WESLEY HENRIQUE BATISTA NUNES

ALGORITMOS HEURÍSTICOS PARA O PROBLEMA DE *NESTING* COM ROTAÇÕES LIVRES

LAVRAS – MG

2021

WESLEY HENRIQUE BATISTA NUNES

ALGORITMOS HEURÍSTICOS PARA O PROBLEMA DE *NESTING* COM ROTAÇÕES LIVRES

Defesa apresentada à Universidade Federal de Lavras, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, área de Inteligência Computacional e Processamento Gráfico, para a obtenção do título de Mestre.

Prof. Dr. Mayron César de Oliveira Moreira
Orientador

Prof. Dra. Marina Andretta
Coorientadora

LAVRAS – MG
2021

Ficha catalográfica elaborada pelo Sistema de Geração de Ficha Catalográfica da Biblioteca Universitária da UFLA, com dados informados pelo(a) próprio(a) autor(a).

Nunes, Wesley Batista.
Algoritmos Heurísticos para o Problema de *Nesting* com Rotações Livres / Nunes, Wesley Batista. - 2021.
90 p. : il.

Dissertação(mestrado acadêmico) - Universidade Federal de Lavras, 2021.
Orientador: Prof. Dr. Mayron César de Oliveira Moreira.
Bibliografia.

1.Nesting. 2. Rotações Livres. 3. Heurísticas. I. Moreira, Mayron César de Oliveira. II. Andretta, Marina. III. Título.

WESLEY HENRIQUE BATISTA NUNES

ALGORITMOS HEURÍSTICOS PARA O PROBLEMA DE *NESTING* COM ROTAÇÕES LIVRES

Defesa apresentada à Universidade Federal de Lavras, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, área de Inteligência Computacional e Processamento Gráfico, para a obtenção do título de Mestre.

APROVADA em 30 de Março de 2021.

Prof. DSc. Mayron César de Oliveira Moreira	UFLA
Prof. DSc. Marina Andretta	USP
Prof. DSc. Maria Antónia Caravilla	UPORTO
Prof. DSc. Dilson Lucas Pereira	UFLA

Prof. Dr. Mayron César de Oliveira Moreira
Orientador

Prof. Dra. Marina Andretta
Co-Orientadora

LAVRAS – MG
2021

AGRADECIMENTOS

Primeiramente gostaria de agradecer a Deus por me dar saúde e entendimento para conseguir concluir esta etapa muito importante da minha vida. Gostaria de agradecer meus pais por me apoiarem desde o começo para eu conseguir chegar onde cheguei hoje. Gostaria de agradecer minha namorada Ana Flávia que sempre esteve comigo desde o inicio da graduação. Gostaria de agradecer todos meus amigos que tive no IFMG que carrego comigo até hoje e agradecer os novos amigos que fiz no LOOP e na UFLA. Agradeço imensamente ao meu orientador Mayron e minha co-orientadora Marina por me acompanhar durante essa jornada, sempre me apoiando, me dando conselhos e me ajudando com as dificuldades encontradas. Agradeço também à UFLA pela estrutura fornecida pelos alunos e pelos excelentes profissionais. Agradeço a todos os professores que participaram da minha formação, desde a escola até hoje. Finalmente, gostaria de agradecer à CAPES pelo apoio financeiro, pois sem isso não seria possível a finalização deste trabalho.

“Do or do not, there is no try".
(Yoda)

RESUMO

O problema de corte e empacotamento de peças irregulares, ou problema de *Nesting*, visa encontrar a melhor posição de peças dentro de uma faixa, tentando minimizar a altura utilizada (e, consequentemente, o espaço utilizado pelas peças). É grande a relevância este problema, visto que o corte e o empacotamento de peças ocorrem no contexto de indústrias de móveis, têxteis e de calçados, entre outras. O problema abordado consiste em um âmbito bidimensional com peças convexas e não convexas, com rotação livre. Nosso objetivo é conseguir responder à seguinte pergunta de pesquisa: "Existe alguma forma de melhorar a eficiência da resolução do problema de *Nesting* bidimensional com rotações livres usando heurísticas?". Buscando responder a esta pergunta, foram implementadas inicialmente cinco regras de posicionamento, cinco regras de ordenação e duas regras de rotação. Foram propostas vinte e cinco heurísticas construtivas usando, cada uma, combinações de uma regra de posicionamento, uma de ordenação e ambas de rotação. Os resultados computacionais feitos com doze instâncias da literatura indicaram que, dentre estas, as que usam duas regras de posicionamento (chamadas α e β) se destacaram. Em particular, uma delas obteve resultados melhores (apesar de similares às demais). Também foi implementada uma heurística chamada de Heurística de Múltiplos Posicionamentos e um Algoritmo Genético, que utilizam a combinação dessas duas regras de posicionamento que se destacaram. Após uma comparação dentre os métodos propostos, foi possível perceber que a Heurística de Múltiplos Posicionamentos obteve resultados melhores em oito das doze instâncias testadas. Depois de definido qual nosso melhor método, comparamos o mesmo com os resultados presentes na literatura. Os resultados da abordagem foram promissores, superando em três das oito instâncias da literatura.

Palavras-chave: *Nesting*. Rotação Livre. Heurísticas. Algoritmo Genético.

ABSTRACT

The problem of cutting and packing irregular pieces, or Nesting's problem, aims to find the best position of pieces within a range, trying to minimize the height used (and, consequently, the space used by the pieces). This problem is of great relevance since the cutting and packing of pieces occur in the furniture, textiles, and footwear industries. The problem addressed is two-dimensional with convex and non-convex pieces, with free rotation. Our goal is to be able to answer the following research question: "Is there any way to improve the efficiency of solving the problem of two-dimensional Nesting with free rotations, using heuristics?". Five positioning rules, five sorting rules, and two rotation rules were initially implemented to answer this question, and twenty-five constructive heuristics were proposed, each using combinations of a positioning rule, a sorting rule, and both rotation rules. Computational experiments made with twelve instances of the literature indicated that, among these heuristics, those that use two positioning rules (called α and β) stood out. In particular, one of them obtained better results (although similar to the others). A heuristic called the Multiple Positioning Heuristic, and a Genetic Algorithm were also implemented, which use the combination of these two position rules that stood out. After comparing the proposed methods, it was possible to notice that the Multiple Positioning Heuristic obtained better results in eight of the twelve tested instances. After defining our best method, we compare it with the results found in the literature. Our approach's results were promising, surpassing the best results in three of the eight instances in the literature.

Keywords: Nesting, Continuous Rotation, Heuristics.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

O problema de corte e empacotamento de peças irregulares, conhecido como *Nesting*, tem como objetivo encontrar um leiaute para uma quantidade finita de peças (itens) convexas e não convexas, minimizando o desperdício de espaço em uma placa. O gerenciamento eficaz de matéria-prima é uma preocupação constante de decisores em contextos como indústrias de móveis, de papéis, metalomecânica, têxteis e de couro. Assim, além do aumento da lucratividade da produção, tais soluções impactam na sustentabilidade do negócio. O *Nesting* também é conhecido no trabalho de Wäscher, Haußner e Schumann (2007) por *irregular cutting and packing problem* (problema de corte e empacotamento irregular).

Neste projeto, iremos tratar de uma variante do problema de *Nesting*, o *Irregular Strip Packing Problem* (ISPP) (WÄSCHER; HAUSSNER; SCHUMANN, 2007), um problema em que se tem a largura da placa fixa, e a altura a ser minimizada (neste caso, a placa pode ser chamada de faixa). As peças que serão alocadas na faixa terão duas dimensões e poderão ser rotacionadas livremente. A faixa em que os itens serão arranjados é retangular e sua dimensão fixa é conhecida previamente.

O problema de *Nesting* contém algumas restrições que devem ser respeitadas para a obtenção de uma solução viável. Tais restrições são:

- restrição de contenção, que garante todas peças dentro da faixa;
- todas as peças devem ser alocadas;
- nenhuma peça deve se sobrepor à outra.

A forma de representação das peças e o mecanismo de identificar a sobreposição entre elas na faixa constituem dois critérios elementares de escolha para a eficiência de um algoritmo. Em especial, considerando a sobreposição de itens, alguns autores propuseram métodos geométricos, como métodos de rasterização, trigonometria direta, *No-Fit Polygon*, função ϕ (BENNELL; OLIVEIRA, 2008), *No-Fit Raster* (NFR) (MUNDIM; ANDRETTA; QUEIROZ, 2017) e Retas Separadoras (PERALTA; ANDRETTA; OLIVEIRA, 2017).

Diversas técnicas para a resolução do problema de corte e empacotamento já foram propostas na literatura, porém são poucos os trabalhos que levam em consideração a rotação livre das peças. Quando considerada, o modelo de otimização para o problema se torna não-linear. Alguns trabalhos que levam essa característica das peças são os de Peralta, Andretta e Oliveira (2017), Liao et al. (2016) e Stoyan, Pankratov e Romanova (2016). Em especial, focamos o nosso estudo no contexto abordado por Peralta, Andretta e Oliveira (2017), que aborda o problema de *Nesting* com rotações livres por meio de um modelo com retas separadoras e uma formulação com parábolas separadoras.

1.1 Objetivos

A pergunta que este trabalho pretende responder pode ser enunciada como:

Existe alguma forma de melhorar a eficiência da resolução o problema de Nesting bidimensional com rotações livres utilizando heurísticas?

Apresentamos na sequência: (i) o objetivo geral, (ii) e os objetivos específicos a respeito da dissertação.

1.1.1 Objetivos gerais

Neste trabalho, vamos propor algoritmos heurísticos para melhorar o estado da arte da resolução do problema de corte e empacotamento de peças irregulares com rotação livre, com o objetivo de minimizar o desperdício gerado pela alocação das peças. Heurísticas construtivas e meta-heurísticas serão propostas para a resolução do problema, visto que até a criação deste trabalho, não se tem estudos de tais técnicas aplicadas ao problema em questão.

1.1.2 Objetivos específicos

Os objetivos específicos deste trabalho são:

- fazer uma revisão de literatura de trabalhos que abordam o problema;
- criar um *software* para auxiliar na visualização das soluções;
- propor heurísticas e meta-heurística para a resolução do problema;
- propor heurísticas específicas para a rotação das peças;

1.2 Estrutura do texto

O restante do texto está organizado da seguinte forma: o Capítulo 2 apresenta a formalização do problema de corte e empacotamento com itens irregulares (*Nesting*). No Capítulo 3, são apresentados alguns métodos para a detecção da sobreposição de itens. Além disso, descrevemos trabalhos que consideram o problema com e sem rotações livres das peças. O Capítulo 4 mostra quais foram os algoritmos utilizados na resolução do problema. No Capítulo 5, são apresentados os resultados finais das heurísticas propostas para o problema de *Nesting* com rotações livres. A partir dos resultados obtidos, apresentamos, no Capítulo 6, nossas considerações finais do trabalho e as propostas para a continuidade da pesquisa. Por fim no Apêndice A são apresentadas algumas imagens de soluções referentes ao Capítulo 5.

2 DEFINIÇÃO DO PROBLEMA

O problema de corte e empacotamento com objetos irregulares (*Nesting*) é definido a seguir. Considere uma faixa retangular no plano cartesiano com dimensões $L \times W$, em que L (x eixo) é a largura faixa e W (y eixo) é a altura que será minimizada. O canto esquerdo inferior da faixa é posicionado em $(0,0)$. A representação e notação das peças que serão utilizadas são baseadas no trabalho de Peralta, Andretta e Oliveira (2017).

As peças que compõem a solução do problema serão definidas como polígonos bidimensionais contidos no plano cartesiano. Considere um conjunto P de n polígonos. Um polígono $p \in P$ é descrito por meio um conjunto de vértices ordenados V_p, em que v_j é um vértice de V_p no espaço bidimensional. O conjunto V_p é definido por:

$$V_p = \{(x_p^1, y_p^1), (x_p^2, y_p^2), ..., (x_p^{|V_p|}, y_p^{|V_p|})\}. \tag{2.1}$$

Quando um polígono p é rotacionado, os novos pontos (x,y) que definem cada vértice $v_j = (\hat{x}_p^j, \hat{y}_p^j)$, para $j = 1, ..., |V_p|$, são dados por:

$$\hat{x}_p^j = x_p^j cos\theta_p - y_p^j sin\theta_p, \hat{y}_p^j = x_p^j sin\theta_p + y_p^j cos\theta_p, \tag{2.2}$$

em que θ_p representa o ângulo de rotação da peça e $(\hat{x}_p^j, \hat{y}_p^j)$ consiste no valor modificado do vértice j, pertencente ao polígono p.

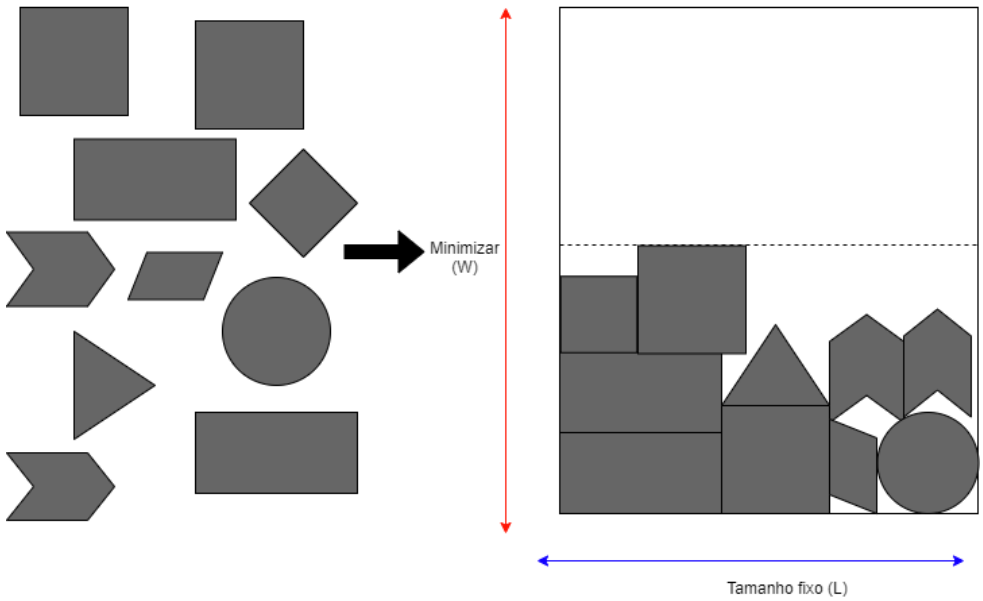
O problema de *Nesting* requer que todas as peças estejam dentro da faixa e que elas não se sobreponham. Com isso, temos as seguintes restrições:

$$0 \leq \hat{x}_p^j \leq L, \qquad \forall p \in P,\ 1 \leq j \leq |V_p|, \tag{2.3}$$

$$Overlap(p, p') = FALSE, \qquad \forall p, p' \in P,\ p \neq p'. \tag{2.4}$$

As restrições (2.3) garantem que o conjunto das coordenadas x dos vértices de $p \in P$ estejam dentro do limite da faixa. As restrições (2.4) estabelecem que o polígono p não deverá se sobrepor ao polígono p', para todo par de polígonos contidos em P. A função *Overlap(p, p') = FALSE* consiste em uma abstração da restrição em questão, retornando *FALSE* no caso dos polígonos p e p' não se sobreporem. A Figura 2.1 mostra um exemplo de solução factível para o problema de *Nesting*.

Figura 2.1 – Instância de exemplo do problema de *Nesting*



Fonte: Do Autor (2021)

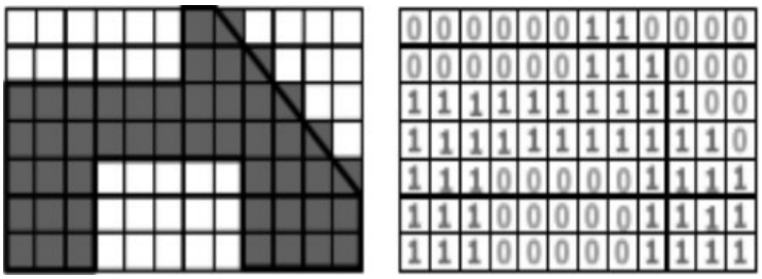
3 REVISÃO DA LITERATURA

Neste capítulo apresentamos, na Seção 3.1, como as peças são representadas na literatura e como os autores tratam a sobreposição de peças. Em seguida, na Seção 3.2, apresentamos alguns trabalhos da literatura que abordam o problema de *Nesting* sem rotações livres, já que esta variação do problema é a mais encontrada. Finalmente, na Seção 3.3, tratamos de trabalhos que consideram rotações livres das peças.

3.1 Representações das peças e como evitar sobreposição entre elas

No trabalho Oliveira e Ferreira (1993) é proposto um esquema de representação de peças utilizando matrizes com valores 0's e 1's. Quando uma célula da matriz contém o valor 1, significa que existe uma peça naquela região. Com esse tipo de representação, analisar se duas peças estão sobrepostas se torna uma tarefa mais fácil. Quando duas peças se sobrepõem, o valor das células de intersecção das mesmas se torna maior que 1. Tal representação de peças pode ser observada na Figura 3.1.

Figura 3.1 – Representação de peças em matrizes (0-1).



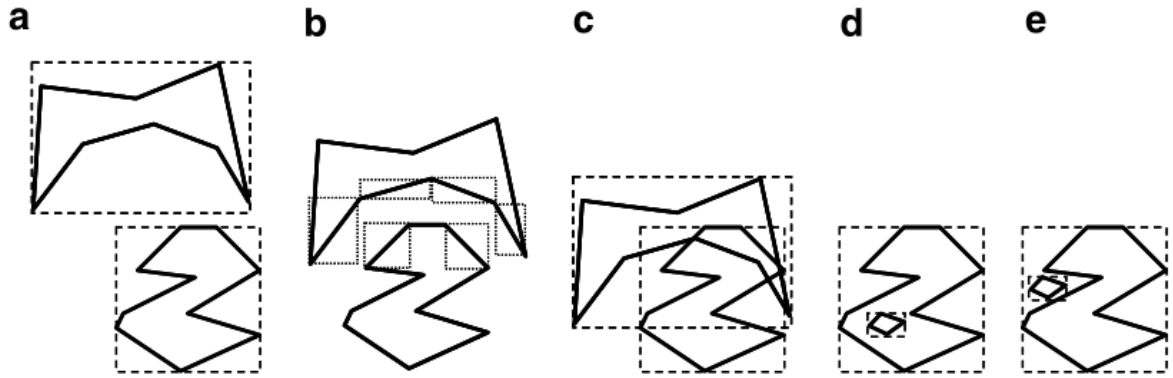
Fonte: Bennell e Oliveira (2008)

Outra representação utilizada consiste em transformar cada peça do problema em um polígono. Cada vértice da peça é representado por um ponto (x, y) no plano cartesiano, fazendo assim a peça se transformar em polígono. Para tal representação, os vértices são armazenados ordenados em uma estrutura de dados Peralta, Andretta e Oliveira (2017). Com a modelagem das peças através de polígonos, as peças podem ser melhor representadas e o uso de memória computacional é menor do que a representação utilizando matrizes. No entanto, a verificação de sua não sobreposição se torna uma tarefa mais difícil.

No trabalho de Bennell e Oliveira (2008), são apresentados alguns métodos para verificação de peças sobrepostas e é mostrada a dificuldade computacional de cada método. Para o caso em que as peças são representadas por polígonos, um método apresentado é denominado trigonometria direta, que consiste em trabalhar diretamente com os vértices e arestas dos polígonos para verificar a não sobreposição de dois polígonos. Uma maneira de fazer essa verificação é realizando os seguintes testes:

- Teste 1: Os retângulos circunscritos do polígono se sobrepõem?
  Não - Polígonos não se sobrepõem (Figura 3.2a).
  Sim - Aplique o Teste 2.
- Teste 2: Para cada par de arestas a_1 e b_1 de um polígono p_1, e a_2 e b_2 de um polígono p_2, o retângulo circunscrito de a_1 se sobrepõe aos de a_2 e b_2? O retângulo circunscrito de a_2 se sobrepõe aos de a_1 e b_1?
  Não para todas - Polígonos não se sobrepõem (Figura 3.2b).
  Sim - Para cada aresta aplique o Teste 3.
- Teste 3: Para cada par de arestas dos diferentes polígonos, existe alguma interseção?
  Não para todas - Aplique o Teste 4 (Figura 3.2c).
  Sim - Polígonos se sobrepõem.
- Teste 4: Para cada vértice do polígono, ele está dentro do outro polígono?
  Não - Polígonos não se sobrepõem (Figura 3.2e).
  Sim - Polígonos se sobrepõem (Figura 3.2d).

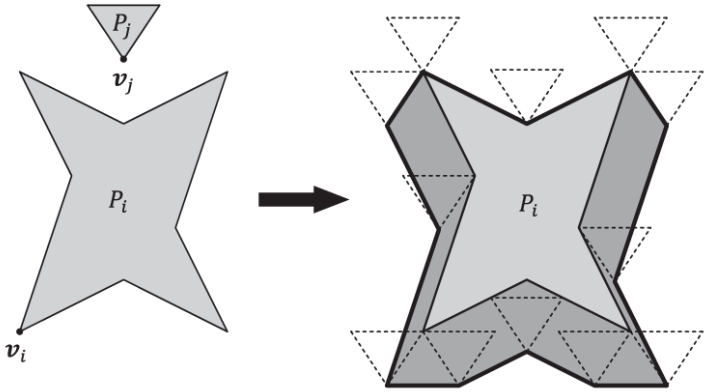
Figura 3.2 – Exemplos de posições relativas para dois polígonos.



Fonte: Bennell e Oliveira (2008)

O método mais utilizado na literatura para a verificação de sobreposição de peças, quando estas são representadas por polígonos, é o *No-Fit Polygon* (NFP). A essência do *NFP* é um polígono derivado da combinação de dois outros polígonos, em que o seu interior representa os pontos de sobreposição entre os itens (Figura 3.3). Depois de construído o NFP entre dois polígonos, a verificação de não sobreposição entre eles se resume a verificar se um ponto está dentro ou fora do NFP. A construção do NFP não é uma tarefa simples, mas ela pode ser feita em uma etapa de pré-processamento, o que faz com que a verificação da não sobreposição possa ser feita de maneira eficiente (para mais detalhes, veja Bennell e Oliveira (2008)).

Figura 3.3 – Representação de um *no-fit polygon*.

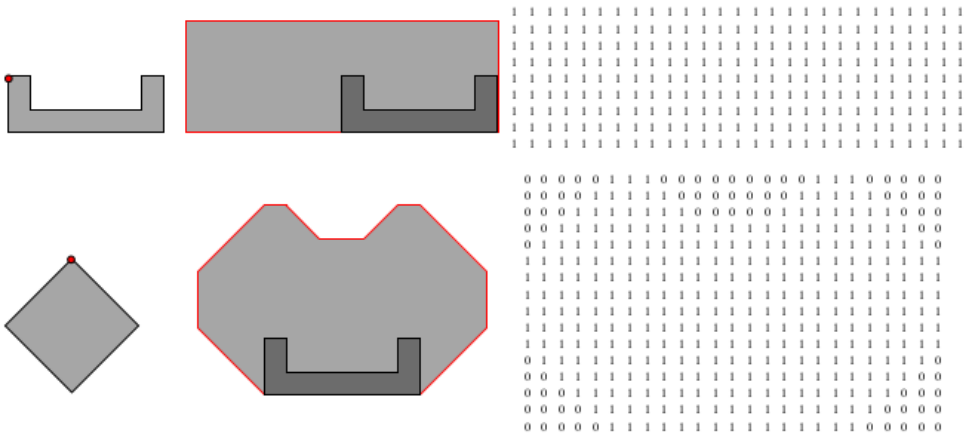


Fonte: Elkeran (2013)

O método baseado em funções *phi*, proposto por Stoyan et al. (2002), consiste em uma expressão matemática gerada a partir de duas peças, representadas por polígonos, círculos, arcos ou combinação destes. A expressão representa a sobreposição ou não das peças. Caso o valor da expressão seja maior que zero, as peças estão separadas; se o valor é igual a zero, as peças se tocam; caso contrário (valores negativos), estão sobrepostas. A desvantagem é a dificuldade em calcular essas funções de uma maneira automática.

Mundim (2017) propõe um método de verificação de não sobreposição chamado *No-Fit Raster*. Tal algoritmo possui semelhanças com o *No-Fit Polygon* padrão. O *No-Fit Raster* representa o NFP_{AB} (*No-Fit Polygon* entra a peça A e B) como uma malha *booleana*, com resolução g. Se $g = 1$, então o plano será discretizado em números inteiros. O valor de cada posição da malha é 1 se ela está dentro do polígono gerado pelo *NFP*, e 0 caso contrário (Figura 3.4).

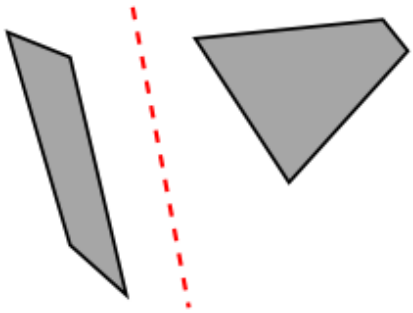
Figura 3.4 – Representação de um *no-fit polygon* e seu respectivo *no-fit raster*.



Fonte: Mundim (2017)

Em Peralta, Andretta e Oliveira (2017), foi proposto um método de separação utilizando retas e parábolas separadoras. Dizemos que dois polígonos convexos não se sobrepõem se os pontos de cada um deles estiverem em lados opostos da reta em questão. No caso de polígonos não convexos, as peças são separadas em polígonos convexos e é criada uma reta para cada parte. A representação das retas separadoras pode ser vista na Figura 3.5.

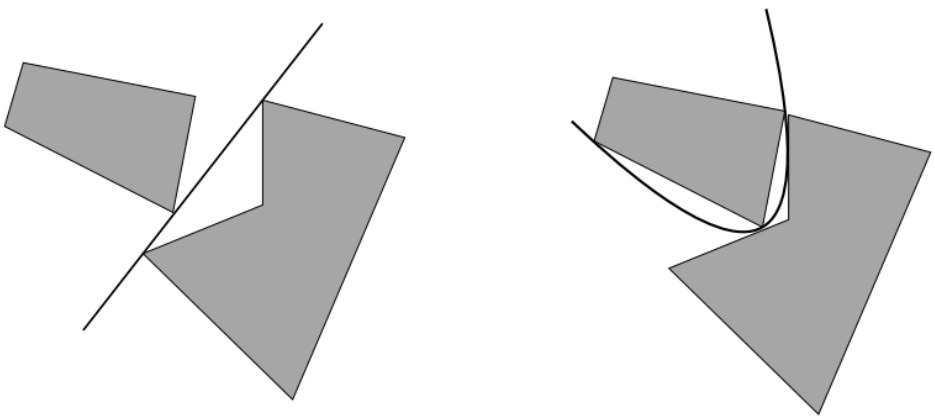
Figura 3.5 – Reta separadora de dois polígonos.



Fonte: Peralta, Andretta e Oliveira (2017)

Quando o problema consiste em empacotamento de peças convexas, as retas separadoras têm um bom desempenho. Porém, quando uma peça é não convexa, Peralta, Andretta e Oliveira (2017) propõe a utilização de parábolas separadoras para garantir a não sobreposição. Uma parábola separa duas peças se uma peça está "dentro" da parábola e a outra "fora". Quando utilizamos retas separadoras na separação de polígonos convexos é sempre possível encostar uma peça à outra. Já no caso de peças não convexas isso nem sempre é possível. Nesse caso pode-se recorrer a uma parábola separadora que poderá permitir que peças não convexas se toquem, tal como está representado na Figura 3.6.

Figura 3.6 – Parábola separadora de dois polígonos.



Fonte: Peralta, Andretta e Oliveira (2017)

3.2 Problemas de *Nesting* sem rotações livres

A maioria dos trabalhos presentes na literatura sobre o problema de *Nesting* trata as peças sem rotação ou com rotações pré-definidas. Nesta seção foram selecionados alguns que julgamos mais importantes para serem comentados. Dividimos os trabalhos em métodos exatos, meta-heurísticas e heurísticas construtivas.

3.2.1 Métodos exatos

No trabalho de Fischetti e Luzzi (2009), é proposto um modelo de programação linear inteira mista (MILP) para um subproblema do problema de *Nesting*, chamado *multiple containment problem* (MCP). As peças são representadas por polígonos e o NFP é usado para verificar não sobreposição. No MCP, dado que algumas peças "grandes" já estão fixas na faixa, tem-se como objetivo reaproveitar o espaço não utilizado, alocando as peças "menores". A proposta dos autores foi comparada com um algoritmo guloso, que sempre tenta alocar a peça no menor espaço possível. As soluções obtidas superaram as melhores conhecidas na literatura do MCP, levando em conta qualidade da solução e tempo de execução.

Alvarez-Valdes, Martinez e Tamarit (2013) propõem um algoritmo *branch and bound* com uma estratégia de ramificação, visando uma melhoria no tempo de resolução do método. Para o sequenciamento de peças (representadas por polígonos), foi utilizada uma ordenação decrescente de suas áreas. O NFP é utilizado para a verificação de não sobreposição das peças. Os autores chegaram à conclusão que, com boas estratégias de ramificação, o tempo computacional para a resolução do problema é reduzido em até 50% se comparado ao *solver CPLEX* e outras estratégias de ramificação propostas.

Toledo et al. (2013) propõem uma formulação em que peças irregulares são alocadas dentro de uma faixa com pontos discretizados. As peças no modelo são representadas por um ponto de referência. Para garantir inclusão das peças na faixa e a não sobreposição dessas, os autores utilizam os conjuntos *Inner-Fit Polygon* e o *No-Fit Polygon*, respectivamente. Um modelo matemático foi proposto e testado com instâncias clássicas da literatura. Os resultados são satisfatórios em instâncias com 16 a 56 peças. No entanto, a formulação gera muitas restrições, fazendo com que seu processamento seja da ordem de 18 mil segundos com um conjunto de somente 16 peças.

No trabalho de Leao et al. (2019), os autores comparam as formulações matemáticas propostas na literatura de problemas de *Nesting*, considerando a orientação da peça, a natureza no modelo matemático, representações da placa, e quais os tipos de geometria utilizadas para verificar a não sobreposição das peças. O trabalho é dividido em classes de modelos: modelos de programação linear inteira-mista, e

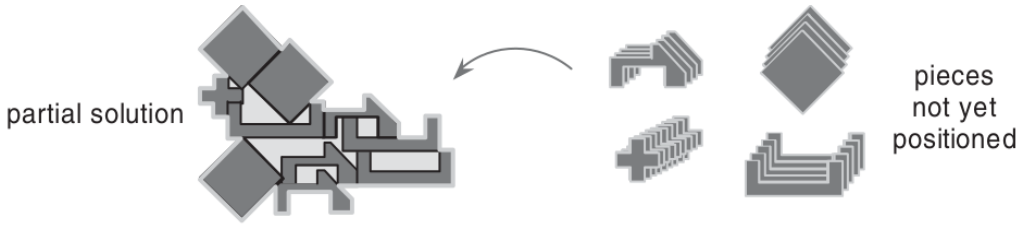
modelos de programação não-linear. A grande diferença entre esses modelos está ligada diretamente às rotações das peças. O modelo se torna linear inteiro-misto quando existem rotações pré-definidas para as peças, ou quando rotações livres não são estabelecidas. Quando uma rotação contínua da peça é permitida, os modelos matemáticos triviais se tornam não-lineares.

3.2.2 Heurísticas construtivas

Heurísticas construtivas possuem, em geral, baixo custo computacional, além de serem boas alternativas para a geração de limitantes superiores. A solução obtida por uma heurística construtiva pode ser utilizada como o ponto de partida de outras heurísticas, meta-heurísticas e algoritmos exatos.

Oliveira, Gomes e Ferreira (2000) propõem uma nova regra no posicionamento de peças que modifica o posicionamento parcial das peças na faixa. Na Figura 3.7, apresentamos uma solução em construção e um possível posicionamento para a próxima peça.

Figura 3.7 – Regra de posicionamento proposta por Oliveira, Gomes e Ferreira (2000).



Fonte:Bennell e Oliveira (2009)

Ao longo da construção da solução, três regras de posicionamento são testadas a fim de tentar minimizar a quantidade de espaço gasto na faixa:

- minimização do retângulo envolvente da solução parcial;
- minimização do comprimento da solução parcial;
- maximização da sobreposição do retângulo envolvente de duas peças, sem que elas se sobreponham.

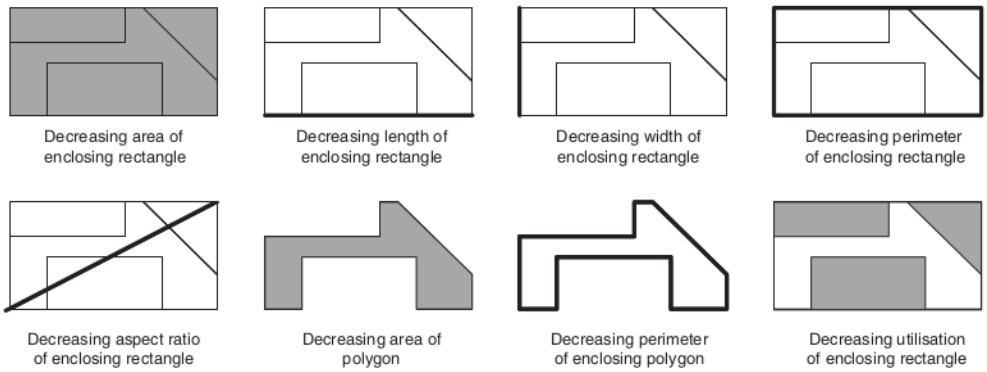
A cada iteração do algoritmo, uma nova peça é inserida na solução parcial, buscando atender às três regras propostas pelos autores. Qualquer lacuna que esteja entre as peças alocadas é considerada como um desperdício de placa.

Outra questão importante durante a construção de uma solução é como as peças são ordenadas. A alteração da ordem de posicionamento das peças tem uma influência significativa na qualidade das

soluções geradas. A abordagem mais simples para se utilizar é a aleatorização do sequenciamento das peças. Dowsland, Vaid e Dowsland (2002) propuseram oito métodos para a ordenação das peças:

- área do retângulo envolvente;
- comprimento do retângulo envolvente;
- largura do retângulo envolvente;
- perímetro do retângulo envolvente;
- diâmetro do retângulo envolvente;
- área da peça;
- perímetro da peça;
- área da não utilização do retângulo envolvente.

Figura 3.8 – Ordenações propostas por Dowsland, Vaid e Dowsland (2002).



Fonte:Bennell e Oliveira (2009)

Uma outra abordagem é a seleção dinâmica das peças, como apresentada em Bennell e Oliveira (2009). Tal método consiste em, a cada iteração, analisar a configuração atual da placa e escolher as próximas peças candidatas a serem alocadas baseando-se na atual solução construída do problema.

O *survey* de Bennell e Oliveira (2009) discute regras de posicionamento e sequenciamento das peças. Uma das regras clássicas descritas, denominada *Bottom-Left*, estabelece que cada peça a ser inserida é deslizada para baixo até tocar em uma outra peça. Quando isso ocorre, a peça candidata é deslocada para a esquerda, até que possa ser deslizada para baixo novamente. O algoritmo para quando a peça não pode ser mais deslocada para esquerda e para baixo. Observe que a ordem dos itens afeta a solução final.

Os autores também ilustram algumas possíveis maneiras de se ordenar os itens durante a construção de uma solução, como por exemplo: (i) ordená-los pela área da peça; (ii) pela área do retângulo que o envolve; (iii) pelo seu diâmetro; (iv) pela área não utilizada do retângulo envolvente; entre outras.

3.2.3 Meta-heurísticas

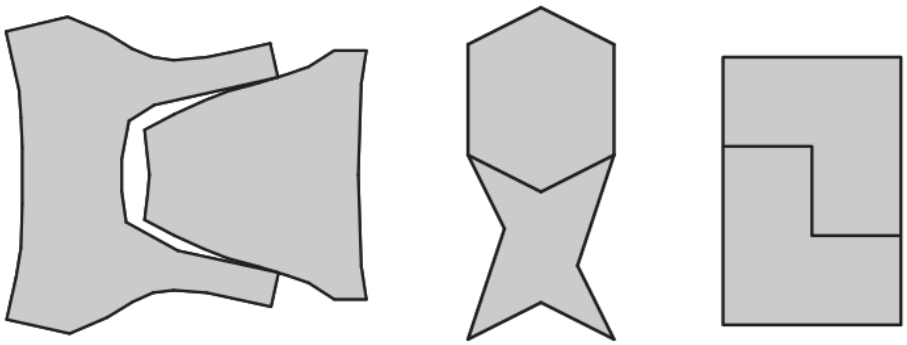
Existem vários trabalhos na literatura que utilizam heurísticas para resolver o problema de *Nesting* sem rotação livre. Porém, alguns autores optaram por utilizar meta-heurísticas para a resolução e obtiveram resultados relevantes. Como nosso foco maior é em problemas de *Nesting* com rotações livres, escolhemos apenas três trabalhos da literatura que utilizam meta-heurísticas que julgamos serem importantes para exemplificar o uso deste tipo de técnica de resolução.

No trabalho de Oliveira e Ferreira (1993), é utilizado uma adaptação da meta-heurística *simulated annealing* para a resolução do problema. Foram estudadas duas variantes do problema: na primeira, as peças são consideradas de uma forma rasterizada; na outra, mais precisa, as peças são consideradas conjuto de pontos em um plano cartesiano. O algoritmo proposto também foi comparado com um algoritmo também implementado pelos autores, em que é escolhida uma posição aleatória para cada peça (aceita sobreposição inicialmente) e depois que todas peças estão encaixadas. Para retirar tal sobreposição, uma peça é escolhida aleatoriamente e ela começa a ser movida em posições pré-definidas (cima, baixo, direita ou esquerda) pela faixa, até ela não sobrepor nenhuma outra. Comparando a abordagem proposta com a técnica de busca local apresentada, o *simulated annealing* obteve melhores resultados, porém o tempo gasto pelo método é maior que a técnica de busca local.

No trabalho de Elkeran (2013) é utilizado um algoritmo de busca cuco guiada (*guided cuckoo search*), juntamente com uma clusterização de pares para resolver o problema de *Nesting*. A clusterização guiada escolhe peças que contém concavidades que se ajustam e as agrupa, como pode ser visto na Figura 3.9. Após os polígonos sofrerem esta clusterização, uma solução inicial é gerada com o algoritmo *Bottom-Left*. Na sequência, *busca cuco guidada* reduz o tamanho da utilização da faixa. Consequentemente, algumas peças irão se sobrepor. Então, as peças que estão se sobrepondo são trocadas de posição, até não se sobreporem a nenhuma outra. Para instâncias clássicas da literatura, o algoritmo proposto pelo autor conseguiu uma compactação de 100%. Atualmente o trabalho de Elkeran (2013) possui os melhores resultados para todas as instâncias da literatura.

Mundim, Andretta e Queiroz (2017) propõem um algoritmo genético com chaves aleatórias viciadas (BRKGA, do inglês: *Biased Random-Key Genetic Algorithm*). Os indivíduos da população são compostos pela ordem em que as peças serão inseridas na solução. Para o empacotamento dos itens, a heurística *Bottom-Left* retângulo é utilizada, juntamente com o *NFR* proposto pelos autores. A partir de

Figura 3.9 – Pares de polígonos agrupados.



Fonte:Elkeran (2013)

experimentos computacionais com duas versões do algoritmo, os autores obtiveram desvios percentuais em relação às melhores soluções reportadas da ordem de $3,13\%$ de diferença para o trabalho de Elkeran (2013). Por outro lado, o BRKGA obteve êxito em 12 instâncias em que a literatura não conseguia obter uma resolução.

3.3 Problemas de *Nesting* com rotações livres

Ao nosso conhecimento, existem poucos trabalhos que abordam o problema de *Nesting* com rotações livres das peças. Rocha et al. (2015) utilizam uma abordagem de duas fases, que é baseada em uma estratégia de compactação das peças maiores dentro do faixa. O trabalho representa as peças através da Cobertura Completa de Circulos (CCC). Na primeira fase, o foco é gerar um leiaute com as peças maiores, para que nos espaços pequenos entre elas caibam as peças menores. A segunda fase consiste em alocar o restante das peças (peças menores) na faixa. A compactação das peças utiliza um modelo não linear. É possível observar que os resultados obtidos pela abordagem de duas fases é melhor ao algoritmo de compactação normal em todas as instâncias testadas. No entanto, o tempo da abordagem proposta pelos autores é maior.

Stoyan, Pankratov e Romanova (2016) propõem um modelo não-linear utilizando a função ϕ para a verificação de não sobreposição das peças. Como ponto de partida para o modelo, foi desenvolvido um algoritmo que busca ótimos locais em um baixo custo de tempo. O algoritmo utilizado, primeiramente, gera sub regiões com pontos factíveis. Fornecendo uma solução já factível para o modelo, ele é capaz de encontrar soluções com maior facilidade, pois a quantidade de desigualdades presentes diminui. Os resultados encontrados pelos autores foram comparados com outros quatro trabalhos da literatura, e se mostraram superiores.

Liao et al. (2016) propõem uma analogia do algoritmo de empacotamento que o compara a um elástico, baseado em um modelo físico. No problema abordado pelo autor, as peças são representadas por polígonos. O algoritmo simula um elástico de borracha que puxa os itens, até compactá-los. Para fazer essa simulação, o algoritmo divide as peças em quatro regiões fora da placa. Em seguida, cálculos de distância entre vértices e massa das peças são feitos para determinar a compactação das mesmas. Essas, por sua vez, são agrupadas para o centro da placa. O elástico irá "apertar" as peças, até que elas respeitem a dimensão fixa da placa.

Peralta, Andretta e Oliveira (2017) propõem um novo método de separação baseado em retas e parábolas separadoras, além de um modelo não-linear para a resolução do problema. Para comprovar a eficiência do formulação proposta, o algoritmo clássico *Bottom-Left* com rotações pré-definidas foi utilizado para construir uma solução inicial. Na sequência, os autores aplicam um algoritmo de pontos interiores para melhorar as soluções.

Os resultados obtidos pelo método na resolução do modelo foram comparados com outros trabalhos da literatura, que levam em consideração a rotação livre das peças (STOYAN; PANKRATOV; ROMANOVA, 2016; LIAO et al., 2016)). Para todas as instâncias testadas em Stoyan, Pankratov e Romanova (2016), os valores das soluções são ligeiramente piores que os já reportados. Em relação aos resultados encontrados por Liao et al. (2016), é possível perceber uma melhoria de até 5% na qualidade das soluções em 8 instâncias. Em outras 2 instâncias, o trabalho de Liao et al. (2016) obteve melhores resultados.

No trabalho de Cherri, Cherri e Soler (2018) é proposto um modelo misto de programação inteira com restrições quadráticas. No modelo, as peças são alocadas de acordo com seu ponto de referência, dadas sua rotação e sua posição. Para eliminar uma quantidade de soluções simétricas que o modelo gera, foram propostas restrições de quebra de simetria. Quando adicionadas as restrições de quebra de simetria, o modelo melhora a qualidade das soluções geradas. Três *solvers* foram testados para a resolução do modelo (COUNNE, SCIP e BARON), sendo o SCIP aquele com o melhor desempenho geral.

4 ABORDAGENS HEURÍSTICAS PROPOSTAS

Bennell e Oliveira (2009) afirmam que o problema de *Nesting* é *NP-Difícil*, já que se reduz ao *BOX-PACK-2* (Nielsen e Odgaard (2003)), que por sua vez também é *NP-Difícil*. O *BOX-PACK-2* consiste em empacotar em uma faixa fixa retângulos de tamanhos diferentes, sem possibilidade de rotação. Diante da dificuldade em encontrar soluções exatas de maneiras eficientes, vamos considerar abordagens heurísticas, inspiradas pelos bons resultados reportados nas literatura em variantes desse problema (como, por exemplo, em Oliveira e Ferreira (1993), Fischetti e Luzzi (2009), Elkeran (2013), Alvarez-Valdes, Martinez e Tamarit (2013), Toledo et al. (2013), Rocha et al. (2015), Stoyan, Pankratov e Romanova (2016), Liao et al. (2016), Peralta, Andretta e Oliveira (2017), Mundim, Andretta e Queiroz (2017), Cherri, Cherri e Soler (2018), Leao et al. (2019)).

Abordagens heurísticas para o problema de corte e empacotamento podem ser construtivas, ao inserir iterativamente cada item na faixa e fornecer uma solução factível para o problema. Existem ainda procedimentos de melhoria, que através de algoritmos de busca local, reduzem o desperdício de espaço na faixa onde as peças estão posicionadas. Para o problema de *Nesting* com rotação livres abordado em neste trabalho, três perguntas devem ser respondidas ao se construir uma solução:

1ª: Em qual ordem sequenciar as peças?

2ª: Em que ângulo cada peça deve ser rotacionada?

3ª: De que maneira posicionar cada peça na faixa?

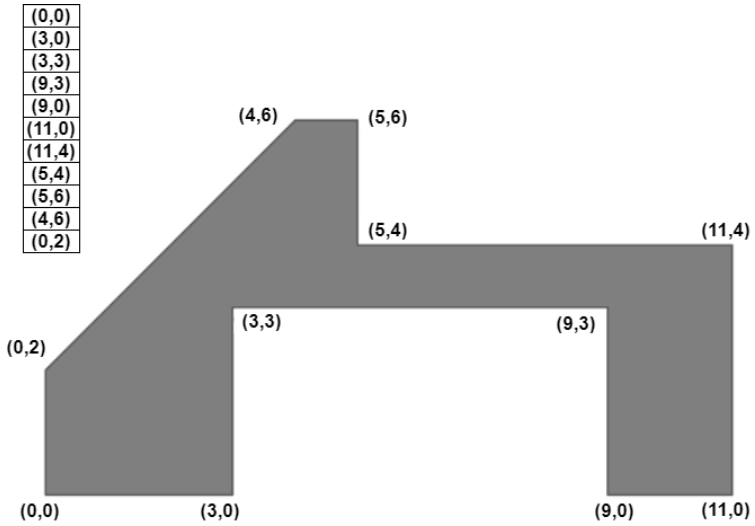
Para responder a estas perguntas, foram implementadas cinco regras de ordenação de peças, duas regras de rotação e cinco regras de posicionamento. Cada regra foi implementada sem a dependência de outras regras, gerando um total de 50 possibilidades de combinação entre os três critérios de geração de soluções para o problema. Nas próximas seções, apresentamos qual representação de peças e da faixa foi utilizada; qual algoritmo de verificação de não sobreposição foi implementado. Além disso, apresentaremos as regras de ordenação utilizadas, regras de rotação e as regras de posicionamento. Por fim, propomos uma adaptação do Algoritmo Genético para o problema.

4.1 Representação das peças e da faixa

Cada uma das n peças é representada por um arranjo de pontos (x,y) ordenados, em que o ponto (x_i,y_i) está ligado com o ponto (x_{i+1},y_{i+1}), para $i = 0,...,|V_p|-1$ (o último ponto está ligado com o primeiro). Com este tipo de representação, é possível definir qualquer polígono, somente armazenando onde

cada vértice está localizado, independentemente de sua complexidade e quantidade de irregularidades. A representação usada está exemplificada na Figura 4.1.

Figura 4.1 – Representação de peças por vetor de pontos.



Fonte: Do Autor (2021)

Esse esquema de representação diminui o espaço de armazenar os polígonos, pois é necessário armazenar somente os vértices. A faixa é definida por uma constante L representando sua largura, além de uma variável W, que representa a altura usada da faixa, e terá seu valor minimizado. Para se calcular o valor da função objetivo em uma solução (ou seja, o valor de W), basta verificar entre todos os polígonos, qual tem o vértice com o maior valor na coordenada y.

Toda vez que uma peça é posicionada, verificamos se todos os pontos da peça estão dentro dos limites da faixa. Assumimos, sem perda de generalidade, que a faixa está posicionada no primeiro quadrante do plano cartesiano (ou seja, seu canto inferior esquerdo está posicionado em $(0,0)$).

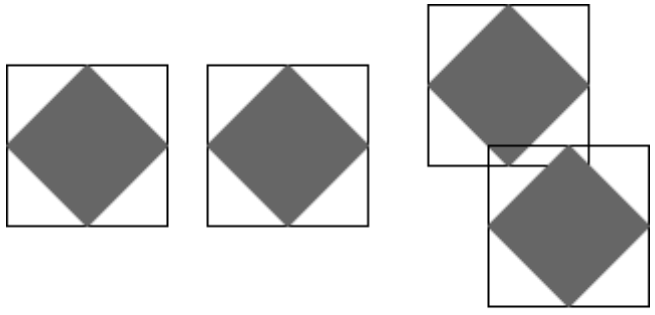
4.2 Verificação de sobreposição de peças

Como apresentado na Seção 3.1, existem algumas formas de verificar a sobreposição de peças não convexas, como, por exemplo, o NFP (BENNELL; OLIVEIRA, 2008), o *No-Fit Raster* (MUNDIM; ANDRETTA; QUEIROZ, 2017), retas e parábolas separadoras (PERALTA; ANDRETTA; OLIVEIRA, 2017), entre outras. Neste trabalho, optamos por utilizar um método baseado em triângulos. O motivo da escolha deste método é devido à dificuldade de se utilizar o NFP e o *No-Fit Raster* quando o problema permite rotações livres. Dado esse ponto, optamos por utilizar a triangulação das peças.

No método proposto, cada peça é particionada em triângulos, em uma etapa de pré-processamento. Para verificar se duas peças se sobrepõem, calculamos a sobreposição entre os triângulos que as com-

põem. Inicialmente, é criado um retângulo em volta de cada peça. Com a criação dos retângulos, é verificado se as peças se intersectam através de testes simples com as posições dos vértices. Caso os retângulos não se sobreponham, então as peças não estão sobrepostas (Figura 4.2(a)).

Figura 4.2 – Teste com o retângulo envolvente. Fonte: próprio autor



Fonte: Do Autor (2021)

Como pode ser observado na Figura 4.2 (b), o fato de os retângulos envolventes das peças estarem sobrepostos não significa necessariamente que as peças também estejam. Então, caso os retângulos estejam se sobrepondo, consideramos a partição das peças em triângulos e realizamos outros testes.

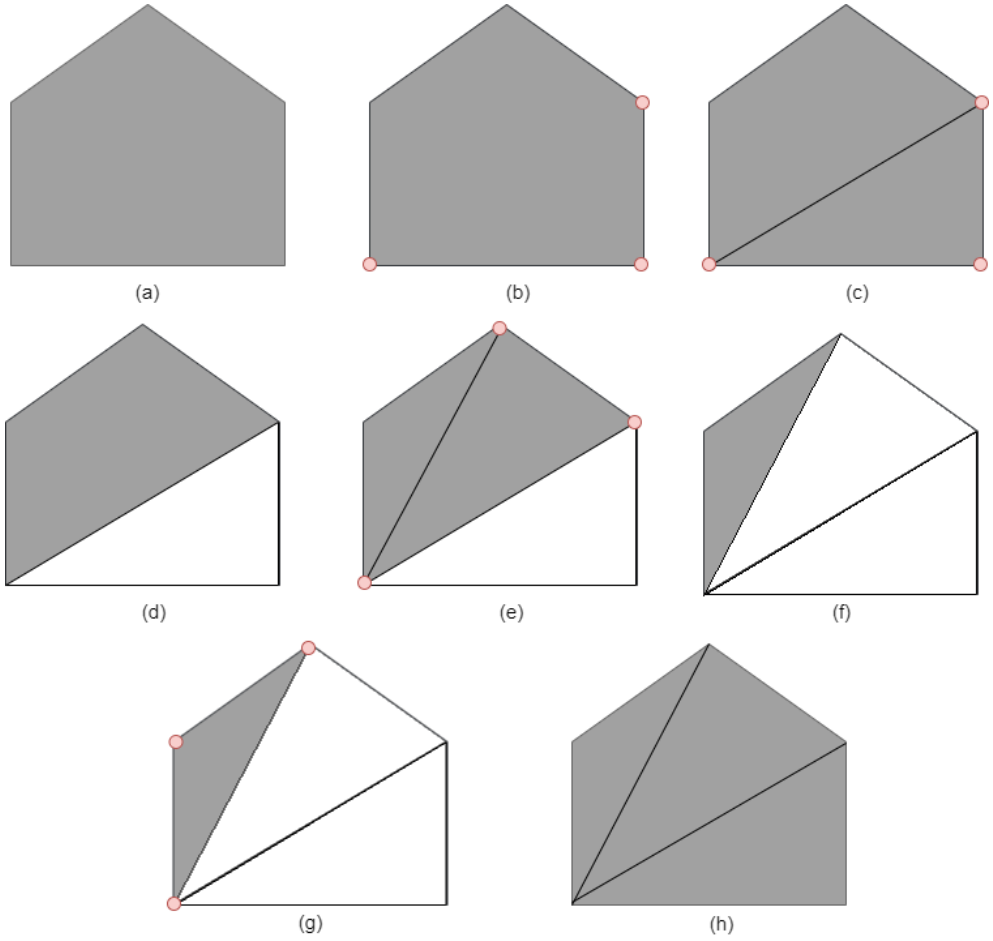
Note que esta verificação usando os retângulos não é necessária. No entanto, como ela é computacionalmente muito mais eficiente, seu uso faz com que o tempo gasto na verificação de sobreposição seja reduzido.

Para particionar um polígono em um conjunto de triângulos, é utilizado um algoritmo chamado de dobra de orelha (EBERLY, 2015). O algoritmo consiste em, inicialmente, verificar se três pontos consecutivos i, $i-1$ e $i+1$ de um polígono formam um triângulo. Caso isso aconteça, o ponto i é marcado como um vértice de orelha e é removido do polígono original. O processo é repetido até que o último triângulo é removido.

Na Figura 4.3 temos um exemplo da execução deste algoritmo. O polígono original é apresentado na Figura 4.3 (a). O algoritmo escolhe, inicialmente, três vértices consecutivos (marcados em vermelho na Figura 4.3 (b)) e testa se eles formam um triângulo (Figura 4.3 (c)). Como eles formam um triângulo, então o mesmo é removido do polígono original (a parte removida do polígono original está em branco na Figura 4.3 (d)).

Após a remoção do triângulo, o algoritmo escolhe novamente três vértices consecutivos e faz o teste para verificar se os três formam um triângulo (Figura 4.3 (e)). Como eles formam um triângulo, este é retirado do polígono (Figura 4.3 (f)). Caso os três pontos não formassem um triângulo, os três próximos pontos consecutivos seriam escolhidos para realizar o teste.

Figura 4.3 – Funcionamento do algoritmo de triangulação.



Fonte: Do Autor (2021)

Na Figura 4.3 (g), o algoritmo escolhe os três últimos vértices do triângulo (que formam, de fato, um triângulo) e o remove, finalizando o processo de triangulação da peça. Na Figura 4.3 (h), é possível observar o resultado final do algoritmo.

Com a utilização desse algoritmo, sempre é possível particionar um polígono convexo ou não convexo (representado por uma lista de vértices) em um conjunto de triângulos (cada um deles representado por uma lista de vértices) (EBERLY, 2015).

Agora que já temos uma forma de particionar as peças em triângulos, resta a verificação de quando dois triângulos estão sobrepostos ou não. Na próxima seção, apresentamos maneiras de verificar como dois triângulos se sobrepõem. Partimos da ideia de que caso dois triângulos de polígonos diferentes de sobreponham, os seus respectivos polígonos se sobrepõem também.

4.2.1 Verificação de sobreposição de triângulos

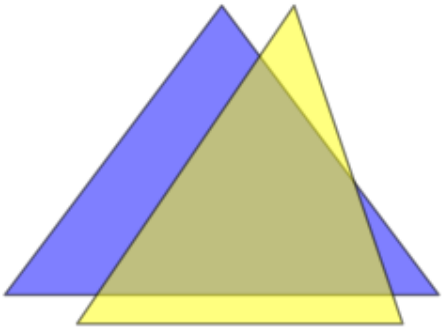
Para verificar a sobreposição de triângulos, utilizaremos três fórmulas: a primeira permite verificar se há intersecção de segmentos de retas; a segunda permite verificar se um ponto no plano está dentro ou fora de um triângulo; a terceira permite verificar se um ponto pertence a um segmento de reta.

Foram identificados quatro casos em que os triângulos estão sobrepostos e as três fórmulas citadas permitem identificar tal sobreposição.

4.2.1.1 Caso 1

O primeiro caso identifica a situação em que ao menos uma aresta de cada um dos dois triângulos se cruzem. Tal circunstância pode ser observada na Figura 4.4.

Figura 4.4 – Caso 1 de sobreposição de triângulos: ao menos duas arestas se cruzam.



Fonte: Do Autor (2021)

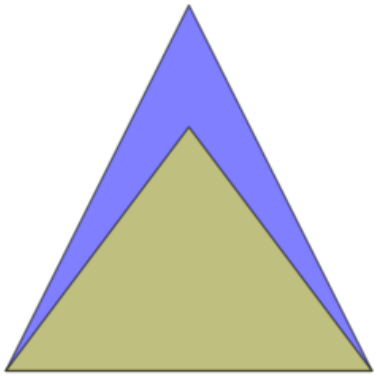
Para a verificação deste caso, foi utilizada a verificação de intersecção de dois segmentos de retas. Todas as arestas de um triângulo são comparadas com todas as arestas do outro. Caso haja alguma intersecção, os triângulos se sobrepõem (consequentemente, as peças às quais eles fazem parte também se sobrepõem).

4.2.1.2 Caso 2

O segundo caso ocorre quando existe um vértice de um triângulo no interior do outro. Tal caso pode ser observado na Figura 4.5.

Para verificar este caso, observamos se cada vértice de um triângulo está no interior do outro triângulo. Caso isso ocorra para algum vértice, o algoritmo acusará que eles se sobrepõem. Consequentemente, seus respectivos polígonos se sobrepõem.

Figura 4.5 – Caso 2 de sobreposição de triângulos: existe um vértice de um triângulo no interior do outro triângulo.

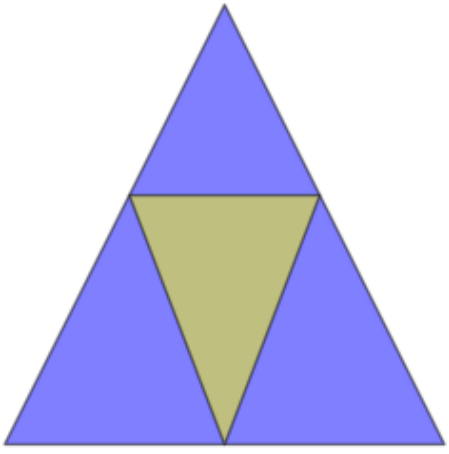


Fonte: Do Autor (2021)

4.2.1.3 Caso 3

O terceiro caso ocorre quando os três vértices de um triângulo pertencem às arestas de outro triângulo. Neste caso, nenhum cruzamento de segmento de reta ocorre entre as arestas destes dois triângulos (Caso 1) e nenhum vértice de um triângulo pertence ao interior de outro triângulo (Caso 2). Tal ocorrência pode ser observada na Figura 4.6.

Figura 4.6 – Caso 3 de sobreposição de triângulos: todos os vértices de um triângulo estão sobre arestas do outro.



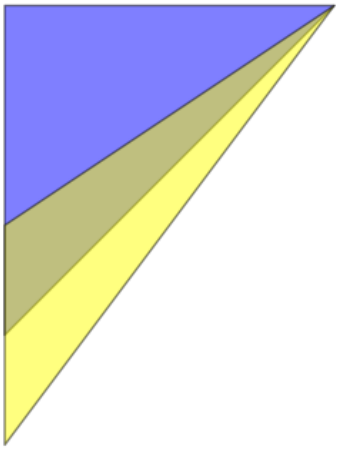
Fonte: Do Autor (2021)

Para verificar esse caso, identificamos se um ponto (vértice de um triângulo) pertence a algum segmento de reta (aresta de outro triângulo). Caso isso ocorra para todos os vértices, o algoritmo identificará que eles se sobrepõem.

4.2.1.4 Caso 4

O quarto caso ocorre quando um dos vértices de um triângulo está exatamente na mesma posição de um vértice de outro triângulo e um dos dois outros vértices do primeiro triângulo está sobre uma aresta do segundo triângulo. Um exemplo pode ser observado na Figura 4.7.

Figura 4.7 – Caso 4 de sobreposição de triângulos: Dois triângulos com um vértice em comum e um de seus vértices é um ponto na aresta de outro.



Fonte: Do Autor (2021)

Para a verificação deste caso, é calculado o ponto médio do segmento de reta que tenham o mesmo vértice em comum e do segmento de reta que tem um vértice na aresta do outro triângulo. Se o ponto médio está no interior do outro triângulo, então o algoritmo irá identificar que os dois triângulos estão sobrepostos.

4.3 Regras de ordenação

A forma como os itens são posicionados na faixa pode alterar drasticamente a qualidade da função ao final da execução do algoritmo. A abordagem mais simples é sequenciar os polígonos aleatoriamente. Porém, neste trabalho, consideramos ordenações pré-definidas para os polígonos.

Em nosso estudo, utilizamos quatro (SA, SB, SD, SE) das oito regras de sequenciamento de peças propostas por Dowsland, Vaid e Dowsland (2002), além de um critério (SC) introduzido nesse trabalho. Os cinco critérios utilizados na ordem decrescente são apresentados a seguir:

SA. área do polígono;

SB. valor absoluto da não utilização retângulo que envolve o polígono;

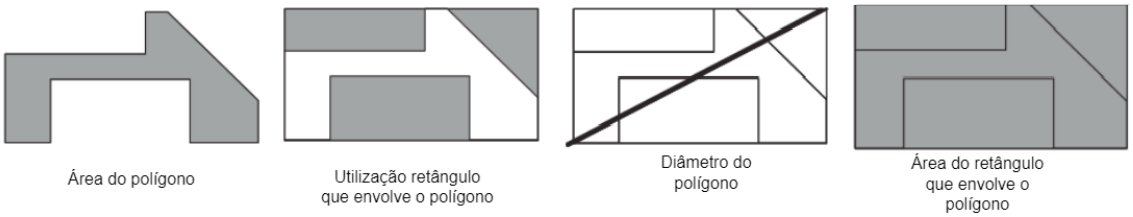
SC. percentual da não utilização do retângulo que envolve o polígono;

SD. diâmetro do polígono;

SE. área do retângulo que envolve o polígono.

Para ilustrar os critérios de ordenações utilizados, observe a Figura 4.8. Na terceira ilustração da Figura 4.8 (critério SD), consideramos a distância Euclidiana entre os pontos (x_{min}, y_{min}) e (x_{max}, y_{max}). As coordenadas (x_{min}, y_{min}) são os menores valores de x e y encontrados nos pontos de polígonos, e (x_{max}, y_{max}) os maiores.

Figura 4.8 – Critérios de ordenação utilizados.



Fonte: Adaptado de Bennell e Oliveira (2009)

O critério de ordenação proposto no presente trabalho (critério SC) efetua o sequenciamento dos polígonos de acordo com o percentual de utilização decrescente do retângulo envolvente, enquanto Dowsland, Vaid e Dowsland (2002) utilizam valores absolutos. Para se calcular o percentual de não utilização, utilizamos a seguinte fórmula:

$$P = 100 * (\mathscr{A} - \mathscr{A}^*)/\mathscr{A}, \tag{4.1}$$

em que $\mathscr{A}$ é o valor da área do retângulo envolvente, e $\mathscr{A}^*$ é a área do polígono. Quando o polígono utiliza toda a área do retângulo, ou seja, o polígono em si é um retângulo, então o valor de P é igual a zero.

4.4 Regras de rotação

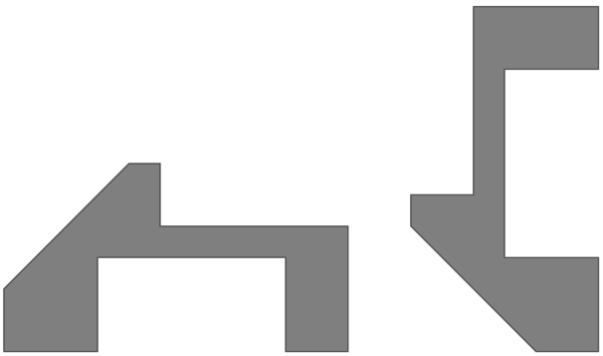
O problema de *Nesting* estudado neste trabalho considera que os polígonos podem rotacionar em qualquer ângulo θ, em que $0 \leq \theta \leq 2\pi$. Neste trabalho, foram propostas duas regras de rotação, que serão usadas para todos os polígonos:

RA. leva em consideração o comprimento e a altura da peça;

RB. leva em consideração a inclinação de uma aresta do polígono.

A regra RA consiste em rotacionar cada polígono em $\pi/2$ caso seu comprimento seja maior que a altura. Um exemplo de rotação pode ser observado na Figura 4.9.

Figura 4.9 – Rotação usando a regra RA.



Fonte: Do Autor (2021)

Após a rotação do polígono, observamos que agora ele ocupará menos espaço no eixo x. Assim, a ideia deste tipo de rotação é fazer com que caibam mais polígonos ao lado do item rotacionado, não havendo necessidade de posicionar outros polígonos acima dele.

A regra RB consiste em escolher uma aresta do polígono e rotacioná-la de forma que a aresta em questão fique paralela ao eixo x ou ao eixo y. Para se calcular tal ângulo, dois parâmetros são necessários (M e N), em que $N = x_0 - x_1$ e $M = y_0 - y_1$, com (x_0, y_0) e (x_1, y_1) os extremos da aresta que será rotacionada.

O ângulo de que o polígono deve ser rotacionado para que a aresta escolhida fique paralela ao eixo y é dado por:

$$\theta = \begin{cases} \arctan(N/M), & \text{se } N \geq M, \\ \arctan(M/N), & \text{se } N < M, \\ \pi/2, & \text{se } M = 0, \\ 0, & \text{se } N = 0. \end{cases} \tag{4.2}$$

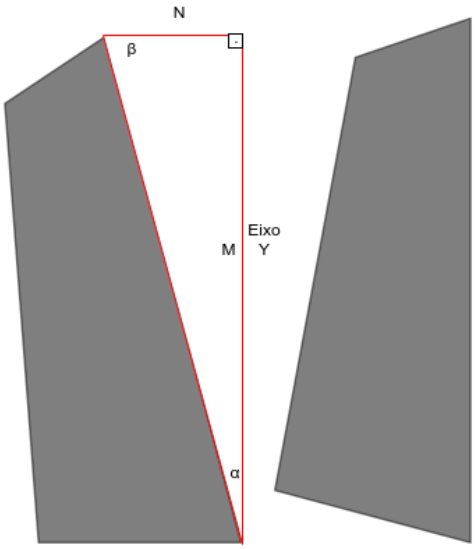
Para encontrar o ângulo de rotação que torna a aresta escolhida paralela ao eixo x, basta utilizar a fórmula descrita acima e adicionar $\pi/2$ ao ângulo obtido. Caso $M = 0$, θ assume o valor de $\pi/2$, pois sua aresta está paralela ao eixo x. Caso $N = 0$, o valor de θ será igual a zero, pois a aresta escolhida já está paralela com o eixo y.

A Figura 4.10 apresenta um exemplo de polígono resultante desta regra de rotação. Neste caso, a aresta escolhida do polígono que antes estava inclinada, agora está paralela com o eixo x. Para obter um polígono rotacionado, devemos calcular para cada coordenada (x_i, y_i) da peça:

$$x_i = x_i \cos\theta - y_i \sin\theta, \tag{4.3}$$

$$y_i = x_i \sin\theta + y_i \cos\theta. \tag{4.4}$$

Figura 4.10 – Rotação usando a regra RB.



Fonte: Do Autor (2021)

4.5 Regras de posicionamento

A maneira como uma peça é inserida na faixa afeta o aproveitamento de espaços livres, que poderiam ser ocupados por peças remanescentes. Neste trabalho, são utilizadas as regras de posicionamento para decidir de que maneira uma peça irá se encaixar. As regras de posicionamento já recebem uma peça com uma rotação definida *a priori*. Ao observar heurísticas clássicas da literatura sobre problemas de *Nesting*, adotamos cinco regras de posicionamento de peças:

PA. Regra *Bottom-Left* (BL);

PB. Regra *Bottom-Left* modificada;

PC. Regra gulosa;

PD. Regra α;

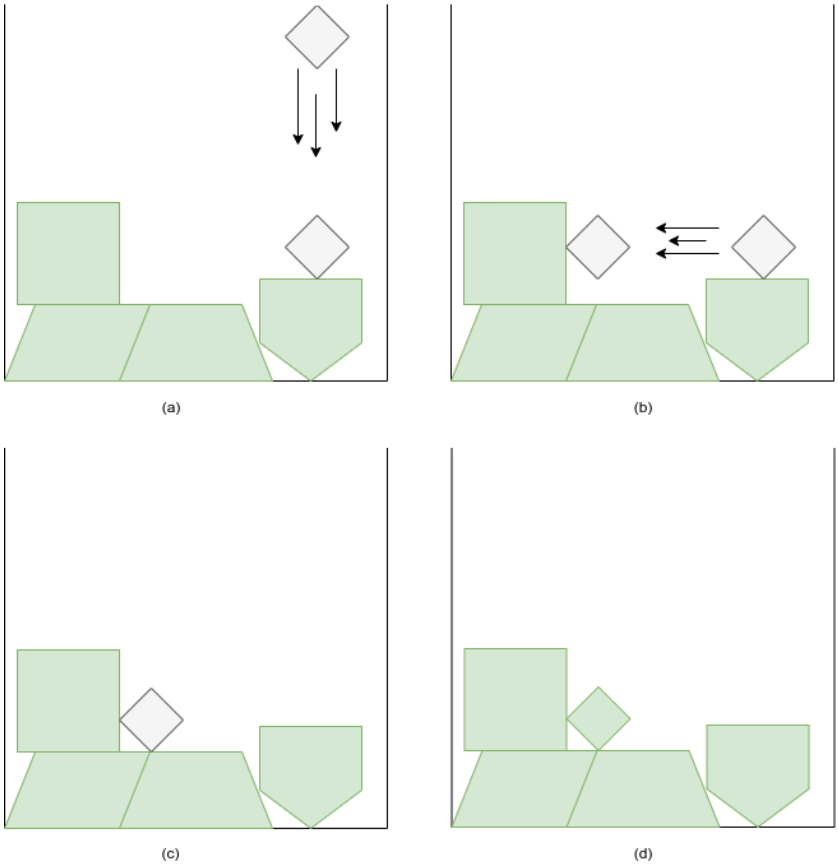
PE. Regra β.

Todos os deslizamentos que estão presentes nas regras de posicionamento a seguir são feitos de uma maneira discreta, utilizando uma constante γ com o valor 0,1.

4.5.1 Regra *Bottom-Left* - PA

A regra *Bottom-Left* (BL) é baseada na heurística proposta por Baker, Coffman Jr e Rivest (1980). A BL move uma peça verticalmente para baixo até que toque em outra. Em seguida, desloca horizontalmente para a esquerda até que encoste em outra peça. Este processo é repetido até que não haja mais movimentos possíveis.

Figura 4.11 – Exemplo da regra *Bottom-Left*.



Fonte: Do Autor (2021)

O procedimento para a alocação de uma nova peça na solução pode ser observado na Figura 4.11. Inicialmente, a peça é deslizada para baixo até tocar em outra peça (Figura 4.11 (a)). Na sequência (Figura 4.11 (b)), ela é deslizada para a esquerda até tocar em outro item. Após a peça não conseguir se mover mais para a esquerda, ela começa a ser novamente deslizada para baixo (Figura 4.11 (c)). Quando

a peça não pode ser deslizada nem para baixo e nem para a esquerda, sua posição final estará estabelecida (Figura 4.11 (d)).

Como pode ser observado no Algoritmo 1, recebemos como parâmetro o conjunto P de peças, qual peça deverá ser alocada (i) e a largura L da faixa. Com isso, o algoritmo moverá a peça para baixo até ela não conseguir se mover mais, depois ela é movida para a esquerda. O método termina quando a peça não consegue se movimentar.

Algorithm 1 Bottom-Left

```
1: Função BOTTOM-LEFT (P, i, L)
2:     Enquanto Peça P[i] puder se mover Faça
3:         Enquanto Peça P[i] consegue mover para baixo Faça
4:             MoveParaBaixo(P[i])
5:         Enquanto Peça P[i] consegue mover para esquerda Faça
6:             MoveParaEsquerda(P[i])
       Retorna P[i]
```

4.5.2 Regra *Bottom-Left* modificada - PB

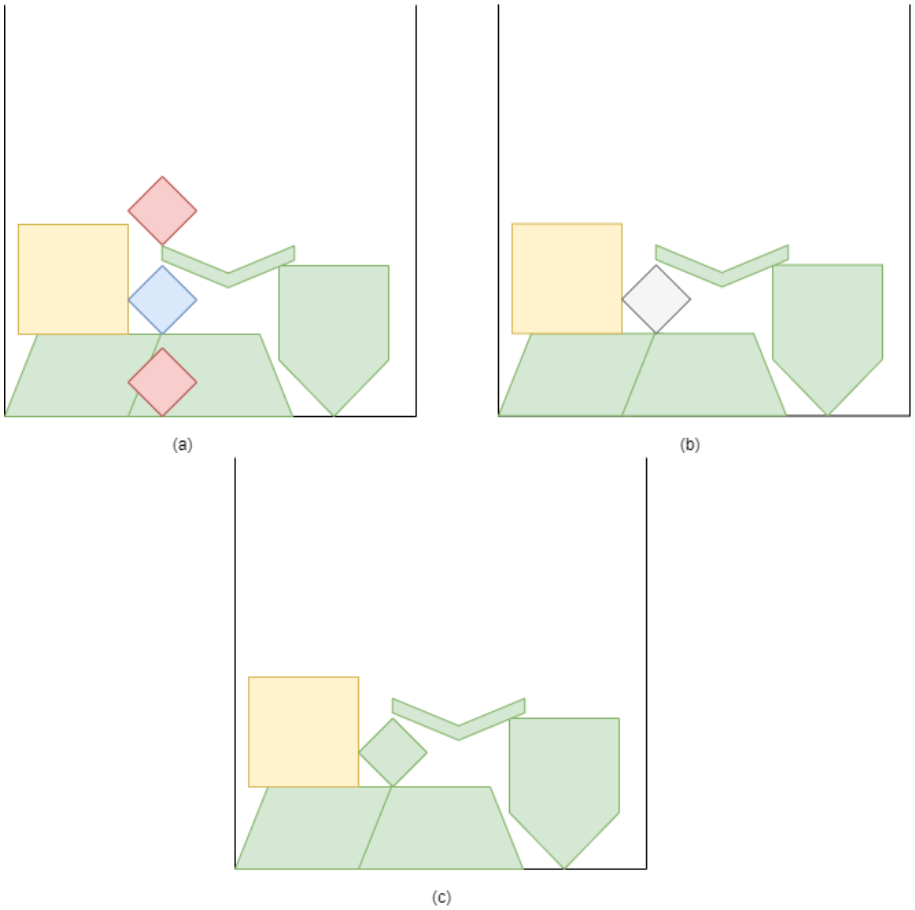
Baseando-se na regra de posicionamento *Bottom-Left*, foi proposta uma modificação no funcionamento do método, para que haja uma melhoria na qualidade das soluções. A modificação proposta tenta inserir o polígono na posição mais abaixo possível, iniciando sempre à direita do último polígono posicionado (caso seja possível). A ideia por trás deste algoritmo é verificar se existem espaços vazios abaixo da última peça inserida.

Na Figura 4.12 (a), vemos que a peça que será inserida (losango) inicia exatamente ao lado da última peça inserida na solução (peça em amarelo), com um de seus vértices posicionados no ponto $y = 0$. Em seguida, a peça é alocada em cima de todas as peças que estão acima dela, ou seja, o menor ponto y da peça será igual ao maior ponto y das outras peças (uma de cada vez). Na Figura 4.12 (b), é verificado que a peça não sobrepõe nenhuma outra peça nessa posição. Então, na Figura 4.12 (c), a peça é alocada. Este procedimento é descrito no Algoritmo 2.

O Algoritmo 2 recebe como parâmetro o conjunto de peças P, o índice i da peça que será alocada e a largura L da faixa. Para a alocação da peça, verificamos se o maior ponto x da última peça alocada, somado com o maior ponto x da peça que está sendo analisada, ultrapassa os limites da faixa. Caso isso ocorra, a peça é reposicionada à origem, para que ela tenha seus pontos y incrementados.

Após a definição da coordenada x da peça, a regra de posicionamento cria em volta dela um retângulo com altura infinita, para determinar quais peças estão acima dela. Em seguida, a peça atual é colocada em cima de cada peça que atualmente está acima dela, tentando encontrar uma posição em

Figura 4.12 – Exemplo da regra *Bottom-Left* modificada.



Fonte: Do Autor (2021)

que ela possa ser inserida (ou seja, em que não haja sobreposição com outras peças e esteja inteiramente contida na faixa).

4.5.3 Regra gulosa - PC

A regra gulosa consiste em iniciar com a peça na origem e ir deslizando-a para a direita, tentando encontrar uma posição em que a peça possa ser inserida. Caso a peça atinja o limite horizontal da faixa, a peça é deslizada à esquerda para tocar o eixo y, é levemente deslizada para cima (constante γ) e o procedimento é repetido. Assim que a primeira posição válida for encontrada, o procedimento para e a peça é inserida neste local.

Uma diferença entre esta regra e a regra *Bottom-Left* modificada é que a busca por espaços vagos inicia da origem, enquanto a Regra *Bottom-Left* modificada inicia ao lado direito da última peça colocada e de baixo para cima.

Algorithm 2 Bottom-Left-modificada

1: **Função** BOTTOM-LEFT-MODIFICADA (P, i, L)
2: **Se** $x_{max}(P[i-1]) + x_{max}(P[i]) > L$ **Então**
3: Coloca P[i] na origem
4: **Senão**
5: $\forall x \in$ P[i], $x \leftarrow x + x_{max}(P[i-1])$
6: poligono_abstrato $\leftarrow (x_{min}(P[i]),0),(x_{max}(P[i]),0),((x_{max}(P[i]),\infty),(0,\infty)$
7: **Para** $j \leftarrow 0$ **até Tamanho**(P) **Faça**
8: **Se** poligono_abstrado sobrepoe $P[j]$ **Então**
9: poligonos_sobrepostos $\cup P[j]$
10: **Para** $j \leftarrow 0$ **até Tamanho**(poligonos_sobrepostos) **Faça**
11: Coloca $P[i]$ em cima poligonos_sobrepostos[j]
12: **Se** $P[i]$ não sobrepõe nenhum polígono **Então**
13: **Retorna** $P[i]$
Retorna $P[i]$

O funcionamento do método pode ser observado na Figura 4.13. Note que na Figura 4.13 (a), o polígono começa na origem e vai deslizando para a direita, tentando encontrar uma posição válida, até tocar o limite da faixa. Na Figura 4.13 (b), o polígono volta à origem do eixo y da faixa e sobe, continuando o processo de deslocamento para a direita. Na Figura 4.13 (c), encontra-se um local para a peça ser inserida, sem que haja sobreposição. Na Figura 4.13 (d) o polígono é empacotado. O pseudocódigo do método pode ser visto no Algoritmo 3.

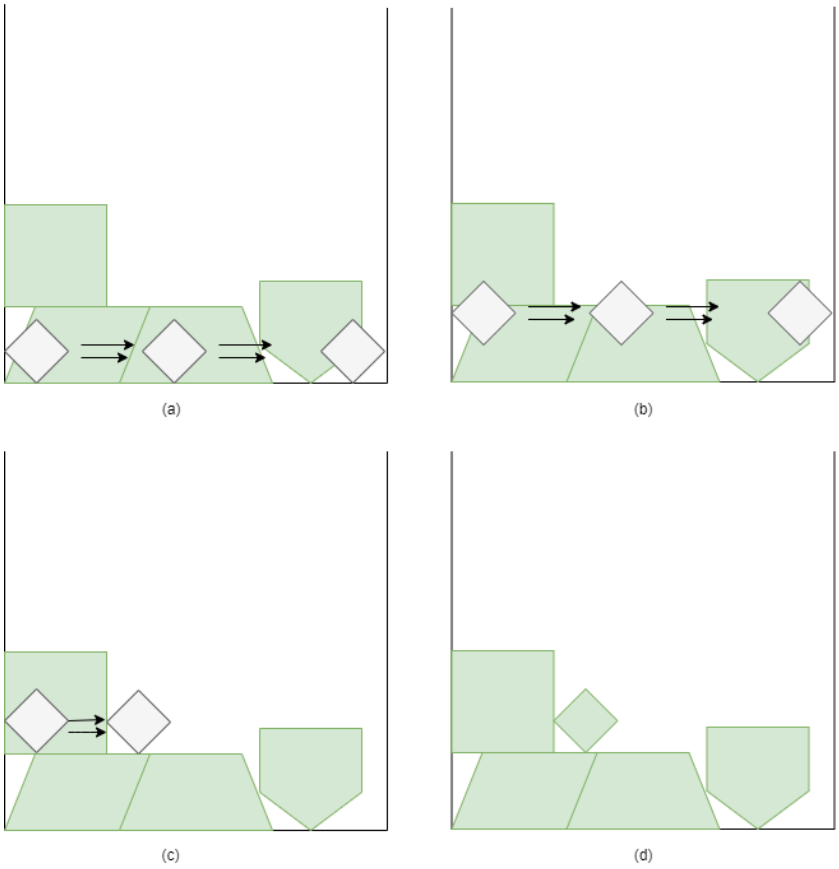
O Algoritmo 3 recebe como parâmetros o conjunto de peças P, o índice da peça que será empacotada e a largura da faixa. A peça que será empacotada é deslocada para a origem. Enquanto a peça $P[i]$ estiver sobrepondo alguma outra peça, o algoritmo desliza a peça no eixo x, fazendo-a deslizar para a direita. Caso a peça atinja o limite da faixa, a peça volta a tocar o eixo y e é levemente deslizada para cima (constante γ), voltando a buscar uma posição válida à direita. Desta forma, em alguma iteração será encontrada uma posição válida, na qual a peça será inserida.

4.5.4 Regra α - PD

A regra α considera todas as combinações em que cada um dos vértices da peça a ser inserida ficam exatamente em cada um dos vértices das peças já alocadas. Caso haja alguma posição viável, aquelas em que a peça a ser inserida fica mais em baixo é escolhida. Caso haja mais de uma posição, seleciona-se quela em que a peça fica mais à esquerda.

Com a posição escolhida da peça, ela é alocada e, caso haja espaço abaixo ou à sua esquerda, a peça sofre um deslocamento similar à regra de posicionamento *Bottom-Left*. O método para quando o polígono não pode ser deslizado em nenhuma das duas direções (abaixo e à esquerda).

Figura 4.13 – Exemplo da regra gulosa.



Fonte: Do Autor (2021)

Como pode ser observado na Figura 4.14 (a), é feita a combinação dos vértices da peça que será alocada com a peça que já está na solução. Na Figura 4.14 (b), as posições inválidas são retiradas, deixando somente as possíveis posições de peças válidas. Na Figura 4.14 (c), é selecionada a posição em que a peça está mais embaixo, e mais à esquerda, realizando na sequência uma realocação similar à regra *Bottom-Left*. Na Figura 4.14 (d), a peça é definitivamente inserida e o método termina.

O Algoritmo 4 recebe como parâmetro o conjunto de peças que estão alocadas na solução (e a peça que será alocada), o índice da peça que será alocada e a largura da faixa. Um laço de repetição é feito para iterar sobre todas as peças do conjunto. O laço subsequente itera sobre todos os vértices da peça que será alocada. Por fim, um terceiro laço é feito para iterar sobre os vértices de cada peça do conjunto. A cada iteração, é feita a movimentação do polígono e sua nova posição é armazenada em uma estrutura de dados.

Após todas as combinações terem sido feitas, as posições inválidas (que causam sobreposição com alguma peça já alocada ou deixam a peça fora da faixa) são removidas deste conjunto. Na sequência, é escolhida a posição em que a peça fica mais para baixo e mais para a esquerda possível (dando

Algorithm 3 Regra gulosa

```
1: Função GULOSA (P, i, L)
2:     Coloca P[i] na origem
3:     Enquanto A peça não se encaixar Faça
4:         move_para_direita(P[i])
5:         Se P[i] nao sobrepõe nenhum polígono em P Então Retorna P[i]
6:         Se x_max de P[i] ≥ L Então
7:             Retorna P[i] para origem do eixo x
8:             move_para_cima(P[i])
       Retorna P[i]
```

prioridade para mais abaixo). Depois de deixar a peça na posição escolhida, ela é deslizada para baixo e para a esquerda, até não conseguir se movimentar novamente. Quando a peça não pode ser movimentar mais, o método termina.

4.5.5 Regra β - PE

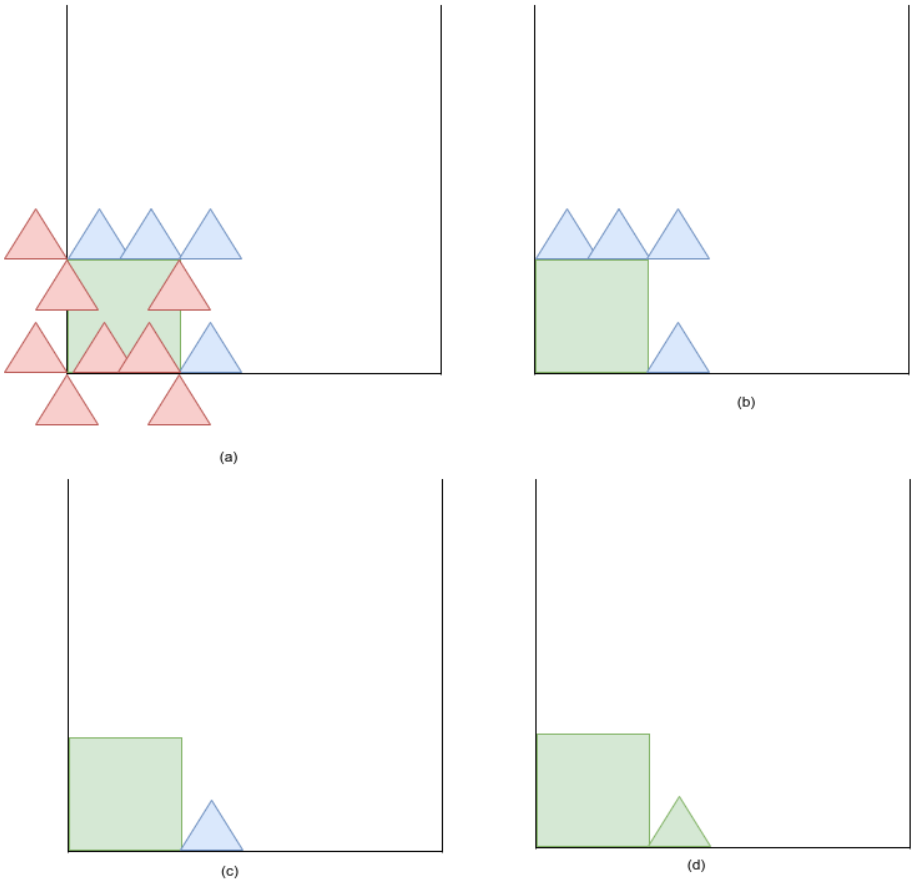
A regra β utiliza conceitos da regra de posicionamento gulosa e da regra *Bottom-Left* modificada. A regra consiste em iniciar uma peça na origem do eixo y, e tenta alocá-la "em cima" das outras peças. Este comportamento é idêntico à regra de posicionamento *Bottom-Left* modificada. Depois de mapeadas todas essas posições (uma posição para cada peça que estava acima dela), a peça é deslizada para a direita e o processo ocorre novamente.

Após todas as posições da faixa para inserção da peça serem armazenadas, as inválidas (que sobrepõem alguma peça ou deixam a peça fora da faixa) são removidas e a posição que está mais abaixo e mais à esquerda é escolhida. Assim, a peça sofre um deslizamento para baixo e para a esquerda, até não conseguir deslizar novamente para nenhuma dessas posições.

Como pode ser observado na Figura 4.15 (a), a peça inicia na origem e é posicionada em cima das peças que estão acima dela. Depois ela é deslizada para a direita e o processo ocorre novamente. Quando a peça toca o limite da faixa, o processo para e então são retiradas todas as posições inválidas, como pode ser visto na Figura 4.15 (b). Na Figura 4.15 (c), é escolhida a posição em que a peça está mais abaixo e mais à esquerda. Por fim, na Figura 4.15 (d), a peça é deslizada para baixo e para a esquerda, e então é alocada. É possível observar que a peça da Figura 4.15 (c) está mais alta que a peça da Figura 4.15 (d).

O Algoritmo 5 recebe como parâmetros o conjunto de peças que estão alocadas na solução (e a peça que será alocada), o índice da peça que será alocada e a largura L da faixa. Então, um laço de repetição é executado até que a peça não possa ser mais deslocada para a direita. A cada iteração do laço, o algoritmo cria um polígono abstrato em volta da peça que será alocada, para saber quais peças estão

Figura 4.14 – Exemplo da regra α.



Fonte: Do Autor (2021)

acima dela. Em seguida, a peça que será alocada é alocada em cima (seu menor ponto y será igual ao maior ponto y das outras peças) das peças que ficaram sobrepostas ao polígono abstrato, e essas posições são armazenadas em uma estrutura de dados. Com isso, é possível saber se a peça $P[i]$ pode ser alocada em tal posição, não sobrepondo nenhuma peça.

Em seguida, cada posição armazenada na estrutura de dados é examinada e o algoritmo retira as posições infactíveis. Na sequência, o algoritmo escolhe aquela posição em que a peça estava mais em baixo e mais a esquerda, e a aloca novamente. Depois da peça ser inserida na posição escolhida, ela é deslizada para baixo e para a esquerda. O método para quando a peça não pode ser deslizada novamente.

4.6 Heurística de Múltiplos Posicionamentos (HMP)

A Heurística de Múltiplos Posicionamentos (HMP) consiste em uma heurística que não utiliza somente uma regra de posicionamento para posicionar suas peças. As peças que serão inseridas na faixa são ordenadas, todas com a mesma regra de ordenação (escolhida a priori), porém, sempre que uma peça for ser inserida na faixa, o algoritmo sorteia, entre as regras α (PD) e β (PE) uma regra

Algorithm 4 Regra α

```
1: Função REGRA α (P, i, L)
2:     posicoes_possiveis ← {}
3:     Para p to P Faça
4:         Para j ← 0 até Tamanho(P[i]) Faça
5:             Para k ← 0 até Tamanho(p) Faça
6:                 posicao_candidata ← combina_vertices(P[i][j], p[k])
7:                 posicao_candidata ← posicao_candidata ∪ posicoes_possiveis
8:     remove_posicoes_invalidas(posicoes_possiveis)
9:     P[i] ← posicao_baixo_esquerda(posicoes_possiveis)
10:    desliza_poligono(P[i])
11: Retorna P[i]
```

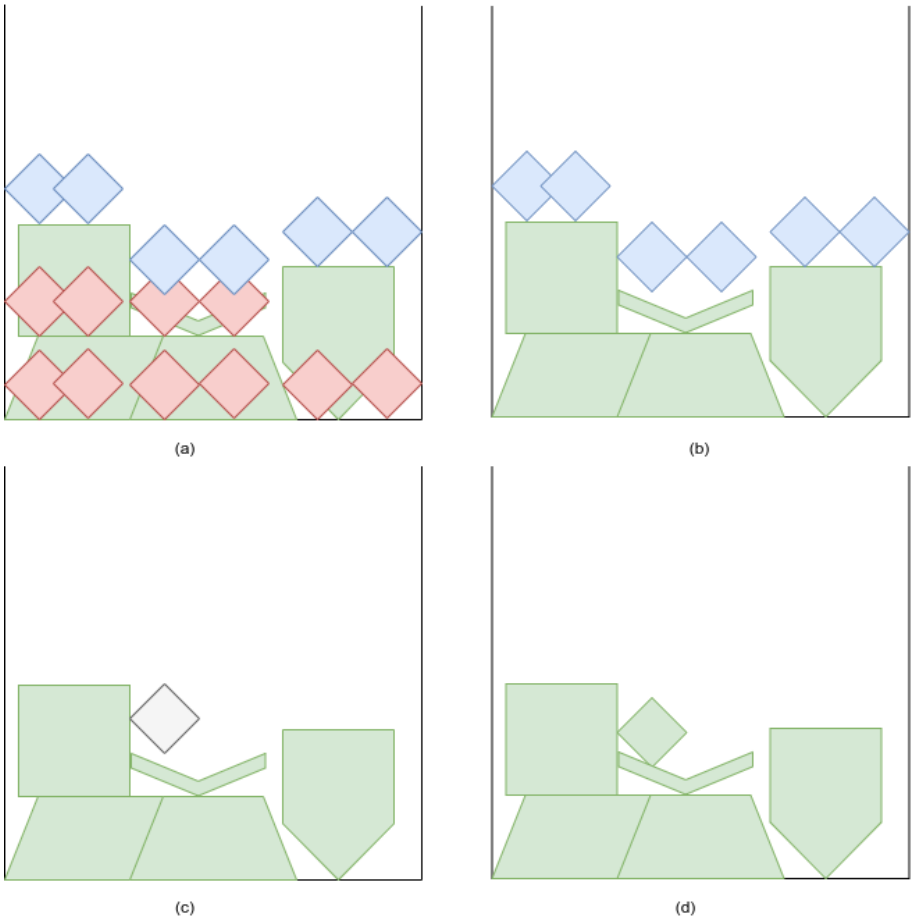
de posicionamento para a peça. Após um número pré-determinado de iterações, escolhemos o melhor resultado encontrado pela HMP. Após os testes feitos somente com as regras de posicionamento, vimos que as regras α e β obtiveram melhores resultados, por isso, a HMP irá utilizar somente uma destas duas regras de posicionamento. Para melhor entendimento da heurística, segue o Algoritmo 6.

O Algoritmo 6 possui como parâmetros o conjunto de polígonos, a largura L da faixa, a quantidade de iterações que a heurística deve utilizar para construir uma solução e a função de ordenação das peças. No inicio de cada iteração as peças são ordenadas de acordo com a função de ordenação recebida. A cada iteração que a heurística gera uma solução, é verificado se tal solução é melhor que a melhor solução já encontrada.

A função *gera_uma_solucao* é a que gera as soluções para serem avaliadas pela heurística HMP. Tal função pode ser observada no Algoritmo 7.

O Algoritmo 7 recebe como parâmetros os polígonos já ordenados e a largura da faixa. Para cada peça presente no conjunto de polígonos, uma função de posicionamento é sorteada com probabilidade igual de 50%. Após o sorteio da função, um laço de repetição passa por todas as arestas da peça, chamando a função rotaciona que irá rotacionar a peça de acordo com as duas regras de rotação propostas. Depois que a peça é rotacionada, é verificado se a aresta escolhida, faz com que a função objetivo diminua, caso seja verdade, o algoritmo salvar as coordenadas da peça em uma variável (peca_final) e o melhor valor de função objetivo. Após todas as arestas serem testadas, a peça P[i] recebe as coordenadas salvas da melhor rotação. O algoritmo para quando todas as peças tiverem sido colocadas, então temos uma nova solução para o HMP.

Figura 4.15 – Exemplo da regra β.



Fonte: Do Autor (2021)

4.7 Algoritmo Genético

O Algoritmo Genético (AG) é uma meta-heurística inspirada na evolução das espécies. O método utiliza o conceito de indivíduos (representa as soluções utilizando cromossomos), seleção dos mais fortes (elitismo), cruzamento (troca de informações entre indivíduos), mutações (alterações de regras que criam as soluções), adaptação dos indivíduos ao meio (avaliação do *fitness* de um indivíduo). O AG foi introduzido por Holland et al. (1992) e, desde então, muitos trabalhos utilizam esta abordagem. No trabalho de Mundim, Andretta e Queiroz (2017) é apresentado um algoritmo genético *BRKGA* para o problema de *Nesting*. Com base naquele trabalho, foi adotada esta abordagem.

Inicializamos o AG ordenando as peças de acordo com uma função de ordenação recebida. Logo após, uma lista de indivíduos vazia é criada para iniciar a geração da população inicial. Então, uma rotina de geração da população inicial é acionada para criar os indivíduos que irão compor a primeira geração. Enquanto um critério de parada não é aceito (geralmente uma quantidade de gerações ou tempo computacional), o algoritmo seleciona os melhores indivíduos da população, ou seja, aqueles que contêm

Algorithm 5 Regra β

Função REGRA β (P, i, L)
    posicoes_possiveis $\leftarrow \{\}$
    Enquanto $P[i]$ não atingita L **Faça**
        move_para_direita(P[i])
        poligono_abstrato $\leftarrow (x_{min}(P[i]),0),(x_{max}(P[i]),0),((x_{max}(P[i]),\infty),(0,\infty)$
        Para $j \leftarrow 0$ **até Tamanho**(A) **Faça**
            Se poligono_abstrado sobrepoe $A[j]$ **Então**
                poligonos_sobrepostos $\leftarrow A[j]$
        Para $j \leftarrow 0$ **até Tamanho**(poligonos_sobrepostos) **Faça**
            Coloca $P[i]$ em cima poligonos_sobrepostos[j]
            Se $P[i]$ não sobrepõe nenhum polígono **Então**
                $P[i] \cup$ posicoes_possiveis
    remove_posicoes_invalidas(posicoes_possiveis)
    $P[i] \leftarrow$ posicao_baixo_esquerda(posicoes_possiveis)
    desliza_poligono($P[i]$)
Retorna $P[i]$

Algorithm 6 Heurística de Múltiplos Posicionamentos

Função HEURÍSTICA DE MÚLTIPLOS POSICIONAMENTOS (P, L, IT, FUNCAO_ORDENACAO)
    melhor_resultado $\leftarrow \infty$
    Para $i \leftarrow 0$ **até** IT **Faça**
        P $\leftarrow$ *ordena*(P, FUNCAO_ORDENACAO)
        resultado $\leftarrow$ GERA_UMA_SOLUCAO(P, L)
        Se melhor_resultado $\leq$ resultado **Então**
            melhor_resultado $\leftarrow$ resultado
    Retorna melhor_resultado

Algorithm 7 Gerador de soluções

Função GERA_UMA_SOLUCAO (P, L)
    Para $i \leftarrow 0$ **até Tamanho** (P) **Faça**
        melhor_sol $\leftarrow \infty$
        funcao_posicionamento $\leftarrow$ sorteia_regras(regra_pd, regra_pe, 50%)
        Para $j \leftarrow$ todas arestas de P[i] **Faça**
            P[i] $\leftarrow$ rotaciona(P, i, j)
            P[i] $\leftarrow$ funcao_posicionamento(P, i, L)
            Se calcula_fo(P) < melhor_sol **Então**
                peca_final $\leftarrow$ P[i]
                melhor_sol $\leftarrow$ calcula_fo(P)
        P[i] $\leftarrow$ peca_final
    Retorna melhor_sol

o melhor *fitness*, e descarta os restantes. Com os melhores indivíduos na população selecionados, novos indivíduos são criados (cruzamento). Após um indivíduo ser criado, um fator de mutação é calculado para saber se ele sofrerá alguma modificação em sua composição, como pode ser visto no Algoritmo 8.

Algorithm 8 Algoritmo Genético para o problema de *Nesting* com rotações livres.

```
1: Função ALGORITMO_GENETICO (P, TAMANHO_POP, GERACOES, PORCENTAGEM_ELITE, VALOR_MUTACAO, FUNCAO_ORDENACAO)
2:     Peças ← ordena(FUNCAO_ORDENACAO, P)
3:     pop ← inicial_pop(TAMANHO_POP, P)
4:     avalia_pop(pop)
5:     Para i ≤GERACOES Faça
6:         pop ← pop_elite(PORCENTAGEM_ELITE)
7:         novos_individuos ← gera_individuos(pop, VALOR_MUTACAO)
8:         pop ← pop ∪ novos_individuos
9:         avalia_pop(pop)
       Retorna melhor_individuo(pop)
```

Na Subseção 4.7.1, apresentamos como os indivíduos do Algoritmo Genético são criados e quais são as informações que compõem cada cromossomo. Na Subseção 4.7.2, mostramos como são feitas as trocas genéticas entre dois indivíduos para gerar um novo, e como este indivíduo pode sofrer uma mutação para escapar de ótimos locais.

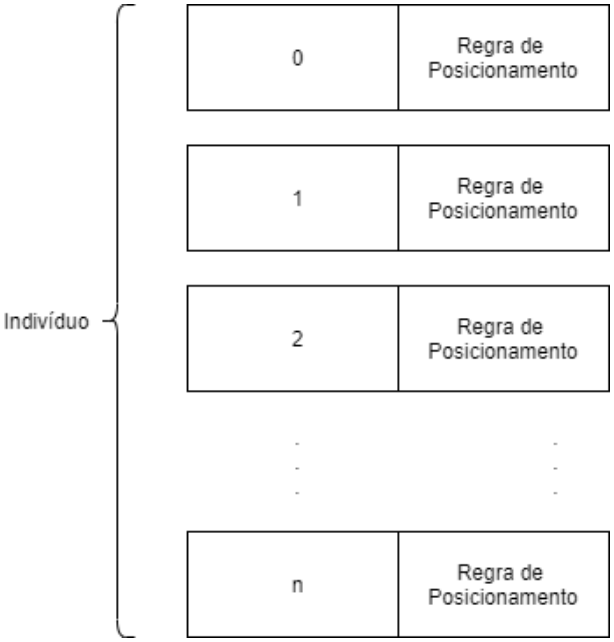
4.7.1 Geração dos indivíduos

Cada indivíduo é composto por uma sequência de cromossomos de tamanho exatamente iguais à quantidade de polígonos recebida de entrada no algoritmo. Cada cromossomo é composto por um par: índice da peça e regra de posicionamento. O índice de cada peça deve estar presente em algum cromossomo do indivíduo e este índice deve estar em um par com uma das regras de posicionamento, sendo elas α (PD) ou β (PE). Tal modelagem pode ser vista na Figura 4.16.

Para gerar uma solução a partir de cada indivíduo, uma rotina percorre cada cromossomo, inserindo na faixa a peça representada no cromossomo usando a regra de posicionamento (que é seu par no cromossomo). Ao inserir a peça, são testadas as rotações dadas pelas regras RA e RB (no caso da regra RB, que precisa escolher uma aresta da peça, todas as arestas são testadas). Dentre todas as rotações testadas, é escolhida aquela que gera um empacotamento parcial com menor altura da faixa (ou seja, com menor valor da função objetivo).

Para criar a população inicial, as peças são ordenadas de acordo com a regra de ordenação recebida como parâmetro para o Algoritmo Genético. Em seguida, cada indivíduo é gerado de forma que o i-ésimo cromossomo tenha o índice da i-ésima peça na ordenação feita. Uma das regras de posicionamento (PD ou PE) é escolhida para cada cromossomo de forma aleatória, com igual probabilidade para ambas as regras. Este processo é repetido até todos indivíduos da população inicial serem criados.

Figura 4.16 – Esquema do indivíduo.

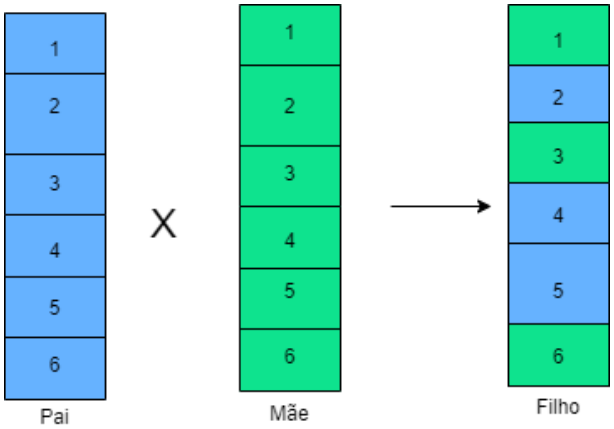


Fonte: Do Autor (2021)

4.7.2 Cruzamento e mutação

Quando os melhores indivíduos são mantidos na população, novos indivíduos serão gerados. Escolhemos aleatoriamente dois indivíduos pais, com probabilidades iguais. Para cada cromossomo que irá compor o novo indivíduo, é escolhido aleatoriamente se o cromossomo virá do "pai" ou da "mãe". Um exemplo pode ser observado na Figura 4.17.

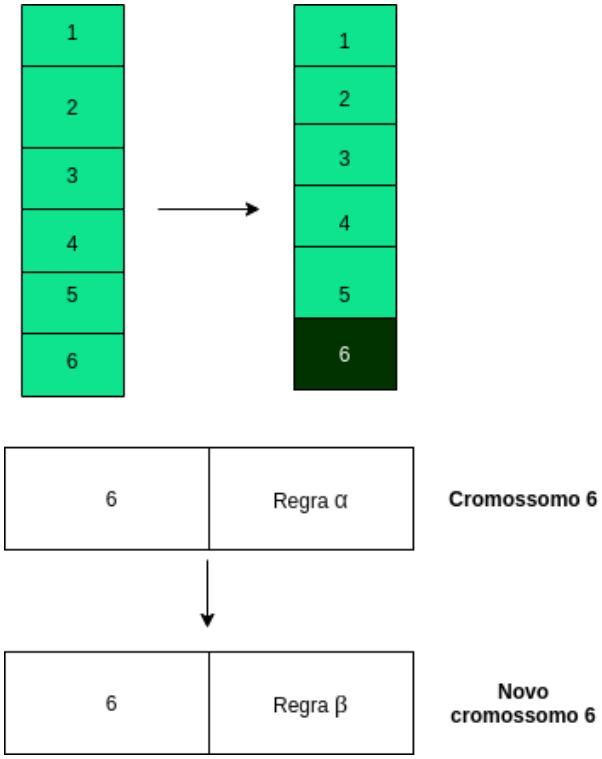
Figura 4.17 – Exemplo de cruzamento.



Fonte: Do Autor (2021)

Quando um novo indivíduo é gerado, uma rotina é aplicada sobre ele para decidir se alguma posição dentro de seus cromossomos será alterada. Dada uma probabilidade de mutação, é trocada a regra de posicionamento do cromossomo (passa de PD para PE ou vice-versa).

Figura 4.18 – Mutação ocorrida.



Fonte: Do Autor (2021)

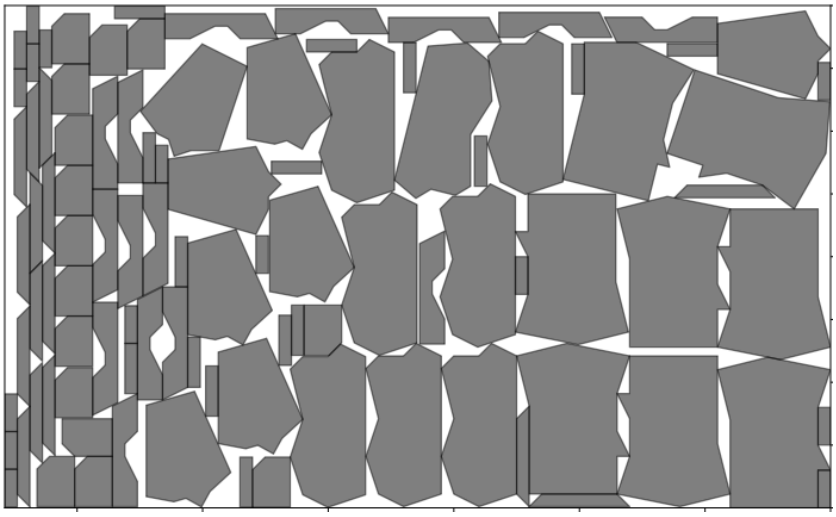
Como pode ser observado na Figura 4.18, um único cromossomo do indivíduo foi escolhido para sofrer a mutação (Cromossomo 6). Após a escolha do cromossomo, sua regra de posicionamento foi alterada de regra α (PD) para regra β (PE).

Cada indivíduo da população do Algoritmo Genético tem um determinado *fitness* (função objetivo). Em nossa abordagem, a população de indivíduos sempre está ordenada de maneira crescente de acordo com o *fitness* (altura do leiaute).

4.8 Visualizador

Para uma melhor visualização da solução, e do comportamento do algoritmo, foi criado um módulo para acompanhar passo a passo como a solução está sendo construída. Além disso, é possível verificar se alguma peça está se sobrepondo a outra, ou se alguma peça está sendo alocada fora da faixa. Um exemplo de saída do visualizador pode ser visto na Figura 4.19.

Figura 4.19 – Teste da instância shirts.



Fonte: Do Autor (2021)

O visualizador foi implementado com ajuda a biblioteca *matplotlib* (HUNTER, 2019) da linguagem Python. Esta biblioteca provê um conjunto de funções para desenhar gráficos, polígonos e pontos. Também foi utilizada uma de suas funcionalidades denominada *animation*, que possibilita desenhar os polígonos sequencialmente, na mesma ordem em que o algoritmo constrói a solução.

A importância de se criar este visualizador é ter a possibilidade de acompanhar o funcionamento de heurísticas ao logo da construção da solução. Além disso, tal ferramenta possibilita observar maneiras de rotacionar os polígonos, ou ordená-los, para que não haja desperdício de espaço na faixa. O código fonte do visualizador está disponível em Nunes (2019).

5 RESULTADOS COMPUTACIONAIS

Este capítulo avalia o desempenho dos algoritmos implementados na resolução de instâncias clássicas do problema de *Nesting* com rotações livres. Comparamos nossos resultados com as soluções reportadas por Peralta, Andretta e Oliveira (2017). A Seção 5.1 descreve as características das instâncias utilizadas para os testes. Na Seção 5.2, mostramos os resultados das heurísticas construtivas que utilizam somente uma combinação de regras de posicionamento, ordenação e rotação. A Seção 5.3 apresenta os resultados obtidos pela Heurística de Múltiplos Posicionamentos. Já a Seção 5.4 apresenta os resultados encontrados pelo Algoritmo Genético. A Seção 5.5 apresenta uma comparação de todos os nossos métodos propostos. Por fim, a Seção 5.6 apresenta uma comparação do nosso melhor método com os resultados presentes no trabalho de Peralta, Andretta e Oliveira (2017).

Os testes foram realizados um computador *AMD Ryzen 5 3600 6-Core Processor 3.59GHz*, com uma arquitetura x86_64, 16 GB de memória RAM, no sistema operacional Ubuntu 18.04 LTS. O código foi implementado na linguagem *Python* 3.

5.1 Instâncias de teste

As instâncias testadas contêm peças convexas e não convexas, e cada tipo de peça pode ter uma quantidade unitária ou réplicas da mesma. O repositório fonte foi o ESICUP (CUTTING; PACKING, 2019), base de dados com instâncias clássicas da literatura, em que cada instância contém uma determinada quantidade de peças e uma largura para a faixa. Os ângulos pré-definidos de algumas peças serão ignorados, levando em conta a característica de rotação livre abordada em nosso contexto.

Na Tabela 5.1 apresenta quais instâncias foram utilizadas, e quantidade de peças que cada uma contém.

5.2 Experimento 1: Heurísticas construtivas

Cada regra de posicionamento apresentada na Seção 4.5 foi implementada e combinada com uma regra de ordenação, gerando vinte e cinco heurísticas construtivas (5 regras de posicionamento $\times$ 5 regras de ordenação = 25 combinações). Para cada uma destas heurísticas construtivas, no momento de alocar uma peça na faixa, é necessário escolher uma rotação. Para isso, foram usadas as regras de rotação RA e RB definidas na Seção 4.4.

A regra de rotação RB precisa de uma aresta e um eixo para ser aplicada. Para as heurísticas que usam as regras de posicionamento *Bottom-Left* (PA), *Bottom-Left* modificada (PB) e Gulosa (PC) (independentemente da regra de ordenação usada), sempre é fornecida a maior aresta do polígono, e

Tabela 5.1 – Instâncias utilizadas.

Instância	Quantidade de peças	Largura da faixa
albano	24	4.900
blaz	24	15
dighe2	10	100
han	23	58
jakobs	25	40
jakobs2	25	70
mao	20	2.550
marques	24	104
poly1a	15	40
shapes	43	40
shirts	99	40
trousers	66	79

Fonte: Do autor (2021).

esta ficará paralela ao eixo *y*. Para as heurísticas que usam as demais regras de posicionamento, foram testadas todas as arestas e os dois eixos (*x* e *y*); o ângulo e eixo em que a função objetivo produz o menor valor logo após a inserção da peça rotacionada serão os escolhidos. Depois de obter a aresta e o eixo que gera o menor valor de função objetivo, o método de rotação RA foi utilizado para testar se a função objetivo teria alguma melhora (em caso positivo, a rotação obtida por esta regra é escolhida).

A escolha das heurísticas que usam as regras *Bottom-Left* (PA), *Bottom-Left* modificada (PB) e Gulosa (PC) terem a característica de receberem a maior aresta e rotacionarem em relação ao eixo *y* é justificada devido a testes preliminares, que mostraram um bom compromisso entre tempo computacional e qualidade de solução. Com as outras regras isso não aconteceu.

Executamos cada heurística construtiva para resolver as instâncias da Tabela 5.1, a fim de analisar qual delas gera os melhores resultados. Na Tabela 5.2, na coluna "*Instância*", temos as instâncias que foram utilizadas. A coluna "*Valor de solução*" mostra qual foi o melhor resultado que a heurística construtiva que usa uma dada regra de posicionamento obteve usando todas as regras de ordenação. A coluna "*t(s)*" mostra o tempo (em segundos) que a solução levou para ser construída. Por fim, a coluna "*Reg ord*" mostra qual foi a regra de ordenação que foi utilizada, juntamente com a regra de posicionamento, para encontrar a solução.

Os tempos de execução de cada instância são relativamente baixos. Assim, essas regras admitem modificações a fim de aprimorar os resultados.

A heurística que utilizou a Regra α (PD) encontrou 5 dos melhores resulta-dos entre as 13 instâncias, enquanto as heurísticas que utilizaram as regras β (PE) e Gulosa (PC) foram superiores em 6

Tabela 5.2 – Melhores resultados das heurísticas implementadas, para cada regra de posicionamento.

	Regra de posicionamento														
	Bottom-Left			*Bottom-Left* modificada			Gulosa			Regra α			Regra β		
Instância	Valor solução	t(s)	Reg Ord	Valor solução	t(s)	Reg Ord	Valor solução	t(s)	Reg Ord	Valor solução	t(s)	Reg Ord	Valor solução	t(s)	Reg Ord
albano	13.364,00	16,98	SC	12.522,00	0,03	SD	11.535,00	7,71	SD	**11.045,80**	54,47	SA	11.175,41	33,04	SD
blaz	37,99	0,58	SC	38,00	0,04	SC	**31,60**	0,77	SE	31,65	22,28	SA	32,51	10,87	SD
dighe2	159,70	0,96	SC	164,46	0,01	SD	141,60	3,40	SA	142,93	1,73	SD	**137,71**	5,29	SC
han	60,69	2,00	SD	66,00	0,02	SA	51,10	3,43	SE	**47,10**	27,07	SA	47,20	21,47	SD
jakobs	17,92	1,26	SD	18,00	0,02	SD	14,00	0,72	SE	13,09	22,81	SA	**12,99**	11,30	SA
jakobs2	35,70	2,31	SB	34,14	0,03	SB	30,70	2,77	SA	28,45	15,27	SE	**28,00**	19,51	SD
mao	2.681,00	8,24	SC	2.510,00	0,01	SB	2.113,00	35,08	SB	**1.913,45**	37,34	SA	1.971,11	84,42	SA
marques	99,10	2,21	SE	98,00	0,02	SD	93,10	11,24	SA	85,89	25,80	SA	**83,70**	36,10	SB
poly1a	23,03	0,55	SE	29,37	0,02	SC	18,30	0,37	SD	**16,28**	2,24	SB	17,08	4,00	SE
shapes	74,33	4,09	SB	78,97	0,08	SE	**62,10**	12,70	SB	63,97	306,17	SA	67,09	103,94	SB
shirts	80,59	16,40	SD	75,00	0,20	SB	68,00	31,07	SB	65,75	358,38	SA	**64,96**	211,74	SD
trousers	322,88	27,30	SD	330,78	0,11	SA	285,69	76,75	SA	264,48	117,46	SE	**253,04**	281,06	SA

Fonte: Do autor (2021).

e 2 problemas teste, respectivamente. As heurísticas que utilizaram as regras *Bottom-Left* (PA) e *Bottom-Left modificada* (PB) não obtiveram resultados tão significativos como os outros métodos propostos.

Tabela 5.3 – Melhores soluções encontradas para cada instância e qual combinação de regra de posicionamento e ordenação que gerou cada solução.

Instância	Regra de Posicionamento	Regra de Ordenação	Valor solução	t(s)
albano	Regra α (PD)	SE	11.045,80	54,47
blaz	Gulosa (PC)	SE	31,60	0,77
dighe2	Regra β (PE)	SC	137,71	5,29
han	Regra α (PD)	SA	47,10	27,07
jakobs	Regra β (PE)	SA	12,99	11,30
jakobs2	Regra β (PE)	SD	28,00	19,51
mao	Regra α (PD)	SA	1.913,45	37,34
marques	Regra β (PE)	SB	83,70	36,10
poly1a	Regra α (PD)	SB	16,28	2,24
shapes	Gulosa (PC)	SB	62,10	12,70
shirts	Regra β (PE)	SD	64,96	211,74
trousers	Regra β (PE)	SA	253,04	281,06

Fonte: Do autor (2021).

Na Tabela 5.3, apresentamos, para cada instância, a combinação de regras que levou ao melhor valor de solução. A maioria dos resultados foram gerados a partir das regras de posicionamento Regra α (PD) e Regra β (PE). Nas regras de ordenação, não houve uma dominância como nas regras de posicionamento. Nas instâncias testadas, o regra SA, que ordena os polígonos de acordo com sua área, foi a melhor em 5 casos, seguida da regra SD, que obteve êxito em 3 problemas. Os critérios SB, SE e SB obtiveram a melhor solução em 2, 2 e 1 instâncias das 13 utilizadas, respectivamente. Para a regra de rotação, todos os melhores resultados obtidos utilizaram o método RB.

Como mencionado anteriormente, as heurísticas que geraram os melhores resultados entre as abordagens implementadas foram as que usaram as regras de posicionamento α (PD) e β (PE). O motivo da eficácia dessas abordagens tem relação as características que são levadas em conta nas possibilidades do posicionamento das peças, pois tais regras testam uma quantidade bem maior de posições que as demais.

A heurística que utilizou a regra de rotação β foi a heurística que obteve na maioria dos casos os melhores resultados. Na instância "dighe2", a regra de ordenação é a SC, na instância "marques" a regra de ordenação é a SB, nas instâncias "shirts" e "jakobs2" a regra de ordenação é SD e nas instâncias "jakobs" e "trouser", SA.

Após analisar os resultados das heurísticas que usam a regra de posicionamento β, ordenação SD, regra de posicionamento β e ordenação SA, notamos que nas demais instâncias, quando a ordenação

se dá pela regra SA, os resultados são melhores. Com isso, concluímos que a heurística construtiva que utiliza a regra de posicionamento β (PE) e ordenação SA (que passaremos a chamar de PA-SE) é a melhor dentre as implementadas. As figuras que representam as soluções encontradas pela PA-SE podem ser vistas no Apêndice A.

5.3 Experimento 2: Heurística de Múltiplos Posicionamentos (HMP)

Nesta seção, apresentamos os resultados computacionais obtidos pela Heurística de Múltiplos Posicionamentos (HMP). O critério de parada da HMP foi definido como 100 iterações ou 1800 segundos. Caso o tempo limite seja atingido, a HMP devolve a melhor solução encontrada até o momento. Tais valores foram definidos a partir de experimentos preliminares. Como a HMP realiza sorteios, cada execução pode gerar soluções diferentes. Por isso, ela foi executada 10 vezes para cada instância.

A Tabela 5.4 mostra os resultados da aplicação da Heurística de Múltiplos Posicionamentos proposta para resolver todas as instâncias. A coluna "*Instância*" mostra o nome da instância. As colunas agrupadas em "*Melhor solução encontrada*" apresentam dados sobre a melhor solução encontrada pela HMP dentre as 10 execuções, enquanto que as colunas agrupadas em "*Pior solução encontrada*" apresentam dados sobre a pior solução encontrada pela HMP dentre as 10 execuções. As colunas "*Número iterações*" apresentam quantas iterações foram realizadas para encontrar cada solução e as colunas "*Tempo (s)*" indicam quanto tempo (em segundos) foi usado para encontrar cada solução. As colunas "*Valor solução*" mostram o valor da solução encontrada, e a coluna "*Iteração encontrou*" mostra em qual iteração a HMP encontrou tal solução. Por fim, a coluna "*Desv M/P*" mostra qual a porcentagem de desvio da melhor solução em relação à pior, utilizando a fórmula dada por: $\frac{s^b - s^*}{s^*} \times 100$, em que s^b é a pior solução encontrada pela HMP e s^* é a melhor solução reportada.

Em instâncias maiores, como a "shapes", "trousers", "shirts" e "han", poucas iterações foram executadas, pois são instâncias maiores, com peças muito complexas.

Podemos observar que o desvio do melhor resultado em relação ao pior resultado de algumas instâncias é um valor elevado, já em outras não. Nas instâncias "jakobs" e "jakobs2" apresentam os maiores valores de desvios, com $16,25\%$ e $18,75\%$, respectivamente. Nas instâncias "blaz", "mao", "marques" e "shapes", os desvios ficaram bem parecidos, todos entre 6% e 7%. O menor desvio encontrado foi na instância "shirts" com o valor de $3,22\%$. As figuras que representam as soluções encontradas pela Heurística de Múltiplos Posicionamentos podem ser vistas no Apêndice B.

Tabela 5.4 – Melhores e piores resultados encontrados pela Heurística de Múltiplos Posicionamentos.

	Melhor solução encontrada				Pior solução encontrada				
Instância	Número iterações	Tempo (s)	Valor solução	Iteração encontrou	Número iterações	Tempo (s)	Valor solução	Iteração encontrou	Desv M/P
albano	50	1.800,00	10.643,66	16	48	1.800,00	11.557,45	45	8,59
blaz	100	861,50	29,65	59	100	901,10	31,59	76	6,54
dighe2	100	104,11	127,25	57	100	96,30	140,73	5	10,59
han	100	1.657,10	44,76	88	100	1537,36	49,71	3	11,06
jakobs	100	893,80	12,00	18	100	669,03	13,95	79	16,25
jakobs2	100	591,79	25,81	27	100	543,31	30,65	91	18,75
mao	21	1.800,00	1.907,31	17	21	1.800,00	2.022,64	20	6,05
marques	71	1.800,00	83,50	69	82	1.800,00	88,60	66	6,11
poly1a	100	88,16	14,55	32	100	89,23	17,34	43	19,18
shapes	7	1.800,00	61,97	1	9	1.800,00	65,97	2	6,45
shirts	7	1.800,00	63,21	4	7	1.800,00	65,53	1	3,67
trousers	10	1.800,00	250,87	4	11	1.800,00	271,70	5	8,30

Fonte: Do autor (2021).

5.4 Experimento 3: Algoritmo Genético

Nesta seção, apresentamos o resultados obtidos pelo Algoritmo Genético. Para a definição de seus parâmetros, utilizamos a ferramenta *HypertOPT* (BERGSTRA; YAMINS; COX, 2013). Os valores testados foram:

- número de gerações: {10, 20, 30, 40, 50};
- tamanho da população: {5, 15, 30, 40, 50};
- elitismo (tamanho da população que sobrevive a cada geração): {10%, 25%, 50% 75%};
- probabilidade de mutação: {10%, 25%, 50% 75%}.

Após a execução da ferramenta utilizando as instâncias "blaz", "dighe2" e "jakobs", os parâmetros calibrados foram:

- número de gerações: 50;
- tamanho da população: 15;
- elitismo (tamanho da população que sobrevive a cada geração): 25%;
- probabilidade de mutação: 25%.

Após calibrar o Algoritmo Genético, decidiu-se que o algoritmo teria um limite de tempo de 1.800 segundos. Caso ele atinja esse limite de tempo, ele devolve o indivíduo com o melhor *fitness* entre os gerados, o que corresponde a uma solução com melhor valor de função objetivo. Por conter aleatoriedade, cada execução do Algoritmo Genético pode gerar soluções diferentes. Por isso, ele foi executado dez vezes para cada instância.

A Tabela 5.5 mostra os resultados da aplicação do Algoritmo Genético proposto para resolver todas as instâncias. A coluna "*Instância*" mostra o nome da instância. As colunas agrupadas em "*Melhor solução encontrada*" apresentam dados sobre a melhor solução encontrada pelo Algoritmo Genético dentre as 10 execuções, enquanto que as colunas agrupadas em "*Pior solução encontrada*" apresentam dados sobre a pior solução encontrada pelo Algoritmo Genético dentre as 10 execuções. As colunas "*Número gerações*" apresentam quantas gerações foram necessárias para encontrar cada solução e as colunas "*Tempo (s)*" indicam quanto tempo (em segundos) foi usado para encontrar cada solução. As colunas "*Valor solução*" mostram o valor da solução encontrada, e a coluna "*Iteração encontrou*" mostra em qual geração o Algoritmo Genético encontrou tal solução. Por fim, a coluna "*Desv M/P*" mostra qual

Tabela 5.5 – Melhores e piores resultados do Algoritmo Genético.

	Melhor solução encontrada				Pior solução encontrada				
Instância	Número gerações	Total t(s)	Valor solução	Iteração encontrou	Número gerações	Total t(s)	Valor solução	Iteração encontrou	Desv M/P
albano	16	1.800,00	10.578,35	8	18	1.800,00	11.260,80	3	6,45
blaz	50	1.123,72	29,59	48	50	1.312,98	32,67	36	10,41
dighe2	50	150,31	134,64	1	50	153,07	151,06	6	12,20
han	37	1.800,00	45,00	1	41	1.800,00	50,00	0	11,11
jakobs	50	1.239,44	12,46	22	50	915,91	14,63	1	17,42
jakobs2	50	240,96	25,78	4	50	704,56	30,73	1	19,20
mao	6	1.800,00	1.933,59	4	7	1.800,00	2.033,14	1	5,15
marques	31	1.800,00	83,70	6	30	1.800,00	89,69	26	7,16
poly1a	50	112,24	15,44	3	50	113,97	17,32	6	12,18
shapes	2	1.800,00	62,10	1	3	1.800,00	65,97	3	6,23
shirts	2	1.800,00	64,01	1	2	1.800,00	66,07	2	3,22
trousers	3	1.800,00	249,17	1	3	1.800,00	265,62	2	6,60

Fonte: Do autor (2021).

a porcentagem de desvio da melhor solução em relação à pior, utilizando a fórmula dada por: $\frac{s^b - s^*}{s^*} \times 100$, em que s^b é a pior solução encontrada pelo Algoritmo Genético e s^* é a melhor solução reportada.

Em instâncias maiores como a "shapes", "trousers" e "shirts", poucas gerações foram executadas, pois são instâncias maiores, com peças muito complexas.

Podemos observar que o desvio do melhor resultado em relação ao pior resultado de algumas instâncias é um valor elevado, já em outras não. As instâncias "jakobs" e "jakobs2" apresentam os maiores valores de desvios com $17,42\%$ e $19,20\%$, respectivamente. Porém em instâncias como "mao" e "shapes", os valores são bem menores: $5,15\%$ e $3,22\%$, respectivamente. As figuras que representam as soluções encontradas pelo Algoritmo Genético podem ser vistas no Apêndice C.

5.5 Comparação entre os métodos propostos

Após a implementação dos métodos propostos e a apresentação de seus resultados, iremos comparar os resultados encontrados com nossas abordagens, para identificar qual delas apresenta os melhores resultados. Assim, iremos comparar os resultados obtidos pela heurística construtiva PE-SA (regra de posicionamento β (PE) e a regra de ordenação SA), com os resultados obtidos pela HMP (Heurística de Múltiplos Posicionamentos) e pelo Algoritmo Genético.

Na Tabela 5.6, temos as comparações dos valores de soluções da PE-SA, da HMP e do Algoritmo Genético. Na coluna *"Instância"*, vemos quais instâncias foram testadas e comparadas. A coluna "*Valor de solução*" mostra qual foi o melhor resultado que o método proposto obteve. A coluna "*t(s)*" mostra o tempo (em segundos) que o método levou para calcular a melhor solução. Por fim, na coluna "*Desv (%) HMP/AG*" mostra qual foi o desvio percentual da Heurística de Múltiplos Posicionamentos em relação ao Algoritmo Genético. Para calcularmos o desvio, utilizamos a fórmula: $\frac{s^b - s^*}{s^*} \times 100$, em que s^b é a melhor solução encontrada pelo Algoritmo Genético proposto e s^* é a melhor encontrada pela Heurística de Múltiplos Posicionamentos.

Como podemos observar na Tabela 5.6, os valores de solução obtidos pela PE-SA, para todas as instâncias, são maiores em relação aos obtidos pela HMP e pelo Algoritmo Genético. No entanto, seu tempo de execução é muito inferior. Isso já era esperado, já que esta heurística é muito mais simples do que as demais.

Comparando o Algoritmo Genético e a HMP, vemos que, para oito das doze instâncias testadas, a HMP obteve o melhor valor de função. Vemos que nas instâncias "poly1a" e "dighe2" há os maiores desvios encontrados a favor da Heurística de Múltiplos Posicionamentos, com os valores de $6,16\%$ e $5,81\%$ respectivamente. Já os maiores desvios encontrados a favor do Algoritmo Genético proposto são para as instâncias "trousers" e "albano", com os valores de $0,68\%$ e $0,61\%$, respectivamente.

Tabela 5.6 – Comparação dos resultados encontrados pela heurística PE-SA, pela HMP e pelo Algoritmo Genético.

	PE-SA		HMP		Algoritmo Genético		
Instância	Valor solução	t(s)	Valor solução	t(s)	Valor solução	t(s)	Desv (%) HMP/AG
albano	11.458,88	**33,33**	10.643,66	1.800,00	**10.578,35**	1.800,00	-0,61
blaz	33,26	**10,56**	29,65	861,50	**29,59**	1,123,72	-0,20
dighe2	141,79	**4,9**4	**127,25**	104,11	134,64	150,31	5,81
han	51,36	**22,52**	**44,76**	1,657,10	45,00	1.800,00	0,54
jakobs	12,99	**11,30**	**12,00**	893,80	12,46	1,239,44	3,83
jakobs2	31,63	2**0,40**	25,81	591,79	**25,78**	240,96	-0,12
mao	1.971,11	**84,42**	**1.907,31**	1.800,00	1.933,59	1.800,00	1,38
marques	87,50	**39,75**	**83,50**	1.800,00	83,70	1.800,00	0,24
poly1a	18,74	**4,00**	**14,55**	88,16	15,44	112,24	6,16
shapes	67,97	**105,59**	**61,97**	1.800,00	62,10	1.800,00	0,21
shirts	65,10	**213,92**	**63,21**	1.800,00	64,01	1.800,00	1,27
trousers	253,04	**281,06**	250,87	1.800,00	**249,17**	1.800,00	-0,68

Fonte: Do autor (2021).

Em relação ao tempo de execução, vemos que em seis das doze instâncias, os algoritmos atingiram o limite de tempo de execução. Em relação às demais instâncias, a HMP foi mais rápida que o Algoritmo Genético em cinco instâncias. Somente na instância "Jakobs2" o Algoritmo Genético conseguiu obter uma solução mais rapidamente do que a HMP.

5.6 Comparação entre a melhor heurística proposta e um método da literatura

Após a comparação entre os métodos propostos, foi possível observar que a HMP obteve os melhores resultados na resolução das instâncias testadas. Assim, iremos comparar seus resultados com os resultados encontrados no trabalho de Peralta, Andretta e Oliveira (2017). Neste trabalho, os autores geram dez pontos iniciais, usam um Método de Pontos Interiores para encontrar uma solução para um modelo de programação não linear usando cada um dos diferentes pontos iniciais. São reportados o melhor valor de solução encontrado e a média de tempo de execução para o cálculo das dez diferentes soluções.

A Tabela 5.7 compara a qualidade das soluções apresentadas na Tabela 5.4 com o trabalho Peralta, Andretta e Oliveira (2017). As colunas agrupadas por "*Heurística de Múltiplos Posicionamentos*" apresentam os dados referentes à HMP, enquanto as colunas agrupadas por "*Literatura*" apresentam os dados referentes a Peralta, Andretta e Oliveira (2017). Em relação aos dados da HMP, a coluna "*Melhor sol. Encontrada*" mostra o melhor valor de função objetivo encontrado. A coluna "*Tempo (s)*" mostra o tempo de execução (em segundos) da HMO para encontrar esta solução. A coluna "*t(s) por iteração*" apresenta a média de tempo (em segundos) gasto em cada iteração (ou seja, é o tempo gasto para encontrar a solução dividido pelo número de iterações gastas para encontrá-la). Em relação aos dados de Peralta, Andretta e Oliveira (2017), a coluna "*Melhor sol. Encontrada*" mostra o valor da melhor solução encontrada e a coluna "*Tempo médio (s)*" mostra a média de tempo (em segundos) gasto para encontrar essa solução (ou seja, calcula-se a média dos tempos usados para encontrar as dez soluções). Avaliamos o desempenho do nosso algoritmo com base no desvio percentual "*Desv.(%) HMP/Lit*" em relação à melhor solução em Peralta, Andretta e Oliveira (2017), dada por: $\frac{s^b - s^*}{s^*} \times 100$, em que s^b é a melhor solução encontrada pela Heurística de Múltiplos Posicionamentos proposta e s^* é a melhor solução relatada por Peralta, Andretta e Oliveira (2017) .

Como podemos observar, em três das oito instâncias comparadas, nossa abordagem tem um valor de função objetivo (coluna "*Melhor sol.*") menor, ou seja, melhor que o de Peralta, Andretta e Oliveira (2017). Porém, nas outras instâncias, nossa abordagem não supera os resultados encontrados pelos autores. A comparação entre os tempos de execução não é muito simples, já que ambientes computacionais diferentes foram usados, além de a forma de cálculo de tempo ter sido diferente. No entanto, se compa-

Tabela 5.7 – Comparação dos melhores encontrados pela Heurística de Múltiplos posicionamentos com o trabalho de Peralta, Andretta e Oliveira (2017).

	Heurística de Múltiplos Posicionamentos			Literatura		
Instância	Melhor Sol. Encontrada	Tempo (s)	t(s) por iteração	Melhor Sol. Encontrada	Tempo médio (s)	Desv(%) HMP/Lit
albano	10.643,66	1.800,00	**36,00**	**10.335,80**	337,24	2,98
blaz	29,65	861,50	**8,62**	**27,82**	50,89	6,58
jakobs	**12,00**	893,80	**8,94**	12,99	23,98	-7,62
jakobs2	**25,81**	591,79	**5,92**	26,00	36,01	-0,73
marques	**83,50**	1.800,00	**25,35**	84,65	124,85	-1,36
poly1a	14,55	88,16	**0,88**	**14,01**	25,30	3,85
shirts	63,21	1.800,00	**257,14**	**62,19**	63,21	1,64
trousers	250,87	1.800,00	**180,00**	249,35	721,36	0,61

Fonte: Do autor (2021).

rarmos o tempo por iteração gastos pela HMP com os tempos médios apresentados em Peralta, Andretta e Oliveira (2017), que é uma comparação minimamente justa, os tempos de execução da HMP são, em geral, muito menores.

Por fim, iremos comparar todos os melhores resultados entre todos nossos métodos propostos, com os resultados presentes no trabalho de Peralta, Andretta e Oliveira (2017). Com essa comparação, conseguimos ver qual o limite de nossas abordagens, quando comparados com os resultados de Peralta, Andretta e Oliveira (2017).

A Tabela 5.8 mostra, na coluna "*Instância*", quais foram as instâncias comparadas com os resultados da literatura. A coluna "*Método*" indica qual dos métodos propostos encontrou a melhor solução. As colunas "*Melhor Sol. Encontrada*" apresentam quais foram os melhores resultados encontrados (tanto métodos propostos como por Peralta, Andretta e Oliveira (2017)). A coluna "*t(s) por iteração*" mostra o tempo médio (em segundos) por iteração gasto pelo método para encontrar a melhor solução. A coluna "*t(s) médio*" mostra o tempo médio gasto em Peralta, Andretta e Oliveira (2017) para calcular sua melhor solução. A coluna "*Desv(%)*" mostra qual foi o desvio entre nossa melhor solução e a melhores solução de Peralta, Andretta e Oliveira (2017).

Comparando com os resultados obtidos pelas nossas abordagens em relação aos melhores resultados obtidos por Peralta, Andretta e Oliveira (2017), vemos que em quatro das oito instâncias, nossos resultados foram melhores. Além disso, nossos tempos médios foram menores para todas as instâncias.

Tabela 5.8 – Comparação entre os melhores resultados encontrados pelos métodos propostos e por (PERALTA; ANDRETTA; OLIVEIRA, 2017)

	Nossas Abordagens			Literatura		
Instância	Método	Melhor Sol. Encontrada	Tempo it (s)	Melhor Sol. Encontrada	t(s) médio	Desv (%)
albano	AG	10.578,35	**112,50**	**10.335,80**	337,24	2,35
blaz	AG	29,59	**22,47**	**27,82**	50,89	6,36
jakobs	HMP	**12,00**	**8,94**	12,99	23,98	-7,62
jakobs2	AG	**25,78**	**4,82**	26,00	36,01	-0,85
marques	HMP	**83,50**	**25,35**	84,65	124,85	-1,36
poly1a	HMP	14,55	**0,88**	**14,01**	25,30	3,85
shirts	HMP	63,21	**257,14**	**62,19**	1.820,28	1,64
trousers	AG	**249,17**	**600,00**	249,35	721,36	-0,07

Fonte: Do autor (2021).

6 CONSIDERAÇÕES FINAIS E PROPOSTAS FUTURAS

Neste trabalho, estudamos o problema de *Nesting* com rotações livres. O problema consiste em encontrar a melhor posição de peças dentro de uma faixa, minimizando a altura em que as peças são posicionadas e, consequentemente, o espaço em que elas ocupam.

O problema que foi abordado neste trabalho é bidimensional, com peças convexas e não convexas, que podem rotacionar em qualquer ângulo. Ele contém algumas restrições que devem ser respeitadas para que uma solução apresentada seja válida: todas as peças devem ser alocadas; todas as peças devem estar dentro do limite fixo da placa; não haverá sobreposição de nenhuma peça.

Para responder à seguinte pergunta: “Existe alguma forma de melhorar a eficiência da resolução do problema de *Nesting* bidimensional com rotações livres, utilizando heurísticas?", foram propostas heurísticas construtivas, uma Heurística de Múltiplos Posicionamentos (HMP) e um Algoritmo Genético. Para compor os métodos propostos, foram implementadas cinco regras de posicionamento, cinco regras de ordenação e duas regras de rotação.

Para a representação das peças, utilizamos a forma de polígonos, em que a peça é um conjunto ordenado de pontos (x, y) no plano cartesiano. A faixa é definida por uma constante L que representa sua largura fixa e uma variável W, que representa a altura utilizada da faixa, e terá seu valor minimizado. Para verificar a sobreposição das peças, foi proposto um algoritmo que transforma as peças do problema em um conjunto de triângulos (triangulação das peças). Com isso feito, para verificar se duas peças estão sobrepostas, basta verificar se seus respectivos triângulos se sobrepõem.

Para testar o desempenho dos métodos propostos, foram utilizadas doze instâncias clássicas da literatura (CUTTING; PACKING, 2019), todas com quantidades de peças variadas e peças convexas e não convexas. Os resultados dos métodos propostos foram comparados com os resultados obtidos por Peralta, Andretta e Oliveira (2017), que obteve seus resultados a partir da solução de um modelo não linear.

Com os resultados das heurísticas construtivas, foi possível constatar que algumas heurísticas que utilizaram certas regras de posicionamento obtiveram melhores resultados que as demais, a saber, as heurísticas que utilizaram as regras α e β. Após os testes foi possível concluir que a heurística que utilizou a regra de posicionamento β (PE) e a regra de ordenação *SA* foi a melhor dentre todas as heurísticas geradas a partir da combinação das regras de posicionamento e ordenação.

Com a utilização da Heurística de Múltiplos Posicionamentos, foi possível notar que, em algumas instâncias, com peças mais complexas, o algoritmo excedeu seu tempo de limite e não conseguiu executar todas as iterações. Também foi possível perceber que o desvio percentual entre o melhor e o pior resultado encontrado tem valores altos em algumas instâncias, chegando à $19,18\%$ no pior dos casos.

O Algoritmo Genético não conseguiu executar todas as gerações, fazendo com que o critério de parada por tempo seja o mais frequente. Tal fato ocorre pois algumas instâncias tem uma quantidade maior de peças e com peças mais complexas. Também foi possível perceber que o desvio percentual da melhor solução em relação a pior, chega em $19,20\%$ no pior dos casos.

Após a implementação dos três métodos propostos, comparamos a melhor heurística construtiva, a Heurística de Múltiplos Posicionamentos e o Algoritmo Genético. Foi possível perceber que, para todas as instâncias, a heurística construtiva é a mais rápida, mas sempre encontra valores de função objetivo piores que os outros dois métodos. Ao comparar os resultados obtidos pela Heurística de Múltiplos Posicionamentos e pelo Algoritmo Genético, notamos que a Heurística de Múltiplos Posicionamentos obteve melhores valores de função objetivo em oito das doze instâncias testadas. Em relação ao tempo de execução, a Heurística de Múltiplos Posicionamentos empatou em seis instâncias e foi mais rápida em cinco instâncias, das doze comparadas.

Por fim, os resultados obtidos pela Heurística de Múltiplos Posicionamentos, que obteve os melhores resultados dentre as heurísticas implementadas, foram comparados com os resultados reportados em Peralta, Andretta e Oliveira (2017). Neste caso, oito instâncias foram usadas. Nesta comparação, a Heurística de Múltiplos Posicionamentos obteve uma melhoria nos resultados em três das oito instâncias. Já em relação ao tempo médio, em todas as instâncias nossa abordagem é mais rápida.

Após a comparação da Heurística de Múltiplos Posicionamentos, comparamos todos os nossos melhores resultados para cada instância com Peralta, Andretta e Oliveira (2017), com a finalidade de descobrir um limite para nossos algoritmos em relação aos resultados de Peralta, Andretta e Oliveira (2017). Vemos que em quatro das oito instâncias comparadas, nossas abordagens conseguiram um resultado melhor. Já em relação ao tempo médio, em todas as instâncias nossa abordagem é mais rápida.

Como trabalhos futuros, um estudo mais aprofundado sobre os resultados obtidos pelo Algoritmo Genético pode ser feito, a fim de identificar o motivo de os resultados apresentados serem piores do que os da Heurística de Múltiplos Posicionamentos. Outras meta-heurísticas podem ser adaptadas para a resolução dos problema, usando as mesmas regras de posicionamento, ordenação e rotação. Além disso, novas regras de posicionamento, ordenação e rotação podem ser estudadas, de forma a aumentar o espaço de busca de soluções da Heurística de Múltiplos Posicionamentos e do Algoritmo Genético.

REFERÊNCIAS

ALVAREZ-VALDES, R.; MARTINEZ, A.; TAMARIT, J. A branch & bound algorithm for cutting and packing irregularly shaped pieces. **International Journal of Production Economics**, Elsevier, v. 145, n. 2, p. 463–477, 2013.

BAKER, B. S.; COFFMAN JR, E. G.; RIVEST, R. L. Orthogonal packings in two dimensions. **SIAM Journal on computing**, SIAM, v. 9, n. 4, p. 846–855, 1980.

BENNELL, J. A.; OLIVEIRA, J. F. The geometry of nesting problems: A tutorial. **European Journal of Operational Research**, Elsevier, v. 184, n. 2, p. 397–415, 2008.

BENNELL, J. A.; OLIVEIRA, J. F. A tutorial in irregular shape packing problems. **Journal of the Operational Research Society**, Taylor & Francis, v. 60, n. sup1, p. S93–S105, 2009.

BERGSTRA, J.; YAMINS, D.; COX, D. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In: PMLR. **International conference on machine learning**. [S.l.], 2013. p. 115–123.

CHERRI, L. H.; CHERRI, A. C.; SOLER, E. M. Mixed integer quadratically-constrained programming model to solve the irregular strip packing problem with continuous rotations. **Journal of Global Optimization**, Springer, v. 72, n. 1, p. 89–107, 2018.

CUTTING, E.; PACKING. **Data sets**. 2019. Disponível em: <https://www.euro-online.org/websites/esicup/data-sets/>.

DOWSLAND, K. A.; VAID, S.; DOWSLAND, W. B. An algorithm for polygon placement using a bottom-left strategy. **European Journal of Operational Research**, Elsevier, v. 141, n. 2, p. 371–381, 2002.

EBERLY, D. **Triangulation by Ear Clipping**. 2015. Disponível em: <https://www.geometrictools.com/Documentation/TriangulationByEarClipping.pdf>.

ELKERAN, A. A new approach for sheet nesting problem using guided cuckoo search and pairwise clustering. **European Journal of Operational Research**, Elsevier, v. 231, n. 3, p. 757–769, 2013.

FISCHETTI, M.; LUZZI, I. Mixed-integer programming models for nesting problems. **Journal of Heuristics**, Springer, v. 15, n. 3, p. 201–226, 2009.

HOLLAND, J. H. et al. **Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence**. [S.l.]: MIT press, 1992.

HUNTER, J. **Matplotlib Documentation**. 2019. Disponível em: <https://matplotlib.org/>.

LEAO, A. A. et al. Irregular packing problems: a review of mathematical models. **European Journal of Operational Research**, Elsevier, 2019.

LIAO, X. et al. Visual nesting system for irregular cutting-stock problem based on rubber band packing algorithm. **Advances in Mechanical Engineering**, SAGE Publications Sage UK: London, England, v. 8, n. 6, p. 1687814016652080, 2016.

MUNDIM, L. R. **Mathematical models and heuristic methods for nesting problems**. Tese (Doutorado) — Universidade de São Paulo, 2017.

MUNDIM, L. R.; ANDRETTA, M.; QUEIROZ, T. A. de. A biased random key genetic algorithm for open dimension nesting problems using no-fit raster. **Expert Systems with Applications**, Elsevier, v. 81, p. 358–371, 2017.

NIELSEN, B. K.; ODGAARD, A. Fast neighborhood search for the nesting problem. **Sci. Report, University of Copenhagen**, 2003.

NUNES, W. H. B. **Polygon Visualizer**. 2019. Disponível em: <https://github.com/WesleyHBNunes/polygons-visualizer>.

OLIVEIRA, J. F.; GOMES, A. M.; FERREIRA, J. S. Topos–a new constructive algorithm for nesting problems. **OR-Spektrum**, Springer, v. 22, n. 2, p. 263–284, 2000.

OLIVEIRA, J. F. C.; FERREIRA, J. A. S. Algorithms for nesting problems. In: **Applied simulated annealing**. [S.l.]: Springer, 1993. p. 255–273.

PERALTA, J.; ANDRETTA, M.; OLIVEIRA, J. F. **Solving Irregular Strip Packing Problems With Free Rotations Using Separation Lines**. [S.l.], 2017.

ROCHA, P. et al. Two-phase approach to the nesting problem with continuous rotations. **IFAC-PapersOnLine**, Elsevier, v. 48, n. 3, p. 501–506, 2015.

STOYAN, Y.; PANKRATOV, A.; ROMANOVA, T. Cutting and packing problems for irregular objects with continuous rotations: mathematical modelling and non-linear optimization. **Journal of the Operational Research Society**, Taylor & Francis, v. 67, n. 5, p. 786–800, 2016.

STOYAN, Y. et al. phi-functions for primary 2d-objects. **Stud. Inform. Univ.**, v. 2, p. 1–32, 01 2002.

TOLEDO, F. M. et al. The dotted-board model: a new mip model for nesting irregular shapes. **International Journal of Production Economics**, Elsevier, v. 145, n. 2, p. 478–487, 2013.

WÄSCHER, G.; HAUSSNER, H.; SCHUMANN, H. An improved typology of cutting and packing problems. **European journal of operational research**, Elsevier, v. 183, n. 3, p. 1109–1130, 2007.

APÊNDICE A – Heurística construtiva PE-SA

Neste capítulo apresentamos as imagens que representam as soluções encontradas pela heurística PE-SA e as melhores soluções obtidas pela Heurística de Múltiplos Posicionamentos (HMP) e pelo Algoritmo Genético. As imagens correspondem às soluções reportadas na Tabela 5.6.

Figura 1 – Resultado da PE-SA, na resolução da instância “albano”, com valor de função objetivo 11.458,88.

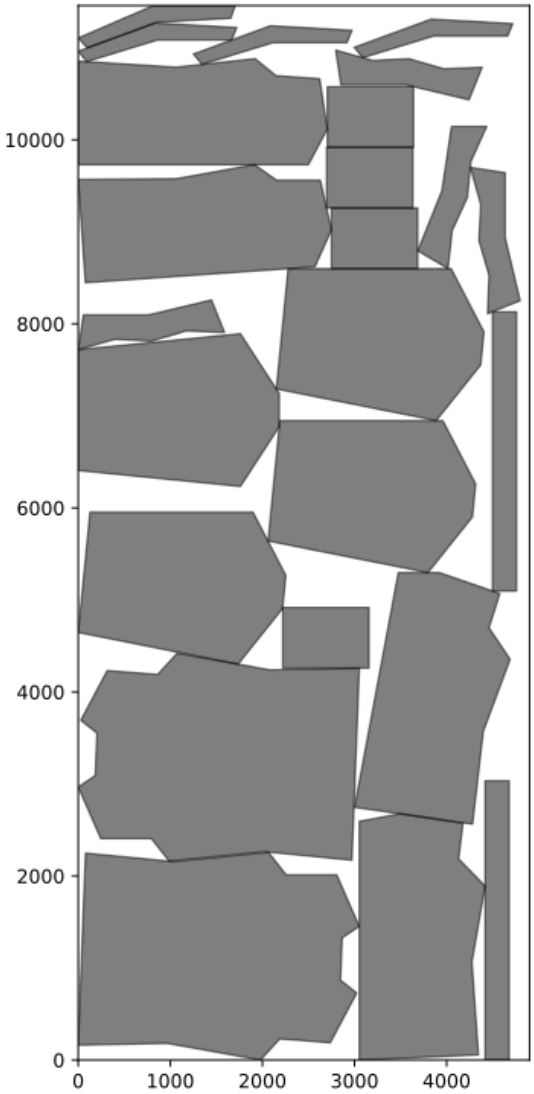


Figura 2 – Resultado da PE-SA, na resolução da instância “blaz”, com valor de função objetivo 33,26.

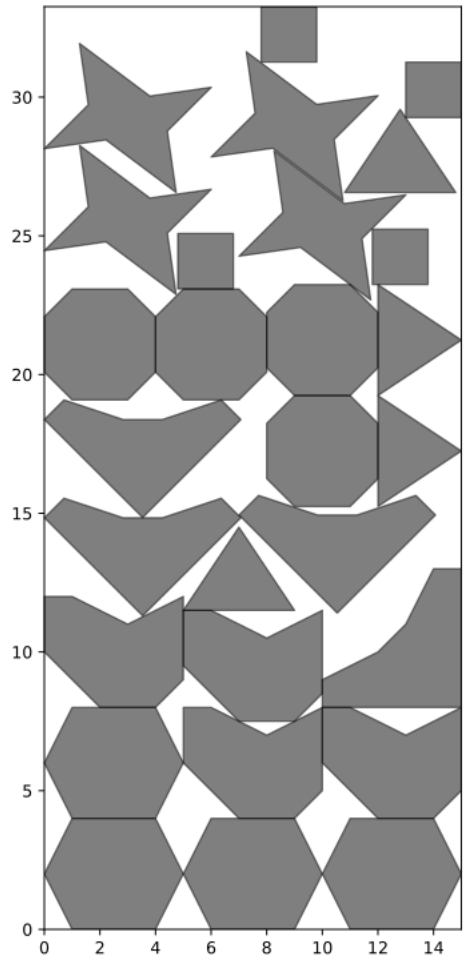


Figura 3 – Resultado da PE-SA, na resolução da instância “dighe2”, com valor de função objetivo 141,79.

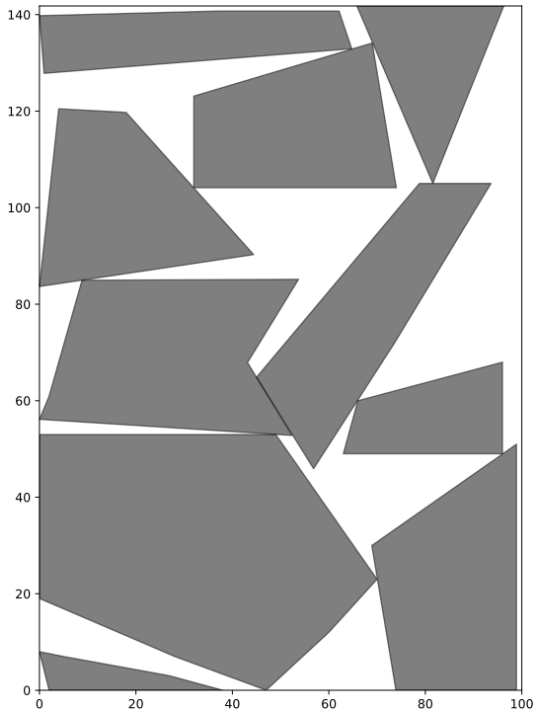


Figura 4 – Resultado da PE-SA, na resolução da instância “han”, com valor de função objetivo 51,36.

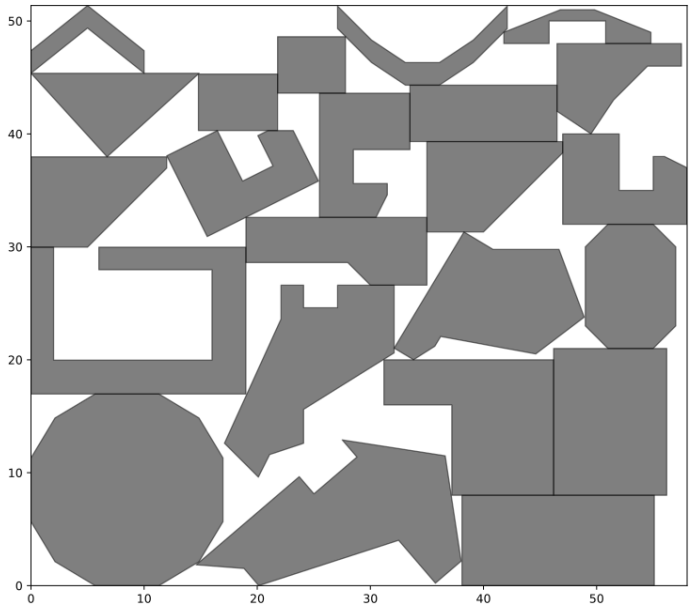


Figura 5 – Resultado da PE-SA, na resolução da instância “jakobs”, com valor de função objetivo 12,99.

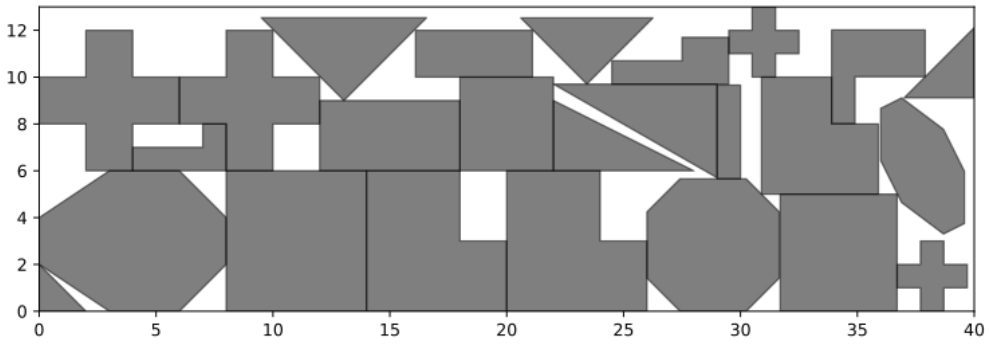


Figura 6 – Resultado da PE-SA, na resolução da instância “jakobs2”, com valor de função objetivo 31,63.

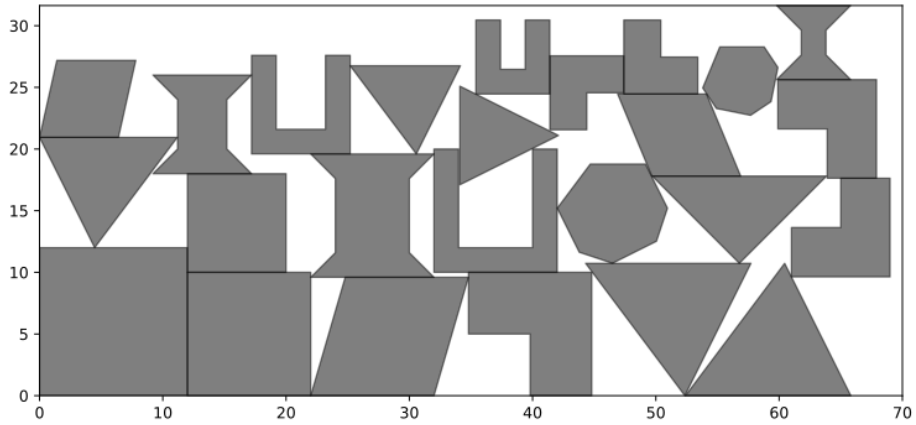


Figura 7 – Resultado da PE-SA, na resolução da instância “mao”, com valor de função objetivo 1.971,11.

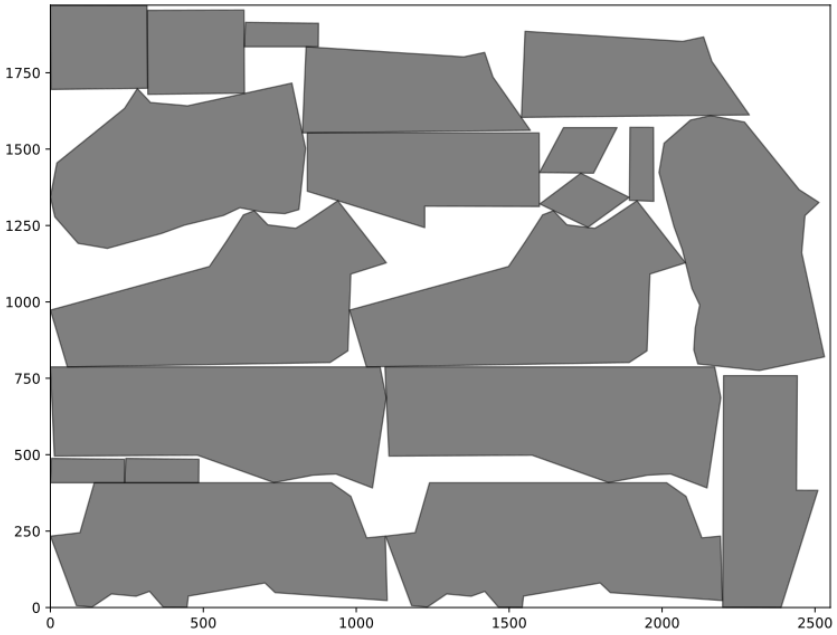


Figura 8 – Resultado da PE-SA, na resolução da instância “marques”, com valor de função objetivo 87,50.

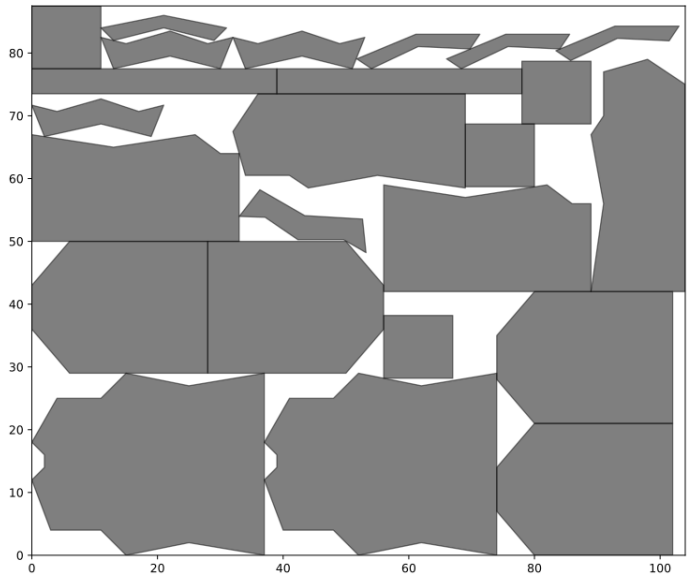


Figura 9 – Resultado da PE-SA, na resolução da instância “poly1a”, com valor de função objetivo 18,74.

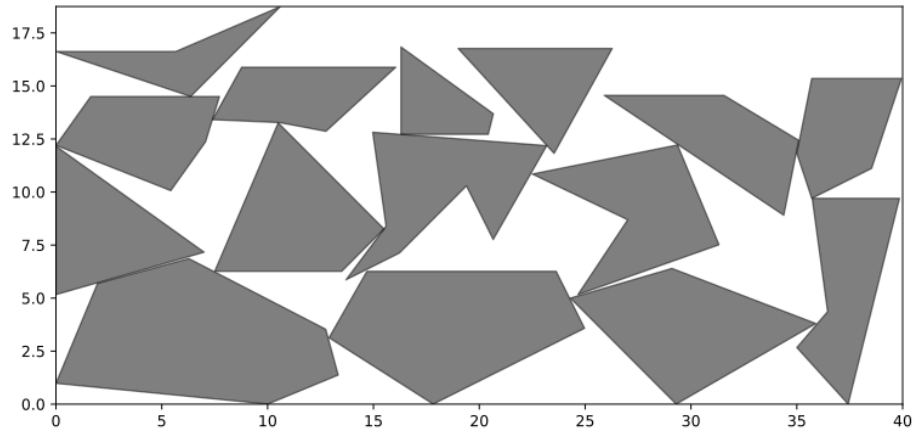


Figura 10 – Resultado da PE-SA, na resolução da instância “shapes”, com valor de função objetivo 67,97.

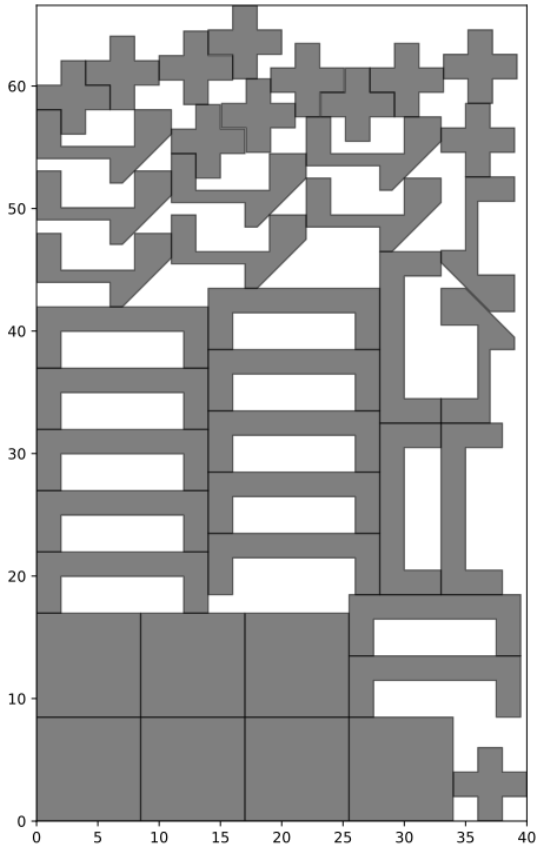


Figura 11 – Resultado da PE-SA, na resolução da instância “shirts”, com valor de função objetivo 65,10.

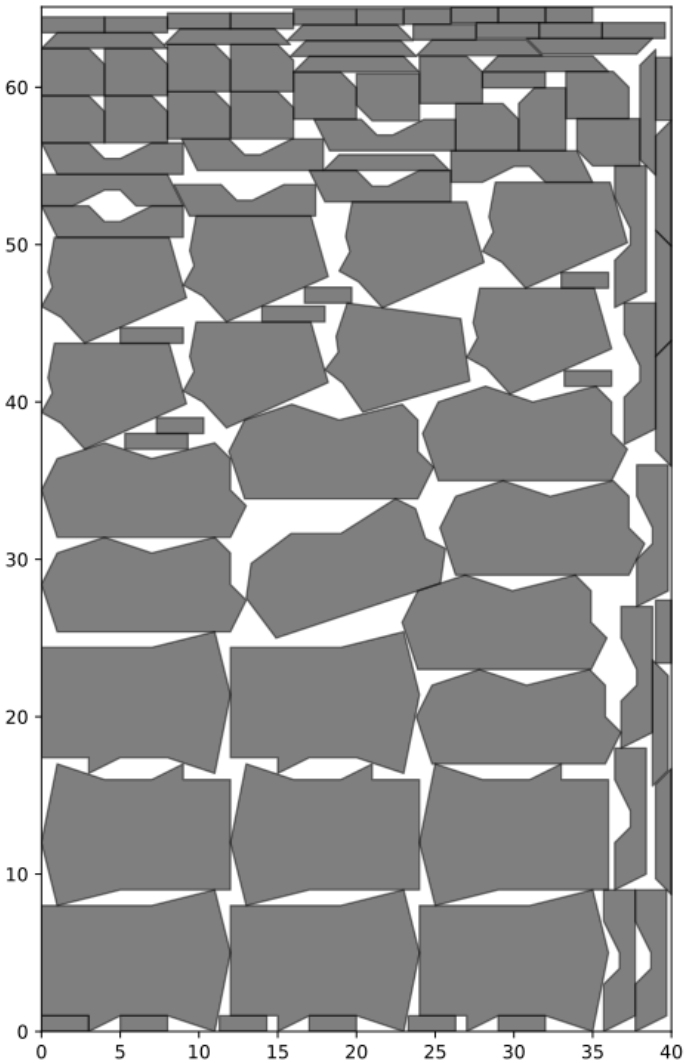
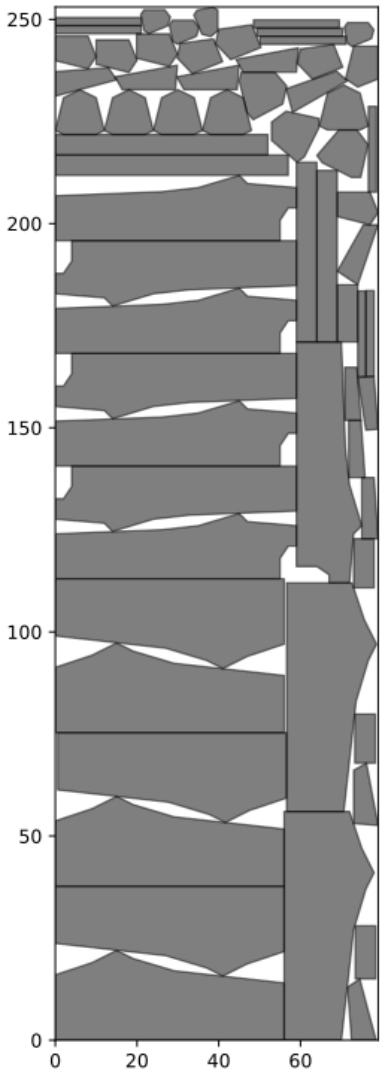


Figura 12 – Resultado da PE-SA, na resolução da instância “trousers”, com valor de função objetivo 253,04.



APÊNDICE B – Heurística de Múltiplos Posicionamentos (HMP)

Figura 13 – Resultado da HMP, na resolução da instância “albano”, com valor de função objetivo 10.643,66.

Figura 14 – Resultado da HMP, na resolução da instância “blaz”, com valor de função objetivo 29,65.

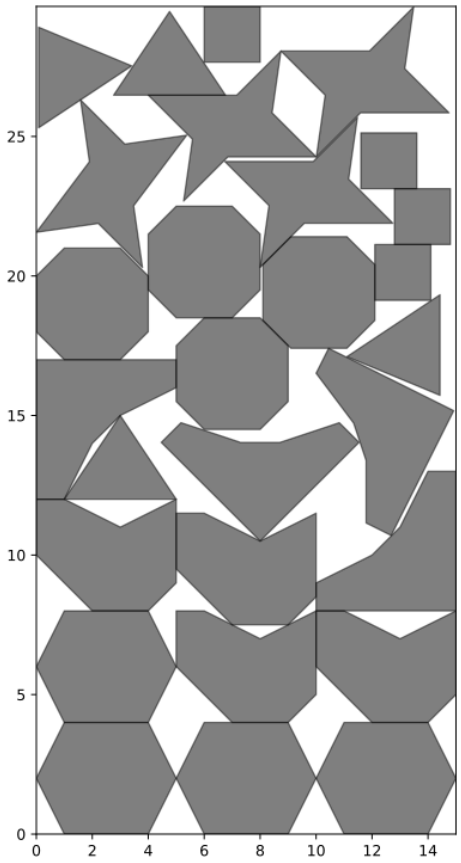


Figura 15 – Resultado da HMP, na resolução da instância “dighe2”, com valor de função objetivo 127,25.

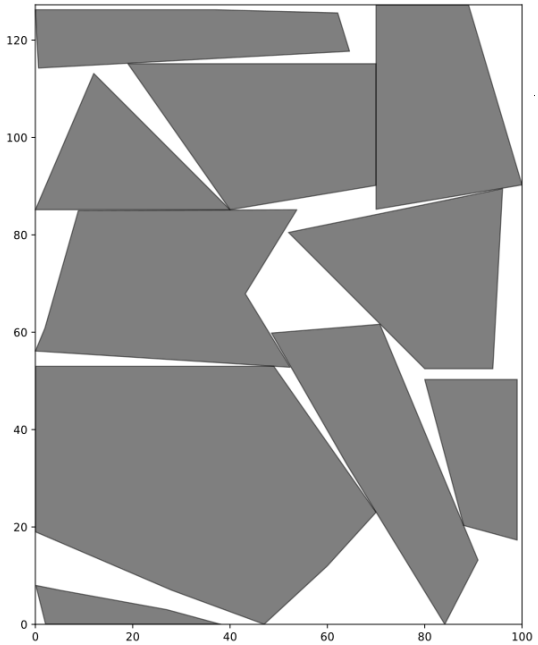


Figura 16 – Resultado da HMP, na resolução da instância “han”, com valor de função objetivo 44,76.

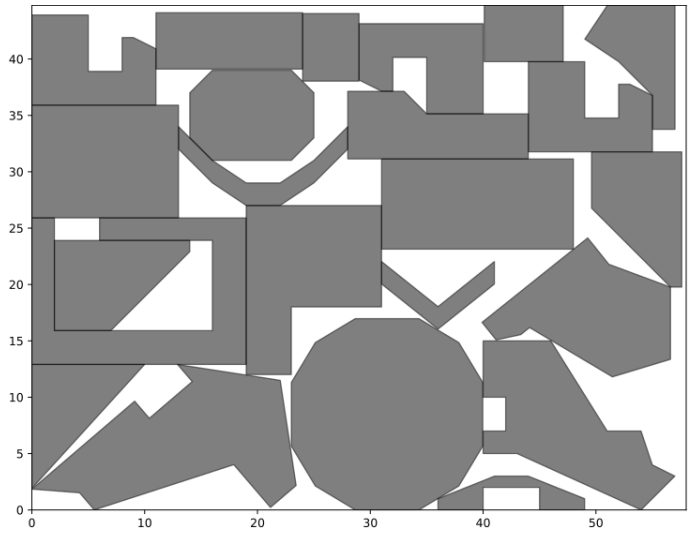


Figura 17 – Resultado da HMP, na resolução da instância “jakobs”, com valor de função objetivo 12,00.

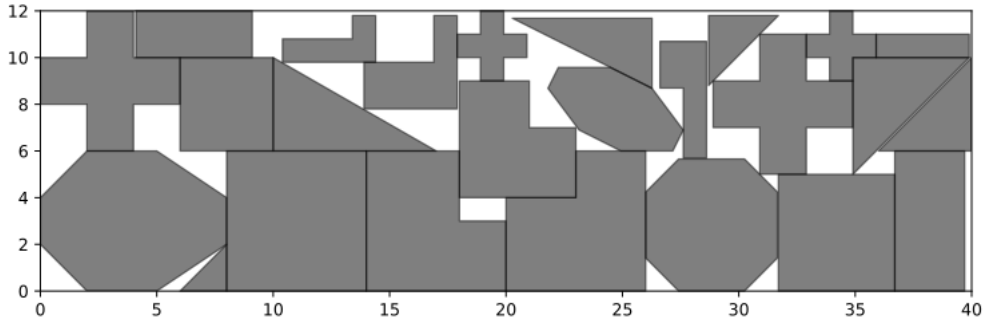


Figura 18 – Resultado da HMP, na resolução da instância “jakobs2”, com valor de função objetivo 25,81.

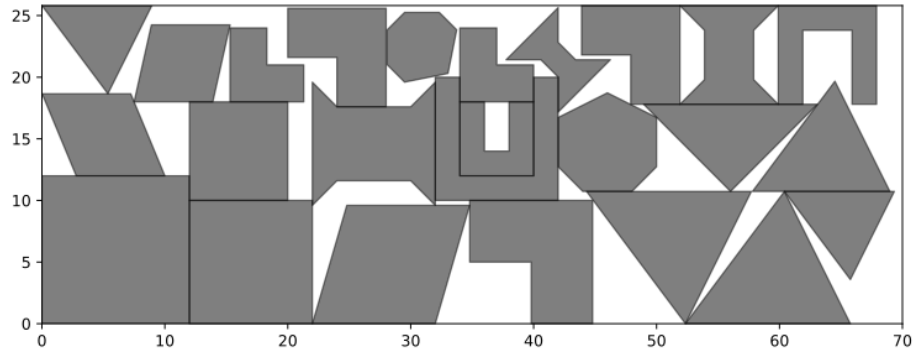


Figura 19 – Resultado da HMP, na resolução da instância “mao”, com valor de função objetivo 1.907,31.

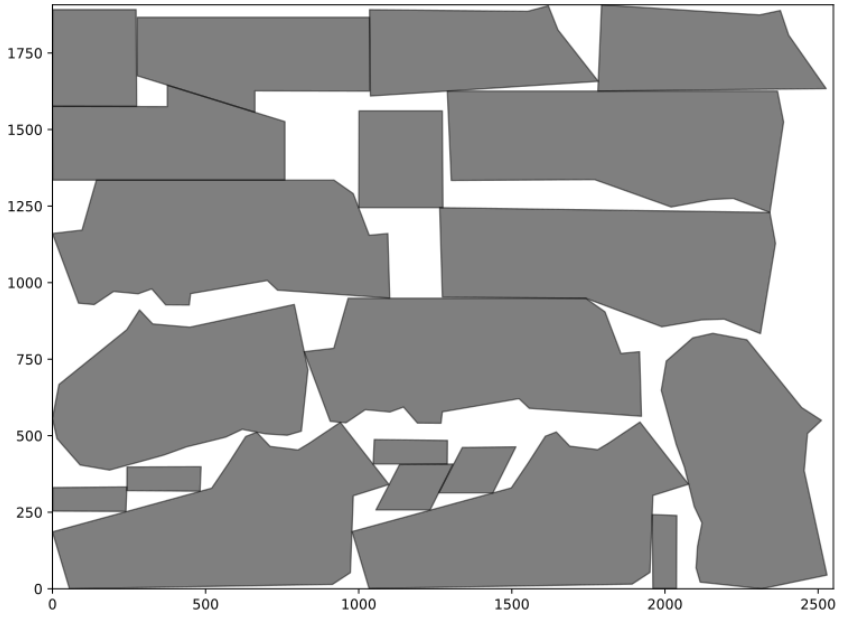


Figura 20 – Resultado da HMP, na resolução da instância “marques”, com valor de função objetivo 83,50.

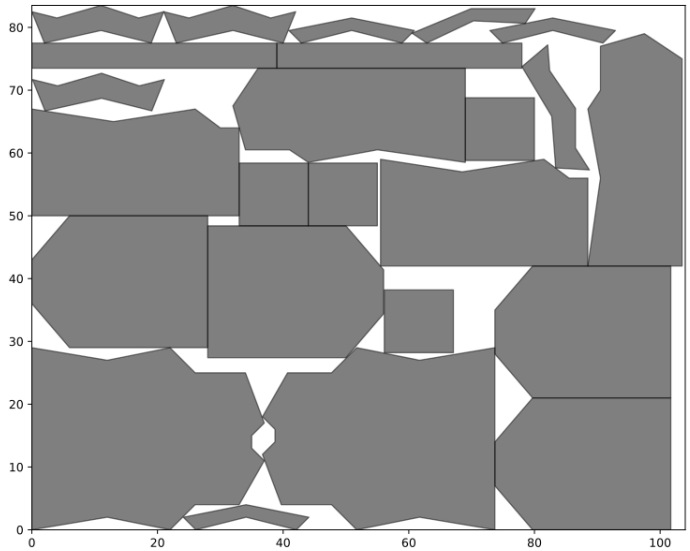


Figura 21 – Resultado da HMP, na resolução da instância “poly1a”, com valor de função objetivo 14,55.

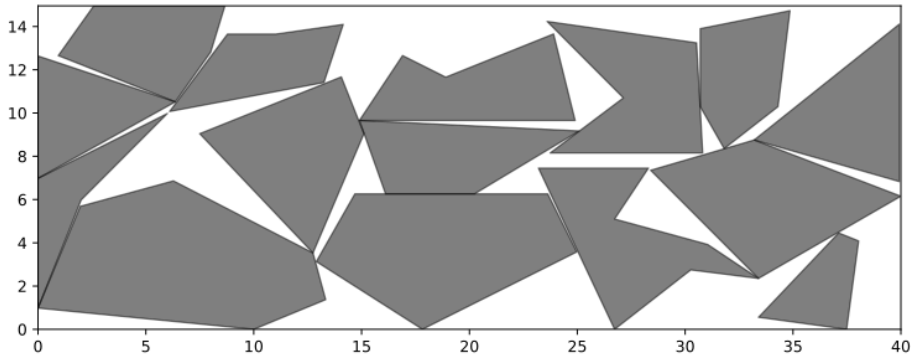


Figura 22 – Resultado da HMP, na resolução da instância “shapes”, com valor de função objetivo 61,97.

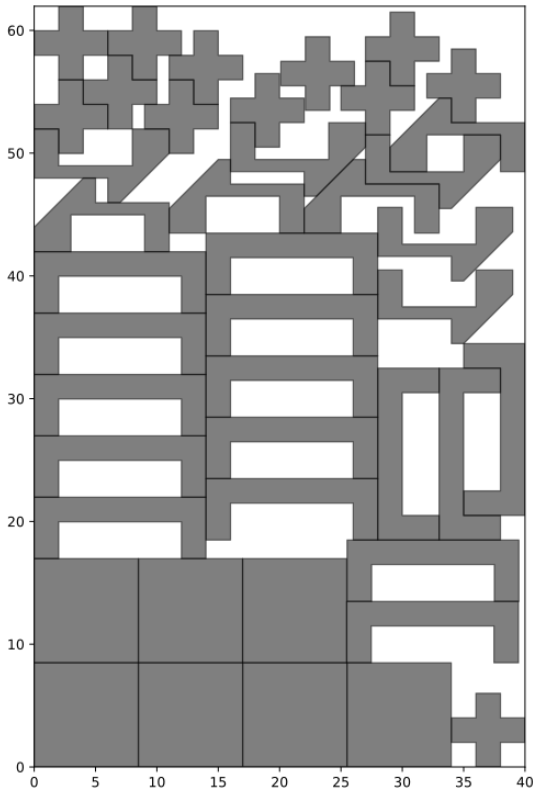


Figura 23 – Resultado da HMP, na resolução da instância “shirts”, com valor de função objetivo 63,21.

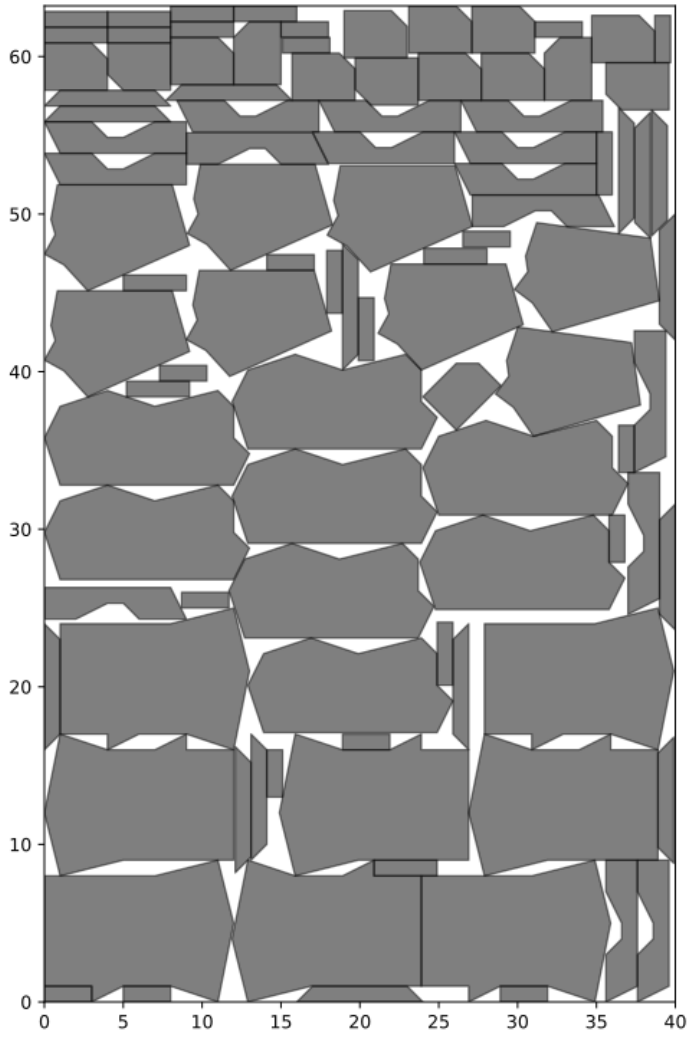
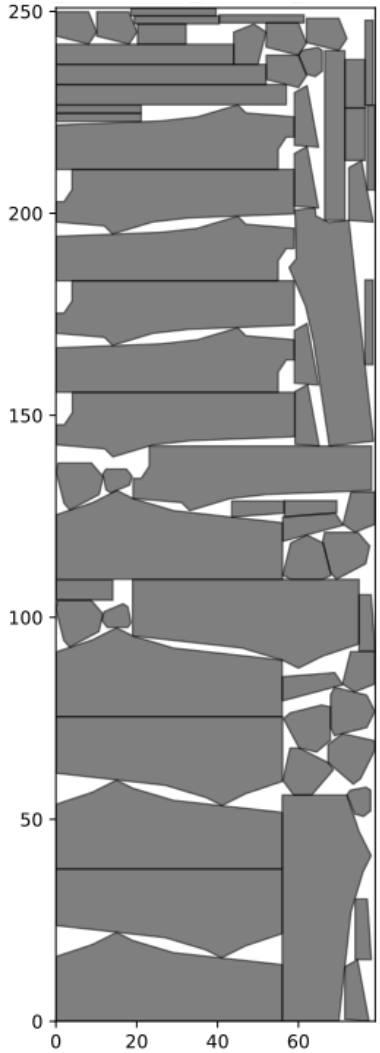


Figura 24 – Resultado da HMP, na resolução da instância “trousers”, com valor de função objetivo 250,87.



APÊNDICE C – Algoritmo Genético

Figura 25 – Resultado do Algoritmo Genético, na resolução da instância “albano”, com valor de função objetivo 10.578,35.

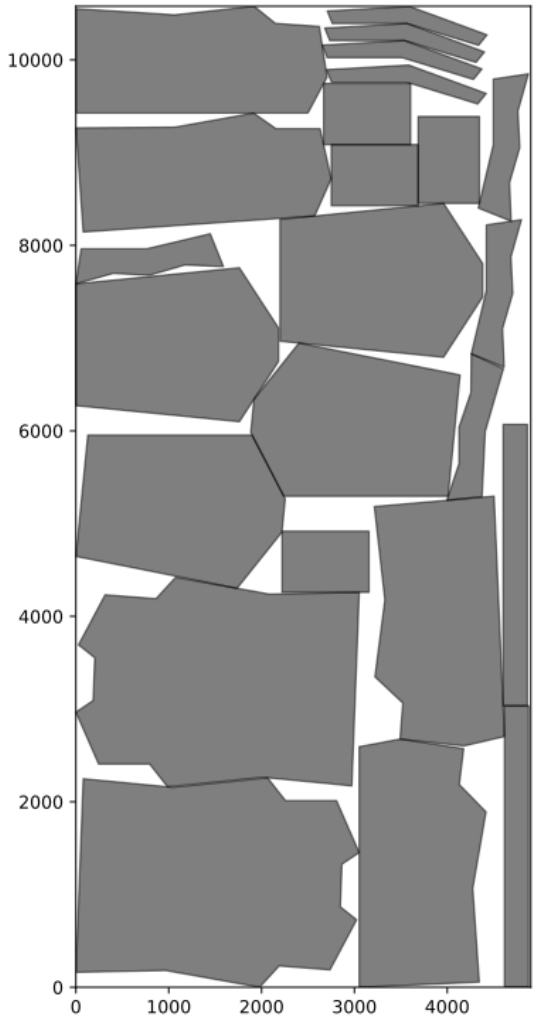


Figura 26 – Resultado do Algoritmo Genético, na resolução da instância “blaz”, com valor de função objetivo 29,59.

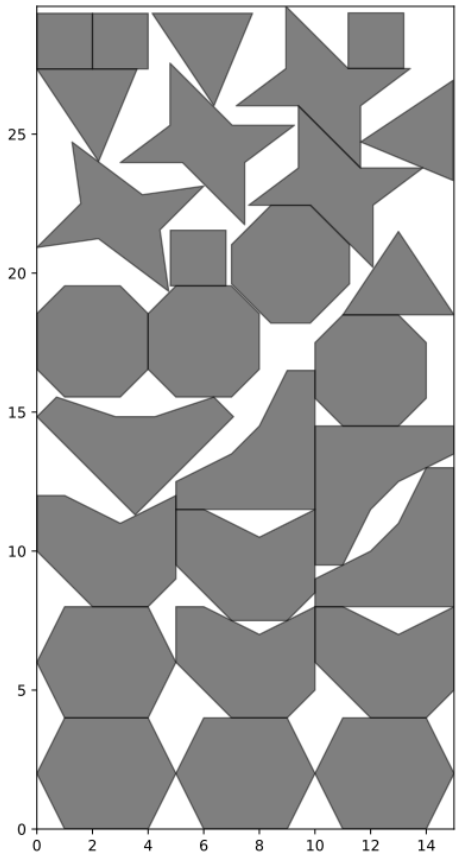


Figura 27 – Resultado do Algoritmo Genético, na resolução da instância “dighe2”, com valor de função objetivo 134,64.

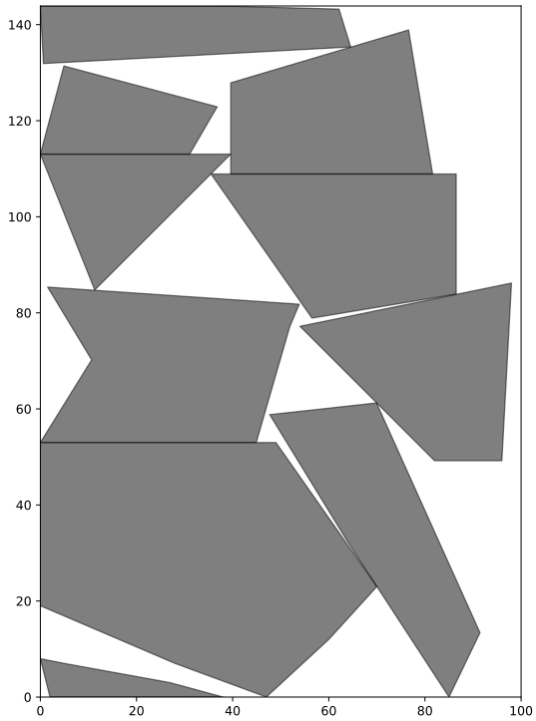


Figura 28 – Resultado do Algoritmo Genético, na resolução da instância “han”, com valor de função objetivo 45,00.

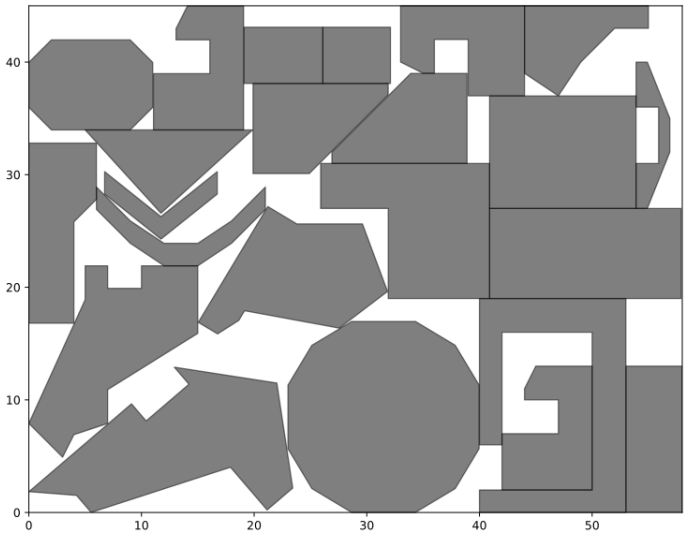


Figura 29 – Resultado do Algoritmo Genético, na resolução da instância “jakobs”, com valor de função objetivo 12,46.

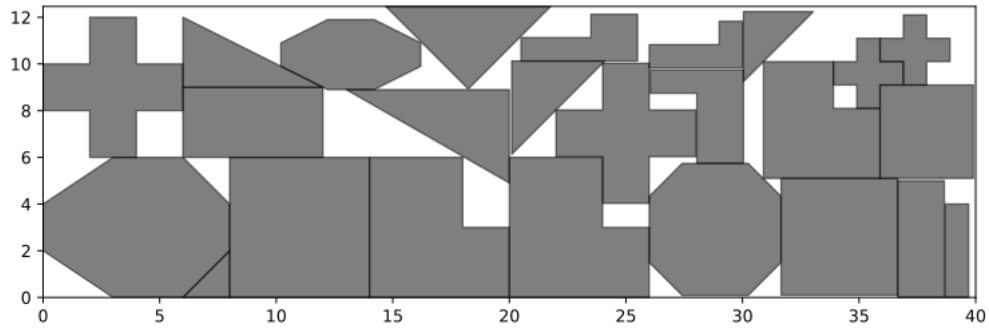


Figura 30 – Resultado do Algoritmo Genético, na resolução da instância “jakobs2”, com valor de função objetivo 25,78.

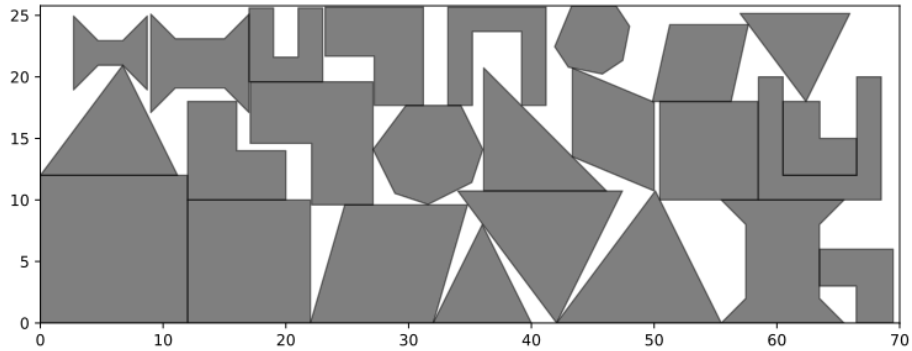


Figura 31 – Resultado do Algoritmo Genético, na resolução da instância “mao”, com valor de função objetivo 1.933,59.

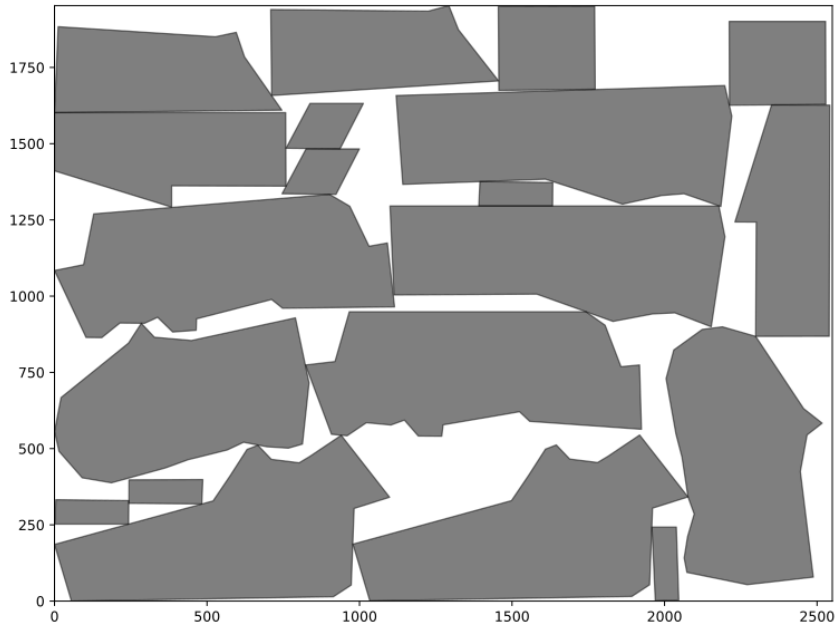


Figura 32 – Resultado do Algoritmo Genético, na resolução da instância “marques”, com valor de função objetivo 83,70.

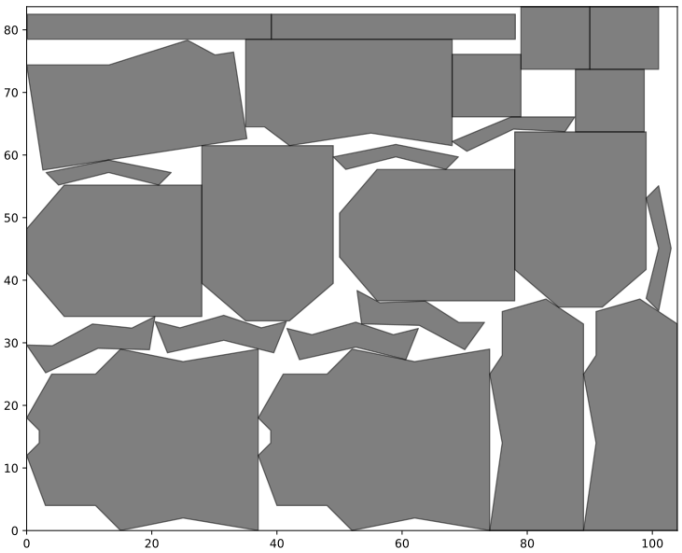


Figura 33 – Resultado do Algoritmo Genético, na resolução da instância “poly1a”, com valor de função objetivo 15,44.

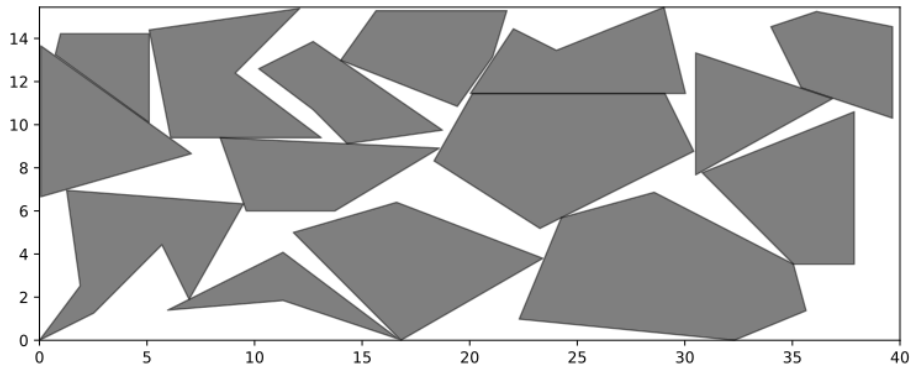


Figura 34 – Resultado do Algoritmo Genético, na resolução da instância “shapes”, com valor de função objetivo 62,10.

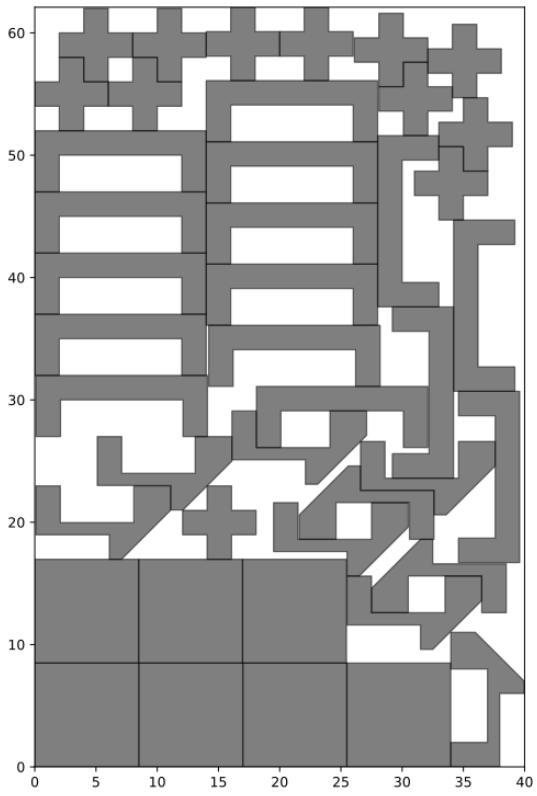


Figura 35 – Resultado do Algoritmo Genético, na resolução da instância “shirts”, com valor de função objetivo 64,01.

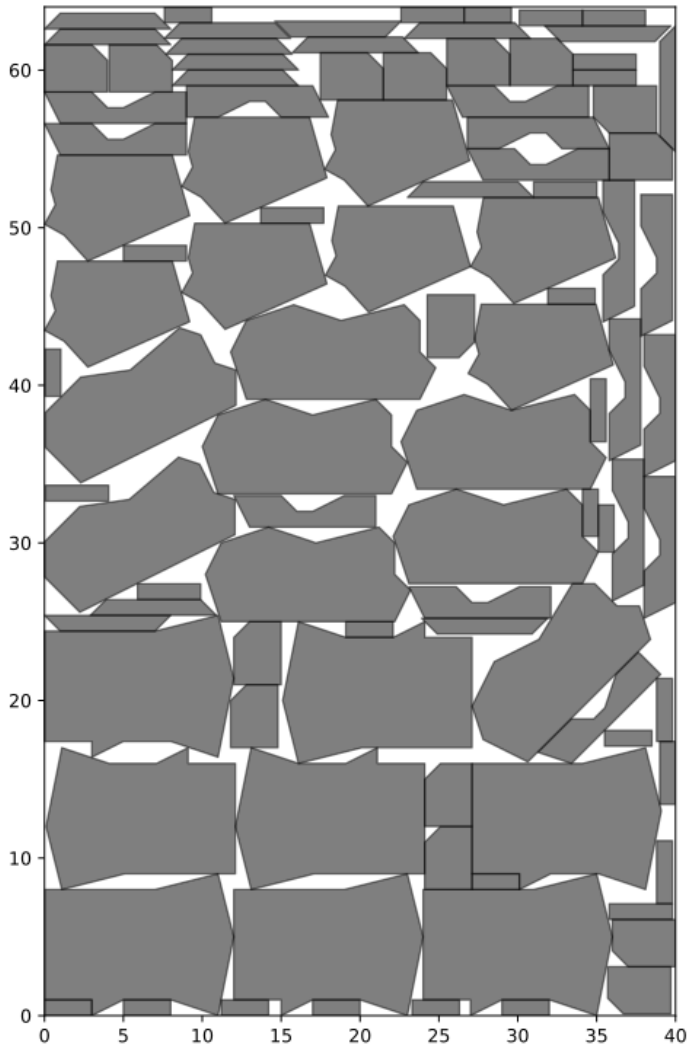


Figura 36 – Resultado do Algoritmo Genético, na resolução da instância “trousers”, com valor de função objetivo 249,17.