDT 2011).

O Bottom-left fill trabalha posicionando os polígonos sempre da esquerda para a direita, de cima para baixo respeitando a tamanho da forma para empacotamento (BRANDT, 2011). A cada adição de um polígono, é realizado o algoritmo de NFP para cada um dos polígonos já empacotados, sendo que, se houver rotações, é realizado novamente o NFP para cada polígono já empacotado (BRANDT, 2011).

2.3 OTIMIZAÇÃO DO PROBLEMA DE CORTE E EMPACOTAMENTO

Na literatura vários métodos podem ser utilizados para obtenção de soluções ótimas para problemas NP-Difícil, entre eles se destacam métodos meta-heurísticos como Tabu Searh, Hill Climbing (HAIMING; JIONG; XINSHENG, 2006). Contudo, devido à natureza

sequencial do problema de corte e empacotamento um outro método meta-heurístico que se destaca é o Algoritmo Genético (GA) devido a sua capacidade de obter sequencias com boa qualidade (HAIMING; JIONG; XINSHENG, 2006).

2.3.1 Hill Climbing

O algoritmo de Hill Climbing é um dos algoritmos meta-heurístico no qual mantém uma solução corrente e avalia uma nova solução mudando a vizinhança de um polígono (BRANDT, 2011). Se o resultado da mudança da vizinhança for melhor que o resultado corrente, o mesmo é substituído, caso contrário a solução é descartada e o processo continua do início (BRANDT, 2011).

2.3.2 Tabu Search

Outro algoritmo de meta-heurística utilizado para empacotamento é o Tabu Search, no qual ao invés de gerar uma solução com uma única vizinhança, o algoritmo procurar gerar subconjuntos de vizinhanças a cada iteração (BRANDT, 2011).

Além disso, outro mecanismo do Tabu Search é utilizar uma memória de longo período de vizinhanças já geradas, para com isso manter a diversidade de vizinhanças (BRANDT, 2011). A cada subconjunto de novas vizinhanças o Tabu Search substitui a solução corrente pela melhor solução do subconjunto, mesmo que a solução seja pior que a solução corrente (BRANDT, 2011).

2.3.3 Algoritmos Genéticos

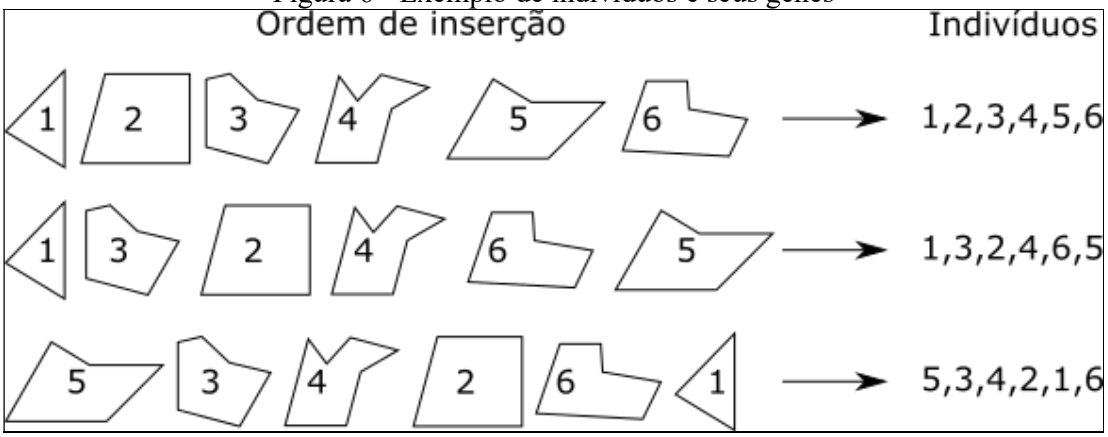
Algoritmos Genéticos fazem parte da computação evolucional que estuda o fenômeno da adaptação para a resolução de problemas da mesma maneira que ocorre na natureza, e de trazer os mecanismos utilizados pela adaptação natural para sistemas computacionais (MITCHELL, 1998). Segundo Mitchell (1998), Algoritmos Genéticos trabalham métodos nos quais se movem populações de cromossomos utilizando-se de uma seleção natural para a geração de uma nova população, essas seleções naturais utilizam operações genéticas como Seleção, Crossover, Mutação e Fitness.

2.3.3.1 População, geração e gene

A população em algoritmos genéticos corresponde a lista de indivíduos que são gerados durante o processo de optimização, o processamento ocorre em intervalos chamados de gerações (WHITWELL, 2004). Os indivíduos das populações são estruturas de dados que

contém uma composição chamada de gene, assim através do processo de Seleção no decorrer de uma nova geração, são selecionados os melhores indivíduos (WHITWELL, 2004). Na Figura 6 é possível visualizar um exemplo de indivíduos que representam uma ordem de inserção no empacotamento.

Figura 6 - Exemplo de indivíduos e seus genes



Fonte: Elaborado pelo autor.

2.3.3.2 Operação de Seleção

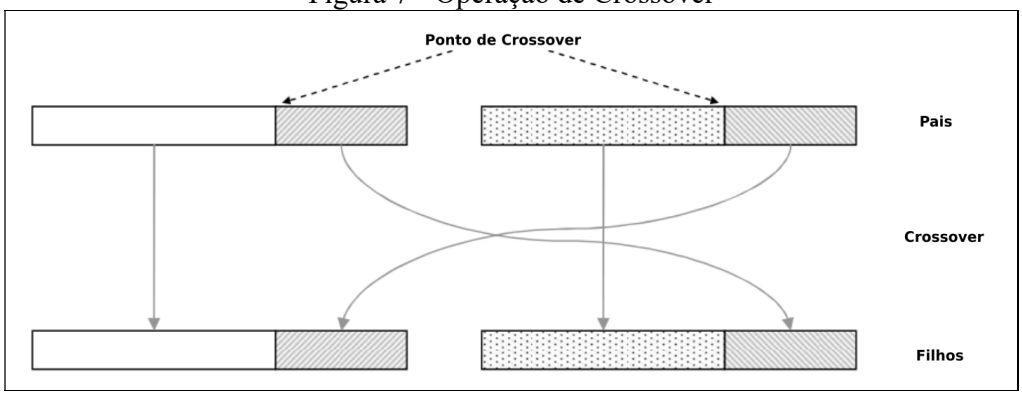
A Seleção é um mecanismo utilizado para garantir que indivíduos com um valor de Fitness elevado tenham chance de gerar novos indivíduos posteriores, reduzindo a chance de indivíduos com Fitness baixo (POPA, 2012). Uma das técnicas de Seleção mais utilizadas é a técnica de Seleção da roleta (SIVANANDAM; DEEPA, 2008). Segundo Sivanandam e Deepa (2008) são implementados os seguintes passos na técnica da roleta:

a) somar em `T` o valor total de Fitness da população;
b) repetir o processo `N` vezes tal que:
   - escolher um número randômico `r` entre 0 e `T`;
   - percorrer toda a população, somando os valores de Fitness em uma variável `s`;
   - continuar até que `s` seja maior ou igual à `r`;
   - escolher o indivíduo que fez com que a soma em `s` fosse igual ou superior à `r`.

2.3.3.3 Operação de Crossover

O operador genético de Crossover é responsável pela geração de novas populações através da população já existente (KENDALL, 2000). Sua execução consiste em pegar indivíduos pais e transferir material genético entre eles para a geração de novos indivíduos filhos, conforme pode ser visualizado na Figura 7.

Figura 7 - Operação de Crossover



Fonte: Affenzeller, Winkler e Beham (2009).

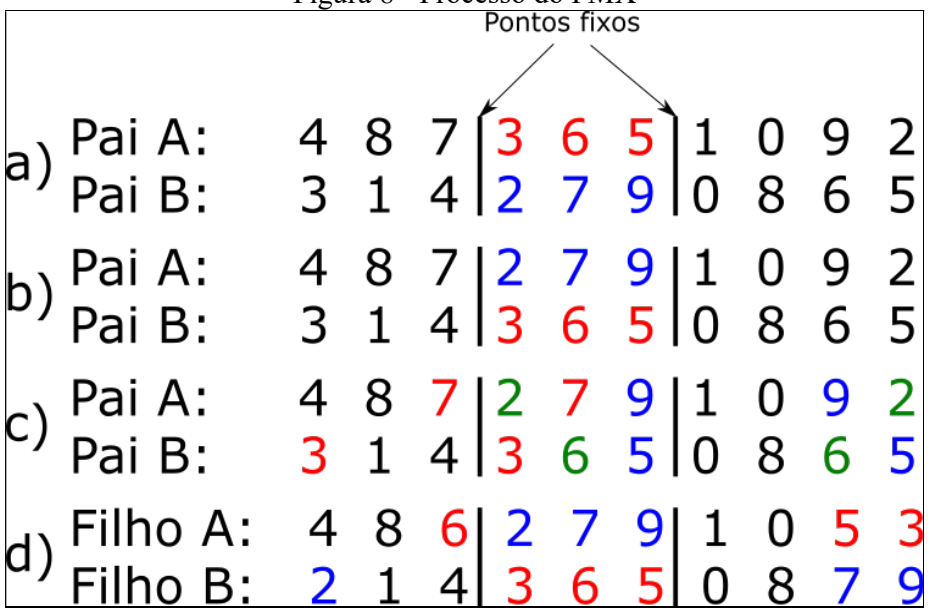
Segundo Affenzeller, Winkler e Beham (2009) a operação de Crossover pode ser dividida em três tipos:

a) Ponto único, no qual um ponto de corte randômico é escolhido, gerando duas cabeças e duas caldas, a troca entre esses cortes produz um novo indivíduo;
b) Múltiplos pontos, onde se faz vários pontos de corte, no qual elementos gerados são montados a partir dos pedaços resultantes;
c) Crossover uniforme, dados dois parentes um indivíduo é criado a partir de uma máscara de maneira randômica.

2.3.3.3.1 Partial Matched Crossover

Alguns problemas têm natureza combinatória, no qual sua representação se deve através da permutação dos seus elementos (GOLDBERG; LINGLE, 1985). Para esses problemas é utilizado um método especial de Crossover chamado de Partial Matched Crossover (PMX), no qual a troca de genes entre os pais ocorre de forma a não gerar genes duplicados ou a falta de um gene em um novo indivíduo, o que acarretaria em um indivíduo inválido (SIVANANDAM; DEEPA, 2008). Segundo Sivanandam e Deepa (2008), o PMX ocorre conforme mostrado na Figura 8.

Figura 8 - Processo do PMX



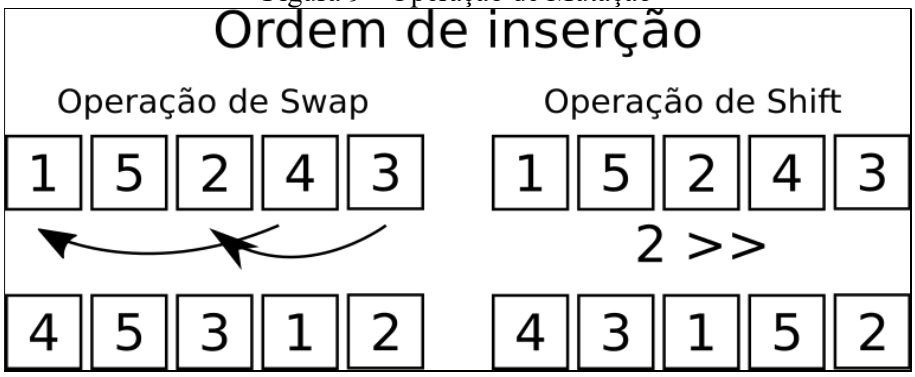
Fonte: Elaborado pelo autor.

a) são selecionados dois pais, e neles são colocados dois pontos fixos randômicos na mesma posição de ambos;
b) os genes entre o ponto fixos são trocados entre cada um dos pais;
c) são identificados os genes que se repetem fora dos pontos fixos;
d) os genes repetidos são substituídos pelos genes pertencentes ao pai antes da troca no passo b.

2.3.3.4 Operação de Mutação

Affenzeller, Winkler e Beham (2009) descrevem a operação de Mutação como pequenos pulos indiretos dentro das possíveis combinações para resolução de um problema. E também que a Mutação deva ocorrer de maneira rara e randômica com uma probabilidade menor que 10%. Além disso segundo Affenzeller, Winkler e Beham (2009) a aplicação da Mutação pode acarretar problemas com a validade de um cromossomo, exemplo, a duplicação de um cromossomo pode ser ilegal. Então, para estes casos Affenzeller, Winkler e Beham (2009) sugerem alternativas que realizam Swap ou Shift dos genes de um indivíduo. As operações de shift e swap podem ser observadas na Figura 9.

Figura 9 - Operação de Mutação

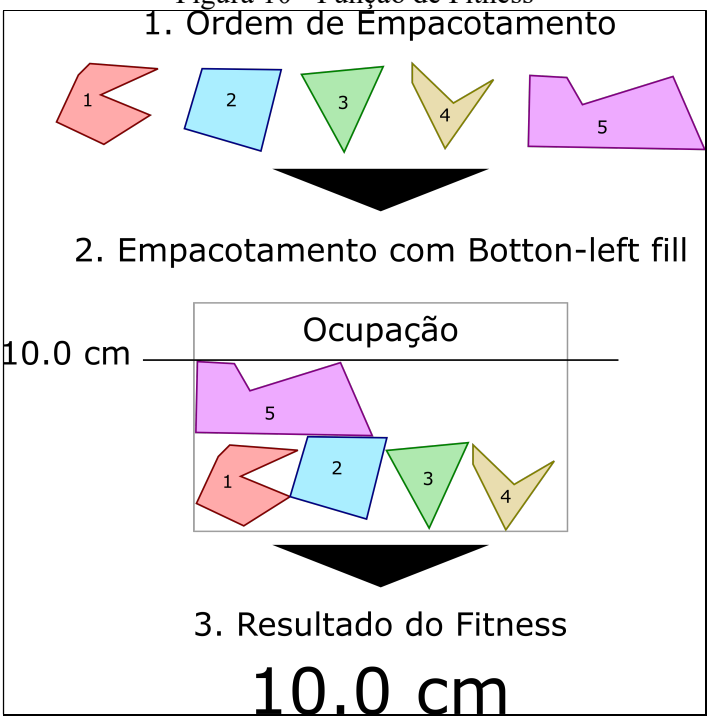


Fonte: Elaborado pelo autor.

2.3.3.5 Função de Fitness

Para Sivanandam e Deepa (2008) em Algoritmos Genéticos o Fitness de um indivíduo é o valor resultante de uma função que leva em consideração as características observáveis de um indivíduo. Durante cada geração os indivíduos de uma população são avaliados e para cada indivíduo é atribuído um valor de Fitness (AFFENZELLER; WINKLER; BEHAM, 2009). A função de Fitness tem o objetivo de garantir que novos indivíduos gerados tenham uma performance relativa à da população anterior (AFFENZELLER; WINKLER; BEHAM, 2009). A abordagem utilizada neste trabalho baseada em Junior, Pinheiro e Saraiva (2013) foi utilizar uma função de Fitness que permitia verificar a ocupação final das peças como pode ser visto na Figura 10 on é feito o empacotamento com uma ocupação de 10 centímetros.

Figura 10 - Função de Fitness



Fonte: Elaborado Pelo Autor

2.3.4 Formato de arquivo SVG

Segundo Dahlström, Dengler e Grasso (2011), o formato de arquivo Scalable Vector Graphics (SVG) define uma sintaxe baseada em XML para desenho vetorial em duas dimensões. O SVG é composto por elementos de desenho como polígonos, círculos, curvas de bézier, segmentos de reta entre outros (DAHLSTRÖM; DENGLER; GRASSO, 2011).

Os segmentos de reta são identificados pela tag XML `path` que contém a propriedade `g` no qual é define comandos de desenho (DAHLSTRÖM; DENGLER; GRASSO, 2011). Cada comando é composto por sua coordenada em duas dimensões relativa à página, seguida do tipo de comando, que pode ser vertical, horizontal ou curva (DAHLSTRÖM; DENGLER; GRASSO, 2011). O funcionamento dos comandos de desenho é análogo a desenhar em uma página, e com a combinação de movimentos de linha e traços de reta é possível desenhar qualquer tipo de polígono em duas dimensões (DAHLSTRÖM; DENGLER; GRASSO, 2011).

2.4 TRABALHOS CORRELATOS

Nesta seção serão apresentados três trabalhos correlatos. Na Seção 2.4.1 será abordado o artigo de Haiming, Jiong e Xinsheng (2006). Seguido da Seção 2.4.2 onde é abordado o

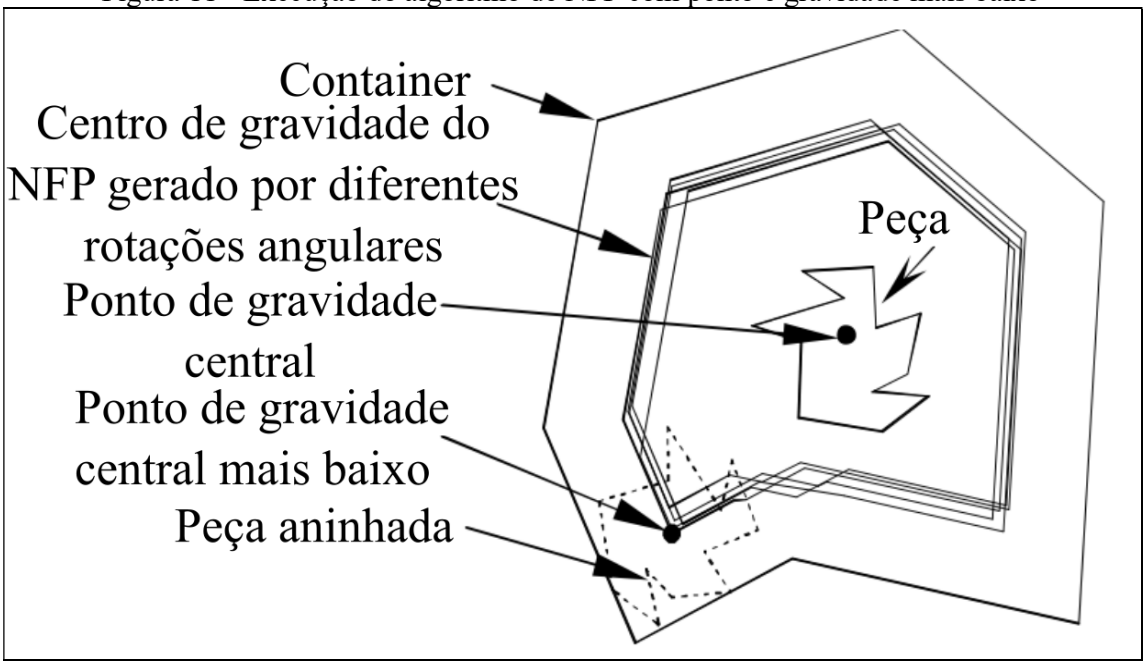
trabalho de Junior, Pinheiro e Saraiva (2013), por fim na seção 2.4.3 é abordado o trabalho de Brandt (2011).

2.4.1 Research and implementation of irregular-shaped nesting problem

O objetivo de Haiming, Jiong e Xinsheng (2006) é apresentar uma nova solução para o problema de disposição de polígonos irregulares em um plano de duas dimensões baseado no algoritmo de NFP. Sua proposta visou substituir o princípio de disposição das peças Bottom-left largamente adotado, pelo princípio de ponto de gravidade mais baixo.

Nesse princípio, o posicionamento da peça é escolhido com base no ponto de menor gravidade. O ponto de menor gravidade é calculado utilizando o método de divisão de triângulos seguido de sucessivas rotações da peça. A execução do algoritmo de NFP baseado em ponto de menor gravidade pode ser visualizada na Figura 11.

Figura 11 - Execução do algoritmo de NFP com ponto e gravidade mais baixo



Fonte: Haiming, Jiong e Xinsheng (2006).

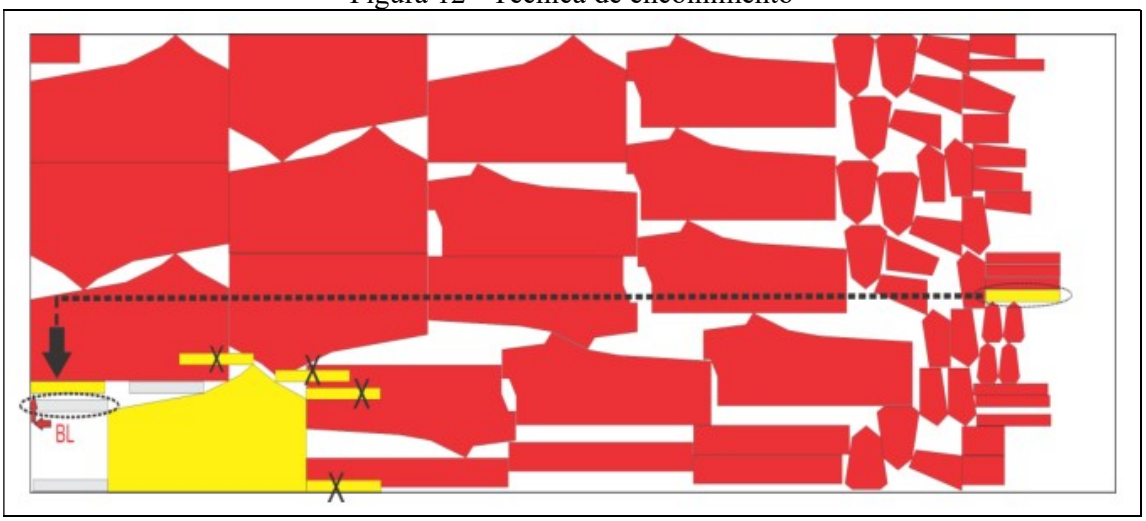
Para determinar a sequência de disposição das peças, o autor utilizou Algoritmo Genético com operador de Crossover. O operador de Crossover remove partes randômicas de dois conjuntos de peças, conhecidos como pais, e realiza a cópia dos elementos selecionados para seus filhos. Os resultados obtidos por Haiming, Jiong e Xinsheng (2006) mostraram uma ocupação de 84.9% de matéria prima em comparação com a ocupação de 77,4%. Os autores apontam sua solução como competitiva em termos de resultado obtidos da utilização da área de matéria-prima e do tempo de execução.

2.4.2 Tackling the irregular strip packing problem by hybridizing Genetic Algorithm and Bottom-left heuristic

No trabalho realizado por Junior, Pinheiro e Saraiva (2013) é proposta a utilização do algoritmo de NFP com disposição Bottom-left para o agrupamento de peças com formatos irregulares e retangulares, o algoritmo NFP foi utilizado em conjunto com um Algoritmo Genético.

Os autores apontaram sua técnica como sendo uma solução híbrida, já que integrou as técnicas de NFP com a disposição Bottom-left utilizando Algoritmo Genético. Além dessas técnicas descritas, os autores utilizam um passo chamado de encolhimento, que visa utilizar o espaço livre entre as peças para adicionar os polígonos retangulares localizados no lado direito mais afastado. A técnica de encolhimento pode ser visualizada na Figura 12.

Figura 12 - Técnica de encolhimento



Fonte: Junior, Pinheiro e Saraiva (2013).

Foram realizados testes com cinco conjuntos de dados obtidos no site da European Working Group in Cutting and Packing[1], o trabalho é comparado com quatro outras técnicas, SAHA, BLF, 2DNest e BS em cinco conjuntos de dados diferentes. Os resultados obtidos da porcentagem de ocupação da matéria-prima por Junior, Pinheiro e Saraiva (2013) podem ser vistos na Tabela 1.

[1] Disponível em: http://paginas.fe.up.pt/~esicup/

Tabela 1 - Resultados obtidos por Junior, Pinheiro e Saraiva

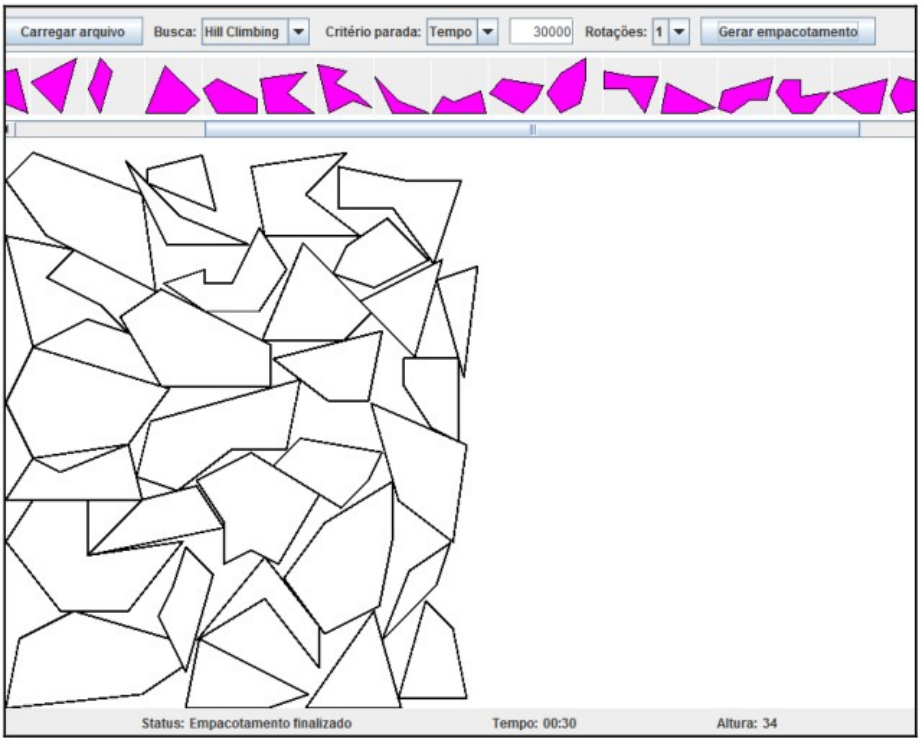
Dataset	SAHA	BLF	2DNest	BS	HM
DIGHE1	100.00	77.40	99.86	100.00	100.00
DIGHE2	100.00	79.40	99.95	100.00	100.00
JAKOBS1	78.89	82.60	89.07	85.96	80.22
JAKOBS2	77.28	74.80	80.41	80.40	73.92
TROUSERS	89.96	88.50	89.84	90.38	88.74

Fonte: Junior, Pinheiro e Saraiva.

2.4.3 Distribuição otimizada de polígonos em um plano bidimensional

Em Brandt (2011) é proposto o desenvolvimento de uma ferramenta capaz de posicionar polígonos irregulares em duas dimensões em um tempo aceitável, com a possibilidade de visualizar todo o processo de posicionamento. Da mesma forma que Liu e He (2006) e Junior, Pinheiro e Saraiva (2013) é utilizado o algoritmo de NFP, no entanto não é utilizado Algoritmo Genético. São utilizadas duas técnicas de heurística local, Hill Climbing e Tabu Search. A interface da ferramenta desenvolvida por Brandt (2011) pode ser vista na Figura 13.

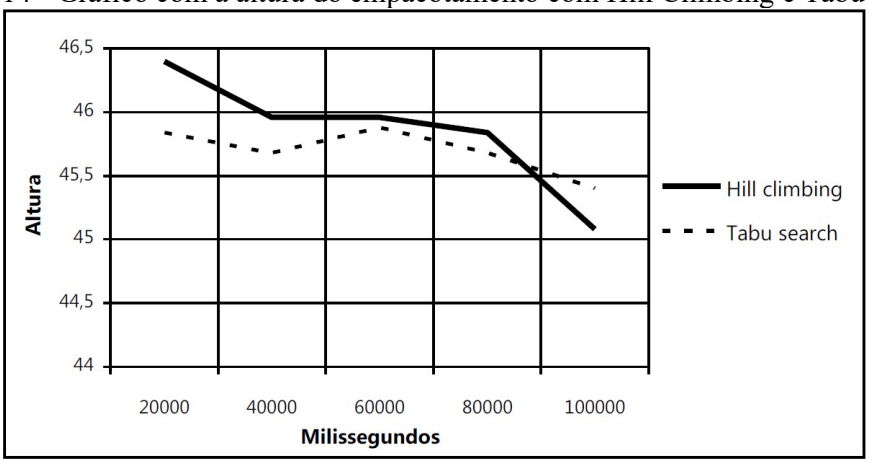
Figura 13 - Interface ferramenta desenvolvida por Brandt (2011)



Fonte: Brandt (2011).

O trabalho de Brandt (2011) compara somente a técnica de Hill Climbing com a de Tabu Search não havendo comparação com nenhum autor externo. De acordo com os resultados apresentados, a técnica de Tabu Search se mostrou superior ao Hill Climbing, alcançando uma altura menor no mesmo para o mesmo tempo de execução na maioria dos testes realizados, contudo, com um tempo de execução de aproximadamente 80000 milissegundos o Hill Climbing se mostra superior, como pode ser visualizado no gráfico da Figura 14.

Figura 14 - Gráfico com a altura do empacotamento com Hill Climbing e Tabu Search



Fonte: Brandt (2011).

3 DESENVOLVIMENTO

As seções a seguir descrevem os requisitos, a especificação, a implementação e a operacionalidade do trabalho, abordando sucintamente as ferramentas e etapas utilizadas no desenvolvimento do projeto. Ao fim, são mostrados os resultados obtidos com este trabalho.

3.1 REQUISITOS

A lista a seguir descreve os requisitos a serem respeitados durante o desenvolvimento do trabalho.

a) disponibilizar parâmetros de linha de comando para interação com o programa (Requisito Funcional – RF);
b) realizar a disposição dos polígonos obedecendo à altura do material (RF);
c) informações sobre altura utilizada e tempo decorrido para efetuar o encaixe dos polígonos (RF);
d) não sobrepor polígonos e não permitir que polígonos tenham partes fora da altura informada pelo usuário (a não ser em casos onde a biblioteca de encaixe falhe) (RF);
e) executar a técnica de NFP com a disposição Bottom-left fill utilizando Algoritmos Genéticos com operador de Crossover, Mutação, Seleção e Fitness (RF);
f) exportar no formato SVG para permitir a alteração manual do resultado obtido através de editores vetoriais ou programas de terceiros (RF);
g) permitir parar por tempo de execução através da parametrização passada pelo usuário (RF),
h) obter um resultado completo em um tempo menor que 20 minutos para uma quantidade menor que 20 peças (Requisito Não Funcional – RNF);
i) ser desenvolvido utilizando a linguagem de programação Java e Python (RNF);
j) ser desenvolvido utilizando o ambiente Eclipse para a programação (RNF).

3.2 ESPECIFICAÇÕES

Para ilustração das especificações do programa JPacking produzido neste trabalho, foi utilizada a ferramenta Inkscape[2] para a criação de diagramas de atividade e casos de uso, além do programa class-visualizer[3] para criação do diagrama de classes.

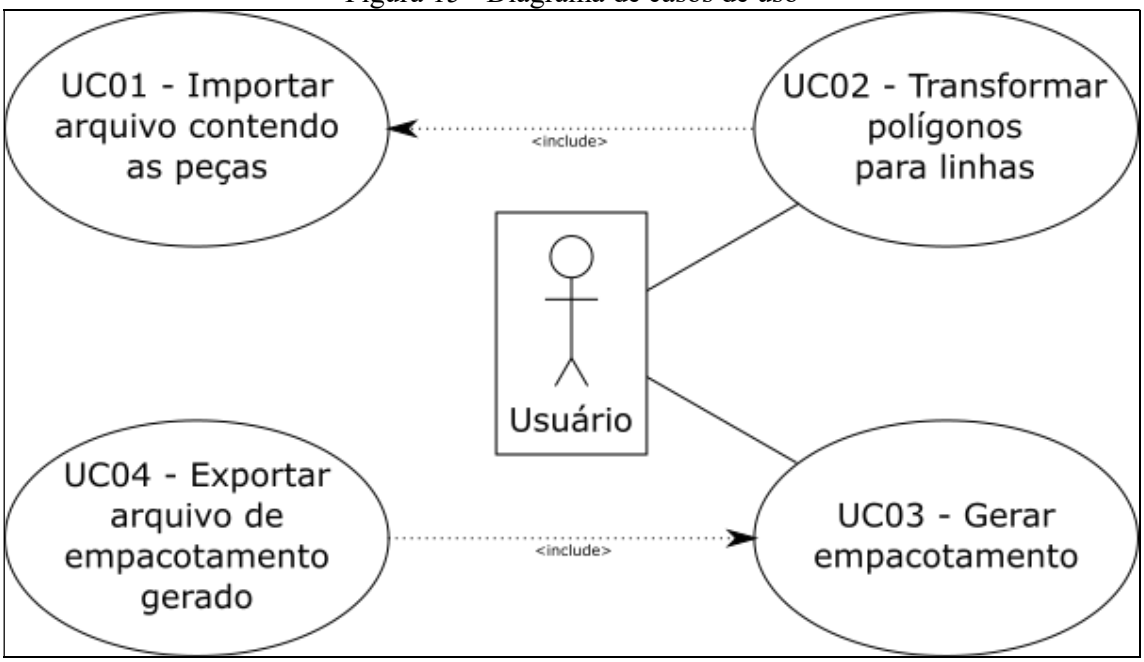
---

[2] Inkscape pode ser encontrado em: https://inkscape.org/en/
[3] Class-visualizer pode ser encontrado em: https://www.class-visualizer.net/

3.2.1 Diagrama de casos de uso

Nesta seção são abordadas as funções exercidas pelo sistema, sendo o ator somente o `Usuário`. Na Figura 15 é possível visualizar os casos de uso.

Figura 15 - Diagrama de casos de uso



Fonte: Elaborado pelo autor.

Os casos de uso foram divididos de acordo com a interação que o usuário realiza para o funcionamento do JPacking, desde a transformação dos polígonos para segmentos de reta, até a geração do empacotamento.

O caso de uso `UC01 – Importar arquivos contendo as peças` é a ação realizada para a importação das peças (polígonos) que serão passadas para o empacotamento. É possível a criação do arquivo contendo as peças com o auxílio de um programa de edição vetorial como o Inkscape.

No editor vetorial as peças podem ser criadas utilizando as ferramentas de polígonos ou a caneta de desenho através da opção de segmentos de linha. É necessário que esteja contido no arquivo todas as peças que serão utilizadas no empacotamento. Se existe a necessidade de mais de uma peça de um determinado tipo, a mesma deve ser copiada dentro do arquivo SVG manualmente, ou utilizando o script de multiplicação de polígonos disponibilizado pelo JPacking.

Já no caso de uso `UC02- Transformar polígonos para linhas` é a ação realizada pelo `Usuário` para transformar todos os elementos `<path>` contendo comandos de verticais horizontais e curvas de bézier para segmentos de reta.

Essa transformação é feita com o auxílio de um script Python, que carrega o arquivo SVG e retorna para a saída padrão outro SVG contendo somente segmentos de reta. O processo de transformação pode ser pulado, se no processo de criação das peças for utilizado a ferramenta de caneta de desenho através de segmentos de reta, ou realizar a conversão dentro do editor vetorial para segmentos de reta.

Seguindo adiante, o caso de uso `UC03 – Gerar empacotamento` é a ação no qual o programa JPacking lê o arquivo de entrada contendo as peças e realiza o empacotamento das peças utilizando Algoritmo Genético.

Neste passo o `Usuário` pode interagir com o JPacking passando parâmetros de linha de comando no qual é possível:

a) selecionar o arquivo de entrada;
b) especificar a altura que pode ser ocupada pela forma de matéria prima, visto que a largura (ocupação) é infinita e não existe restrição de forma na largura;
c) atribuir a quantidade de rotação que a peça pode fazer;
d) quantidade de gerações que o Algoritmo Genético irá executar;
e) tamanho da população do Algoritmo Genético;
f) tempo máximo de execução para o empacotamento;
g) especificar parâmetros avançados como:
    - fator de Crossover,
    - fator de Mutação,
    - idade máxima de um indivíduo pai,
    - quantidade máxima de gerações no qual um indivíduo permanece como melhor.

Por fim, o caso de uso `UC04 – Exportar arquivo de empacotamento gerado` descreve a ação no qual é exportado o arquivo SVG contendo o resultado do empacotamento. O arquivo de saída pode ser editado com qualquer editor vetorial que suporte SVG da mesma forma que o arquivo de entrada.

3.2.2 Diagrama de atividade

Na Figura 16 é possível visualizar o diagrama de atividades, no qual são descritos os passos realizados pelo programa JPacking para realizar o empacotamento dos polígonos em uma superfície bidimensional.

O passo 1 realiza o parse do arquivo SVG, nele são extraídas somente as tags `<path>`. No passo 2 é realizada a extração dos polígonos no qual é feita a leitura dos pontos dos segmentos de reta de todas as tags `<path>`.

Adiante, o passo 3 realiza a translação dos vértices de todos os polígonos para a origem, isso se deve, pois, os segmentos de reta guardam a posição com relação ao desenho e o algoritmo de NFP necessita que todos os polígonos tenham sua posição a partir da origem.

Com todos os passos anteriores realizados, é possível executar o passo 4 que consiste na execução do Algoritmo Genético. O Algoritmo genético inicia com passo 4.1 onde é feita a leitura dos parâmetros da quantidade de gerações, tamanho da população e tempo máximo de execução, além de uma lista contendo o `id` de cada polígono. Com esses parâmetros é criado a primeira população realizando o Partial Matched Crossover entre os indivíduos, juntamente com a Mutação através do Swap Mutation.

Todos os indivíduos são então submetidos ao passo 4.2, onde a Seleção realiza a avaliação de cada indivíduo a partir de seu valor de Fitness. A função de Fitness consiste na execução do algoritmo de Bottom-left fill com NFP seguindo a ordem de empacotamento dos polígonos retornada pelo Algoritmo Genético.

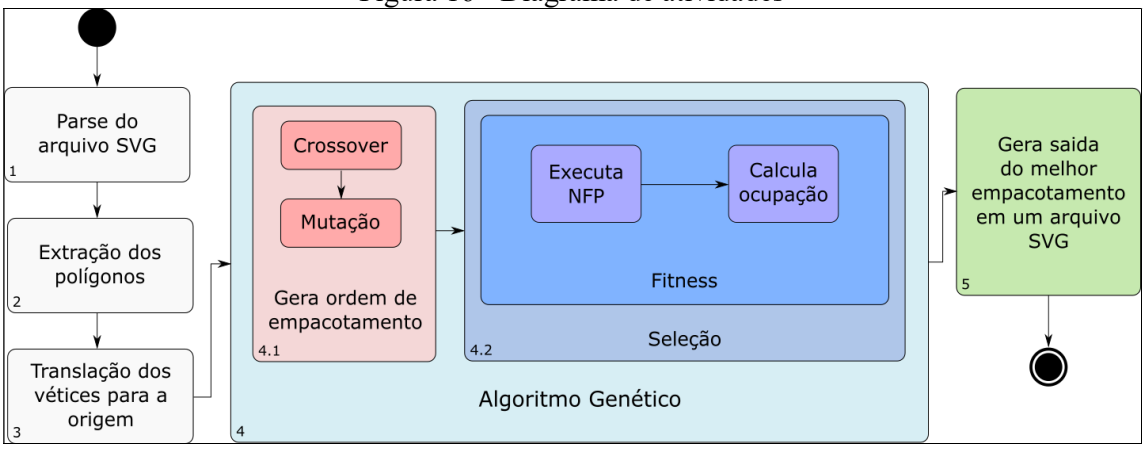
Após a execução do algoritmo de Bottom-left fill, é executado o cálculo de ocupação, que corresponde ao valor do `X` do último vértice à direita. Quanto menor o valor melhor a ocupação, ou seja, a Seleção busca indivíduos que tenham o Fitness menor, que consequentemente realizaram melhor ocupação de matéria prima.

A finalização da execução do Algoritmo Genético se dá de três maneiras:

a) foi realizado a execução até a última geração;
b) se não houve um indivíduo com um Fitness melhor no decorrer 30% das gerações;
c) até a geração na qual o tempo máximo de execução foi atingido.

Por fim, no passo 5 é gerado o arquivo de saída contendo o empacotamento do melhor indivíduo retornado pelo Algoritmo Genético. O arquivo de saída contém todos os polígonos desenhados utilizando segmentos de reta e coloridos para uma melhor visualização.

Figura 16 - Diagrama de atividades



Fonte: Elaborado pelo autor.

3.2.3 Diagrama de classes

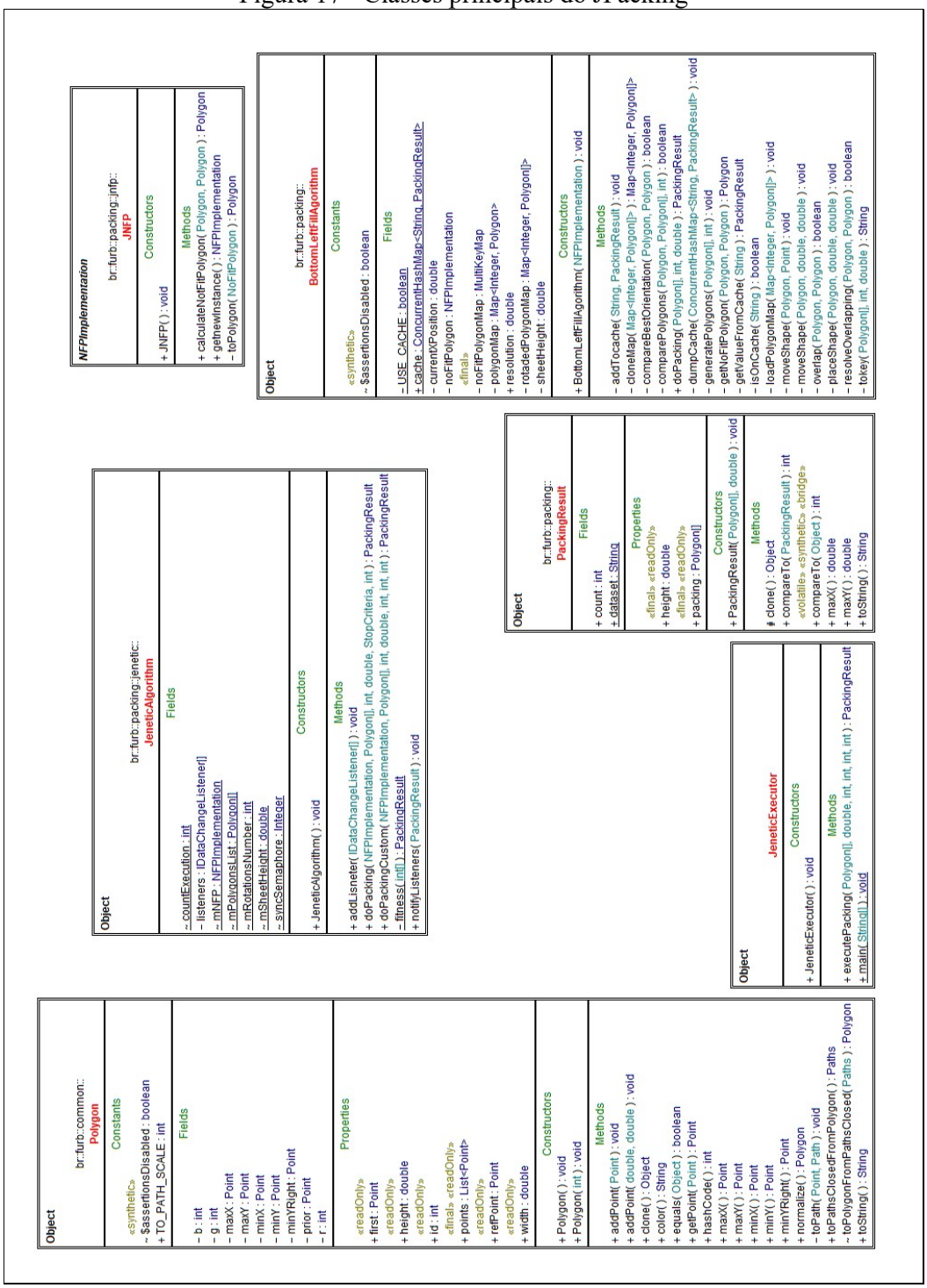
Nessa seção é apresentado o diagrama de classes do programa JPacking. Na Figura 17 é possível visualizar a implementação das classes principais do JPacking. Já na Figura 29, Figura 30 e Figura 31 localizadas nos anexos F e G são mostradas as ligações das classes principais.

A implementação do JPacking é baseada nas classes reaproveitáveis do trabalho realizado por Brandt (2011). São reaproveitados classes básicas como `Polygon`, `Point`, `MathHelper`, `Transform`, `TouchingEdge`, `IntersectionCalculator`, `FeasibleTranslator`, `PackingResult`, `HeightComparatorm`, `WidthComparator` e `BottonLeftFillAlgorithm`.

A classes principais do programa JPacking são `JeneticsExecutor` e `JeneticsAlgorithm`, responsáveis pela leitura dos parâmetros e de dar início ao Algoritmo Genético descrito em 3.2.2.

Por fim, foi implementado a classe `JNFP` para execução do algoritmo de NFP, além das classes `SVGReader` e `SVGWriter` para leitura e escrita dos arquivos SVG.

Figura 17 - Classes principais do JPacking



Fonte: Elaborado pelo autor.

3.3 IMPLEMENTAÇÃO

A seguir são mostradas as técnicas e ferramentas utilizadas na implementação.

3.3.1 Técnicas e ferramentas utilizadas

A implementação do JPacking utilizou as seguintes ferramentas e bibliotecas:

a) Eclipse Luna na versão SR2;
b) linguagem de programação Java 8;
c) linguagem de programação Python 3.5;
d) classes básicas e funções matemáticas implementadas por Brandt (2011);
e) algoritmo de Bottom-left fill implementado por Brandt (2011);
f) biblioteca de No-fit Polygon JNFP (WAUTERS, 2016);
g) biblioteca de Algoritmo Genético Jenetics (WILHELMSTÖTTER, 2017);
h) biblioteca de leitura de arquivos SVG svg.path (REGEBRO, 2017).

O desenvolvimento do JPacking foi dividido nas seguintes etapas:

a) *fork* do código fonte do trabalho de Brandt (2011) para a adição do Algoritmo Genético;
b) substituição do algoritmo de NFP existente para utilizar a biblioteca JNFP;
c) implementação do Algoritmo Genético utilizando a biblioteca Jenetics;
d) criação das funções de leitura e escrita do arquivo SVG;
e) adição da leitura dos parâmetros via de linha de comando;
f) desenvolvimento do script em Python para transformar elementos de um arquivo SVG em segmentos de reta;
g) desenvolvimento do script em Python para multiplicação dos polígonos dentro de um arquivo SVG.

3.3.1.1 Fork do código fonte existente

O programa JPacking criou uma bifurcação (*fork*) do programa implementado por Brandt (2011). Foram realizadas a implementação de novas interfaces para permitir o acoplamento do Algoritmo Genético ao código fonte reaproveitado. O *fork* do código fonte teve o intuito de reaproveitar, evitando a reimplementação de classes básicas, funções matemáticas e do algoritmo de Bottom-left fill.

3.3.1.2 Algoritmo de NFP

O algoritmo de NFP utilizado pelo JPacking foi providenciado da biblioteca JNFP, ela é uma biblioteca Java que realiza o algoritmo de NFP para polígono convexos e não convexos utilizando técnica de órbita.

O trabalho de Brandt (2011) também continha uma implementação do algoritmo de NFP, contudo, após testes realizados utilizando a implementação de Brandt (2011) e o JNFP foi concluído que a biblioteca JNFP foi mais performática, por este motivo a mesma foi escolhida como implementação do algoritmo de NFP.

3.3.1.3 Algoritmo Genético

A implementação do Algoritmo Genético foi feita utilizando a biblioteca Jenetics, que é uma biblioteca de Algoritmos Genéticos para a linguagem Java. A biblioteca Jenetics permite criar motores genéticos que obedecem a um determinado formato de manipulação de genes de indivíduos. A implementação do Algoritmo Genético pode ser visualizada no Quadro 1.

Quadro 1 – Pseudocódigo da implementação do Algoritmo Genético

```
1 motor = novoMotor (
            FUNÇÃO-DE-FITNESS,
            Permutação(QUANTIDADE-POLIGONOS));
2 motor.otimização(MÍNIMO)
            .idadeMáxima(20%)
            .tamanhoPopulação(TAMANHO-DA-POPULAÇÃO);
3 motor.alteradores(
            SwapMutator(0.2),
            PartiallyMatchedCrossover(0.35));
4 melhor = motor.executa()
            .limita(porMelhorEstável(30%))
            .limita(porTempo(TEMPO-EM-MILISSEGUNDOS))
            .limita(porQuantidadeDeGerações(QUANTIDADE-DE-GERAÇÕES))
            .coletaOMelhor();
5 return melhor;
```

Fonte: Elaborado pelo autor.

No Quadro 1, a linha 1 mostra a criação de um novo motor, que recebe a implementação da função de Fitness, além do formato de manipulação dos genes de novos indivíduos. O formato de manipulação escolhido foi através da permutação, no qual permite criar indivíduos novos sem a repetição de genes, evitando a geração de indivíduos inválidos.

Adiante na linha 2, é realizado o processo de otimização, no qual é atribuída uma idade máxima de 20% das gerações para um indivíduo permanecer entre os melhores, permitindo assim que mais indivíduos novos possam gerar filhos após esse tempo. Além disso é limitado o número de indivíduos para o tamanho escolhido da população.

Continuando a montagem do motor, temos na linha 3, a aplicação dos alteradores, no qual foi usado o algoritmo de SwapMutator com um valor fator padrão de 0.2 (alterável via linha de comando) para realizar a Mutação do Algoritmo Genético. E também a escolha do algoritmo de PartiallyMatchedCrossover com um fator padrão de 0.35 (alterável via linha de comando) para realizar a operação de Crossover do Algoritmo Genético.

Em seguida, na linha 4 é feita a execução do motor, limitando a execução pelo melhor indivíduo após 30% das gerações. Além disso existe a limitação por tempo de execução em milissegundos e pela quantidade máxima de gerações escolhida pelo usuário. Por fim, na linha 5 é retornado o melhor indivíduo após a finalização da execução do Algoritmo Genético.

3.3.1.3.1 Função de Fitness

A função de Fitness tem a tarefa de realizar o empacotamento e verificar a ocupação. No Quadro 2 é possível visualizar a implementação da função de Fitness.

Quadro 2 - Pseudocódigo da implementação da função de Fitness

```
1 Função Fitness(int [] PERMUTAÇÃO, LISTA-POLIGONOS) {
2       ordemDeInserção = new Polígono [QUANTIDADE-DE-POLIGONOS]
3       for (int i=0;i<QUANTIDADE-DE-POLIGONOS;i++) {
4             ordemDeInserção [i] = LISTA-POLIGONOS [PERMUTAÇÃO [i]];
5       }
6              ocupação =  bottomLeftFill.usarImplementaçãoNFP(JNFP)

                        .executaEmpacotamento(
                              ordemDeInserção,
                              NUMERO-DE-ROTAÇÕES,
                              ALTURA-DA-FORMA);
7               return ocupação;
8 }
```

Fonte: Elaborado pelo autor.

No Quadro 2 na linha 1 são passados como parâmetros a permutação gerada pelo Algoritmo Genético que serve para criar a ordem de inserção dos polígonos na forma de uma lista utilizada para realizar o empacotamento.

Adiante na linha 2 é criado o objeto no qual é armazenado a ordem de inserção para ser utilizado pelo algoritmo de Bottom-left fill. Nas linhas 3-5 são atribuídos os polígonos ao objeto de ordem de inserção seguindo a permutação gerada pelo Algoritmo Genético.

Na linha 6 é atribuído o valor da ocupação, através da execução do algoritmo de Bottom-left fill utilizando a implementação do NFP pela biblioteca JNFP. A execução do empacotamento considera a ordem de inserção, o número de rotações permitidas e a altura máxima da forma. Por fim, na linha 7 é retornado o valor da ocupação.

3.3.1.4 Leitura e escrita do arquivo SVG

Tanto a entrada como a saída do JPacking são arquivos do tipo SVG, que podem ser manipulados em qualquer editor vetorial com suporte a SVG. A função de leitura pode ser observada no Quadro 3 e a função de escrita no Quadro 4.

Quadro 3 - Pseudocódigo da Leitura do arquivo SVG

```
1  Função LerSVG(CAMINHO) {
2        ArrayList poligonos = new ArrayList ();
3        leitorXML.Ler(CAMINHO)
                              .ParaTodasAsTAGS('svg>path')
                              .lerAtributo('d')
                              .limparComandos('z ,Z, m, M, l, L')
                              .transformarParaPoligonos()
                              .deslocarParaOrigem()
                              .adicionaParaLista(poligonos);
4        return poligonos;
5  }
```

Fonte: Elaborado pelo autor.

No Quadro 3 na linha 1, é passado o caminho da localização do arquivo SVG de entrada. Adiante na linha 2 é criado uma lista para armazenar os polígonos extraídos do arquivo SVG.

Em seguida na linha 3 é criado um objeto para ler a estrutura XML do arquivo SVG passando o caminho. Após isso são lidas todas as tags `<path>` dentro do SVG e os atributos `d` de todas as tags `<path>`. Na próxima etapa são limpos da propriedade `d` os comandos de fechamento de linha, movimentação e linha, sobrando somente as coordenadas `X` e `Y` de cada segmento de reta.

Ainda na linha 3, todas as coordenadas são passadas como vértices para a criação de um polígono. O polígono em seguida é deslocado para a origem e adicionado dentro da lista de polígonos. Por fim, na linha 4 é retornada uma lista com todos os polígonos extraídos de cada tag `<path>`.

Quadro 4 - Pseudocódigo da escrita do arquivo SVG

```
1  Função EscreverSVG(poligonos) {
2        escritorXML.paraTodosAspoligonos(poligonos)
                              .transformarVéticesParaSegmentosDeReta()
                              .escreverNaSaidaPadrão();
3  }
```

Fonte: Elaborado pelo autor.

No Quadro 4 linha 1 é passado a lista de polígonos com a melhor ocupação, em seguida na linha 2 é criado um objeto escritor de XML que para cada polígono. Em seguida é realizado o processo de transformação de todos os vértices de cada polígono para segmentos de reta.

A transformação envolve criar uma tag `<path>`, com um atributo `d` e criar um comando de desenho de segmentos de reta e fechar este segmento. Por fim, o arquivo SVG é escrito na saída padrão de texto.

3.3.1.5 Linha de comando

O programa JPacking pode ser utilizado tanto como uma biblioteca Java como uma aplicação para terminal de texto. Por este motivo foram implementadas as entradas via comando de texto dos parâmetros que constituem as variáveis do Algoritmo Genético.

No Quadro 5 na linha 2 até a linha 7 são lidos os parâmetros de entrada, que determinam o comportamento do JPacking em modo de execução. O primeiro parâmetro é o arquivo SVG de entrada, seguido da altura máxima, a quantidade de rotações, quantidade máxima de gerações, o tamanho da população e o tempo máximo de execução.

Quadro 5 - Pseudocódigo da leitura dos parâmetros de entrada

```
1 public static void main(args) {
2               SVG-ENTRADA = args[0];
3               ALTURA      = args[1];
4               ROTAÇÕES    = args[2];
5               GERAÇÕES    = args[3];
6               POPULAÇÃO   = args[4];
7               TEMPO       = args[5];
8 ...
```

Fonte: Elaborado pelo autor.

Na Tabela 2 é possível visualizar o valor mínimo e máximo de todos os parâmetros de entrada do JPacking.

Tabela 2 - Quantidade mínima e máxima dos parâmetros

Parâmetro	Min	Max
SVG-ENTRADA	1 peça	43330 peças
ALTURA	0	2^32
ROTAÇÕES	0	359
GERAÇÕES	1	2^32
POPULAÇÃO	1	2^32
TEMPO	0	2^64

Fonte: Elaborado pelo autor.

Quadro 6 - Pseudocódigo leitura dos parâmetros avançados

```
1 Função LerPropriedadesAvançadas(prop) {
2               IDADE-MÁXIMA       = prop [0];
3               FATOR-CROSSOVER    = prop [1];
4               FATOR-MUTAÇÃO      = prop [2];
5               MELHOR-ESTÁVEL     = prop [3];
6 ...
```

Fonte: Elaborado pelo autor.

No Quadro 6 é possível visualizar os parâmetros avançados do JPacking. Todos os parâmetros avançados têm seu valor padrão, não necessitando sua atribuição a cada execução. Na linha 2 até a linha 6 são lidos os parâmetros avançados que são passados para o JPacking via propriedades da Máquina Virtual do Java.

O primeiro parâmetro avançado é a idade máxima no qual um indivíduo permanece como pai, seguido do fator de Crossover, fator de Mutação, quantidade de gerações e indivíduo estável.

Na Tabela 3 é possível visualizar o valor padrão, valor mínimo e máximo que é possível passar para os parâmetros avançados.

Tabela 3 - Valores padrão, mínimo e máximo das propriedades avançadas

Parâmetro	Valor padrão	Mínimo	Máximo
IDADE-MÁXIMA	20%	1%	100%
FATOR-CROSSOVER	0.35	0.0	1.0
FATOR-MUTAÇÃO	0.2	0.0	1.0
MELHOR-ESTÁVEL	30%	1%	100%

Fonte: Elaborado pelo autor.

3.3.1.6 Script de conversão para segmentos de reta

O programa JPacking aceita arquivos SVG, desde de que todos os comandos de desenho das tags `<path>` sejam compostos por segmentos de reta. Contudo, para os arquivos SVG que tenham tags `<path>` com outros tipos de comandos como curvas de bézier, verticais e horizontais, é necessário convertê-los para segmentos de reta. Para isso, foi implementado um script em Python utilizando a biblioteca svg.path que realiza a conversão para segmentos de reta para todas as tags `<path>` dentro de um arquivo SVG. Na Figura 18 é possível visualizar o resultado da conversão de um polígono contendo curvas para um polígono contendo segmentos de reta.

No Quadro 7 na linha 1 é iniciado o script com a leitura do caminho onde se encontra o arquivo SVG, seguido da linha 2 onde é atribuído o fator de pontos, que implica na quantidade de segmentos que uma curva pode ter após a navegação pelo seu perímetro.

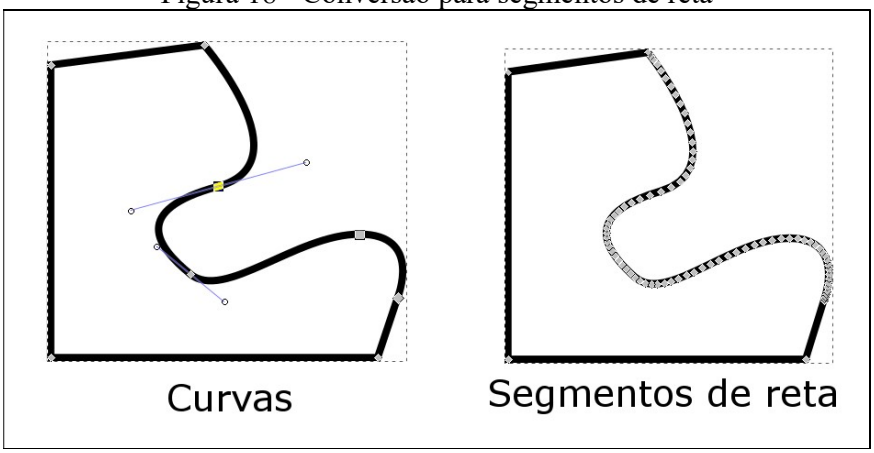
Adiante na linha 3 para cada tag `<path>` é feita a navegação pelo perímetro do polígono interpretando cada comando de desenho como segmento de reta. Por fim, na linha 4 é escrito cada segmento de reta para a saída padrão.

Quadro 7 – Pseudocódigo do script conversão para segmentos de reta

```
1 | CAMINHO                = args[0]
2 | FATOR-PONTOS = args[1]
3 | segmentos = leitorXML.Ler(CAMINHO)
  |       .ParaTodasAsTAGS('svg>path')
  |       .NagegarNoPerimetro(FATOR-PONTOS)
  |       .TransformarParaSegmentoDeReta()
4 | segmentos.EscreverSVGParaSaidaPadrão();
```

Fonte: Elaborado pelo autor.

Figura 18 - Conversão para segmentos de reta



Fonte: Elaborado pelo autor.

3.3.1.7 Script de multiplicação de polígonos

Para casos onde é necessário o empacotamento de várias peças do mesmo conjunto, pode ser utilizado o script de multiplicação de peças. Este script escreve `N` vezes o conjunto em um novo arquivo SVG.

No Quadro 8 é possível visualizar o funcionamento do multiplicador, na linha 1 é fornecida via linha de comando o caminho do arquivo SVG, em seguida é lido o parâmetro de multiplicação. Na linha 3 para todas as tags `<path>` é feito a multiplicação, que consiste na cópia da mesma dentro do arquivo SVG. Por fim o arquivo é escrito na saída padrão.

Quadro 8 - Pseudocódigo script de multiplicação

```
1 | CAMINHO                = args[0]
2 | MULTIPLICADOR = int(args[1])
3 | polygonos = leitorXML.Ler(CAMINHO)
  |       .ParaTodasAsTAGS('svg>path')
  |       .MutiplicarPor(MULTIPLICADOR);
4 | segmentos.EscreverSVGParaSaidaPadrão();
```

Fonte: Elaborado pelo autor.

3.3.2 Repositório do JPacking

O programa JPacking pode ser acessado no seu repositório oficial[4] localizado no GitHub. Lá podem ser encontrados a implementação em Java, além dos scripts desenvolvidos em Python, tal como *datasets* para realização de testes.

3.4 RESULTADOS

Para obtenção de dados da fundamentação dos resultados do JPacking foram realizados seis tipos de testes com diferentes parâmetros de execução e diferentes *datasets*. Salientando que a ocupação da matéria-prima deve ser sempre otimizada, sendo assim, para cada teste os resultados obtidos com ocupação menor são os melhores resultados.

3.4.1 Ferramentas utilizadas para levantamento de dados

Para realização dos testes foram utilizados scripts Bash para a automatização, a linguagem de programação Python para manipulação dos dados de saída e para a geração de arquivos CSV. E por fim foi utilizado a linguagem de programação R em conjunto com a biblioteca ggplot2 para geração dos gráficos.

3.4.2 Testes realizados

Os testes realizados tiveram o intuito de coletar os resultados da performance do programa JPacking em diferentes situações. O Teste 1 visou a comparação do parâmetro de quantidade de rotações.

A seguir o Teste 2 explorou a comparação da execução do Algoritmo Genético com diferentes populações e gerações. Já no Teste 3 foi realizada a coleta dos resultados comparando o Algoritmo Genético com os algoritmos de Hill Climbing e Tabu Search implementados por Brandt (2011).

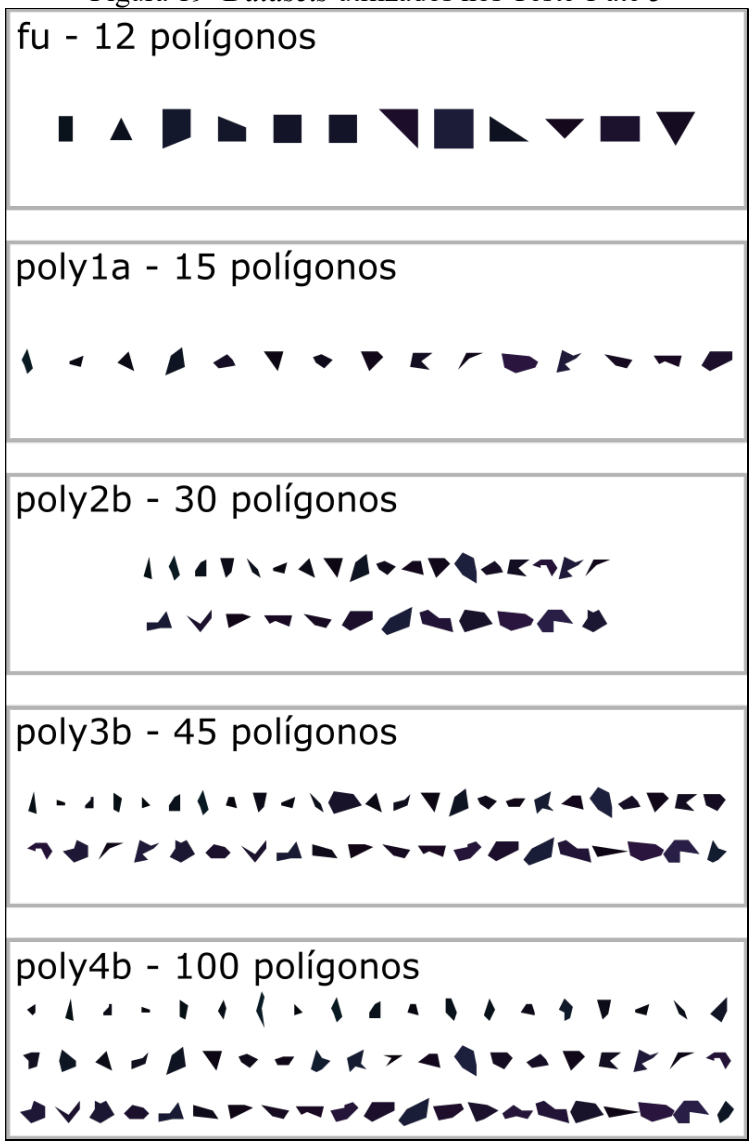
Dando continuidade aos testes, o Teste 4 verifica a performance do Algoritmo Genético variando os fatores de Crossover e de Mutação. O Teste 5 compara a ocupação do Algoritmo Genético e dos algoritmos de Hill Climbing e Tabu Search variando a altura do material. Os Testes 1 até 5 utilizaram os *datasets* com os polígonos mostrados na Figura 19.

Por fim, o Teste 6 observa o comportamento do JPacking utilizando *datasets* que se aproximam mais do mundo real.

---

[4] Repositório do JPacking: https://github.com/rodra55d/TCC2/tree/master/packingProblem/packing-problem

Figura 19- *Datasets* utilizados nos Teste 1 até 5



Fonte: Elaborado pelo autor.

3.4.3 Teste 1: Rotações

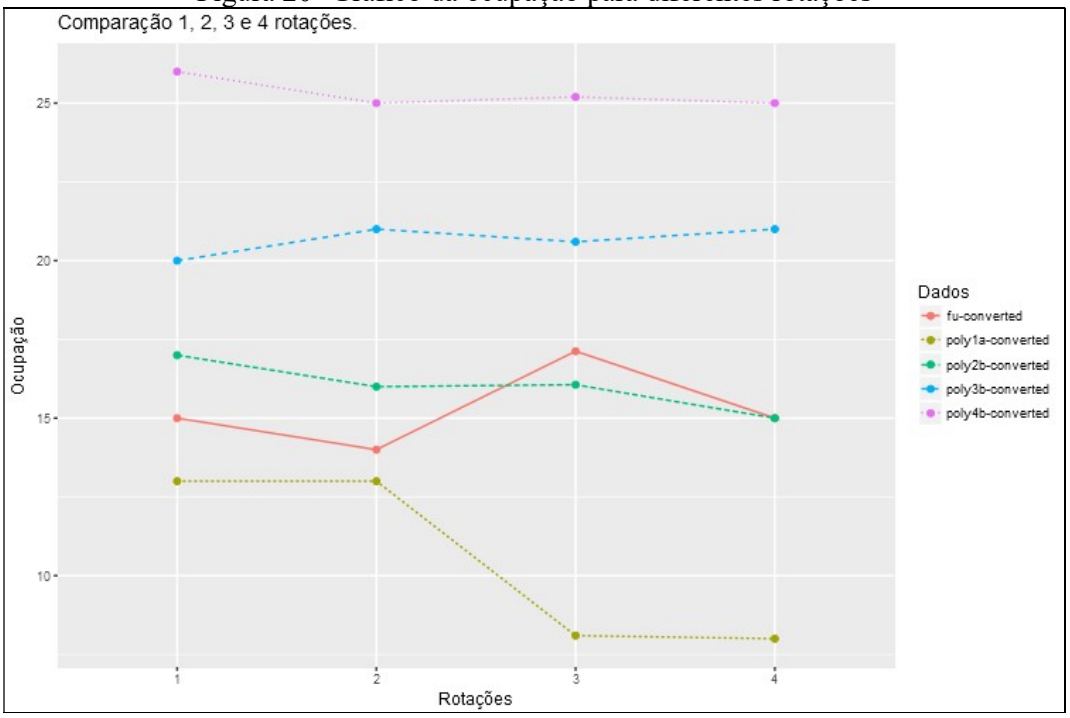
O Teste 1 verifica o quanto a rotação influencia na ocupação. Os parâmetros de execução do Teste 1 podem ser vistos no Quadro 9. Na Tabela 4 localizada no Apêndice A é possível visualizar os valores dos resultados e na Figura 20 é mostrado o gráfico dos resultados obtidos.

Quadro 9 - Parâmetros de execução do Teste 1

Parâmetro	Valores	Parâmetro	Valores
Altura	100	Idade Máxima	20%
Rotação	0°, 90°, 120°, 180°	Fator Crossover	0.35
Gerações	20	Fator Mutação	0.2
Populações	100	Melhor Estável	30%
Tempo	Fim execução	*Datasets*	fu, poly1a, poly2b, poly3b e poly4b

Fonte: Elaborado pelo autor.

Figura 20- Gráfico da ocupação para diferentes rotações



Fonte: Elaborado pelo autor.

Na Tabela 4 são destacados em negrito os melhores resultados para cada tipo de dataset. No gráfico da Figura 20 é possível visualizar que a rotação das peças ajudou a melhorar a ocupação nos *datasets* `poly1a`, `poly2b` e `poly4b`, com exceção de 2 *datasets* `fu` e `poly3b`.

Foi possível concluir com a execução do Teste 1 que a rotação tem influência no melhoramento da ocupação em todos os *datasets*. Contudo, a cada rotação adicional faz com que o tempo de execução cresça linearmente (O(n)).

3.4.4 Teste 2: Gerações e populações

O Teste 2 teve o intuito de verificar qual combinação entre quantidade de gerações e tamanho de populações obtém as melhores ocupações. Os parâmetros de execução do Teste 2 podem ser vistos no Quadro 10. Na Tabela 5 localizada no Apêndice B são mostrados os

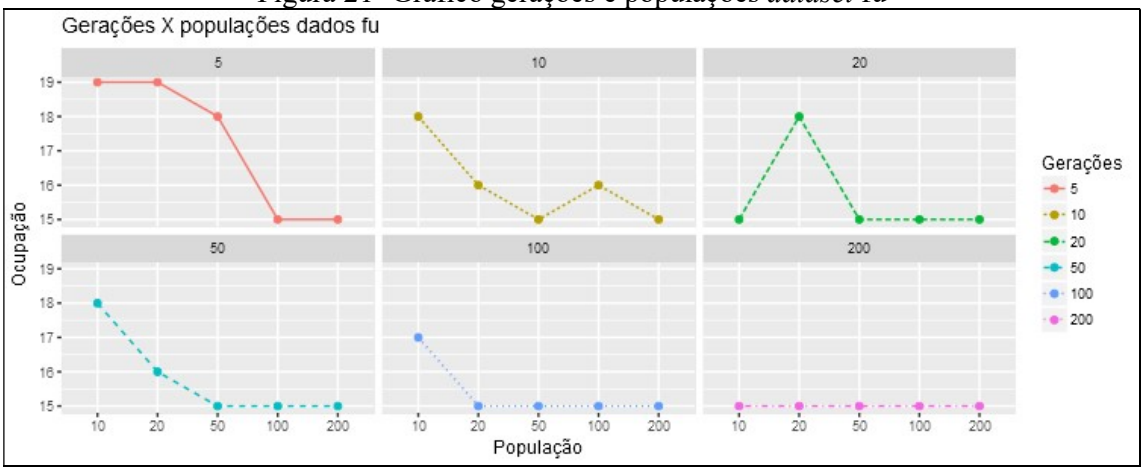
valores para o *dataset* `fu` e também Tabela 6 localizada no Apêndice B são mostrados os valores para o *dataset* `poly3b`. Na Figura 21 é mostrado o gráfico para o *dataset* `fu` e Figura 22 é mostrado o gráfico dos resultados obtidos para o *dataset* `poly3b`.

Quadro 10 - Parâmetros de execução do Teste 2

Parâmetro	Valores	Parâmetro	Valores
Altura	100	Idade Máxima	20%
Rotação	0°	Fator Crossover	0.35
Gerações	5, 10, 20, 50, 100 e 200	Fator Mutação	0.2
Populações	10, 20, 50, 100 e 200	Melhor Estável	30%
Tempo	Fim execução	*Datasets*	fu, poly3b

Fonte: Elaborado pelo autor.

Figura 21- Gráfico gerações e populações *dataset* fu



Fonte: Elaborado pelo autor.

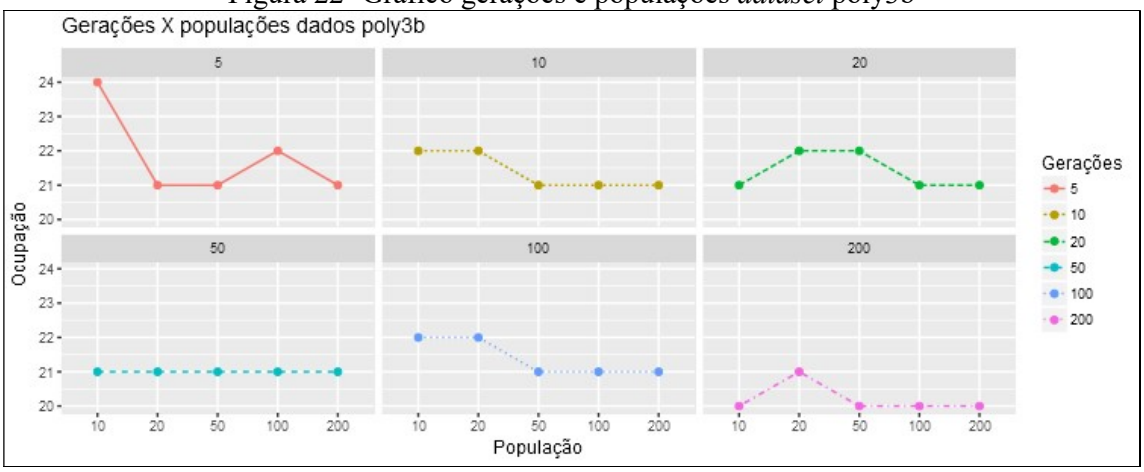
Na Tabela 5 são destacados em negrito as execuções que obtiveram as melhores ocupações para o *dataset* `fu`. Já no gráfico apresentado na Figura 21 é possível visualizar que para cada quantidade de gerações e o aumento da população ajudaram a obter uma melhor ocupação.

É possível notar que para cada geração o tamanho de população 200 foi o único a obter a melhor ocupação independentemente da quantidade de gerações. Também é possível observar que a quantidade de gerações 200 foi a única a obter melhor ocupação independentemente do tamanho da população.

Com esses resultados é possível afirmar que para o *dataset* `fu` um tamanho de população de 200 e uma quantidade de gerações também 200, permitem obter os melhores resultados. No entanto também foram obtidos resultados de melhor ocupação com valores de geração e população menores que 200 como pode ser visto na Tabela 5.

Isto leva a concluir que executar vários testes calibrando os valores de gerações e população torna possível obter bons resultados em menor tempo.

Figura 22- Gráfico gerações e populações *dataset* poly3b



Fonte: Elaborado pelo autor.

A Tabela 6 apresenta em negrito os melhores resultados de ocupação para o *dataset* `poly3b`. O intuito desse segundo teste foi verificar se uma quantidade de gerações alta e um tamanho de população alta continua a apresentar melhores resultados para um *dataset* diferente.

Como pode ser visualizado no gráfico da Figura 22 a quantidade de 200 gerações foi a única a obter melhores ocupações. No entanto houve mais de um tamanho de população que obteve melhor ocupação como pode ser visto na Tabela 6.

Com esses resultados é possível afirmar que para o *dataset* `poly3b` uma quantidade de gerações de 200 permitem obter os melhores resultados. Isto leva a concluir que da mesma forma que no *dataset* `fu`, executar vários testes calibrando os valores de gerações e população pode ser possível obter de bons resultados em um tempo menor.

3.4.5 Teste 3: Comparação diferentes fatores de Crossover e Mutação

O programa JPacking por padrão executa com um fator de Crossover de 0.35 e um fator de Mutação de 0.2. Nestes testes foram realizados empacotamentos variando o fator de Crossover e de Mutação. O Teste verificou se os melhores empacotamentos se encontram próximos aos fatores padrões de Crossover e Mutação.

3.4.5.1 Crossover

Este Teste foi realizado variando o valor do fator de Crossover. Os parâmetros de execução do Teste de variação de fator de Crossover podem ser vistos no Quadro 11. Na Tabela 7 localizada no Apêndice C é possível visualizar os valores dos resultados obtidos e na Figura 23 pode ser visualizado o gráfico com os resultados.

Quadro 11 - Parâmetros de execução do Teste 3 para Crossover

Parâmetro	Valores	Parâmetro	Valores
Altura	100	Idade Máxima	20%
Rotação	0°	Fator Crossover	0.1, 0.3, 0.35, 0.5, 0.7, e 0.9
Gerações	Infinita	Fator Mutação	0.2
Populações	50	Melhor Estável	30%
Tempo	2 minutos	*Datasets*	fu, poly1a, poly2b, poly3b e poly4b

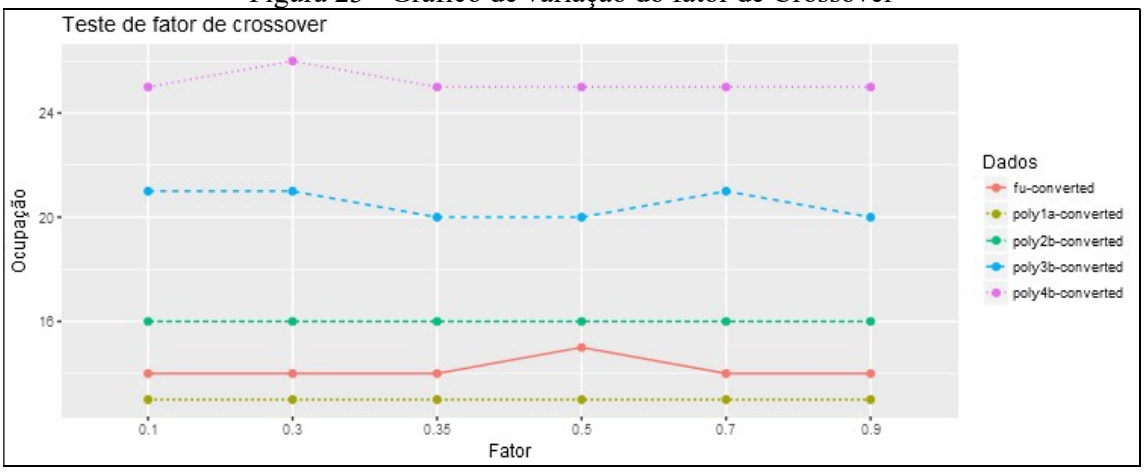
Fonte: Elaborado pelo autor.

Na Tabela 7 pode ser visualizado em destaque os valores da ocupação para a variação do fator de Crossover. No gráfico apresentado na Figura 23 é possível visualizar que o fator de 0.35 esteve sempre presente nas melhores ocupações para cada um dos *datasets*.

É possível observar que o fator padrão de Crossover obteve as melhores ocupações para todo os *datasets*. No entanto é possível visualizar na Tabela 7 que outros valores de fator de Crossover também obtiveram melhores ocupações.

Assim pode-se concluir que o valor do fator de Crossover padrão obtém melhores resultados, contudo calibrando o valor do fator de Crossover pode ser possível resultados ainda melhores.

Figura 23 - Gráfico de variação do fator de Crossover



Fonte: Elaborado pelo autor.

3.4.5.2 Mutação

Este Teste foi realizado variando o valor do fator de Mutação. Os parâmetros de execução do Teste de variação de fator de Mutação podem ser vistos no Quadro 12. Na Tabela 8 localizada no Apêndice C é possível visualizar os valores dos resultados obtidos e na Figura 24 pode ser visualizado o gráfico com os resultados.

Quadro 12 - Parâmetros de execução do Teste 3 para Mutação

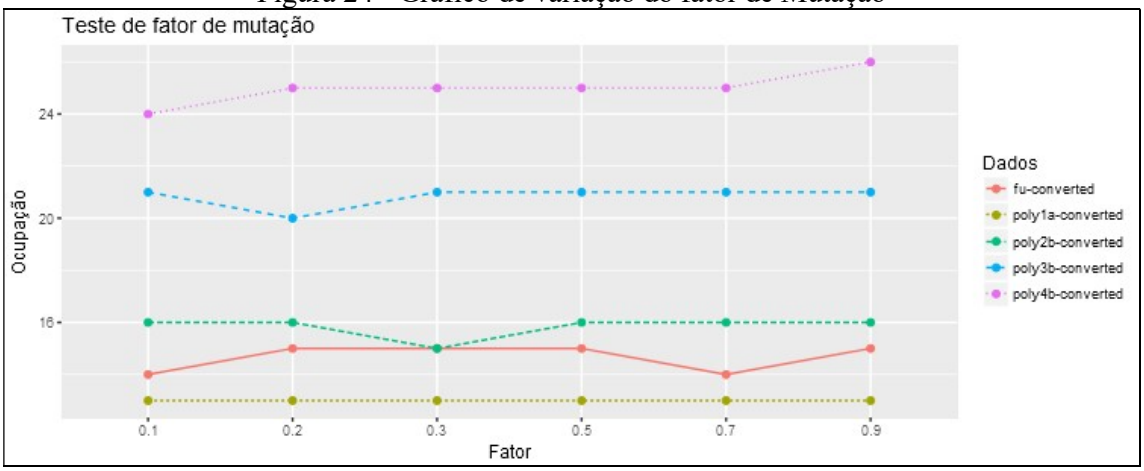
Parâmetro	Valores	Parâmetro	Valores
Altura	100	Idade Máxima	20%
Rotação	0°	Fator Crossover	0.35
Gerações	Infinita	Fator Mutação	0.1, 0.2, 0.3, 0.5, 0.7, 0.9
Populações	50	Melhor Estável	30%
Tempo	2 minutos	*Datasets*	fu, poly1a, poly2b, poly3b e poly4b

Fonte: Elaborado pelo autor.

Na Tabela 8 é possível visualizar em negrito os valores dos fatores que obtiveram a menor ocupação para cada um dos *datasets*. Já no gráfico apresentado na Figura 24 é possível visualizar que não existe um valor do fator de Mutação onde todos os *datasets* convergem para uma ocupação menor.

Assim é possível concluir que o valor do fator de Mutação é influenciado pelo *dataset,* um valor padrão não obtém melhores resultados em todos os *datasets*. Contudo calibrando o valor do fator de Mutação através de vários testes pode ser possível achar o fator de Mutação de cada *dataset* que gere melhores resultados.

Figura 24 - Gráfico de variação do fator de Mutação



Fonte: Elaborado pelo autor.

3.4.6 Teste 4: Comparação Genético, Hill Climbing e Tabu Search

O Teste 4 teve o intuito de comparar os o Algoritmo Genético com os algoritmos de Hill Climbing e Tabu Search implementados por Brandt (2011). Os parâmetros de execução do Teste 4 podem ser vistos no Quadro 13. Na Tabela 9 localizada no Apêndice D é possível visualizar os valores dos resultados obtidos e na Figura 25 pode ser visualizado o gráfico com os resultados. As variações do Algoritmo Genético são identificadas como: `G_p50` (executado com tamanho de população de 50 indivíduos), `G_p100` (100 indivíduos) e `G_p200` (200 indivíduos).

Quadro 13 - Parâmetros de execução do Teste 4

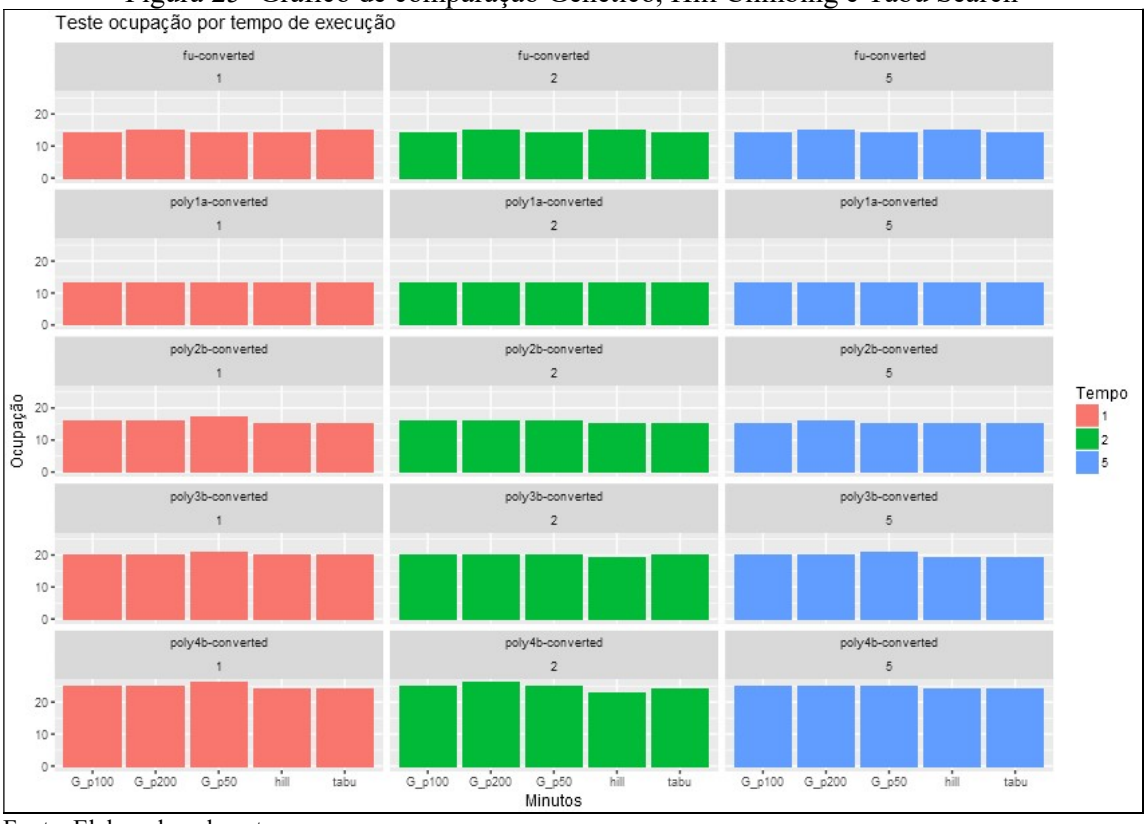
Parâmetro	Valores	Parâmetro	Valores
Altura	100	Idade Máxima	20%
Rotação	0°	Fator Crossover	0.35
Gerações	Infinita	Fator Mutação	0.2
Populações	50, 100, 200	Melhor Estável	30%
Tempo	1, 2 e 5 minutos	*Datasets*	fu, poly1a, poly2b, poly3b e poly4b
Algoritmos	Genético, Hill Climbing e Tabu Search		

Fonte: Elaborado pelo autor.

Também na Tabela 9 são identificados em negrito os testes que obtiveram melhor ocupação para cada dataset. No gráfico apresentado na Figura 25, cada linha representa um *dataset* diferente, sendo que cada coluna separa os minutos, 1, 2, e 5. O eixo X representa os algoritmos utilizados e o eixo Y a ocupação.

Visualizando o gráfico apresentado na Figura 25 e verificando os valores apresentados na Tabela 9 é possível concluir que algoritmos de Hill Climbing e Tabu Search obtiveram os melhores resultados, contudo para os *datasets* (`fu`, `poly1a` e `poly2b`) o Algoritmo Genético foi capaz de alcançar os mesmos resultados do Hill Climbing e Tabu Search.

Figura 25- Gráfico de comparação Genético, Hill Climbing e Tabu Search



Fonte: Elaborado pelo autor.

3.4.7 Teste 5: Variação altura

Este Teste comparou o Algoritmo Genético com os algoritmos de Hill Climbing e Tabu Search implementados por Brandt (2011), variando a altura da forma. Os parâmetros de execução do Teste 5 podem ser vistos no Quadro 14. Na Tabela 10 localizada no Apêndice E é possível visualizar os valores dos resultados obtidos e na Figura 26 pode ser visualizado o gráfico com os resultados.

Quadro 14 - Parâmetros de execução do Teste 5

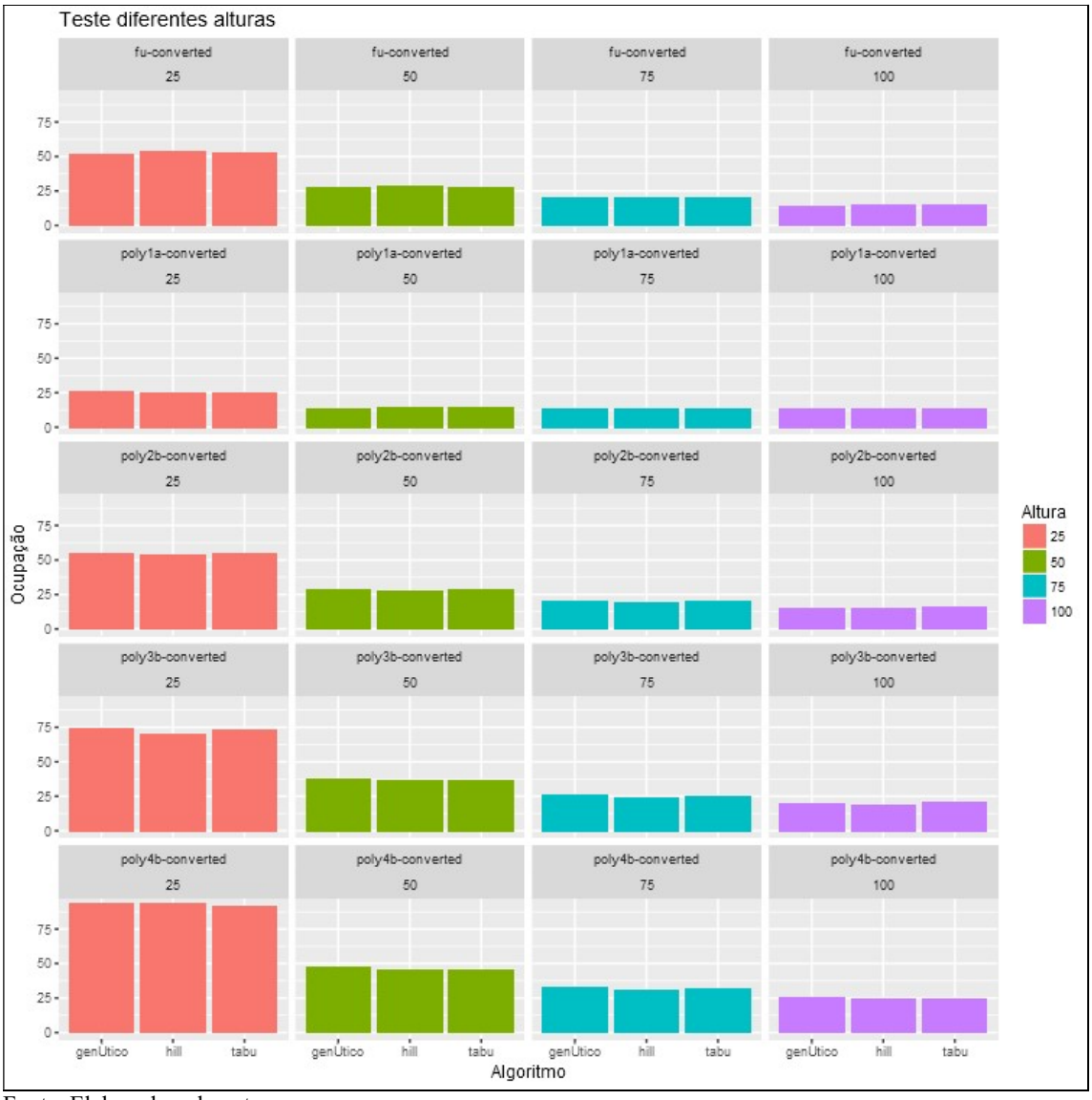
Parâmetro	Valores	Parâmetro	Valores
Altura	25, 50, 75, 100	Idade Máxima	20%
Rotação	0°	Fator Crossover	0.35
Gerações	Infinita	Fator Mutação	0.2
Populações	200	Melhor Estável	30%
Tempo	2 minutos	*Datasets*	fu, poly1a, poly2b, poly3b e poly4b
Algoritmos	Genético, Hill Climbing e Tabu Search		

Fonte: Elaborado pelo autor.

Na Tabela 10 estão destacados em negrito as melhores ocupações para cada *dataset* e altura. No gráfico da Figura 26 cada linha representa um *dataset* diferente e cada coluna uma altura. O eixo `X` representa os algoritmos utilizados e o eixo `Y` a ocupação.

É possível concluir através dos valores da Tabela 10 e no gráfico apresentado na Figura 26 que o algoritmo de Hill Climbing obteve a melhor ocupação na maioria dos testes. O Algoritmo Genético obteve a melhor ocupação no teste realizado com o *dataset* `fu` e altura de 25, os demais testes conforme Tabela 10 o Algoritmo Genético alcançou os mesmos resultados do Hill Climbing e do Tabu Search.

Figura 26- Gráfico de comparação da variação de altura para o Genético, Hill Climbing e Tabu Search



Fonte: Elaborado pelo autor.

3.4.8 Teste 6: Avaliação resultados de empacotamento outros *datasets* com JPacking e SVGNest

O Teste 6 tem o intuito de observar o comportamento do JPacking utilizando *datasets* que se aproximam mais do mundo real do que os *datasets* `fu`, `poly1a`, `poly2b`, `poly3b`, poly4b utilizados anteriormente nos testes. Para isso foram criados 7 *datasets* novos (`alma1`, `alma2`, `alma3`, `alma4`, `metal1`, `roupas1`, `roupas2`) utilizando figuras de partes de roupas e peças de metal. O Teste 6 também avalia o resultado do empacotamento gerado pelo do JPacking com o empacotamento gerado pelo programa de código livre SVGNest desenvolvido por Qiao (2017).

Os parâmetros para execução do teste podem ser vistos no Quadro 15. Os resultados com as imagens de cada empacotamento de cada um dos programas para cada *dataset* podem ser encontrados nos seguintes Quadros:

a) Quadro 17 - Resultado empacotamento *dataset* alma1 localizado no Apêndice H;
b) Quadro 18 - Resultado empacotamento *dataset* alma2 localizado no Apêndice I;
c) Quadro 19 - Resultado empacotamento *dataset* alma3 localizado no Apêndice J;
d) Quadro 20 - Resultado empacotamento *dataset* alma4 localizado no Apêndice K;
e) Quadro 21 - Resultado empacotamento *dataset* metal1 localizado no Apêndice L;
f) Quadro 22 - Resultado empacotamento *dataset* roupas1 localizado no Apêndice M;
g) Quadro 23 - Resultado empacotamento *dataset* roupas2 localizado no Apêndice N.

Quadro 15 - Parâmetros de execução do Teste 6

JPacking			
Parâmetro	Valores	Parâmetro	Valores
Altura	100	Idade Máxima	20%
Rotação	0°	Fator Crossover	0.35
Gerações	200	Fator Mutação	0.2
Populações	200	Melhor Estável	30%
SVGNest			
Parâmetro	Valores	Parâmetro	Valores
Space between parts	0	Curve tolerance	0.3
Part rotations	1	GA population	200
GA mutation rate	10	Iterations	200
Datasets			
alma1, alma2, alma3, alma4, metal1, roupas1, roupas2			

Fonte: Elaborado pelo autor.

Foi possível visualizar no Teste 6 que o JPacking obteve resultados de empacotamento sem erro nos *datasets* (`alma1, alma2, alma3, alma4, roupas1, roupas2`) com exceção do *dataset* (`metal1`). Já o programa SVGNest desenvolvido por Qiao (2017), obteve resultados sem erros somente nos *datasets* (`alma2` e `roupas2`), o restante dos *datasets* o programa SVGNest realizou o empacotamento com erros.

Assim é possível concluir a partir da visualização dos resultados dos empacotamentos do Teste 6 que o programa JPacking é capaz de realizar o empacotamento de *datasets* com peças que se aproxima do mundo real.

3.4.9 Comparação JPacking e trabalhos correlatos

Nesta seção é apresentado a comparação entre os trabalhos de Haiming, Jiong e Xinsheng (2006), Junior, Pinheiro e Saraiva (2013) e Brandt (2011) com JPacking desenvolvido neste trabalho.

Quadro 16 - Comparação trabalhos correlatos e trabalho atual

Funcionalidade	Haiming, Jiong e Xinsheng (2006)	Junior, Pinheiro e Saraiva (2013)	Brandt (2011)	Trabalho atual
Estratégia de encaixe	NFP, low gravity center	NFP, Bottom-left fill	NFP, Bottom-left fill	NFP, Bottom-left fill
Algoritmo	Genético	Genético	Hill Climbing e Tabu Search	Genético
Técnica adicional	-	Encolhimento	-	-
Interface Gráfica	-	-	Sim	Não
Interface linha de comando	-	-	Não	Sim
Alteração manual	-	-	Sim	Sim
Arquivo de entrada	-	-	XML	SVG
Arquivo de saída	não especificado	não especificado	não tem	SVG

Fonte: Elaborado pelo autor.

Como pode ser observado no Quadro 16 todos os trabalhos utilizam o NFP e a sua maioria como o JPacking utiliza disposição Bottom-left. A maioria dos trabalhos utiliza algoritmo genético, com exceção do trabalho de Brandt (2011) no qual são utilizados os algoritmos de Hill Climbing e Tabu Search.

Adiante, o trabalho de Junior, Pinheiro e Saraiva (2013) utiliza a técnica de encolhimento para melhora o encaixe. O JPacking não utiliza técnica adicional visto que

utiliza um arquivo SVG como entrada e saída, desta forma é possível realimentar o JPacking com o arquivo do resultado gerado anteriormente e permite realizar mudanças nos parâmetros do Algoritmo Genético para obter um encaixe melhor.

A seguir, alguns trabalhos não mencionam a utilização de uma interface gráfica, contudo o trabalho de Brandt (2011) apresenta uma interface gráfica para visualização do processo de empacotamento passo-à-passo. O JPacking oferece a saída em SVG, podendo assim ser manipulada em qualquer editor vetorial e visualizado em vários navegadores web.

Já o item Interface linha de comando, o JPacking é o único que disponibiliza uma interface para linha de comando, permitindo a automatização do processo de empacotamento através de scripts. Quanto ao item alteração manual, o trabalho de Brandt (2011) permite a alteração através da manipulação do resultado no programa implementado no trabalho, o JPacking permite a alteração utilizando qualquer editor vetorial com suporte a SVG.

Por fim, o JPacking utiliza o tipo de arquivo SVG como entrada e também como saída, ao invés de um formato XML único. Assim são possíveis a transferência e a manipulação e conversão dos resultados em programas que suportem a leitura do tipo de arquivo SVG.

3.4.10 Outros testes e resultados

No repositório oficial do JPacking[5] podem ser encontrados todos os testes realizado em conjunto com as tabelas, gráficos e arquivos SVG contendo os resultados obtidos.

3.5 DISCUSSÃO

Nesta seção são abordados os problemas encontrados durante o desenvolvimento além de investigações sobre o problema encontrado para servir de ponto de início para futuras intenções do trabalho.

3.5.1 Erros de empacotamento

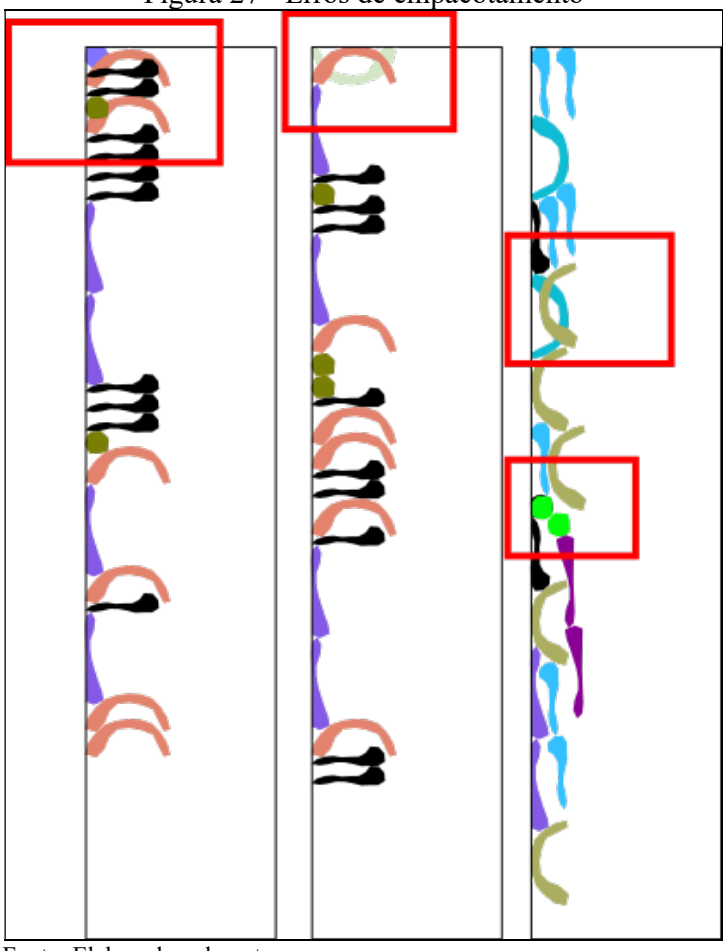
Durante a execução de alguns testes, houveram erros no empacotamento em *datasets* que continham polígonos muito irregulares ou côncavos. Desta forma a execução do NFP acusa que não foi possível criar o polígono de NFPAP.

Foi investigado se o problema era devido a existir polígonos com seus vértices em sentido horário e outros em sentido anti-horário. Foi concluído que a biblioteca de NFP utilizada, neste caso o JNFP pode tratar com polígonos em ambos os sentidos. Assim sendo,

[5] Os testes estão localizados em: https://github.com/rodra55d/TCC2/tree/master/tests

não foi possível concluir o real motivo deste problema. Na Figura 27 é possível visualizar alguns casos onde houve erros de empacotamento.

Figura 27 - Erros de empacotamento



Fonte: Elaborado pelo autor.

3.5.2 Limite de polígono e performance do Bottom-left fill

Durante os testes com *datasets* maiores foi verificado que a biblioteca Jenetics utilizando o formato de manipulação dos genes via permutação tem um limite de 43330 genes por indivíduo. Na Figura 28 é possível visualizar o erro gerado durante a execução.

Figura 28 - Erro de execução limite de genes

```
$ java -cp "packing.jar;jenetics.jar;cc.jar" JeneticExecutor poly4b-converted-10
000 100 1 999999 200 300000
Root element svg
Information of all polygons
Exception in thread "main" java.lang.IllegalArgumentException: n*sub.length > In
teger.MAX_VALUE (600000*600000 = 360000000000 > 2147483647)
        at org.jenetics.internal.math.base.checkSubSet(base.java:321)
        at org.jenetics.internal.math.base.subset(base.java:234)
        at org.jenetics.internal.math.base.subset(base.java:168)
        at org.jenetics.internal.math.base.subset(base.java:138)
        at org.jenetics.PermutationChromosome.of(PermutationChromosome.java:248)
        at org.jenetics.PermutationChromosome.ofInteger(PermutationChromosome.ja
va:338)
        at org.jenetics.PermutationChromosome.ofInteger(PermutationChromosome.ja
va:318)
        at org.jenetics.PermutationChromosome.ofInteger(PermutationChromosome.ja
va:293)
        at org.jenetics.engine.codecs.ofPermutation(codecs.java:595)
        at br.furb.packing.jenetic.JeneticAlgorithm.doPackingCustom(JeneticAlgor
ithm.java:95)
        at JeneticExecutor.executePacking(JeneticExecutor.java:26)
        at JeneticExecutor.main(JeneticExecutor.java:54)
```

Fonte: Elaborado pelo autor.

No entanto, o maior problema é a performance do algoritmo de Bottom-left fill implementado por Brandt (2011). Com *datasets* com mais de 150 peças o tempo de execução chega a levar mais de 5 horas, o que levou ao cancelamento de alguns testes com *datasets* maiores.

Uma investigação do problema levou a conclusão que o método de implementação utilizado realiza o deslocando em pequenas proporções, o que acaba levando muito tempo para achar uma posição de encaixe válida.

4 CONCLUSÕES

O objetivo deste trabalho foi desenvolver o JPacking um programa capaz de realizar o empacotamento de polígonos irregulares um uma superfície bidimensional utilizando Algoritmos Genéticos. Através dos resultados obtidos foi possível observar que o objetivo foi alcançado, visto que o programa oferece como entrada e saída um arquivo SVG, o que permite sua retroalimentação e a edição dos arquivos em qualquer editor vetorial com suporte a SVG.

Pode se destacar também através dos resultados obtidos que não existem parâmetros fixos para todos os tipos de *datasets* contendo os mais variados tipos de polígonos. O processo de utilização do programa JPacking necessita que sejam escolhidos os parâmetros corretos para cada situação. Quantidades elevadas de gerações e tamanhos grandes de populações podem ajudar a obter os resultados melhores. No entanto a calibração do fator de crossover, mutação, alterar a quantidade de rotações também pode ajudar a obter resultados melhores com tempos menores.

Foi também possível notar através dos testes que algoritmos como o Hill Climbing e o Tabu Search podem obter resultados melhores que o Algoritmo Genético. Contudo, o Algoritmo Genético permite uma maior configuração de parâmetros de execução o que permite igualar ou até superar os resultados obtidos pelo Hill Climbing e Tabu Search.

Para concluir, o programa JPacking não é totalmente perfeito, apesar de ter resultados melhores que o SVGNest em *datasets* que se aproximam do mundo real, existem ainda problemas de empacotamento que ocorrem com certos *datasets*. Sua performance para quantidades maiores de polígonos e sua incapacidade de utilizar formas de matéria prima que tenha limitação de forma, justificam fazer melhorias na ferramenta. Assim através da implementação de suas extensões e a aplicação de melhoramentos todos os impedimentos podem ser resolvidos e podem tornar o JPacking utilizável em um ambiente de produção.

4.1 EXTENSÕES

Nesta seção serão listados melhoramentos e extensões para o programa JPacking:

a) reimplementar o algoritmo de Bottom-left fill utilizando o método de Qiao (2017), no qual é feita a junção todos os polígonos NFPAB e realizado o deslocando para através dos vértices para encontrar uma posição ideal;
b) procurar por um método heurístico alternativo para determinar a melhor rotação de um polígono durante o processo de empacotamento;
c) adicionar espaço entre os polígonos, pois existem métodos de corte que consomem

parte da matéria prima;

d) explorar o empacotamento de polígonos maiores primeiro e menores depois, como é feito no método de Qiao (2017);
e) explorar buracos e regiões côncavas de polígonos;
f) procurar aproveitar pedaços de matéria prima com limitações de formato;
g) criar método para invalidar partes da matéria prima que contém problemas como furos e manchas;
h) criação de mecanismo de cache para o Bottom-left fill, para evitar refazer o mesmo empacotamento para a mesma ordem de empacotamento;
i) integrar o JPacking como uma extensão para o programa Inkscape.

REFERÊNCIAS

AFFENZELLER, Michael; WINKLER, Stephan; BEHAM, Andreas. **Genetic Algorithms and Genetic Programming: modern concepts and practical applications**. Boca Raton: Chapman & Hall/crc, 2009.

AUTOMATION, Aeronaut. **Aeronaut Automation Automated Cutting Systems & Software**. 2017. Disponível em: <http://www.aeronaut.org/>. Acesso em: 28 maio 2017.

BRANDT, Denise. **Distribuição otimizada de polígonos em um plano bidimensional**. 2011. 65 f. TCC (Graduação) - Curso de Ciência da Computação, Fundação Universidade Regional de Blumenau, Blumenau, 2011.

BURKE, Edmund Kieran. Complete and robust no-fit polygon generation for the irregular stock cutting problem. **European Journal of Operational Research**. Nottingham, p. 27-49. 16 mar. 2007.

DAHLSTRÖM, Erik; DENGLER, Patrick; GRASSO, Anthony**. Scalable Vector Graphics (SVG) 1.1. 2011**. Disponível em: <https://www.w3.org/TR/SVG/>. Acesso em: 28 maio 2017.

GOLDBERG, David; LINGLE, Robert. Alleles Lociand the Traveling Salesman Problem. **Proceedings of The 1st International Conference On Genetic Algorithms**. Hillsdale, p. 154-159. jan. 1985.

HAIMING, Liu; JIONG, Zhou; XINSHENG, Wu**.** Optimization algorithm based on niche genetic algorithm for irregular nesting problem. **Chinese Control and Decision Conference**. China, p. 4259-4263. 25 mar. 2015.

JUNIOR, Bonfim A.; PINHEIRO, Plácido R.; SARAIVA, Rommel D.. Tackling the Irregular Strip Packing Problem by Hybridizing Genetic Algorithm and Bottom-left Heuristic. **IEEE Congress on Evolutionary Computation**. Cancún, México, p. 3012-3018. 20 jun. 2013.

KENDALL, Graham. **Applying Meta-Heuristic Algorithms to the Nesting Problem Utilising the No Fit Polygon**. 2000. 242 f. Tese (Doutorado) - Curso de Computer Science And Information Technology, University Of Nottingham, Nottingham, 2000.

LIU, Huyao; HE, Yuanjun. Research and Implementation of Irregular-Shaped Nesting Problem. **Proceedings of The 6th World Congress On Intelligent Control And Automation**. Dalian, p. 1397-1400. 21 jun. 2006.

MITCHELL, Melanie. **An Introduction to Genetic Algorithms**. Cambridge: Mit Press, 1998.

MUNDIM, Leandro R.; QUEIROZ, Thiago Alves de. A hybrid heuristic for the 0–1 knapsack problem with items of irregular shape. **Conferencia Latinoamericana en Informática**. Medellin, p. 1-6. 01 out. 2012.

POPA, Rustem. **Genetic Algorithms in Applications**. Rijeka, Croatia: In Tech, 2012.

QIAO, Jack. **SVGnest: An open source vector nesting tool**. 2017. Disponível em: <https://github.com/Jack000/SVGnest>. Acesso em: 28 maio 2017.

REGEBRO, Lennart. **Svg.path: SVG path objects and parser**. 2017. Disponível em: <https://pypi.python.org/pypi/svg.path>. Acesso em: 01 jun. 2017.

SIVANANDAM; DEEPA. **Introduction to Genetic Algorithms**. Coimbatore, India: Springer, 2008.

WAUTERS, Tony. **JNFP: A robust and open-source no-fit polygon generator library in Java**. 2016. Disponível em: <https://github.com/TonyWauters/JNFP>. Acesso em: 01 jun. 2017.

WHITWELL, Glenn. **Novel Heuristic and Metaheuristic Approaches to Cutting and Packing**. 2004. 314 f. Tese (Doutorado) - Curso de Doctor Of Philosophy, University Of Nottingham, Nottingham, 2004.

WILHELMSTÖTTER, Franz. **Jenetics: Java Genetic Algorithm Library**. 2017. Disponível em: <http://jenetics.io/>. Acesso em: 01 jun. 2017.

APÊNDICE A – Tabela de resultados do Teste 1

Este apêndice apresenta (Tabela 4) os resultados obtidos na execução do teste 1 utilizando 0°, 90°, 120° e 180° de rotações.

Tabela 4 - Resultados da ocupação para 1,2,3 e 4 rotações.

Dataset	Altura	Rot.	Ger.	Pop.	Ocupação	Tempo/ms
fu	100	1	20	100	15,0	1947
fu	**100**	**2**	**20**	**100**	**14,0**	**1735**
fu	100	3	20	100	17,1	4296
fu	100	4	20	100	15,0	4014
poly1a	100	1	20	100	13,0	1602
poly1a	100	2	20	100	13,0	2555
poly1a	100	3	20	100	8,1	4540
poly1a	**100**	**4**	**20**	**100**	**8,0**	**5533**
poly2b	100	1	20	100	17,0	6515
poly2b	100	2	20	100	16,0	14758
poly2b	100	3	20	100	16,1	25172
poly2b	**100**	**4**	**20**	**100**	**15,0**	**22834**
poly3b	**100**	**1**	**20**	**100**	**20,0**	**26974**
poly3b	100	2	20	100	21,0	24380
poly3b	100	3	20	100	20,6	36630
poly3b	100	4	20	100	21,0	49832
poly4b	100	1	20	100	26,0	30988
poly4b	**100**	**2**	**20**	**100**	**25,0**	**60732**
poly4b	100	3	20	100	25,2	111422
poly4b	100	4	20	100	25,0	102600

Fonte: Elaborado pelo autor.

APÊNDICE B – Tabelas de resultados do Teste 2

Este apêndice apresenta (Tabela 5 e Tabela 6) os resultados obtidos na execução do teste 2, utilizando combinações de 5, 10, 20, 50, 100 gerações para tamanho de população de 10, 20, 50 100 e 200 indivíduos para os datasets fu e poly3b.

Tabela 5 - Teste gerações e população *dataset* fu

Dataset	Altura	Rot.	Ger.	Pop.	Ocupação	Tempo/ms
fu	100	1	5	10	19	308
fu	100	1	5	20	19	390
fu	100	1	5	50	18	574
fu	**100**	**1**	**5**	**100**	**15**	**808**
fu	**100**	**1**	**5**	**200**	**15**	**1641**
fu	100	1	10	10	18	370
fu	100	1	10	20	16	483
fu	**100**	**1**	**10**	**50**	**15**	**964**
fu	100	1	10	100	16	1151
fu	**100**	**1**	**10**	**200**	**15**	**1842**
fu	**100**	**1**	**20**	**10**	**15**	**429**
fu	100	1	20	20	18	653
fu	**100**	**1**	**20**	**50**	**15**	**977**
fu	**100**	**1**	**20**	**100**	**15**	**2016**
fu	**100**	**1**	**20**	**200**	**15**	**2259**
fu	100	1	50	10	18	733
fu	100	1	50	20	16	841
fu	**100**	**1**	**50**	**50**	**15**	**1660**
fu	**100**	**1**	**50**	**100**	**15**	**2837**
fu	**100**	**1**	**50**	**200**	**15**	**3644**
fu	100	1	100	10	17	1041
fu	**100**	**1**	**100**	**20**	**15**	**1156**
fu	**100**	**1**	**100**	**50**	**15**	**2479**
fu	**100**	**1**	**100**	**100**	**15**	**3284**
fu	**100**	**1**	**100**	**200**	**15**	**4979**
fu	**100**	**1**	**200**	**10**	**15**	**1376**
fu	**100**	**1**	**200**	**20**	**15**	**2203**
fu	**100**	**1**	**200**	**50**	**15**	**2939**
fu	**100**	**1**	**200**	**100**	**15**	**4052**
fu	**100**	**1**	**200**	**200**	**15**	**7058**

Fonte: Elaborado pelo autor.

Tabela 6 - Teste gerações e população *dataset* poly3b

Dataset	Altura	Rot.	Ger.	Pop.	Ocupação	Tempo/ms
poly3b	100	1	5	10	24	1363
poly3b	100	1	5	20	21	2519
poly3b	100	1	5	50	21	3930
poly3b	100	1	5	100	22	5931

poly3b	100	1	5	200	21	10613
poly3b	100	1	10	10	22	2091
poly3b	100	1	10	20	22	3624
poly3b	100	1	10	50	21	6251
poly3b	100	1	10	100	21	9514
poly3b	100	1	10	200	21	17303
poly3b	100	1	20	10	21	2259
poly3b	100	1	20	20	22	3674
poly3b	100	1	20	50	22	9263
poly3b	100	1	20	100	21	18422
poly3b	100	1	20	200	21	32894
poly3b	100	1	50	10	21	5491
poly3b	100	1	50	20	21	8272
poly3b	100	1	50	50	21	17911
poly3b	100	1	50	100	21	25591
poly3b	100	1	50	200	21	50805
poly3b	100	1	100	10	22	6136
poly3b	100	1	100	20	22	11398
poly3b	100	1	100	50	21	24852
poly3b	100	1	100	100	21	44771
poly3b	100	1	100	200	21	83965
poly3b	**100**	**1**	**200**	**10**	**20**	**16547**
poly3b	100	1	200	20	21	19098
poly3b	**100**	**1**	**200**	**50**	**20**	**56292**
poly3b	**100**	**1**	**200**	**100**	**20**	**120636**
poly3b	**100**	**1**	**200**	**200**	**20**	**199594**

Fonte: Elaborado pelo autor.

APÊNDICE C – Tabelas de resultados do Teste 3

Este apêndice apresenta (Tabela 7 e Tabela 8) a variação do fator de Crossover em (0,1), (0,3), (0,35), (0,5), (0,7), (0,9). E variação do fator de Mutação de (0,1), (0,2), (0,3), (0,5), (0,7) e (0,9).

Tabela 7 - Variação fator de Crossover

Dataset	Ocupação	Fator		Dataset	Ocupação	Fator
fu	14	0,1		fu	15	0,5
poly1a	13	0,1		poly1a	13	0,5
poly2b	16	0,1		poly2b	16	0,5
poly3b	21	0,1		poly3b	20	0,5
poly4b	25	0,1		poly4b	25	0,5
fu	14	0,3		fu	14	0,7
poly1a	13	0,3		poly1a	13	0,7
poly2b	16	0,3		poly2b	16	0,7
poly3b	21	0,3		poly3b	21	0,7
poly4b	26	0,3		poly4b	25	0,7
fu	**14**	**0,35**		fu	14	0,9
poly1a	**13**	**0,35**		poly1a	13	0,9
poly2b	**16**	**0,35**		poly2b	16	0,9
poly3b	**20**	**0,35**		poly3b	20	0,9
poly4b	**25**	**0,35**		poly4b	25	0,9

Fonte: Elaborado pelo autor.

Tabela 8 - Variação fator de Mutação

Dataset	Ocupação	Fator		Dataset	Ocupação	Fator
fu	**14**	**0,1**		fu	15	0,5
poly1a	**13**	**0,1**		**poly1a**	**13**	**0,5**
poly2b	16	0,1		poly2b	16	0,5
poly3b	21	0,1		poly3b	21	0,5
poly4b	24	0,1		poly4b	25	0,5
fu	15	0,2		**fu**	**14**	**0,7**
poly1a	**13**	**0,2**		**poly1a**	**13**	**0,7**
poly2b	**16**	**0,2**		poly2b	16	0,7
poly3b	20	0,2		poly3b	21	0,7
poly4b	25	0,2		poly4b	25	0,7
fu	15	0,3		fu	15	0,9
poly1a	**13**	**0,3**		**poly1a**	**13**	**0,9**
poly2b	**15**	**0,3**		poly2b	16	0,9
poly3b	21	0,3		poly3b	21	0,9
poly4b	25	0,3		poly4b	26	0,9

Fonte: Elaborado pelo autor.

APÊNDICE D – Tabelas de resultados do Teste 4

Este apêndice apresenta (Tabela 9) a comparação do Algoritmo Genético com variação de tamanho de população de 50, 100 e 200. Em comparação com o algoritmo de Hill Climbing e Tabu Search implementados por Brandt (2011). Todos os algoritmos foram executados com tempos de 1, 2, e 5 minutos.

Tabela 9 - Comparação Genético, Hill Climbing e Tabu Search

Dataset	Ocupação	Tempo	Algorit.		Dataset	Ocupação	Tempo	Algorit.
fu	**14,0**	**1**	**G_p50**		poly2b	16,0	5	G_p200
fu	**14,0**	**2**	**G_p50**		**poly2b**	**15,0**	**1**	**hill**
fu	**14,0**	**5**	**G_p50**		**poly2b**	**15,0**	**2**	**hill**
fu	**14,0**	**1**	**G_p100**		**poly2b**	**15,0**	**5**	**hill**
fu	**14,0**	**2**	**G_p100**		**poly2b**	**15,0**	**1**	**tabu**
fu	**14,0**	**5**	**G_p100**		**poly2b**	**15,0**	**2**	**tabu**
fu	15,0	1	G_p200		**poly2b**	**15,0**	**5**	**tabu**
fu	15,0	2	G_p200		poly3b	21,0	1	G_p50
fu	15,0	5	G_p200		poly3b	20,0	2	G_p50
fu	**14,0**	**1**	**hill**		poly3b	21,0	5	G_p50
fu	15,0	2	hill		poly3b	20,0	1	G_p100
fu	15,0	5	hill		poly3b	20,0	2	G_p100
fu	15,0	1	tabu		poly3b	20,0	5	G_p100
fu	**14,0**	**2**	**tabu**		poly3b	20,0	1	G_p200
fu	**14,0**	**5**	**tabu**		poly3b	20,0	2	G_p200
poly1a	**13,0**	**1**	**G_p50**		poly3b	20,0	5	G_p200
poly1a	**13,0**	**2**	**G_p50**		poly3b	20,0	1	hill
poly1a	**13,0**	**5**	**G_p50**		**poly3b**	**19,0**	**2**	**hill**
poly1a	**13,0**	**1**	**G_p100**		**poly3b**	**19,0**	**5**	**hill**
poly1a	**13,0**	**2**	**G_p100**		poly3b	20,0	1	tabu
poly1a	**13,0**	**5**	**G_p100**		poly3b	20,0	2	tabu
poly1a	**13,0**	**1**	**G_p200**		**poly3b**	**19,0**	**5**	**tabu**
poly1a	**13,0**	**2**	**G_p200**		poly4b	26,0	1	G_p50
poly1a	**13,0**	**5**	**G_p200**		poly4b	25,0	2	G_p50
poly1a	**13,0**	**1**	**hill**		poly4b	25,0	5	G_p50
poly1a	**13,0**	**2**	**hill**		poly4b	25,0	1	G_p100
poly1a	**13,0**	**5**	**hill**		poly4b	25,0	2	G_p100
poly1a	**13,0**	**1**	**tabu**		poly4b	25,0	5	G_p100
poly1a	**13,0**	**2**	**tabu**		poly4b	25,0	1	G_p200
poly1a	**13,0**	**5**	**tabu**		poly4b	26,0	2	G_p200
poly2b	17,0	1	G_p50		poly4b	25,0	5	G_p200
poly2b	16,0	2	G_p50		poly4b	24,0	1	hill
poly2b	**15,0**	**5**	**G_p50**		**poly4b**	**23,0**	**2**	**hill**
poly2b	16,0	1	G_p100		poly4b	24,0	5	hill
poly2b	16,0	2	G_p100		poly4b	24,0	1	tabu

poly2b	**15,0**	**5**	**G_p100**		poly4b	24,0	2	tabu
poly2b	16,0	1	G_p200		poly4b	24,0	5	tabu
poly2b	16,0	2	G_p200					

Fonte: Elaborado pelo autor.

APÊNDICE E – Tabelas de resultados do Teste 5

Este apêndice apresenta (Tabela 10) a comparação do Algoritmo Genético com variação de tamanho de população 200 indivíduos. Em comparação com o algoritmo de Hill Climbing e Tabu Search implementados por Brandt (2011). Todos os algoritmos foram executados com variação de altura de 25, 50, 75 e 100.

Tabela 10 - Comparação da variação de altura para o Genético, Hill Climbing e Tabu Search.

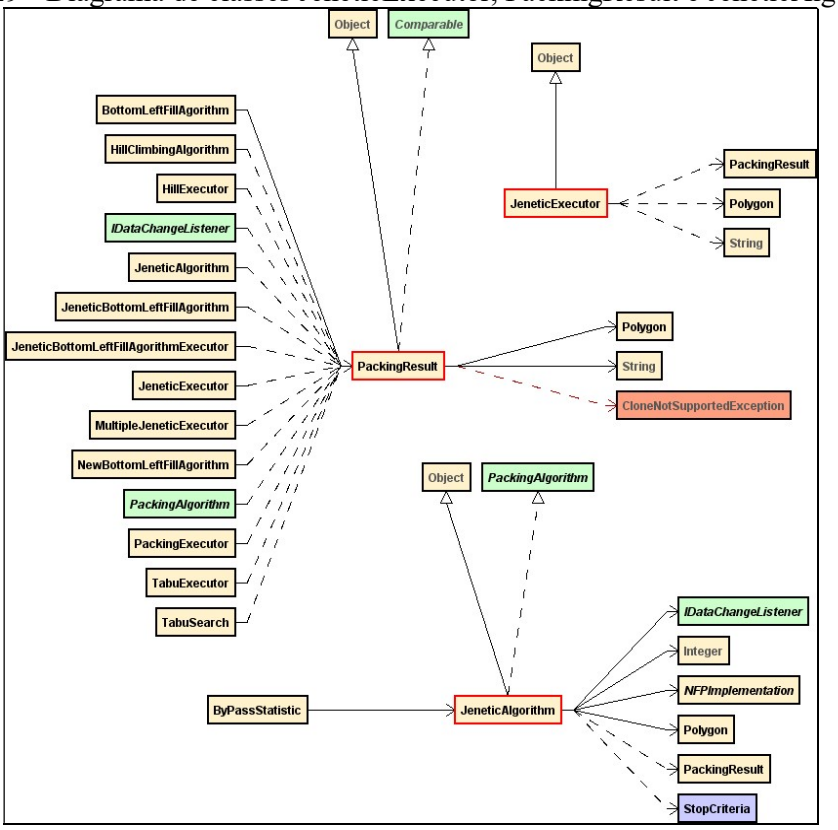
Dataset	Altura	Ocupação	Algoritmo		Dataset	Altura	Ocupação	Algoritmo
fu	**25**	**52**	**genético**		poly2b	75	19,75	genético
fu	25	54	hill		**poly2b**	**75**	**19,06**	**hill**
fu	25	53	tabu		poly2b	75	19,58	tabu
fu	**50**	**27,5**	**genético**		**poly2b**	**100**	**15**	**genético**
fu	50	28	hill		**poly2b**	**100**	**15**	**hill**
fu	**50**	**27,5**	**tabu**		poly2b	100	16	tabu
fu	**75**	**20,5**	**genético**		poly3b	25	73,75	genético
fu	**75**	**20,5**	**hill**		**poly3b**	**25**	**70**	**hill**
fu	**75**	**20,5**	**tabu**		poly3b	25	73,25	tabu
fu	**100**	**14**	**genético**		poly3b	50	37,5	genético
fu	100	15	hill		**poly3b**	**50**	**36**	**hill**
fu	100	15	tabu		poly3b	50	36,5	tabu
poly1a	25	25,75	genético		poly3b	75	26,25	genético
poly1a	**25**	**25**	**hill**		**poly3b**	**75**	**24,25**	**hill**
poly1a	25	25,25	tabu		poly3b	75	24,75	tabu
poly1a	**50**	**13,5**	**genético**		poly3b	100	20	genético
poly1a	50	14	hill		**poly3b**	**100**	**19**	**hill**
poly1a	50	14	tabu		poly3b	100	21	tabu
poly1a	**75**	**13**	**genético**		poly4b	25	93,5	genético
poly1a	**75**	**13**	**hill**		poly4b	25	93,25	hill
poly1a	**75**	**13**	**tabu**		**poly4b**	**25**	**91**	**tabu**
poly1a	**100**	**13**	**genético**		poly4b	50	47,5	genético
poly1a	**100**	**13**	**hill**		**poly4b**	**50**	**45**	**hill**
poly1a	**100**	**13**	**tabu**		poly4b	50	45,5	tabu
poly2b	25	54,5	genético		poly4b	75	33	genético
poly2b	**25**	**53,5**	**hill**		**poly4b**	**75**	**30,75**	**hill**
poly2b	25	55	tabu		poly4b	75	31,25	tabu
poly2b	50	28	genético		poly4b	100	25	genético
poly2b	**50**	**27,5**	**hill**		**poly4b**	**100**	**24**	**hill**
poly2b	50	28,5	tabu		**poly4b**	**100**	**24**	**tabu**

Fonte: Elaborado pelo autor.

APÊNDICE F – Diagrama de classes JeneticExecutor e PackingResult, JeneticAlgorithm, BottomLeftFillAlgorithm e JNFP

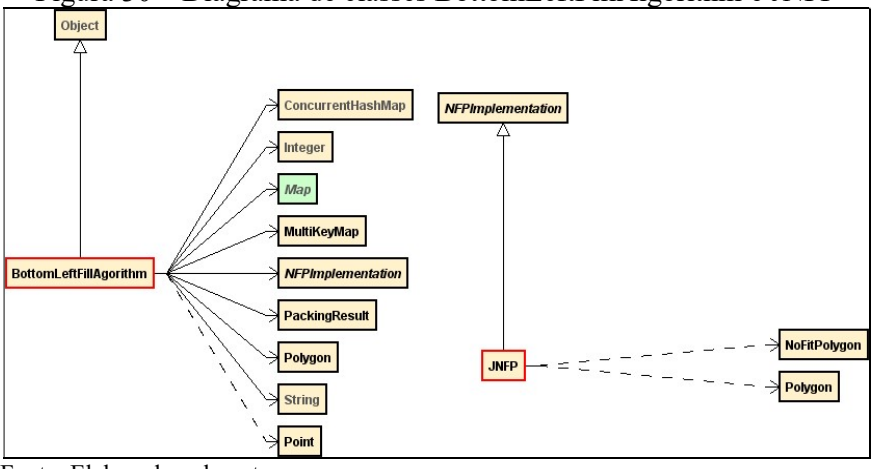
Neste apêndice são apresentadas (Figura 29 e Figura 30) as ligações das classes `JeneticExecutor`, `PackingRersult`, `JeneticAlgorithm`, `BottonLeftFillAlgorithm` e `JNFP`.

Figura 29 - Diagrama de classes JeneticExecutor, PackingResult e JeneticAlgorithm



Fonte: Elaborado pelo autor

Figura 30 – Diagrama de classes BottomLeftFillAlgorithm e JNFP

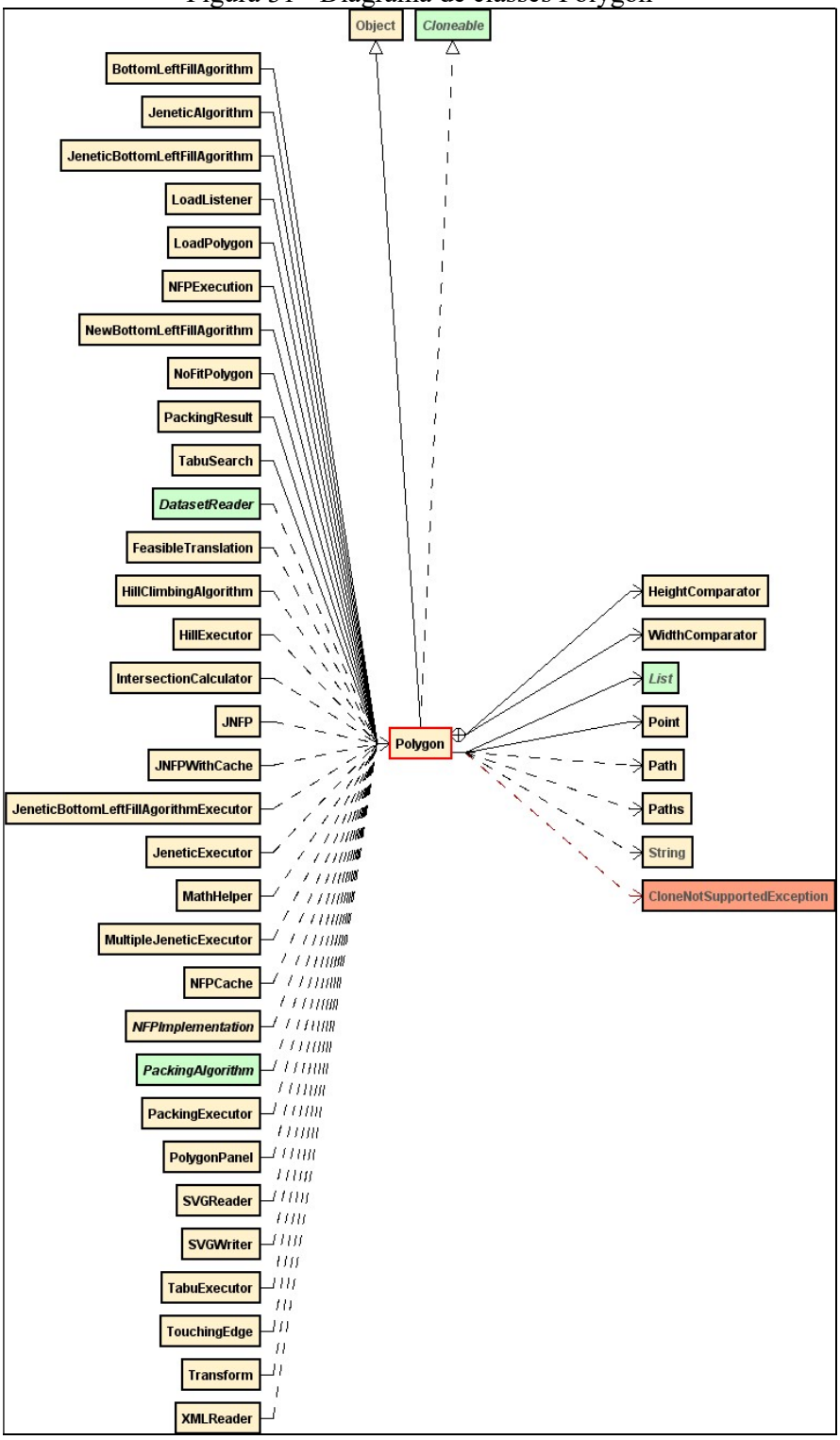


Fonte: Elaborado pelo autor.

APÊNDICE G – Diagrama de classes Polygon

Este apêndice apresenta (Figura 31) as ligações da classe `Polygon`.

Figura 31 - Diagrama de classes Polygon



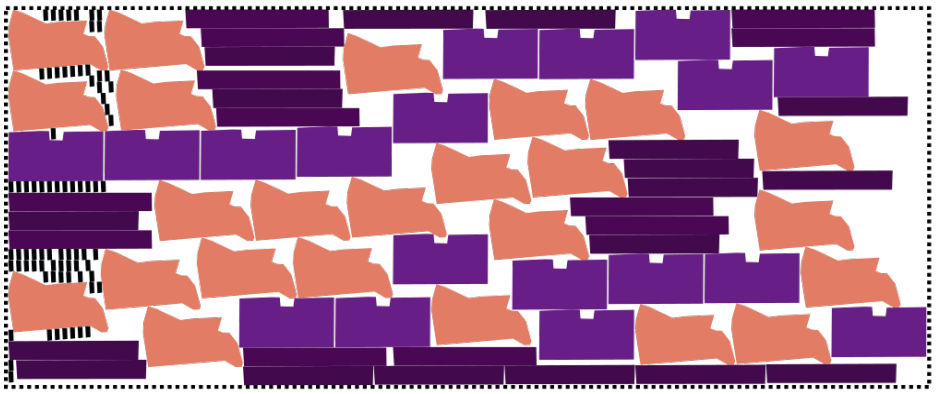
Fonte: Elaborado pelo autor.

APÊNDICE H – Comparação JPacking e SVGNest *dataset* alma1

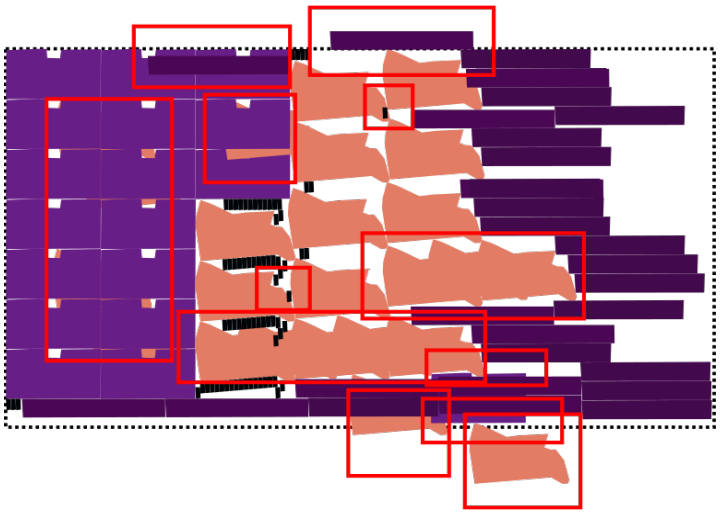
Este Apêndice apresenta (Quadro 17) a comparação entre o empacotamento realizado pelo programa JPacking e o programa SVGNest desenvolvido por Qiao (2017) para o *dataset* `alma1`. Nos quadros vermelhos estão destacados erros de empacotamento.

Quadro 17 - Resultado empacotamento *dataset* alma1

JPacking



SVGNest



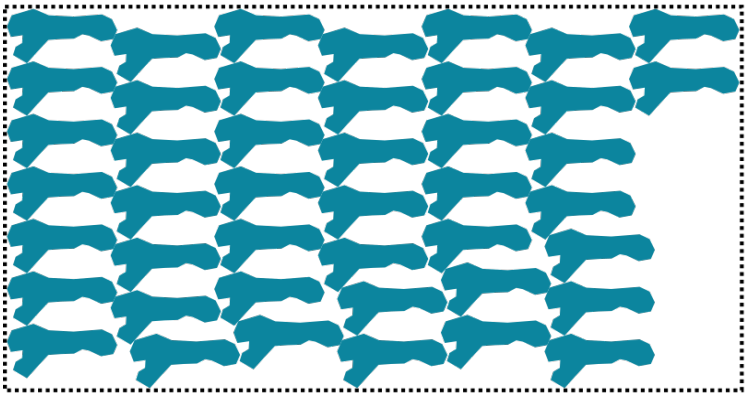
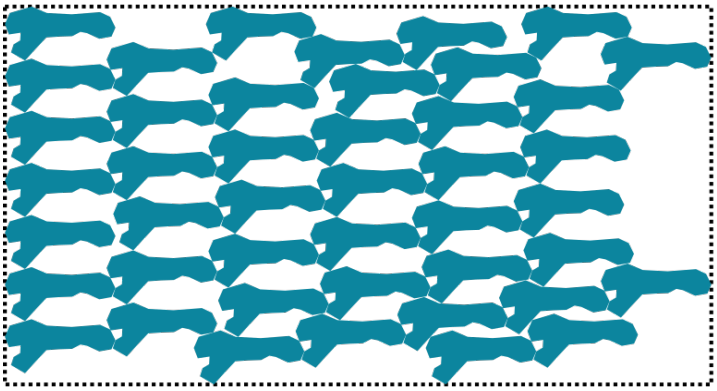
Resultados do empacotamento	
JPacking	241
SVGNest	185 com erros.

Fonte: Elaborado pelo autor.

APÊNDICE I – Comparação JPacking e SVGNest *dataset* alma2

Este Apêndice apresenta (Quadro 18) a comparação entre o empacotamento realizado pelo programa JPacking e o programa SVGNest desenvolvido por Qiao (2017) para o *dataset* `alma2`.

Quadro 18 - Resultado empacotamento *dataset* alma2

JPacking

SVGNest


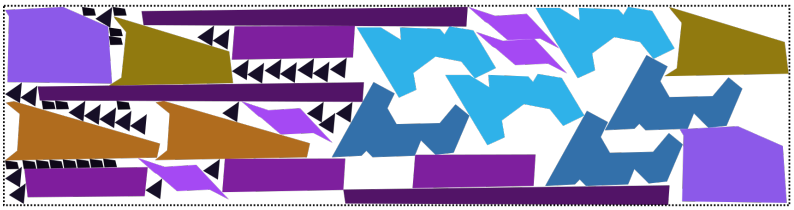
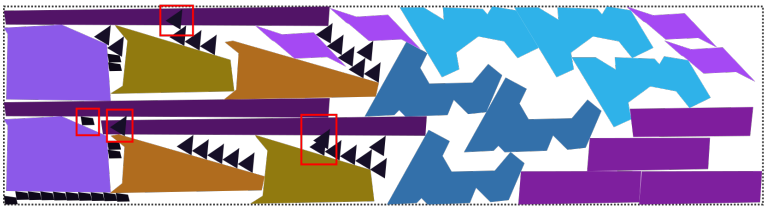
Resultados do empacotamento	
JPacking	192
SVGNest	184

Fonte: Elaborado pelo autor.

APÊNDICE J – Comparação JPacking e SVGNest *dataset* alma3

Este Apêndice apresenta (Quadro 19) a comparação entre o empacotamento realizado pelo programa JPacking e o programa SVGNest desenvolvido por Qiao (2017) para o *dataset* `alma3`. Nos quadros vermelhos estão destacados erros de empacotamento.

Quadro 19 - Resultado empacotamento *dataset* alma3

JPacking

SVGNest


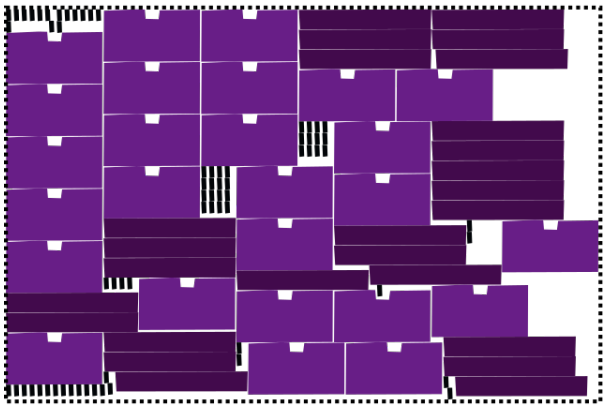
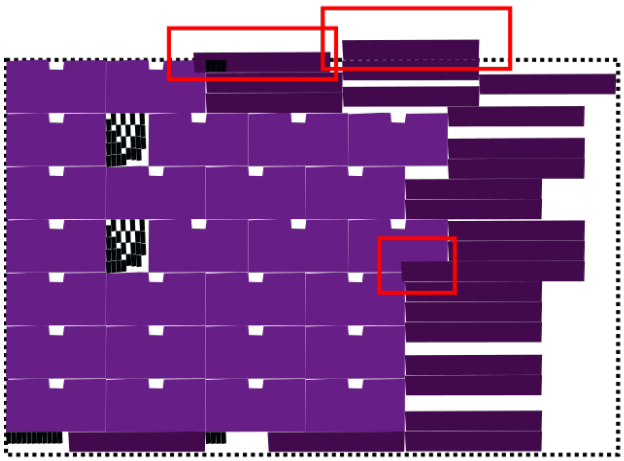
Resultados do empacotamento	
JPacking	409
SVGNest	396 com erros.

Fonte: Elaborado pelo autor.

APÊNDICE K – Comparação JPacking e SVGNest *dataset* alma4

Este Apêndice apresenta (Quadro 20) a comparação entre o empacotamento realizado pelo programa JPacking e o programa SVGNest desenvolvido por Qiao (2017) para o *dataset* `alma4`. Nos quadros vermelhos estão destacados erros de empacotamento.

Quadro 20 - Resultado empacotamento *dataset* alma4

JPacking

SVGNest


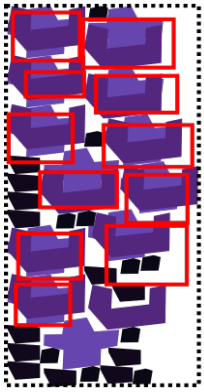
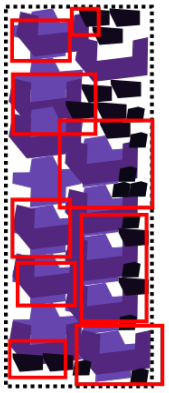
Resultados do empacotamento	
JPacking	153
SVGNest	152 com erros.

Fonte: Elaborado pelo autor.

APÊNDICE L – Comparação JPacking e SVGNest *dataset* metal1

Este Apêndice apresenta (Quadro 18) a comparação entre o empacotamento realizado pelo programa JPacking e o programa SVGNest desenvolvido por Qiao (2017) para o *dataset* `metal1`. Nos quadros vermelhos estão destacados erros de empacotamento.

Quadro 21 - Resultado empacotamento *dataset* metal1

JPacking

SVGNest


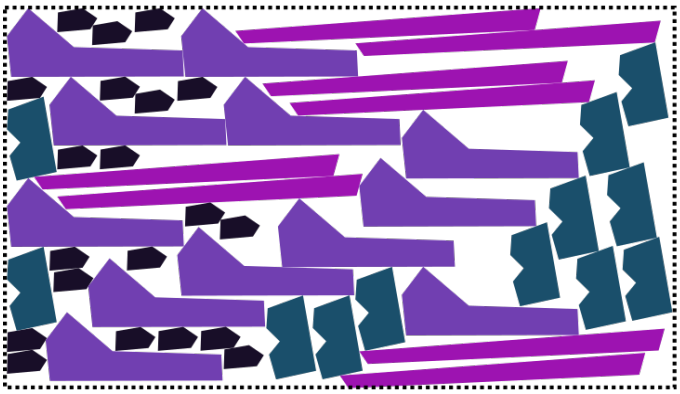
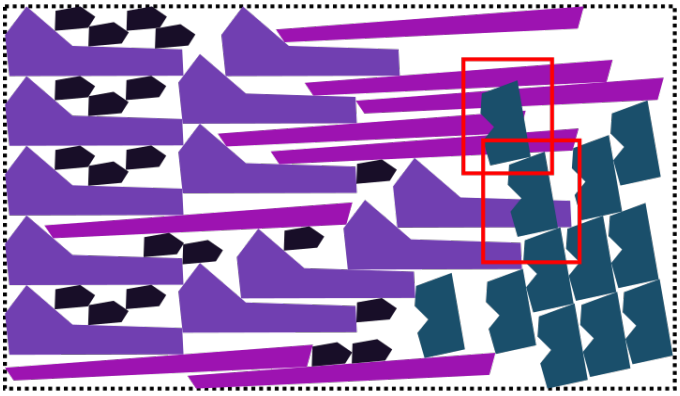
Resultados do empacotamento	
JPacking	48 com erros.
SVGNest	39 com erros.

Fonte: Elaborado pelo autor.

APÊNDICE M – Comparação JPacking e SVGNest *dataset* roupas1

Este Apêndice apresenta (Quadro 22) a comparação entre o empacotamento realizado pelo programa JPacking e o programa SVGNest desenvolvido por Qiao (2017) para o *dataset* `roupas1`. Nos quadros vermelhos estão destacados erros de empacotamento.

Quadro 22 - Resultado empacotamento *dataset* roupas1

JPacking

SVGNest


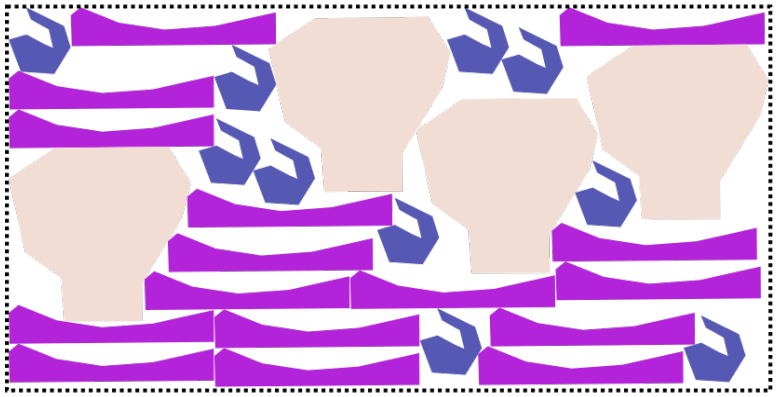
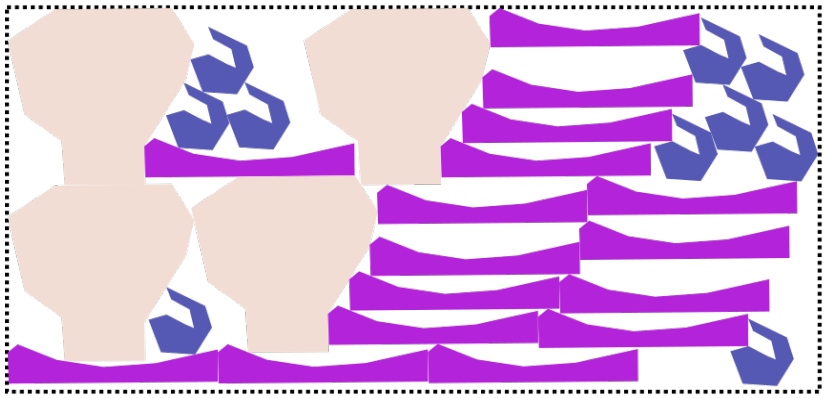
Resultados do empacotamento	
JPacking	173
SVGNest	172 com erros.

Fonte: Elaborado pelo autor.

APÊNDICE N – Comparação JPacking e SVGNest *dataset* roupas2

Este Apêndice apresenta (Quadro 23) a comparação entre o empacotamento realizado pelo programa JPacking e o programa SVGNest desenvolvido por Qiao (2017) para o *dataset* `roupas2`.

Quadro 23 - Resultado empacotamento *dataset* roupas2

JPacking

SVGNest


Resultados do empacotamento	
JPacking	197
SVGNest	205

Fonte: Elaborado pelo autor.