ALGORITMO GENÉTICO COM CHAVES ALEATÓRIAS PARA O PROBLEMA DE CORTE GUILHOTINADO BIDIMENSIONAL EM TRÊS ESTÁGIOS E COM RESTRIÇÕES DE PRECEDÊNCIA

MARCOS VINÍCIUS ALMEIDA GUIMARÃES

ALGORITMO GENÉTICO COM CHAVES ALEATÓRIAS PARA O PROBLEMA DE CORTE GUILHOTINADO BIDIMENSIONAL EM TRÊS ESTÁGIOS E COM RESTRIÇÕES DE PRECEDÊNCIA

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Orientador: Thiago Ferreira de Noronha
Coorientador: Sebastián Alberto Urrutia

Belo Horizonte
Outubro de 2020

Guimarães, Marcos Vinícius Almeida.

G963a Algoritmo genético com chaves aleatórias para o problema de corte guilhotinado bidimensional em três estágios e com restrições de precedência [manuscrito] / Marcos Vinícius Almeida Guimarães. — 2020.
xx, 45 f. il.

Orientador: Thiago Ferreira de Noronha.
Coorientador: Sebastián Alberto Urrutia.
Dissertação (mestrado) - Universidade Federal de Minas Gerais, Instituto de Ciências Exatas, Departamento de Ciência da Computação.
Referências: f.41-45

1. Computação – Teses. 2. Problema de corte bidimensional – Teses . 3. Algoritmos genéticos – Teses. 4. Restrições de precedência – Teses. 5. Software -Reutilização – Teses. I. Noronha, Thiago Ferreira de. II. Urrutia, Sebastián Alberto. III. Universidade Federal de Minas Gerais, Instituto de Ciências Exatas, Departamento de Ciência da Computação. IV. Título.

CDU 519.6*82(043)

Ficha catalográfica elaborada pela bibliotecária Belkiz Inez Rezende Costa
CRB 6ª Região nº 1510

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

Algoritmo Genético com Chaves Aleatórias para o Problema de Corte Guilhotinado Bidimensional em Três Estágios e com Restrições de Precedência.

MARCOS VINÍCIUS ALMEIDA GUIMARÃES

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

Thiago F. Noronha
PROF. THIAGO FERREIRA DE NORONHA - Orientador
Departamento de Ciência da Computação - UFMG

PROF. SEBASTIÁN ALBERTO URRUTIA - Coorientador
Departamento de Ciência da Computação - UFMG

PROFA. ISABEL CRISTINA MELLO ROSSETI
Instituto de Computação - UFF

Vinicius Fernandes dos Santos
PROF. VINÍCIUS FERNANDES DOS SANTOS
Departamento de Ciência da Computação - UFMG

Belo Horizonte, 29 de Outubro de 2020.

"Happiness can be found, even in the darkest of times, if one only remembers to turn on the light."

(Albus Dumbledore)

Resumo

Esta dissertação aborda o Problema de Corte Guilhotinado Bidimensional em 3 Estágios e com Restrições de Precedência (2DCSP-PC). O último é uma generalização do Problema de Corte Guilhotinado Bidimensional em 3 Estágios (2DCSP) com restrições de precedência entre os itens a serem cortados. O objetivo é minimizar a quantidade de material utilizado para cortar todos os itens. Até onde pode-se dizer, esta nova restrição de precedência impede o uso da maioria dos algoritmos na literatura para o 2DCSP, pois eles não foram projetados para considerá-la.

Primeiramente, duas heurísticas construtivas presentes na literatura foram adaptadas para o 2DCSP-PC, uma heurística *First-Fit* Finita (FFF) e um Procedimento Heurístico Sequencial (SHP). Como na etapa de pré-processamento das instâncias é escolhida e fixada uma orientação para os itens, dois Algoritmos Genéticos com Chaves Aleatórias Tendenciosos (BRKGA) foram propostos, o primeiro fixa os itens horizontalmente, ou seja, com sua largura maior ou igual à sua altura (BRKGA), e o segundo escolhe qual a orientação dos itens (BRKGA-R). Os dois últimos foram comparados com o Algoritmo Evolucionário com Representação por Elementos (EAe) presente na literatura. Os experimentos mostraram que a heurística BRKGA obteve resultados quase que estritamente melhores que as outras para a grande maioria das instâncias testadas.

Palavras-chave: Problema de corte bidimensional, BRKGA, Algoritmos Genéticos, Restrições de Precedência.

Abstract

This dissertation addresses the 3-staged 2D Guillotine Cutting Stock Problem With Precedence Constraints (2DCSP-PC). This problem is a generalization of the 3-staged 2D Guillotine Cutting Stock Problem (2DCSP), which deals with precedence constraints between the items to be cut. The goal is to minimize the amount of material used to cut all the items. As far as one can tell, this new precedence constraint prevents the use of most algorithms in the literature for 2DCSP, because they have not been designed to consider it.

First, two constructive heuristics present in the literature have been adapted for the 2DCSP-PC, a Finite First-Fit heuristic (FFF) and a Sequential Heuristic Procedure (SHP). Since the pre-processing stage of the instances chooses an orientation for the items and fixes it, two Biased Random-Key Genetic Algorithms (BRKGA) were proposed, the first fixes the items horizontally, in other words, with their width greater than or equal to their height (BRKGA), and the second chooses the orientation of the items (BRKGA-R). The latter and the former were compared with the Evolutionary Algorithm with Elements Representation (EAe) present in the literature. The experiments showed that the BRKGA heuristic achieved strictly better results than the others for the vast majority of the instances that have been tested.

Keywords: Cutting Stock Problem, BRKGA, Genetic Algorithms, Precedence Constraints.

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo 1

Introdução

Muitas indústrias enfrentam o desafio de encontrar soluções mais econômicas para o problema de cortar objetos maiores para produzir objetos menores comumente com o intuito de atender a alguma demanda. Esses problemas de corte são frequentemente encontrados em diferentes processos industriais, como corte de chapas de aço, de vidro, dentre outros. Nesta dissertação, os objetos maiores serão referidos como *placas* e os objetos menores como *itens*. Além disso, assume-se que os objetos são bidimensionais e têm uma forma retangular.

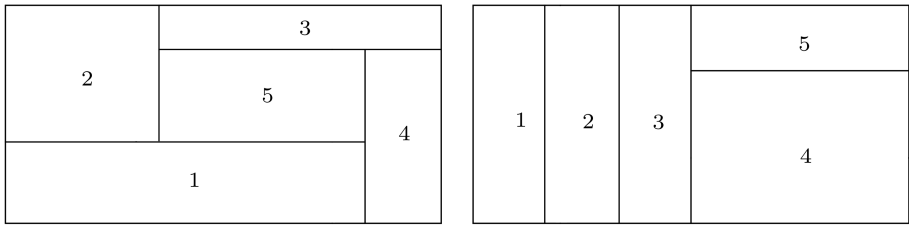

Figura 1.1: Exemplo de cortes não guilhotinados (a) e guilhotinados (b).

Uma restrição comum para problemas de corte é que os itens devem ser obtidos apenas através de cortes guilhotinados, ou seja, cortes que são paralelos a um dos lados do objeto e vão de um lado a outro da placa, sempre dividindo-a em duas. A figura 1.1 mostra exemplos de cortes não guilhotinados (a) e cortes guilhotinados (b).

Outra restrição frequente para esse tipo de problema é a necessidade de separar os cortes em estágios. Um estágio de corte consiste em um conjunto de cortes guilhotinados paralelos, e um padrão de corte do tipo k-estágios é uma sequência de no máximo k estágios de corte. Em cada estágio, os cortes são feitos na direção oposta à etapa anterior, sendo executados inicialmente de forma vertical. A notação α-*cut* é utilizada para indicar o número do estágio em que o corte foi feito.

Um padrão de corte é uma descrição de como uma placa é cortada para se obter um subconjunto de itens. Ele pode ser representado como uma árvore, onde o nó raiz corresponde à placa inicial e os outros nós representam partes da placa que foram separadas com os cortes guilhotinados. Portanto, as folhas representam os itens e os pedaços individuais da placa que não foram utilizados. Cada nível da árvore corresponde a um estágio, ou seja, todos os cortes no mesmo nível têm a mesma orientação. Na prática, os cortes são realizados utilizando uma abordagem de profundidade em primeiro lugar, para evitar a alteração do segmento da placa que está na guilhotina. Por exemplo, a representação em árvore do padrão de corte em 3-estágios da Figura 1.2 é fornecido na Figura 1.3. Primeiro, um corte vertical *1-cut* é aplicado para separar um pedaço da placa não utilizado (destacado com linhas diagonais). Em seguida, é realizado um corte horizontal *2-cut* para extrair o item 1. Em seguida, três cortes verticais sucessivos são executados para obter os itens 2, 3 e 4, além de outro pedaço de placa não utilizado.

A diferença entre o problema abordado nesta dissertação e outros na literatura é que aqui os itens são organizados em lotes. Cada lote representa um pedido de um cliente e define a ordem na qual os itens devem ser cortados. Ou seja, se o item i preceder o item j em um lote, i deverá ser cortado antes de j. Não há restrição de precedência entre itens em lotes diferentes. Essa restrição vem de uma aplicação real da indústria referente a cortes guilhotinados em três estágios em placas de vidro [Lydia & Quentin, 2018]. Como as peças de vidro são frágeis, elas devem ser empilhadas e enviadas na ordem exata em que são utilizadas pelos clientes, para evitar rachaduras

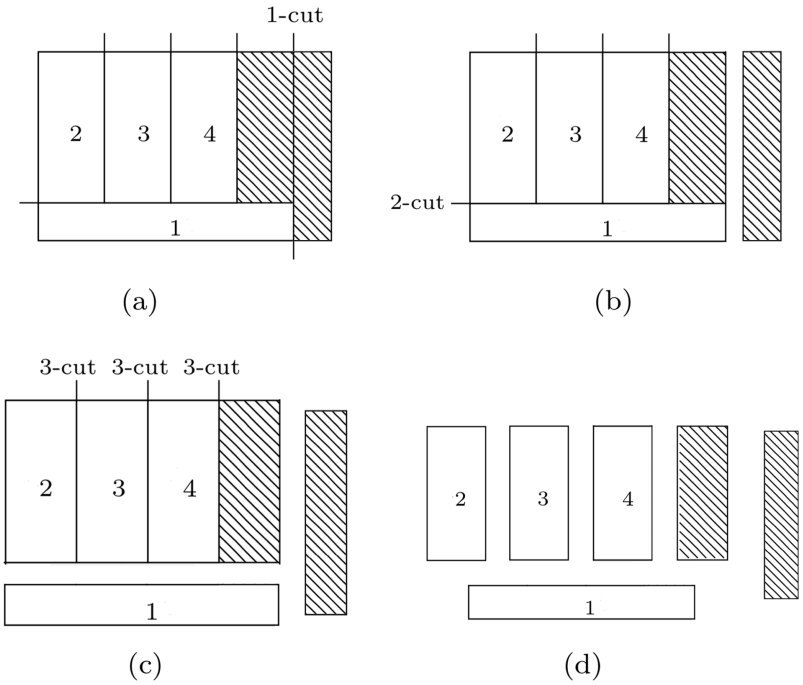

Figura 1.2: Um padrão de corte inicial (a), o primeiro, segundo e terceiro estágios de cortes presentes em (b), (c) e (d), respectivamente.

ao se tentar acessar as peças de vidro no meio das pilhas. A mesma restrição de precedência pode ser encontrada no gerenciamento da cadeia de suprimentos de peças metálicas, devido ao grande peso dessas peças.

O Problema de Corte Bidimensional sujeito a restrições de cortes guilhotinados, estágios e precedência é referido aqui como o Problema de Corte Guilhotinado Bidimensional em 3 Estágios e com Restrições de Precedência (2DCSP-PC). Seja W e H a largura e a altura das placas disponíveis para corte, e I o conjunto de itens a serem cortados, onde cada item $i \in I$ tem largura w_i e altura h_i. Seja também S o conjunto de pilhas que representam os pedidos dos clientes, onde cada pilha $s = (\pi_1^s, \pi_2^s, \ldots, \pi_{n_s}^s)$ descreve a ordem em que os itens devem ser cortados, de modo que $\pi_k^s \in I$ seja cortado antes de $\pi_{k+1}^s \in I$, para todos os $k = 1, \ldots, n_s - 1$, onde n_s é o número de itens em s. Uma solução para 2DCSP-PC consiste em uma sequência de padrões de corte $\mathcal{S}$ que descreve como, e em qual ordem, as placas devem ser cortadas. Esta solução deve atender a todas as seguintes restrições: (i) as placas não podem ser rotacionadas; (ii)

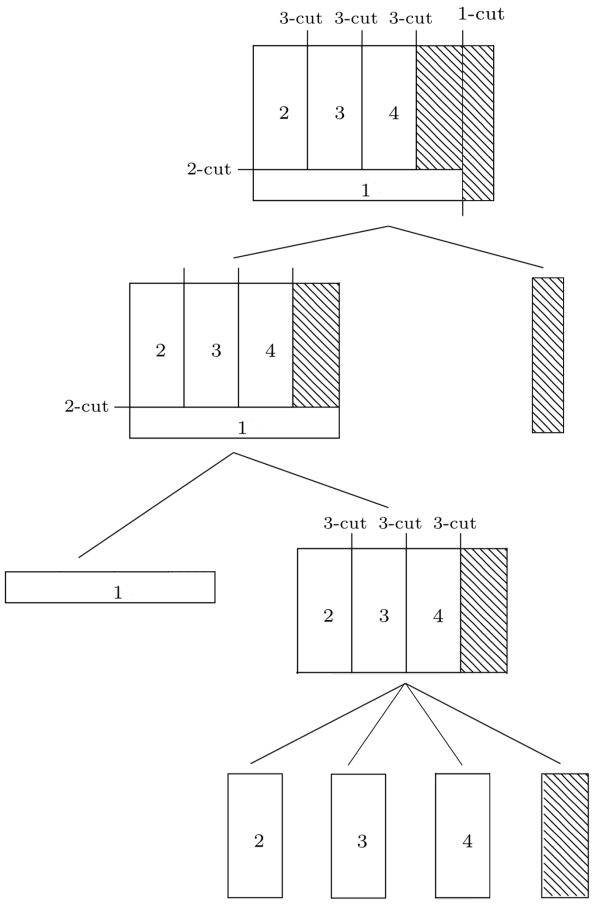

Figura 1.3: Exemplo de um padrão de corte da Figura 1.2.

os itens só podem ser rotacionados em 90°; (iii) todos os itens em I devem ser cortados exatamente uma vez; (iv) se $i \in I$ preceder $j \in I$ em sua pilha, i deve ser completamente cortado antes de j; (v) apenas cortes guilhotinados são permitidos; e (vi) o número de estágios de corte é no máximo 3. No entanto, como em [Hifi & Roucairol, 2001; Yanasse & Morabito, 2006; Alvarez-Valdes et al., 2007; Andrade et al., 2016], é permitido um corte adicional do tipo *4-cut* apenas para separar um único item de um pedaço da placa não utilizado. Esse processo é conhecido na literatura como *trimming*. A Figura 1.4a mostra um exemplo de *trimming*, onde o item 3 é separado de um pedaço da placa não utilizado por um único corte do tipo *4-cut*, enquanto a Figura 1.4b mostra um exemplo inválido de corte do tipo *4-cut* utilizado para separar o item 3 do item 4.

O custo de uma solução $\mathcal{S}$ para 2DCSP-PC é definido como a quantidade de

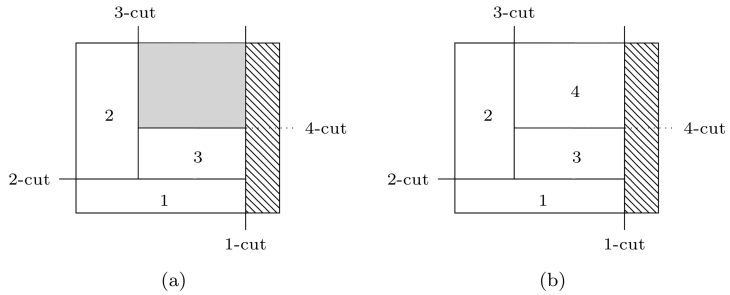

Figura 1.4: Exemplo de um corte do tipo *4-cut* permitido (a) e um não permitido (b).

material utilizado para cortar todos os itens. Como em [Cherri et al., 2009; Chen et al., 2015; Andrade et al., 2016; Cui et al., 2017], os pedaços de placa não utilizados resultantes dos padrões de corte em $\mathcal{S}$ são divididos em dois tipos: *área residual* e *desperdício*. *Resíduo* é o material à direita do último corte do tipo *1-cut* aplicado à última placa. Supõe-se que este pedaço de placa pode ser reutilizado futuramente. Portanto, ele não é considerado no custo de $\mathcal{S}$. Todos os outros pedaços de placa não utilizados são considerados *desperdícios*, porque assume-se que eles não podem ser reutilizados. Por exemplo, na Figura 1.4a, o pedaço de placa não utilizado destacado em cinza é considerado desperdício, enquanto o destacado com linhas diagonais é considerado resíduo. Portanto, o custo $f(\mathcal{S})$ de uma solução $\mathcal{S}$ é definido de acordo com a equação (1.1),

$$f(\mathcal{S}) = \quad H \times W \times (|\mathcal{S}| - 1) + H \times r(\mathcal{S}) \tag{1.1}$$

onde $|\mathcal{S}|$ indica o número de placas utilizadas, $H \times W \times (|\mathcal{S}| - 1)$ é a área total das primeiras $|\mathcal{S}| - 1$ placas, $r(\mathcal{S})$ é a posição do último corte do tipo *1-cut* no último padrão de corte de $\mathcal{S}$, e $H \times r(\mathcal{S})$ representa a área utilizada da última placa.

O 2DCSP-PC consiste em encontrar a solução $\mathcal{S}^* = \text{argmin}_{\mathcal{S}\in\Delta} f(\mathcal{S})$ que usa a menor quantidade de material para cortar todos os itens em I, onde Δ é o conjunto de soluções viáveis. Quando há apenas um item por pilha, esse problema é equivalente ao Problema de Corte Guilhotinado Bidimensional em 3 Estágios (2DCSP) [Lai & Chan, 1997]. Portanto ele também é NP-Difícil.

Como não existem técnicas conhecidas para projetar algoritmos exatos em tempo

polinomial para problemas NP-Difíceis, esta dissertação foca em algoritmos heurísticos. No entanto, tanto quanto pode-se dizer, a nova restrição de precedência introduzida em [Lydia & Quentin, 2018] impede o uso da maioria dos algoritmos na literatura para o 2DCSP-PC, porque eles não foram projetados para considerar esta precedência entre os itens. Além disso, não é evidente como adaptar os algoritmos heurísticos e exatos mais avançados para levar em consideração esta precedência. Portanto, nesta dissertação, as heurísticas construtivas FFF [Berkey & Wang, 1987] e SHP [Suliman, 2006], e o algoritmo genético EAe [Puchinger et al., 2004] foram estendidos para o 2DCSP-PC. Além disso, foi avaliado qual das heurísticas construtivas encontra os melhores resultados; e foram propostos dois *Algoritmos Genéticos com Chaves Aleatórias Tendenciosos* (do inglês, *Biased Random-Key Genetic Algorithm, BRKGA*) que utilizam junto com o EAe uma das heurísticas construtivas como algoritmo decodificador. A decisão de se utilizar o *framework* BRKGA deu-se ao fato de que em [Gonçalves & Resende, 2013] ele foi utilizado para problemas de corte bidimensionais e tridimensionais com resultados bem satisfatórios quando comparado com algoritmos da literatura.

O restante desta dissertação está organizada da seguinte forma. Na seção 2, são apresentados os trabalhos relacionados. Nas seções 3.1, 3.2, 4.1 e 4.2, são descritas, respectivamente, as duas heurísticas construtivas FFF e SHP, as duas heurísticas propostas baseadas no *framework* BRKGA e o algoritmo genético EAe para o 2DCSP-PC. Na Seção 5, são discutidos os experimentos computacionais que foram realizados com instâncias obtidas no *ROADEF Challenge* 2018. Por fim, na Seção 6, são traçadas considerações finais e são apontados alguns possíveis trabalhos futuros.

Capítulo 2

Trabalhos Relacionados

Até o presente momento, nenhum trabalho na literatura trata especificamente o problema definido na Seção 1. Dessa forma, a revisão bibliográfica realizada foca em problemas semelhantes, ou seja, problemas de corte guilhotinado bidimensional em estágios tradicionais. Análises abrangentes para o problema de corte bidimensional são fornecidas por [Dyckhoff & Finke, 1992], [Lodi et al., 2002b] e [Lodi et al., 2002a].

2.1 Problema de corte guilhotinado bidimensional sem restrições de número de estágios

Em [Berkey & Wang, 1987] foi descrita a heurística clássica gulosa *First-Fit* Finita (do inglês, *Finite First-Fit*, FFF) para um problema de corte guilhotinado bidimensional. Primeiramente, os itens são rotacionados de forma que suas larguras sejam maiores ou iguais às suas respectivas alturas. Em seguida, eles são ordenados de forma decrescente levando em consideração as suas respectivas larguras. O primeiro item é adicionado ao canto inferior esquerdo da primeira placa e o tamanho do primeiro corte do tipo *1-cut* é definido como a largura desse item. Em seguida, FFF posiciona o item seguinte acima do primeiro. Logo após, FFF tenta posicionar os próximos itens no espaço restante

entre o final do item anterior e o corte do tipo *1-cut* atual. Caso não seja possível, FFF insere o novo item acima dos anteriores. Caso não seja possível, FFF tenta inseri-lo na base na placa e usa a sua largura para definir o tamanho de um novo corte do tipo *1-cut*. Se o último não for possível, uma nova placa é inicializada. O procedimento se repete até que todos os itens sejam cortados. É importante destacar que FFF sempre verifica se é possível posicionar os itens em todos os espaços que sobram à direita dos itens já posicionados. A heurística proposta foi comparada com outros 3 procedimentos heurísticos (*Finite Next-Fit*, *Finite Best-Strip* e *Finite Bottom-Lefit*) e, apesar de não ter obtido os melhores resultados em questão de tempo, foi um dos procedimentos que obteve os melhores resultados no quesito qualidade das soluções.

Em [Aryanezhad et al., 2012] foi abordado um problema de corte guilhotinado bidimensional com itens de orientação fixa. A heurística proposta atribui pesos baseados na largura de cada item a ser cortado, de forma que os itens com maior largura tem maior prioridade na fila de corte. Esta decisão foi tomada com base no princípio de que a maior parte do desperdício nos cortes guilhotinados é criada durante o processo de corte, portanto, na abordagem sugerida, as larguras das peças em cada placa de vidro são iguais ou as mais próximas possíveis às larguras dos cortes. Para os experimentos realizados, esta abordagem foi utilizada ao lado de três outros métodos apresentados em [Whitwell, 2004] (*metaheuristic bottom-left-fill (MBLF), best-fit (BF) + MBLF*, e *interactive approach (IA)*). Como apenas o tempo de execução das heurísticas foi levado em consideração, nada pôde ser concluído a respeito da qualidade das soluções obtidas pelo algoritmo proposto.

Um problema de corte guilhotinado bidimensional sem limitação de estágios foi estudado em [Furini et al., 2016] e uma formulação de Programação Linear Inteira foi proposta para resolvê-lo. Essa modelagem consegue lidar com instâncias de até trinta itens e, através da utilização de técnicas descritas em [Dyckhoff, 1981], requer um número pseudo-polinomial de variáveis e restrições. Para os experimentos foram levados em consideração o tamanho, a efetividade e a qualidade das soluções obtidas.

As instâncias utilizadas estão presentes em [Hifi & Roucairol, 2001] e a formulação descrita neste trabalho foi comparada com as descritas por [Messaoud et al., 2008] e [Dolatabadi et al., 2012]. A abordagem proposta foi capaz de resolver de forma ótima várias das instâncias utilizadas.

2.2 Problema de corte guilhotinado bidimensional com 3 estágios

Em [Puchinger et al., 2004] foram apresentadas heurísticas para resolver o problema de corte guilhotinado bidimensional em três estágios. Duas estratégias baseadas em *Branch & Bound* foram propostas para resolver o problema, porém essas estratégias não garantem resultados ótimos porque utilizam técnicas heurísticas. Tal decisão deu-se ao fato dos autores julgarem impossível resolver o problema acima descrito para instâncias de tamanho prático. Também foram propostos algoritmos evolucionários, dentre eles o algoritmo evolucionário EAe, que utilizam técnicas de recombinação e mutação para gerar soluções viáveis para o problema. Experimentos, realizados em 31 instâncias reais, mostraram que os algoritmos evolucionários apresentaram resultados superiores aos demais no quesito qualidade de solução.

Um problema de corte guilhotinado bidimensional em três estágios foi abordado em [Suliman, 2006] e a heurística sequencial SHP foi proposta para resolvê-lo. Primeiramente, SHP posiciona o primeiro item no canto inferior esquerdo da placa e tenta posicionar os seguintes à sua direita. O processo se repete até que toda a extensão da placa esteja preenchida. Quando não é mais possível adicionar itens à direita dos outros, SHP define o tamanho do primeiro corte do tipo *1-cut* como o comprimento ocupado por todos os itens já posicionados e a altura do primeiro corte do tipo *2-cut* como a altura do maior item anteriormente posicionado. SHP repete o procedimento acima descrito até que não seja mais possível adicionar itens na placa atual. Caso isso

aconteça, uma nova placa é inicializada. O procedimento se repete até que todos os itens tenham sido inseridos na sequência de padrões de corte. É importante destacar que SHP apenas considera o espaço existente no nível atual de corte em que se encontra, ou seja, os novos itens podem ser inseridos apenas acima do último corte do tipo *2-cut* realizado. Os autores concluíram que o desempenho da heurística proposta é melhor do que heurísticas que utilizam medidas fixas para definir os tamanhos dos cortes realizados.

Um problema de corte guilhotinado bidimensional em 3 estágios foi tratado em [Dusberger & Raidl, 2015]. Um procedimento heurístico baseado em *Variable Neighborhood Search* (VNS) foi proposto para resolvê-lo. Para construir uma solução inicial, foram utilizadas três heurísticas baseadas na abordagem *first-fit* presentes em [Dusberger & Raidl, 2014] (*3-staged First Fit Decreasing Height with rotations, matching step* e *Fill Strip*), de modo que a melhor solução fornecida por elas é selecionada. Nos experimentos computacionais foram utilizadas adaptações de instâncias presentes em [Berkey & Wang, 1987] e [Martello & Vigo, 1998] e foi concluído que a heurística proposta neste trabalho supera as abordagens VNS presentes em [Dusberger & Raidl, 2014], pois forneceu soluções melhores em qualidade.

Em [Chen et al., 2015] foi proposta uma heurística que combina técnicas recursivas conhecidas como *Beam Search* ([Hifi et al., 2008]) para resolver um problema de corte guilhotinado com 3 estágios. Nessa abordagem, os pedaços retangulares de material que foram descartados pelo último padrão de corte podem ser reutilizados para cortes futuros. A recursão é utilizada para gerar segmentos que consistem em tiras homogêneas, e uma heurística *Beam Search* é utilizada para obter os padrões de corte em três estágios, levando em consideração as sobras que podem ser utilizadas. A heurística acima descrita foi comparada com um algoritmo para cortes guilhotinados presente em [Andrade et al., 2016]. Experimentos computacionais mostraram que a heurística proposta nesse trabalho obteve soluções melhores do que os de [Andrade et al., 2016].

2.3 Problema de corte guilhotinado bidimensional que trata defeitos nas placas

Um problema de corte guilhotinado bidimensional onde as placas podem conter defeitos e ter tamanhos variados foi tratado em [Jin et al., 2015]. Como não existem restrições de precedência, a heurística proposta primeiramente ordena de forma crescente os lados de maior dimensão das placas, e os itens são ordenados de forma decrescente seguindo o mesmo critério. O algoritmo então tenta posicionar os itens mais largos nas placas menores, e caso isso não seja possível, o item é posicionado na primeira placa que o couber. Depois que todos os itens são posicionados, verifica-se se algum item foi colocado em alguma área com defeito; se sim, esse item é removido, e caso não seja possível posicioná-lo em alguma das placas já utilizadas, ele é inserido ao final. Experimentos computacionais mostraram que esta heurística obteve melhores resultados que os de [Vassiliadis, 2005].

2.4 Problema de corte guilhotinado bidimensional com tipos de precedência

Em [Bennell et al., 2013] foi abordada uma variante do problema de corte guilhotinado bidimensional onde são atribuídos *prazos* a cada um dos itens. Um prazo p_z atribuído a qualquer item $i \in I$, onde I representa o conjunto de itens a serem cortados, indica que o item i deve ser cortando antes que a placa de número $p_z + 1$ seja inicializada. Portanto, o objetivo não é apenas minimizar o número de placas a serem utilizadas mas também minimizar o atraso máximo dos itens. Foi proposto um algoritmo genético para pesquisar o espaço de soluções que utiliza uma heurística de posicionamento para decodificar o gene de melhor *fitness*. Esse algoritmo genético utiliza um operador de cruzamento que considera vários filhos diferentes para cada par de pais. Uma

heurística de busca tabu e uma *randomized descent* também foram propostas, sendo esta última utilizada como referência para comparar os resultados. Os experimentos computacionais realizados mostraram que a busca tabu obteve bons resultados em minimizar as placas, mas o algoritmo genético teve um desempenho um pouco melhor. Quando também considerado o atraso máximo dos itens, o algoritmo genético se mostrou consideravelmente melhor.

No trabalho recente [Freire et al., 2019] foi abordado um problema de corte guilhotinado bidimensional em três estágios e com restrições e precedência onde as placas utilizadas podem possuir defeitos em determinados pontos. Foram propostas uma heurística *multistart* gulosa randomizada e uma técnica de aprimoramento que combina a heurística com uma modelagem de programação lógica por restrições. Os experimentos foram realizados utilizando 2 dos 3 conjuntos de instâncias para o *ROADEF Challenge* 2018. Foram realizadas 2 baterias de experimentos, a primeira com tempo limite de 360 segundos e a segunda com tempo limite de 3600 segundos. Os resultados mostraram que a heurística proposta utilizando a técnica de aprimoramento obteve, em média, $18,00\%$ de desperdício.

Com exceção do trabalho recente [Freire et al., 2019], os algoritmos apresentados até o momento não podem ser aplicados diretamente para o 2DCSP-PC por causa das restrições impostas pela precedência entre os itens. Tais restrições dificultam até que técnicas de agrupamento de itens de mesmo tipo sejam aplicadas, como o caso de [Puchinger et al., 2004]. Portanto, os algoritmos propostos nas seções seguintes foram projetados e adaptados para considerá-las.

Capítulo 3

Heurísticas Construtivas

Nesta seção, apresentamos duas heurísticas construtivas para o 2DCSP-PC baseadas nas heurísticas FFF [Berkey & Wang, 1987] e SHP [Suliman, 2006]. Ao invés de descrever essas heurísticas utilizando a representação em árvore usual de uma solução, nesta dissertação foi adotada uma nova representação com base no que chamamos de representação *k-box*. Uma *0-box* é uma placa por definição. Portanto, sempre tem altura H e largura W. Toda *0-box* é dividida em uma sequência de *1-boxes*. Portanto, uma *1-box* sempre tem altura H, mas pode ter largura variável. Supõe-se que as *1-boxes* sejam colocadas contiguamente da esquerda para a direita em suas respectivas *0-boxes*. Portanto, toda a área de uma *0-box* não coberta por uma *1-box* é uma área não utilizada (resíduo ou desperdício, dependendo se a placa em questão é a última ou não). Analogamente, uma *1-box* é dividida em uma sequência de *2-boxes*. Portanto, uma *2-box* tem sempre a mesma largura da sua respectiva *1-box*, mas pode ter altura variável. Supõe-se que as *2-boxes* sejam colocadas contiguamente de baixo para cima em suas respectivas *1-boxes*. Portanto, toda a área de uma *1-box* não coberta por uma *2-box* é considerada desperdício. Da mesma forma, uma *2-box* é dividida em uma sequência de *3-boxes*, cada uma associada a exatamente um item. Portanto, uma *3-box* tem sempre a mesma altura de sua respectiva *2-box*, mas sua largura é exatamente

igual à do item correspondente. Vale ressaltar que, a partir da largura da *3-box*, pode-se inferir se o item correspondente foi rotacionado ou não. Além disso, está implícito que, se a altura de uma *3-box* for maior que a do item correspondente, ocorrerá um corte de *trimming* (*4-cut*). Também é assumido que as *3-boxes* são colocadas contiguamente da esquerda para a direita de suas respectivas *2-boxes*. Portanto, toda a área do respectivo corte do tipo *2-cut* não coberta por uma *3-box* é considerada desperdício. Ressalta-se o fato de que existe uma correspondência direta entre uma k-*box* e um nó de nível k na árvore de padrões de corte. A vantagem da representação k-*box* de uma solução para o 2DCSP-PC é que ela é mais intuitiva e não é necessário levar em conta a posição exata dos k-*cuts*, mas apenas o tamanho das k-*boxes*, que pode ser inferido a partir da largura e da altura dos itens em cada k-*box*.

3.1 Heurística FFF

Nesta seção, foi estendida a heurística FFF (do inglês, *Finite First-Fit*) de [Berkey & Wang, 1987] para lidar com as restrições de precedência. Como sugerido em [Puchinger et al., 2004], essa heurística assume que todos os itens estejam rotacionados de forma que sua largura seja maior ou igual à altura, e não executa nenhuma rotação posterior. FFF insere os itens na solução na ordem em que aparecem em uma permutação Π dos itens em I. Esta permutação é tal que, se um item i precede um item j em qualquer pilha, i também precede j em Π. Essa condição é necessária para garantir que sempre haja um lugar para inserir os itens sem que as restrições de precedência sejam violadas. Na pior das hipóteses, o próximo item será inserido em uma nova placa vazia sem violar as restrições de precedência, pois todos os itens que o precedem já foram inseridos. Esta ordem é gerada de acordo com um algoritmo de ordenação por seleção (do inglês, *Selection Sort*). Seja c_i um valor-chave associado a um item $i \in I$. A cada iteração do algoritmo de ordenação, o item com o maior valor de c_i, que está no topo de uma pilha $s_i \in S$, é retirado de s_i e adicionado a Π. Foram experimentados três valores para c_i:

(i) a largura do item, (ii) a área do item e (iii) um número gerado aleatoriamente. Essas variantes de FFF são referidas aqui como FFF-W, FFF-A e FFF-R, respectivamente. FFF começa com uma solução vazia e, a cada iteração, insere o próximo item de Π na solução usando a abordagem *first-fit* descrita abaixo.

O pseudocódigo de FFF é descrito no Algoritmo 1. Esta heurística recebe uma instância I para o 2DCSP-PC e uma permutação Π dos itens em I, e retorna uma solução correspondente a uma sequência de padrões de corte $\mathcal{S}$ que descreve como cortar todos os itens em I sem quebrar as restrições de precedência. Na linha 1, uma solução parcial vazia $\mathcal{S}$ é inicializada. A cada iteração do laço das linhas 2 a 22, o próximo item i, de acordo com a permutação Π, é inserido na solução. A seguir, a cada iteração do laço das linhas 3 a 20, FFF percorre cada *0-box* b^0 na mesma ordem em que aparecem em $\mathcal{S}$. Então, a cada iteração do laço das linhas 4 a 15, FFF também avalia cada *1-box* b^1 na ordem em que aparecem em b^0. De maneira análoga, a cada iteração do laço das linhas 5 a 10, FFF analisa cada *2-box* b^2 na ordem em que aparecem em b^1. Se o item i couber em b^2, e todos os itens que precedem i tiverem sido cortados antes do último item em b^2 (ver linha 6), uma nova *3-box* contendo i é anexada a b^2 na linha 7, e a heurística continua para o próximo item em Π na linha 8. Por outro lado, se i não couber em b^2, mas se encaixar como uma nova *2-box* em cima de b^1 sem quebrar as restrições de precedência (consulte a linha 11), uma nova *2-box* (contendo uma única *3-box* com i) é anexada a b^1 na linha 12, e FFF continua para o próximo item em Π na linha 13. Além disso, se o último não for possível, mas i se encaixar como uma nova *1-box* à direita de b^0 sem quebrar as restrições de precedência (consulte a linha 16), uma nova *1-box* (contendo uma única *2-box* com i) é anexada a b^0 na linha 17, e FFF continua para o próximo item em Π na linha 18. Finalmente, se i não couber em nenhuma *k-box* da solução atual, uma nova *0-box* (placa) é acrescentada a $\mathcal{S}$ na linha 21 com uma única *1-box* contendo i. Quando todos os itens são inseridos em $\mathcal{S}$, a solução é retornada na linha 23.

A Figura 3.1 mostra um exemplo de execução de FFF para a instância de exemplo

Algorithm 1 *First-Fit* Finita (FFF)

Entrada: I e Π
Saída: $\mathcal{S}$

```
1: S ← []
2: for each i em Π do
3:     for each b^0 em S do
4:         for each b^1 em b^0 do
5:             for each b^2 em b^1 do
6:                 if i cabe em b^2 sem quebrar as restrições de precedência then
7:                     b^2 ← b^2 : i
8:                     continue para o próximo item em Π
9:                 end if
10:            end for
11:            if [i] cabe em b^1 sem quebrar as restrições de precedência then
12:                b^1 ← b^1 : [i]
13:                continue para o próximo item em Π
14:            end if
15:        end for
16:        if [[i]] cabe em b^0 sem quebrar as restrições de precedência then
17:            b^0 ← b^0 : [[i]]
18:            continue para o próximo item em Π
19:        end if
20:    end for
21:    S ← S : [[[i]]]
22: end for
23: retorna S
```

da Tabela 3.1. Considera-se que as placas têm altura e largura fixas: $H = 3210$ e $W = 3500$. O critério guloso c_i utilizado foi a largura dos itens. Como os mesmos já estão rotacionados de forma que suas larguras sejam maiores ou iguais às suas respectivas alturas, na etapa de pré-processamento é necessário apenas ordená-los (de acordo com o algoritmo de ordenação por seleção descrito acima) pelo valor decrescente de suas larguras para gerar a permutação $\Pi = \{3, 4, 1, 2, 5\}$. Se não existisse precedência entre os itens 1 e 2, o empate entre suas larguras seria decidido através de suas respectivas alturas. Caso as suas alturas também fossem iguais, o empate seria decidido de forma aleatória.

Primeiramente, o item de maior largura 3 é inserido na solução vazia $\mathcal{S}$ definindo como 2000 e 1000 a largura e altura dos primeiros cortes dos tipos *1-cut* e *2-cut*, respectivamente, como mostra a Figura 3.1(a). Em seguida, o item 4 é inserido acima de 1 (consulte a Figura 3.1(b)), definindo a altura do segundo corte do tipo *2-cut* e a

largura do primeiro corte do tipo *3-cut*. Posteriormente, o item 1 é inserido acima do item 4, pois o espaço à direita do item 4 é menor que a largura do item 1, como pode ser observado na figura 3.1(c). De maneira análoga, o item 2 é inserido à direita do item 1 (veja a Figura 3.1(d)). Finalmente, o item 5 é inserido à direita do item 4, como mostra a Figura 3.1(e). As respectivas larguras e alturas dos cortes subsequentes dos tipos *3-cut* e *4-cut* realizados podem ser inferidas a partir das dimensões dos próprios itens. É importante destacar que nenhuma das inserções realizadas quebrou a restrição de precedência entre os itens de mesma pilha. Toda área à esquerda do primeiro e último corte do tipo *1-cut* de comprimento 2000 que não é ocupada por um item é tida como desperdício (cinza claro) e a área à direita do mesmo é considerada resíduo (cinza escuro). A solução presente na figura 1(e) apresenta um desperdício de $26,51\%$.

3.2 Heurística SHP

A heurística SHP (do inglês, *Sequential Heuristic Procedure*) de [Suliman, 2006] também foi estendida para o 2DCSP-PC para lidar com as restrições de precedência. Essa heurística é semelhante à FFF, pois também assume que os itens sejam rotacionados para que eles estejam orientados horizontalmente. Além disso, os itens também são ordenados de acordo com um critério guloso c_i. Novamente foram experimentados três valores para c_i: (i) a largura do item, (ii) a área do item e (iii) um número gerado aleatoriamente. Essas variantes de SHP foram referidas aqui como SHP-W, SHP-A e SHP-R, respectivamente. No entanto, ao invés de definir a largura de uma *1-box*

Tabela 3.1: Instância de exemplo para as heurísticas construtivas FFF e SHP

	Pilha	Posição na Pilha	Largura	Altura
Item 1	1	1	1000	700
Item 2	1	2	1000	600
Item 3	2	1	2000	1000
Item 4	3	1	1500	800
Item 5	3	2	480	400

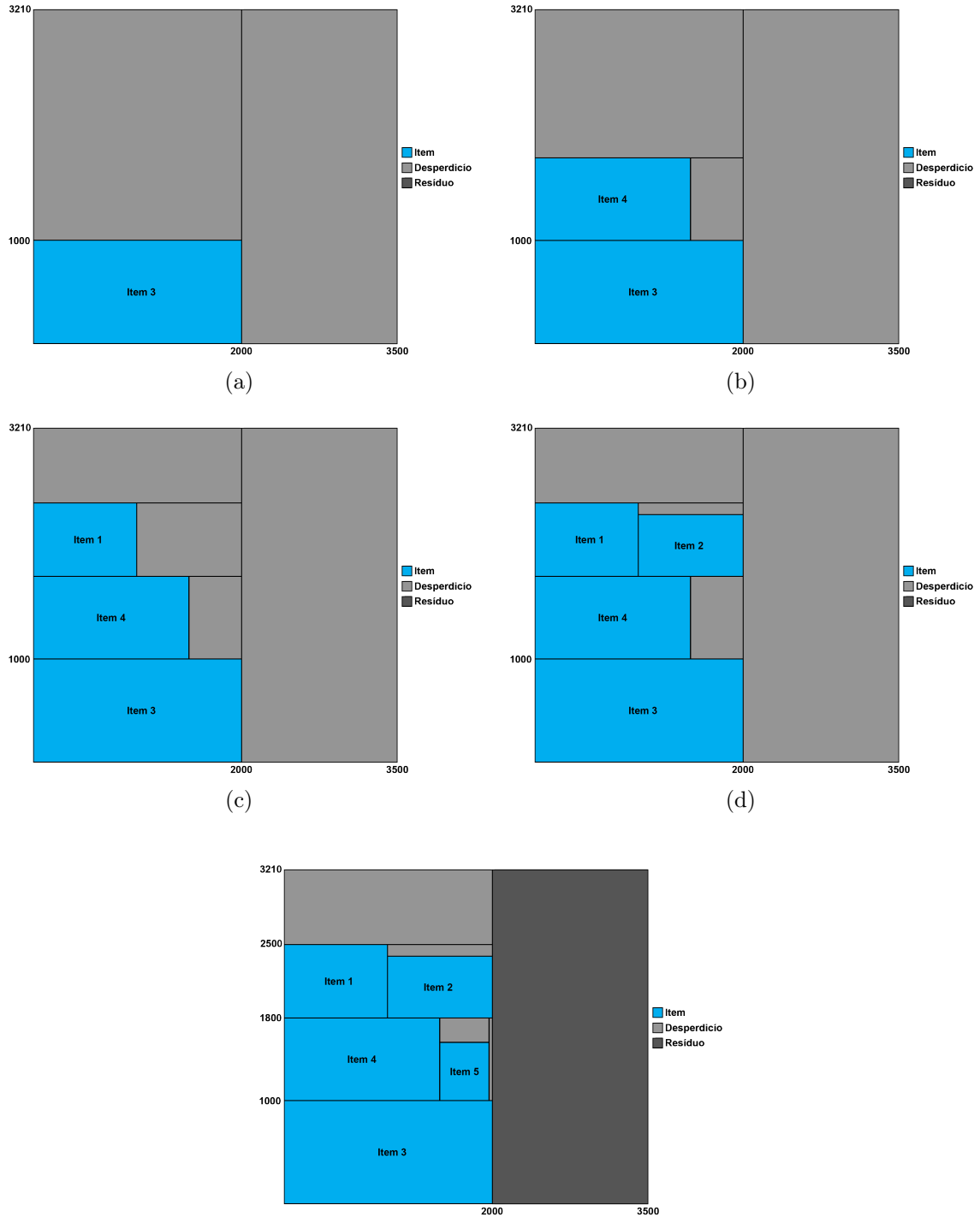

Figura 3.1: Exemplo de execução de FFF para a instância da tabela 3.1

como a largura do primeiro item adicionado a ela, SHP usa uma abordagem diferente, descrita pelo Algoritmo 2.

Os parâmetros de entrada para o Algoritmo 2 formam a tupla $\langle W, H, I, S\rangle$ contendo as dimensões das placas e o conjunto de pilhas com os itens a serem cortados, além de um parâmetro $\overline{W}$ que representa o tamanho máximo permitido para um corte do tipo *1-cut*. A saída correspondente é uma sequência de padrões de corte $\mathcal{S}$ que descreve como cortar todos os itens em I. Denota-se por $T(S) \subseteq I$ como o conjunto de itens que estão no topo das pilhas não vazias de S, e $s(i) \in S$ como a pilha que contém o item $i \in I$. Além disso, uma k-box b^k é representada como uma lista de $(k+1)$-boxes e denota-se $w(b^k)$ e $h(b^k)$ como a largura e a altura de b^k, respectivamente. Para um valor par de k, $w(b^k)$ representa a soma da largura de todas as $(k+1)$-boxes em b^k e $h(b^k)$ representa a altura do item mais alto em b^k. Para um valor ímpar de k, $h(b^k)$ representa a soma da altura de todas as $(k+1)$-boxes em b^k e $w(b^k)$ representa a largura do item mais largo em b^k.

Na primeira linha do Algoritmo 2, uma solução parcial vazia $\mathcal{S}$ é inicializada. A cada iteração do laço das linhas 2 a 17, o padrão de corte de uma placa é decidido. Esse laço é repetido até que todos os itens estejam na solução $\mathcal{S}$. Uma *0-box* vazia b^0 é inicializada na linha 3 e, a cada iteração do laço das linhas 4 a 16, uma nova *1-box* é adicionada a b^0. Esse laço é repetido até que nenhum item em $T(S)$ caiba no espaço restante em b^0. O item i com o maior valor de c_i, que cabe no espaço restante de b^0, é identificado na linha 4. Uma *2-box* (aqui chamada $\bar{b}^2$) é inicializada na linha 5, contendo apenas i, e i é retirado de sua pilha. No entanto, o laço das linhas 6 a 7 continua adicionando itens a $\bar{b}^2$, enquanto $w(\bar{b}^2) \leq \overline{W}$. Conforme sugerido por [Suliman, 2006], a largura de $\bar{b}^2$ define a largura de b^1, que é inicializada na linha 8 com apenas $\bar{b}^2$. A cada iteração do laço das linhas 9 a 13, uma nova *2-box* é adicionada a b^1. Quando não for possível adicionar mais *2-boxes* a b^1, ela será adicionada a b^0 na linha 14. Da mesma forma, quando não for possível adicionar mais *1-boxes* a b^0, ela será adicionada na solução $\mathcal{S}$ na linha 15. Em seguida, quando todos os itens estiverem

em $\mathcal{S}$, ela será retornada na linha 13.

Algorithm 2 *Procedimento Heurístico Sequencial* (SHP)

Entrada: $\langle W, H, I, S\rangle, \overline{W}$
Saída: $\mathcal{S}$

1: Seja $\mathcal{S} \leftarrow [\,]$
2: **while** $T(S) \neq \emptyset$ **do**
3: Seja $b^0 \leftarrow [\,]$
4: **while** $i \neq \emptyset$, onde $i \leftarrow \text{argmax}_{i \in T(S): w_i \leq W - w(b^0) \wedge h_i \leq H} c_i$ **do**
▷ `Inicializa 1-box` b^1.
5: Seja $\bar{b}^2 \leftarrow [i]$, $pop(s(i))$
6: **while** $\ell \neq \emptyset$, onde $\ell \leftarrow \text{argmax}_{\ell \in T(S): w_\ell \leq \min(\overline{W} - w(\bar{b}^2), W - (w(b^0) + w(\bar{b}^2))) \wedge h_\ell \leq H} c_\ell$ **do**
7: $\bar{b}^2 \leftarrow \bar{b}^2 : \ell$, $pop(s(\ell))$
8: **end while**
9: Seja $b^1 \leftarrow [\bar{b}^2]$
▷ `Insere uma sequência de 2-box em` b^1.
10: **while** $j \neq \emptyset$, onde $j \leftarrow \text{argmax}_{j \in T(S): w_j \leq w(b^1) \wedge h_j \leq H - h(b^1)} c_j$ **do**
11: Seja $b^2 \leftarrow [j]$, $pop(s(j))$
12: **while** $k \neq \emptyset$, onde $k \leftarrow \text{argmax}_{k \in T(S): w_k \leq w(b^1) - w(b^2) \wedge h_k \leq h(b^2)} c_k$ **do**
13: $b^2 \leftarrow b^2 : k$, $pop(s(k))$
14: **end while**
15: $b^1 \leftarrow b^1 : b^2$
16: **end while**
17: $b^0 \leftarrow b^0 : b^1$
18: **end while**
19: $\mathcal{S} \leftarrow \mathcal{S} : b^0$
20: **end while**
21: **retorna** $\mathcal{S}$

A Figura 3.2 mostra um exemplo de execução da heurística SHP para a instância de exemplo da Tabela 3.1. Considera-se que as placas têm altura e largura fixas: $H = 3210$ e $W = 3500$. O tamanho máximo dos cortes do tipo *1-cut* foi definido como $\overline{W} = 3500$ e o critério guloso c_i utilizado foi a largura dos itens.

Primeiramente, a fila de prioridades $T(S)$ é construída com o primeiro item de cada pilha e os itens são ordenados de forma decrescente de acordo com suas as larguras. Desta forma, $T(S) = \{3, 4, 1\}$. Em seguida, o primeiro item de $T(S)$ é inserido em $\mathcal{S}$, como se observa na Figura 3.2(a), e removido da fila de prioridades. Como não existem outros itens na pilha do item 3, $T(S)$ permanece com apenas 2 itens. Logo após, o item 4 é inserido à direita do item 3. Como a soma das larguras dos itens 3 e 4 é igual à $\overline{W}$, o tamanho do primeiro corte do tipo *1-cut* é definido como 3500 e a altura do primeiro corte do tipo *2-cut* é definida como a altura do maior dos dois

itens inseridos: 1000 (veja a Figura 3.2(b)). O item 4 é removido de $T(S)$ e o próximo elemento de sua pilha é adicionado de forma ordenada à fila de prioridades resultando em $T(S) = \{1, 5\}$. Posteriormente, o item 1 é inserido acima do item 3 (consulte a Figura 3.2(c)) e removido de $T(S)$. De maneira análoga, o próximo item da pilha de 1 é adicionado à fila de prioridades resultando em $T(S) = \{2, 5\}$. Logo depois, o item 2 é inserido à direita de 1 (veja a Figura 3.2(d)) e removido de $T(S)$. Finalmente, o item 5 é inserido à direita do item 2, como mostra a Figura 3.2(e), e também é removido de $T(S)$ que permanece vazia, pois todos os itens já foram cortados. As respectivas larguras e alturas dos cortes subsequentes dos tipos *3-cut* e *4-cut* realizados podem ser inferidas a partir das dimensões dos próprios itens. É importante destacar que nenhuma das inserções realizadas quebrou as restrições de precedência entre os itens de mesma pilha. Toda área à esquerda do primeiro e último corte do tipo *1-cut* (realizado ao final da placa) que não é ocupada por um item é tida como desperdício (cinza claro). A solução presente na Figura 3.2(e) não possui área residual e apresenta um desperdício de $58,24\%$.

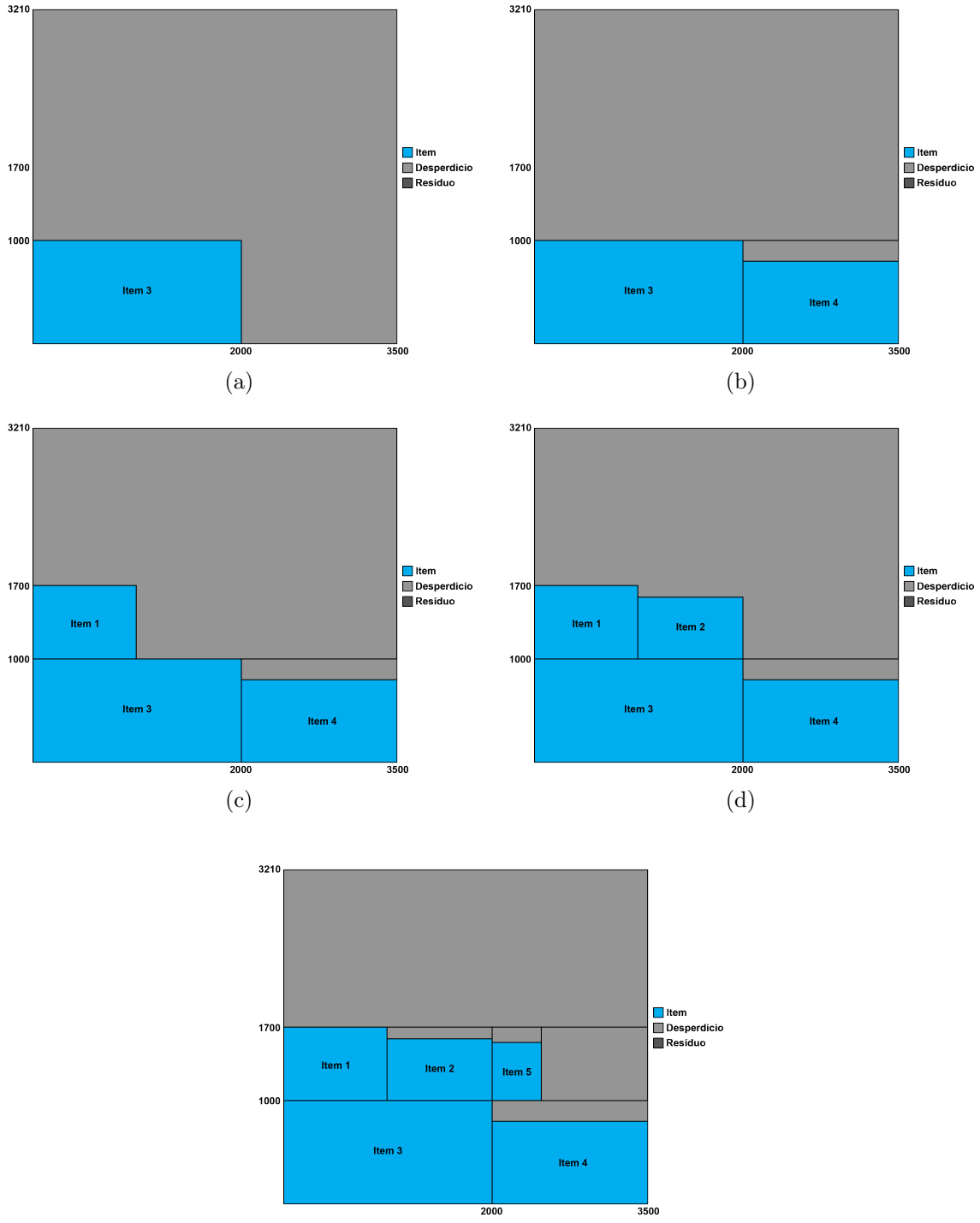

Figura 3.2: Exemplo de execução de SHP para a instância da tabela 3.1

Capítulo 4

Algoritmos Evolucionários

Nesta seção, são apresentados três algoritmos genéticos para o 2DCSP-PC. Os dois primeiros utilizando o *framework* BRKGA de [Gonçalves & Resende, 2011] e o terceiro sendo uma adaptação do *Algoritmo Evolucionário com Representação por Elementos* (do inglês, *Evolutionary Algorithm with Element Representation*, EAe) presente em [Puchinger et al., 2004].

4.1 Algoritmos Genéticos com Chaves Aleatórias Tendenciosos (BRKGA)

Algoritmos genéticos com chaves aleatórias (do inglês, *Random Key Genetic Algorithms*, RKGA) foram introduzidos pela primeira vez por Bean [Bean, 1994] para problemas de otimização combinatória para os quais as soluções podem ser representadas como um vetor de permutação. As soluções são representadas como vetores de números reais gerados aleatoriamente chamados chaves. Um algoritmo determinístico, chamado decodificador, toma como entrada um vetor de solução e associa a ele uma solução viável para o problema de otimização combinatória, para o qual um valor objetivo ou *fitness* pode ser calculado. Dois pais são selecionados aleatoriamente de toda a popu-

lação para implementar a operação de cruzamento na implementação de um RKGA. Os pais podem ser selecionados para acasalar mais de uma vez em uma determinada geração.

Um algoritmo genético com chaves aleatórias tendencioso (BRKGA) difere de um RKGA na maneira como os pais são selecionados para os cruzamentos, [Gonçalves & Resende, 2011]. Em um BRKGA, cada elemento é gerado combinando um elemento selecionado aleatoriamente entre as soluções de elite da população atual, enquanto o outro é uma solução que não é de elite. Dizemos que a seleção é enviesada, pois um pai é sempre um indivíduo de elite e porque essa solução de elite tem uma probabilidade maior de transmitir seus genes para os filhos, ou seja, para a nova geração. Um BRKGA fornece uma melhor implementação da essência do princípio de Darwin de "sobrevivência do mais apto", uma vez que uma solução de elite tem uma maior probabilidade de ser selecionada para acasalamento e os descendentes têm uma maior probabilidade de herdar os genes dos pais elite.

Um algoritmo genético de chaves aleatórias tendencioso para 2DCSP-PC evolui uma população de cromossomos que consiste em vetores de números reais (chamados chaves). Foram utilizados dois tipos diferentes de representação dos vetores de solução.

Representação por ordem simples. Cada solução é representada por um vetor de tamanho $|I|$, no qual cada componente é um número real no intervalo $[0, 1)$ associado a um item em I. Seja k_i a chave associada ao item $i \in I$, o decodificador consiste em executar uma variante das heurísticas construtivas presente na Seção 3 com $c_i = k_i$.

Representação por ordem que permite rotação. Cada solução é representada por um vetor de tamanho $2 \cdot |I|$, no qual a primeira metade dos elementos deste vetor é também um número real no intervalo $[0, 1)$ associado a um item em I conforme a representação por ordem simples. A segunda metade do vetor é composta por chaves que podem assumir os valores 1 ou 0, que ditam se os itens devem ou não, respectivamente, ser rotacionados na etapa de preprocessamento antes da solução ser decodificada. Desta forma, a cada item i são associadas duas chaves diferentes, k_i e k_{n+i}.

Cada solução representada por um cromossomo é decodificada por uma heurística de decodificação que recebe o vetor de chaves e cria uma solução viável para o 2DCSP-PC. Qualquer uma das variantes de FFF e SHP, apresentadas na Seção 3, pode ser utilizada como uma heurística de decodificação para o BRKGA. Como os experimentos computacionais que serão apresentados na Seção 5 mostraram que FFF-W obteve os melhores resultados entre as seis heurísticas construtivas gulosas testadas, a heurística de decodificação escolhida é baseada na FFF-W.

Para respeitar as restrições de precedência existente entre os itens pertencentes à mesma pilha, sempre que chega a vez de um item ser posicionado no padrão de corte, o elemento que está no topo de sua pilha que é escolhido. O custo da solução retornado por essa variante da heurística FFF é utilizado como a função de *fitness* do cromossomo.

Foi utilizado o esquema de cruzamento uniforme parametrizado proposto em [Spears & deJong, 1991] para combinar duas soluções pais e produzir uma solução filho. Nesse esquema, a prole herda cada uma das chaves do pai com melhor *fitness* com probabilidade $\rho > 0,5$ e do pai com menor *fitness* com probabilidade $1 - \rho$. Este algoritmo genético não utiliza o operador de mutação padrão, onde partes dos cromossomos são alteradas com uma pequena probabilidade. Em vez disso, o conceito de mutação utilizado foi o seguinte: um número fixo de soluções mutantes é introduzido na população em cada geração, gerado aleatoriamente da mesma maneira que na população inicial. Os indivíduos que sofrem mutação desempenham o mesmo papel do operador de mutação nos algoritmos genéticos tradicionais, diversificando a pesquisa e ajudando o procedimento a escapar de soluções ótimas locais.

As chaves associadas a cada item são geradas aleatoriamente na população inicial. Em cada geração, a população é particionada em dois conjuntos: TOP e $REST$. Consequentemente, o tamanho da população é $|TOP| + |REST|$. O subconjunto TOP contém as melhores soluções da população. O subconjunto $REST$ é formado por dois subconjuntos separados: MID e BOT, com o subconjunto BOT sendo formado pelos

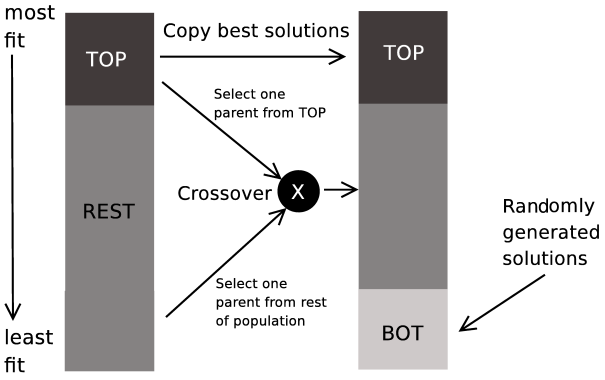

Figura 4.1: Evolução populacional entre gerações consecutivas de um BRKGA.

piores elementos da população atual.

Conforme ilustrado na Figura 4.1, os cromossomos em TOP são simplesmente copiados para a população da próxima geração. Os elementos em BOT são substituídos por indivíduos mutantes recém-criados que são colocados no novo conjunto BOT. Os demais elementos da nova população são obtidos por cruzamento com um pai escolhido aleatoriamente entre TOP e o outro entre $REST$. Isso distingue um AG com chaves aleatórias tendencioso do algoritmo genético de chaves aleatórias de Bean [Bean, 1994] (onde ambos os pais são selecionados aleatoriamente de toda a população). Desde que uma solução pai possa ser escolhida para o cruzamento mais de uma vez em uma determinada geração, as soluções de elite têm maior probabilidade de passar chaves aleatórias para a próxima geração. Dessa forma, $|MID| = |REST| - |BOT|$ soluções descendentes são criadas. O algoritmo para quando um tempo máximo decorrido é atingido.

O pseudocódigo do procedimento heurístico BRKGA é descrito no Algoritmo 3. Esta heurística recebe como parâmetros de entrada $n,\ p,\ p_e,\ p_m$ e ρ que representam os tamanhos dos vetores de chaves, da população, dos subconjuntos TOP e BOT e a probabilidade de herança, respectivamente, e retorna o indivíduo com melhor *fitness* $\mathcal{X}^*$. A linha 1 é responsável por gerar a população inicial $\mathcal{P}$ com p indivíduos, cada um com n chaves geradas aleatoriamente no intervalo $[0, 1)$. Cada execução do laço da linha 2 é responsável pelo processo de evolução populacional entre gerações. Na

linha 3, o algoritmo decodificador é aplicado para calcular o *fitness* de cada novo indivíduo na população atual $\mathcal{P}$. A linha 4 é responsável por particionar a população $\mathcal{P}$ nos subconjuntos $\mathcal{P}_e$ com os p_e melhores indivíduos (elite) e $\mathcal{P}_{\bar{e}}$ com os $p - p_e$ indivíduos restantes. A próxima população $\mathcal{P}^+$ é inicializada na linha 5 contendo todos os indivíduos do subconjunto de soluções de elite p_e. A linha 6 é responsável por gerar o subconjunto $\mathcal{P}_m$ com p_m indivíduos mutantes, cada um tendo n chaves geradas de forma aleatória no intervalo $[0, 1)$. Esses indivíduos mutantes são adicionados em $\mathcal{P}^+$ na linha 7. O laço das linhas 8 a 20 é responsável por preencher o restante da nova população $\mathcal{P}^+$ com novos indivíduos obtidos através das operações de cruzamento. Um pai de elite a e um pai que não pertence ao subconjunto de soluções elite b são selecionados nas linhas 9 e 10, respectivamente. O laço das linhas 11 a 18 é responsável por gerar o indivíduo filho o através das operações de cruzamento entre os pais a e b. A cada iteração deste laço, uma moeda enviesada é lançada com a probabilidade ρ do resultado ser *cara*. Se o resultado for *cara*, a chave atual j é transferida do pai de elite a para o filho o na linha 14, caso contrário, a chave atual j é transferida do pai que não pertence ao subconjunto de soluções elite b para o filho o na linha 16. O filho o recém criado é adicionado à nova população $\mathcal{P}^+$ na linha 19. A linha 21 é responsável por tornar $\mathcal{P}^+$ a população atual $\mathcal{P}$ do algoritmo. Por fim, quando o critério de parada é alcançado, a solução com o melhor valor de *fitness* encontrada até o momento é retornada na linha 23.

4.2 Algoritmo Evolucionário com Representação por Elementos (EAe)

Como parte da adaptação do EAe presente em [Puchinger et al., 2004] para o 2DCSP-PC foi utilizada apenas a *representação por ordem simples* descrita na Seção 4.1 para codificar as soluções candidatas no EAe, porque os experimentos que serão apresen-

Algorithm 3 *Algoritmo Genético com Chaves Aleatórias Tendencioso (BRKGA)*

Entrada: n, p, p_e, p_m, ρ
Saída: Melhor Indivíduo $\mathcal{X}^*$

```
Gera a população P;
while o critério de parada não é alcançado do
    Aplica o algoritmo decodificador para calcular o fitness de cada novo indivíduo em P;
    Particiona P nos subconjuntos P_e e P_ē;
    P+ ← P_e;
    Gera o subconjunto P_m;
    P+ ← P+ ∪ P_m;
    for i ← 1 até p − p_e − p_m do
        Seleciona o pai a aleatoriamente de P_e;
        Seleciona o pai b aleatoriamente de P_ē;
        for j ← 1 até n do
            Joga uma moeda enviesada;
            if resultado da moeda for cara then
                o[j] ← a[j];
            else
                o[j] ← b[j];
            end if
        end for
        P+ ← P+ ∪ {o};
    end for
    P ← P+;
end while
retorna X* ← argmin{f(X)|X ∈ P};
```

tados na Seção 5 mostraram que este modelo de representação apresentou melhores resultados. Com relação à recombinação, o *crossover* de ordem 3 (OX3) de [Davis, 1991] se mostrou particularmente adequado porque respeita parcialmente as posições absolutas e as ordens relativas dos itens, o que é uma condição crucial para o 2DCSP-PC. Além disso, dois operadores de mutação são utilizados, a troca recíproca (do inglês, *reciprocal exchange*, RX) onde duas posições são escolhidas aleatoriamente e trocadas, e a troca de blocos (do inglês, *block exchange*, BX) que corresponde à troca de dois blocos não sobrepostos escolhidos aleatoriamente de comprimento 2^R, com R sendo um valor aleatório no intervalo $(0, ld\ \frac{n}{2}]$, para permitir que blocos menores sejam escolhidos com mais probabilidade.

Os operadores descritos foram integrados em um *algoritmo evolucionário de estado estacionário padrão* (do inglês, *standard steady-state evolutionary algorithm*). As soluções iniciais são criadas aleatoriamente. Em cada iteração, dois pais são selecio-

nados também de forma aleatória conforme descrito em [Eiben & Smith, 2003]. Em seguida, o operador de cruzamento OX3 é sempre aplicado nas soluções pais para gerar uma nova solução filho no qual os operadores de mutação RX e BX são sempre aplicados com igual probabilidade. O número de mutações aplicadas a cada nova solução candidata é escolhido como uma distribuição de Poisson com média esperada 2, tal como utilizada em [Puchinger et al., 2004]. A solução filho recém-criada substitui a pior solução na população. O procedimento é executado até que um tempo máximo estabelecido seja atingido.

Capítulo 5

Experimentos Computacionais

As heurísticas FFF, SHP, as duas variantes do BRKGA, a primeira utilizando a *representação por ordem simples* (aqui chamada de BRKGA) e a segunda utilizando a *representação por ordem que permite rotação* (aqui chamada de BRKGA-R) presentes na Seção 4.1, e a EAe foram implementadas em C++ e compiladas com o GNU g++ versão 4.2.1.

A probabilidade de herança ρ do operador de cruzamento das variantes das heurísticas baseadas no *framework* BRKGA foi definido como $0,7$, conforme utilizado e recomendado em [Brandão et al., 2015; Gonçalves et al., 2005; Noronha et al., 2011; Brandão et al., 2016]. O tamanho da população V para todos os algoritmos evolucionários foi definido como $V = 1000$ como utilizado em [Puchinger et al., 2004]. Para as duas variantes do BRKGA foi utilizada a configuração $|TOP| + |MID| + |BOT| = |V|$. Os tamanhos dos conjuntos TOP, $REST$ e BOT foram configurados para $0,25 \times |V|$, $0,7 \times |V|$ e $0,05 \times |V|$, respectivamente, mais uma vez como sugerido e utilizado em [Noronha et al., 2011] para o problema de atribuição de roteamento e comprimento de onda. Assume-se que o conjunto de placas é infinito e com as seguintes dimensões fixadas: altura $H = 3210$ e largura $W = 6000$. Os experimentos foram realizadas em uma CPU Intel Core de 2.30 GHz i5-7360U com 16 GB de RAM.

Três conjuntos de instâncias foram utilizados nos experimentos, nomeados como *A*, *B* e *X*. Esses conjuntos de instâncias foram utilizados no *ROADEF Challenge* 2018 e podem ser obtidos no site `http://www.roadef.org/challenge/2018/en/instances.php`. Essas instâncias foram adaptadas para o 2DCSP-PC ignorando os defeitos nas placas. Como pode ser observado na Tabela 5.1, as instâncias utilizadas possuem características bem variadas: 1 à 247 pilhas e 5 à 656 itens por instância. A largura dos itens varia de 345 à 3495 e a altura varia de 123 à 2010.

Tabela 5.1: Dados das instâncias dos grupos A, B e X

	Grupo A	Grupo B	Grupo X
Qtd. instâncias	20	15	15
Mín. itens	5	68	124
Média itens	101,95	303,87	284,33
Máx. itens	392	656	412
Mín. pilhas	1	2	2
Média pilhas	11,20	32,00	28,53
Máx. pilhas	72	241	247
Larg. mín. itens	345	351	353
Larg. média itens	1317,35	1410,69	1317,40
Larg. máx. itens	3495	2952	2813
Alt. mín. itens	137	123	193
Alt. média itens	594,82	668,63	600,03
Alt. máx. itens	2010	1759	1828

No primeiro experimento, foram avaliadas e comparadas a qualidade das soluções fornecidas pelas heurísticas FFF e SHP para as instâncias nos conjuntos A, B e X. Foram avaliadas três variantes de FFF: (*i*) *FFF-Width* (FFF-W); (*ii*) *FFF-Area* (FFF-A); e (*iii*) *FFF-Random* (FFF-R). Cada variante ordena os itens de acordo com uma regra diferente. FFF-W ordena os itens de forma decrescente de acordo com sua dimensão máxima, FFF-A ordena os itens de forma decrescente de acordo com a área e FFF-R seleciona aleatoriamente um item para ser inserido no padrão de corte. Também foram avaliadas três variantes de SHP: (*i*) *SHP-Width* (SHP-W); (*ii*) *SHP-Area* (SHP-A); e (*iii*) *SHP-Random* (SHP-R). Cada variante ordena os itens de acordo com uma regra diferente, iguais às de FFF-W, FFF-A e FFF-R, respectivamente. Para as

variantes de SHP, o parâmetro $\overline{W}$ foi definido como a largura do maior item em cada instância. Como FFF-R e SHP-R usam uma ordem aleatória para os itens, elas foram executadas 20 vezes para cada instância usando sementes diferentes para o gerador de números pseudo-aleatórios [Matsumoto & Nishimura, 1998]. Para cada heurística, foi mostrado seu *gap* de otimalidade relativo calculado em relação ao limite inferior $\ell b = \sum_{i \in I} w_i h_i$, ou seja, a soma da área de todos os itens em I.

Os resultados para este experimento são exibidos na Figura 5.1. Esta figura mostra o *gap* de otimalidade relativo médio para as heurísticas FFF-W, FFF-A, FFF-R, SHP-W, SHP-A e SHP-R. Pode-se observar que as variantes da heurística FFF obtiveram *gaps* de otimalidade relativos médios menores que as variantes da heurística SHP para todos os conjuntos de instâncias avaliados. Esses resultados podem estar relacionados ao fato de SHP posicionar itens com a mesma largura máxima lado a lado. Assim, há uma pequena variação na largura dos itens nos estágios finais da SHP, o que pode forçar um padrão de corte muito irregular. Os melhores resultados foram obtidos pela heurística FFF-W. Seu *gap* de otimalidade relativo médio foi de 29,09% para o conjunto de instâncias A, 20,77% para o conjunto de instâncias B e 23,34% para o conjunto de instâncias X, enquanto FFF-A obteve 30,09% para o conjunto de instâncias A, 20,83% para o conjunto de instâncias B e 23,34% para o conjunto de instâncias X, por fim, FFF-R obteve 40,81% para as instâncias do conjunto A, 32,90% para as instâncias de B e 34,95% para as instâncias de X. Esses resultados indicam que FFF-W é a mais promissora entre as heurísticas construtivas avaliadas para o 2DCSP-PC. Portanto, FFF-W será utilizada como heurística decodificadora para todos os algoritmos evolucionários descritos na Seção 4 para o 2DCSP-PC. Para simbolizar tal relação, será acrescentado o sufixo $-FW$ aos nomes dos procedimentos heurísticos aqui descritos.

No segundo experimento, foi investigado se as variantes do BRKGA e o EAe utilizando FFF-W como seus decodificadores identificam eficientemente as relações entre as chaves e boas soluções e converge para soluções quase ideais. Para esse fim, foi

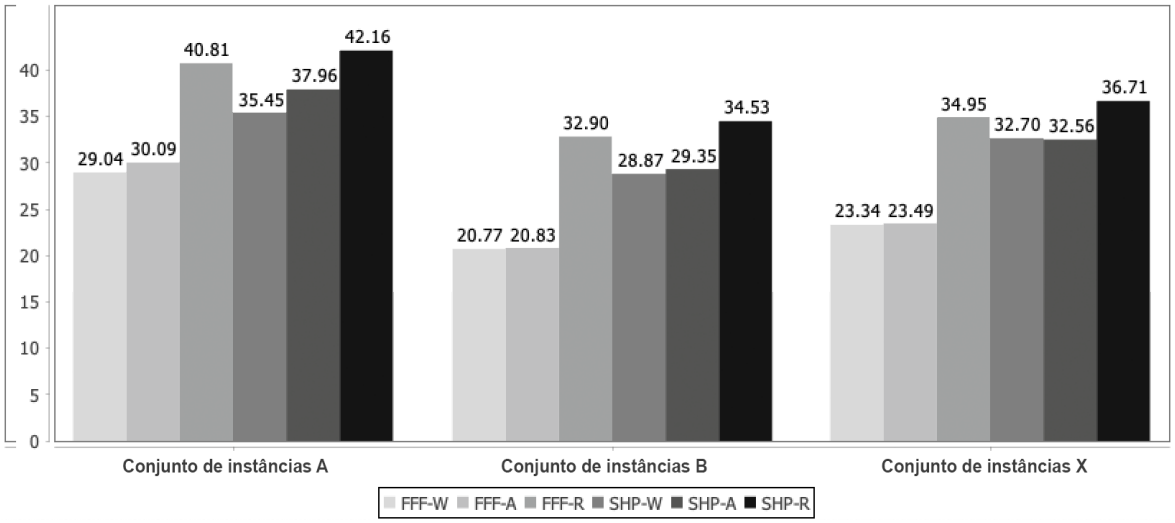

Figura 5.1: *Gap* de otimalidade relativo médio para as heurísticas FFF-W, FFF-A, FFF-R, SHP-W, SHP-A, SHP-R para os conjuntos de instâncias A, B e X.

comparado o desempenho das 3 com o de um procedimento *multistart* (MS) que usa a mesma heurística de decodificação FFF-W. Cada iteração do procedimento *multistart* aplica esta mesma heurística de decodificação a partir de um vetor de chaves gerado de forma aleatória. Ou seja, nada é aprendido de uma iteração do MS para a seguinte. O procedimento *multistart* também utiliza a *representação por ordem simples* presente na Seção 4.1.

O tempo de execução de BRKGA-FW, BRKGA-R-FW, EAe-FW e do MS-FW foi definido como 10 minutos. 20 execuções de cada heurística foram realizadas para cada instância utilizando diferentes sementes para o gerador de números pseudo-aleatórios [Matsumoto & Nishimura, 1998]. Os resultados para os conjuntos de instâncias A, B e X são exibidos respectivamente nas Tabelas 5.2, 5.3 e 5.4. As duas primeiras colunas fornecem o nome da instância e o limite inferior (LB). Os valores médios obtidos pelo BRKGA-FW são exibidos na próxima coluna. O *gap* de otimalidade médio relativo e o coeficiente de variação ($CV = \sigma/_{avg}$) são mostrados nas duas colunas a seguir, onde σ é o desvio padrão da amostra. Os mesmos dados são exibidos para BRKGA-R-FW, EAe-FW e MS-FW nas colunas seguintes.

As tabelas 5.2, 5.3 e 5.4 mostram que o BRKGA-FW encontrou soluções melhores

Tabela 5.2: Resultados obtidos pelo BRKGA-FW, BRKGA-R-FW, EAe-FW e MS-FW para o conjunto A

		BRKGA-FW			BRKGA-R-FW			EAe-FW			MS-FW		
instância	*LB*	média	*gap*(%)	*cv*(%)	média	*gap*(%)	*cv*(%)	média	*gap*(%)	*cv*(%)	média	*gap*(%)	*cv*(%)
A1	4514704	5065380	**10.87**	0.00	5065380	**10.87**	0.00	5065380	**10.87**	0.00	5065380	**10.87**	0.00
A2	77201851	89062734	**13.32**	1.74	103048704	25.08	0.28	89395771.50	13.64	0.92	112307628	31.26	0.24
A3	41796990	52106806.50	**19.79**	1.10	59866500	30.18	0.00	52771116	20.80	1.19	67105692	37.71	0.30
A4	41796990	52179834	**19.90**	0.92	59324652	29.55	3.76	52948308	21.06	1.17	66920796	37.54	0.52
A5	56570007	66987082.50	**15.55**	1.25	78829896	28.24	3.31	68797843.50	17.77	1.16	87880170	35.63	0.39
A6	43254870	50529733.50	**14.40**	0.82	50332800	14.06	0.33	51379260	15.81	0.11	65426220	33.89	1.54
A7	70195170	87764770.50	**20.02**	1.16	93312132	24.77	3.52	90280608	22.25	1.17	111920502	37.28	0.13
A8	138045196	173705779.50	**20.53**	1.92	195067848	29.23	1.09	178017130.50	22.45	0.80	217863984	36.64	0.07
A9	44879034	57150198	**21.47**	4.63	61718670	27.28	0.00	60268552.50	25.53	0.81	74625438	39.86	3.64
A10	71100239	90379636.50	**21.33**	1.36	101291550	29.81	0.66	97504231.50	27.08	2.44	117702996	39.59	3.01
A11	64444211	79741696.50	**19.18**	0.23	88327002	27.04	1.72	81957559.50	21.37	1.43	99309054	35.11	3.23
A12	29180006	36711646.50	**20.52**	2.58	42567810	31.45	0.00	41106778.50	29.01	0.71	49911006	41.54	0.36
A13	213400977	241819732.50	**11.75**	0.73	293020998	27.17	1.73	248723640	14.20	1.02	292087530	26.94	0.82
A14	226360542	275980552.50	**17.98**	0.99	308498334	26.63	1.34	285428224.50	20.69	1.03	346676148	34.71	0.88
A15	238633039	287074954.50	**16.87**	1.33	332077710	28.14	0.35	298544445	20.07	0.71	369954426	35.50	0.72
A16	37325677	48079380	**22.37**	0.00	48079380	**22.37**	0.00	48846570	23.59	1.21	55812912	33.12	0.12
A17	19623149	43447350	54.83	0.00	36455970	**46.17**	0.00	43447350	54.83	0.00	43447350	54.83	0.00
A18	60282102	77705433	**22.42**	2.00	81420366	25.96	1.13	82651561.50	27.06	2.25	102697530	41.30	0.00
A19	41044876	51516808.50	**20.33**	1.23	57435246	28.54	4.19	57188557.50	28.23	4.46	69775770	41.18	0.00
A20	14710475	22717170	35.25	0.00	18313050	**19.67**	0.00	22717170	35.25	0.00	25868106	43.13	0.16
Average			**20.93**			26.61			23.58			36.38	

Tabela 5.3: Resultados obtidos pelo BRKGA-FW, BRKGA-R-FW, EAe-FW e MS-FW para o conjunto B

		BRKGA-FW			BRKGA-R-FW			EAe-FW			MS-FW		
instância	*LB*	média	*gap*(%)	*cv*(%)	média	*gap*(%)	*cv*(%)	média	*gap*(%)	*cv*(%)	média	*gap*(%)	*cv*(%)
B1	77110392	85795596	10.12	1.00	90451380	14.75	1.60	84841102.50	**9.11**	0.24	102887562	25.05	0.01
B2	315354085	386794567.50	**18.47**	1.13	479755686	34.27	2.80	399389484	21.04	0.66	489664314	35.60	0.33
B3	349989487	413512200	**15.36**	0.72	457845510	23.56	0.60	420631017	16.79	0.62	514256766	31.94	0.07
B4	148205615	176400895.50	**15.98**	0.23	194164554	23.67	1.98	181107558	18.17	0.74	217947444	32.00	0.08
B5	319711555	514094340	**37.81**	0.00	516464604	38.10	1.68	514094340	**37.81**	0.00	552614340	42.15	0.00
B6	192874073	228760810.50	**15.69**	1.45	257330934	25.05	1.00	237782836.50	18.89	0.87	287522268	32.92	1.22
B7	187746291	212908867.50	**11.82**	1.41	261597666	28.23	0.70	226677681	17.17	1.03	248054034	24.31	0.04
B8	339397811	392079511.50	**13.44**	0.65	431302662	21.31	0.61	407338407	16.68	0.68	485745546	30.13	0.52
B9	293827643	334129060.50	**12.06**	0.61	361471680	18.71	1.36	343059762	14.35	0.82	416792820	29.50	0.00
B10	345904837	403732453.50	**14.32**	1.23	454169418	23.84	0.38	424282392	18.47	0.99	492562944	29.77	0.32
B11	336052870	392322990	**14.34**	0.84	442156956	24.00	1.65	413888733	18.81	0.53	510982566	34.23	0.10
B12	259876763	317668341	**18.19**	0.80	380235414	31.65	1.31	335368602	22.51	0.54	391566072	33.63	0.01
B13	484072875	588971281.50	**17.81**	0.94	684040728	29.23	0.53	623506387.50	22.36	0.66	731049894	33.78	0.39
B14	176124110	212317906.50	**17.05**	1.32	273121566	35.51	0.26	223866042	21.33	0.88	265370058	33.63	0.09
B15	432558079	521598999	**17.07**	0.71	581375940	25.60	1.07	542040118.50	20.20	0.91	626649780	30.97	0.11
Average			**12.48**			19.87			14.68			23.98	

que o BRKGA-R-FW, EAe-FW e MS-FW para a grande maioria das instâncias nos conjuntos A, B e X. BRKGA-FW, BRKGA-R-FW e EAe-FW obtiveram resultados estritamente melhores que MS-FW, o que mostra que eles efetivamente convergem para soluções de melhor custo que um algoritmo aleatório de *multistart*.

Em relação ao conjunto de instâncias A, o *gap* de otimalidade médio observado para o BRKGA-FW foi de apenas 20,93%, enquanto os *gaps* de otimalidade médios para BRKGA-R-FW, EAe-FW e MS-FW foram de 26,61%, 23,58% e 36,38%, respectivamente. Em relação ao conjunto de instâncias B, a Tabela 5.3 mostra que o

Tabela 5.4: Resultados obtidos pelo BRKGA-FW, BRKGA-R-FW, EAe-FW e MS-FW para o conjunto X

		BRKGA-FW			BRKGA-R-FW			EAe-FW			MS-FW		
instância	*LB*	média	*gap*(%)	*cv*(%)	média	*gap*(%)	*cv*(%)	média	*gap*(%)	*cv*(%)	média	*gap*(%)	*cv*(%)
X1	244983403	296505774	**17.38**	0.76	333387390	26.52	0.57	298489233	17.93	0.74	379006626	35.36	0.03
X2	147062803	167277273	**12.08**	1.22	217779240	32.47	1.37	178930054.50	17.81	0.81	199020000	26.11	0.00
X3	156830774	186547866	**15.93**	1.16	224961936	30.29	1.90	193683535.50	19.03	1.33	233533278	32.84	1.01
X4	241257058	283733986.50	**14.97**	1.12	354893106	32.02	0.86	293411013	17.78	0.51	346695408	30.41	0.78
X5	78796003	98487775.50	**19.99**	1.05	119191152	33.89	0.95	102141076.50	22.86	0.59	124980066	36.95	0.28
X6	248147317	293961849	**15.59**	0.68	334620672	25.84	0.35	306751773	19.10	0.98	376208790	34.04	0.09
X7	369443070	438367230	**15.72**	0.88	482443740	23.42	1.76	455261299.50	18.85	0.77	534408504	30.87	0.18
X8	134427339	255320190	47.35	1.57	223755618	**39.92**	1.28	256652340	47.62	0.00	295172340	54.46	0.00
X9	361189695	427407969	**15.49**	1.06	450398952	19.81	2.06	448845793.50	19.53	1.09	528156066	31.61	0.31
X10	333756718	393396253.50	**15.16**	0.57	455869434	26.79	1.28	412772455.50	19.14	0.87	485979234	31.32	0.53
X11	223408308	276445039.50	**19.19**	1.12	311501610	28.28	0.27	290673204	23.14	1.26	353681652	36.83	0.36
X12	242393345	291204138	**16.76**	1.44	329759448	26.49	0.86	310651762.50	21.97	1.08	361575684	32.96	0.17
X13	259467828	301387542	**13.91**	0.78	368667216	29.62	0.92	321968938.50	19.41	0.98	382007976	32.08	0.07
X14	158409238	196369824	**19.33**	1.00	217893516	27.30	2.13	205005205.50	22.73	0.85	238304622	33.53	0.33
X15	242691036	299699563.50	**19.02**	1.71	341847666	29.01	1.67	322427166	24.73	1.27	386605980	37.23	0.62
Average			**13.89**			21.58			16.58			25.83	

BRKGA-FW encontrou soluções melhores que BRKGA-R-FW, EAe-FW e MS-FW para quase todas as instâncias deste conjunto, perdendo para EAe-FW em apenas uma. Os *gaps* de otimalidade médios foram de apenas 12,48% para BRKGA-FW e 19,87%, 14,68% e 23,98% para BRKGA-R-FW, EAe-FW e MS-FW, respectivamente. Resultados semelhantes foram observados para o conjunto de instâncias X. Pode ser observado na Tabela 5.4 que o BRKGA-FW encontrou soluções melhores que para quase todas as instâncias avaliadas neste conjunto, perdendo apenas para BRKGA-R-FW em uma instância. Os *gaps* de otimalidade médios alcançados foram de 13,89% para o BRKGA-FW e 21,58%, 16,58% e 25,83% para BRKGA-R-FW, EAe-FW e MS-FW, respectivamente.

Para avaliar a qualidade das soluções encontradas em função do tempo de execução foram escolhidas 3 instâncias: A13, B7 e X7, uma de cada um dos 3 grupos de instâncias A, B e X. O critério de seleção utilizado foi a variabilidade dos itens e das pilhas em cada instância. Foram avaliadas as 2 heurísticas que obtiveram os melhores resultados: BRKGA-FW e EAe-FW. Foi utilizado um gráfico do tipo *boxplot* que mostra o indivíduo com melhor *fitness* em cada minuto de execução para cada uma das 20 execuções que foram realizadas.

As Figuras 5.2 e 5.3 apresentam a qualidade das soluções encontradas ao longo

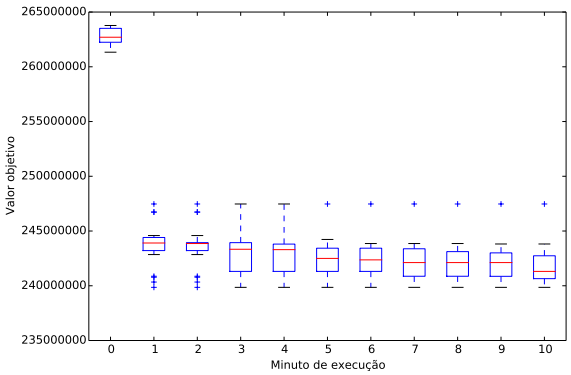

Figura 5.2: Qualidade da solução em função do tempo para o BRKGA-FW na instância A13.

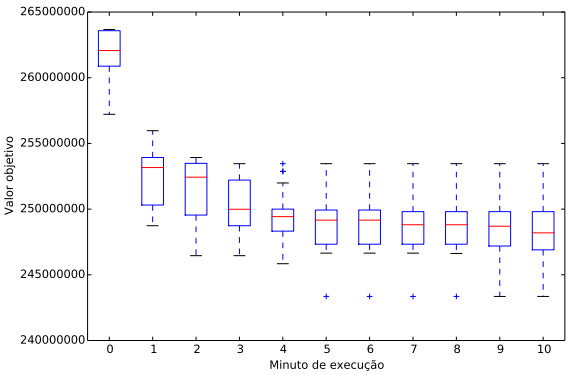

Figura 5.3: Qualidade da solução em função do tempo para o EAe-FW na instância A13.

dos 10 minutos de execução para as heurísticas BRKGA-FW e EAe-FW, respectivamente, para a instância A13. Pode-se observar que existe uma melhora significativa das soluções encontradas entre início de execução dos algoritmos e o primeiro minuto de execução dos mesmos. Os minutos seguintes apresentam pouca melhora e no minuto final pode-se observar uma melhora maior. Entretanto, a heurística BRKGA-FW obteve melhores resultados em média. Um comportamento similar pode ser observado nas Figuras 5.4 e 5.5 que apresentam a qualidade das soluções encontradas ao longo dos 10 minutos de execução para as heurísticas BRKGA-FW e EAe-FW, respectivamente, para a instância B7. Por fim, nas Figuras 5.6 e 5.7, que mostram a qualidade das soluções encontradas as longo dos 10 minutos de execução para as heurísticas BRKGA-FW e EAe-FW, respectivamente, para a instância X7, observa-se um comportamento parecido, com melhoras significativas entre o início de execução dos algoritmos e o primeiro minuto de execução dos mesmos e pouca melhora nos minutos seguintes. A heurística BRKGA-FW também obteve os melhores resultados em média.

Todos esses resultados indicam que a heurística BRKGA-FW foi eficiente na identificação da relação entre chaves e boas soluções, sendo capaz de obter resultados melhores que BRKGA-R-FW, EAe-FW e MS-FW para a grande maioria das instâncias avaliadas.

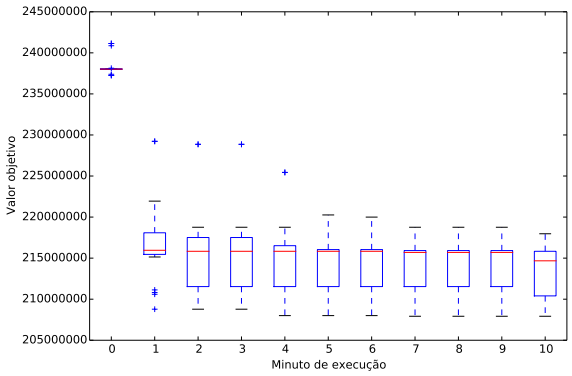

Figura 5.4: Qualidade da solução em função do tempo para o BRKGA-FW na instância B7.

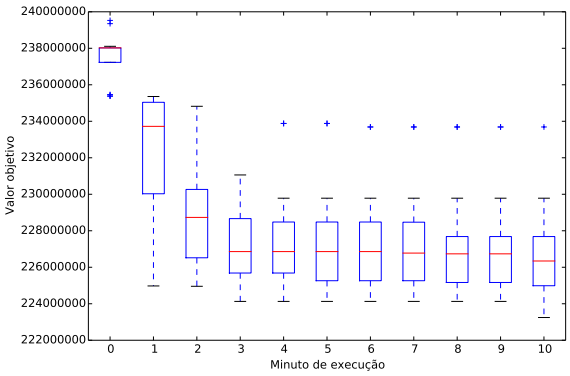

Figura 5.5: Qualidade da solução em função do tempo para o EAe-FW na instância B7.

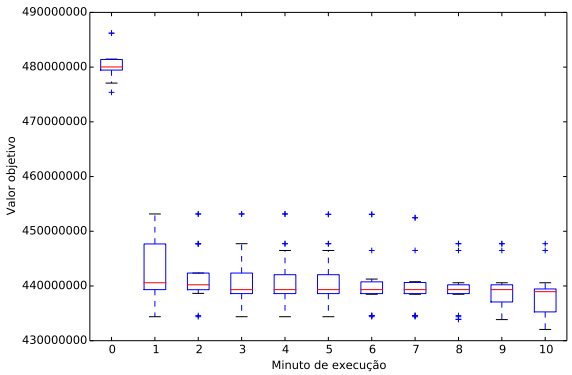

Figura 5.6: Qualidade da solução em função do tempo para o BRKGA-FW na instância X7.

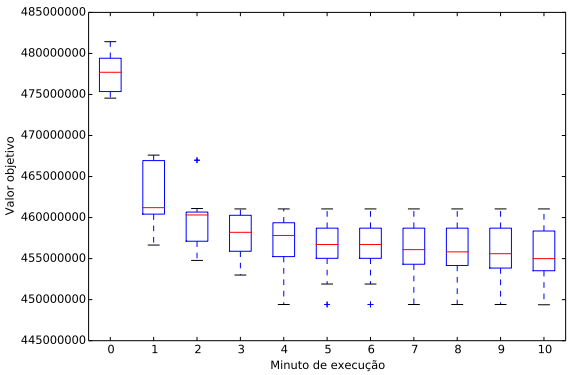

Figura 5.7: Qualidade da solução em função do tempo para o EAe-FW na instância X7.

Capítulo 6

Conclusões

Neste trabalho, foi abordado o problema de corte guilhotinado bidimensional em três estágios e com restrições de precedência. Foram introduzidas duas heurísticas construtivas e três algoritmos genéticos, sendo dois variantes do *framework* BRKGA de [Gonçalves & Resende, 2011] e uma adaptação do *algoritmo evolucionário de estado estacionário padrão* (EAe) de [Puchinger et al., 2004] que usam a melhor das heurísticas construtivas como seus algoritmos decodificadores. Os experimentos que foram realizados mostraram que a variante da heurística FFF que ordena os itens de forma decrescente de acordo com sua maior dimensão (FFF-W) obteve os melhores resultados entre as heurísticas construtivas, com um *gap* de otimalidade de 29,09% para o conjunto de instâncias A, 20,77% para o conjunto de instâncias B e 23,34% para o conjunto de instâncias X.

Além disso, foram comparados os resultados de duas heurísticas BRKGA, uma considerando que os itens podem ser rotacionados e a outra não (BRKGA-R-FW e BRKGA-FW, respectivamente), com o algoritmo genético EAe de [Puchinger et al., 2004] e um procedimento *multistart*. Todos os quatro utilizando a heurística FFF-W como algoritmo decodificador. Os resultados mostram que o BRKGA-FW encontrou soluções melhores que as outras três heurísticas para a grande maioria das instâncias

nos conjuntos A, B e X, alcançando um *gap* de otimalidade médio de 20,93% para o conjunto de instâncias A, 12,48% para o conjunto de instâncias B e 13,89% para o conjunto de instâncias X.

Observa-se que o algoritmo genético BRKGA-R-FW obteve resultados em média piores que os algoritmos genéticos BRKGA-FW e EAe-FW. Isso pode-se dar ao fato de que dobrar o vetor de chaves para decidir se os itens devem ou não ser rotacionados antes de serem posicionados nos padrões de corte aumenta significativamente a complexidade do problema e o espaço de busca de soluções gerando resultados quase que majoritariamente piores que BRKGA-FW e EAe-FW para as instâncias testadas no espaço de tempo estipulado de 10 minutos.

Também pode ser observado que o BRKGA-FW obteve resultados quase que estritamente melhores que o EAe-FW. Duas hipóteses foram levantadas para justificar este fato, sendo a primeira relacionada ao fato de o BRKGA-FW fornecer uma melhor implementação da essência do princípio de Darwin de "sobrevivência do mais apto" ao sempre selecionar uma solução pai do conjunto de soluções elite para gerar filhos que, em tese, também terão *fitness* bons. A segunda está relacionada ao fato de que EAe-FW, por se tratar de um *algoritmo evolucionário de estado estacionário padrão* e trabalhar sempre com a mesma população, não possui um espaço de soluções amplo e diverso como o BRKGA-FW.

Para melhorar os limites inferiores obtidos, as possíveis extensões deste trabalho incluem a proposta de métodos exatos, como Programação Linear Inteira, para resolver o problema. Como alternativa, outros métodos heurísticos que não dependem de algoritmos genéticos podem ser criados para o problema, como heurísticas baseadas em Programação Dinâmica, Programação Por Restrições e técnicas que combinam heurísticas e Mineração de Dados.

Referências Bibliográficas

Alvarez-Valdes, R.; Martí, R.; Tamarit, J. M. & Parajón, A. (2007). Grasp and path relinking for the two-dimensional two-stage cutting-stock problem. *INFORMS Journal on Computing*, 19(2):261--272.

Andrade, R.; Birgin, E. G. & Morabito, R. (2016). Two-stage two-dimensional guillotine cutting stock problems with usable leftover. *International Transactions in Operational Research*, 23(1-2):121--145.

Aryanezhad, M.-B.; Hashemi, N. F.; Makui, A. & Javanshir, H. (2012). A simple approach to the two-dimensional guillotine cutting stock problem. *Journal of Industrial Engineering International*, pp. 8--21.

Bean, J. C. (1994). Genetic algorithms and random keys for sequencing and optimization. *ORSA Journal on Computing*, 2:154--160.

Bennell, J. A.; Lee, L. S. & Potts, C. N. (2013). A genetic algorithm for two-dimensional bin packing with due dates. *International Journal of Production Economics*, 145(2):547--560.

Berkey, J. O. & Wang, P. Y. (1987). Two-dimensional finite bin-packing algorithms. *Journal of the Operational Research Society*, 38(5):423–429.

Brandão, J. S.; Noronha, T. F.; Resende, M. G. C. & Ribeiro, C. C. (2015). A biased random-key genetic algorithm for single-round divisible load scheduling. *International Transactions Operational Research*, 22:823--839.

Brandão, J. S.; Noronha, T. F. & Ribeiro, C. C. (2016). A biased random-key genetic algorithm to maximize the number of accepted lightpaths in wdm optical networks. *Journal of Global Optimization*, 65(4):813--835.

Chen, Q.; Chen, Y.; Cui, Y.; Lu, X. & Li, L. (2015). A heuristic for the 3-staged 2d cutting stock problem with usable leftover. In *2015 International Conference on Electrical, Automation and Mechanical Engineering*, pp. 776--779, Phuket. Atlantis Press. ISSN 2352-5401.

Cherri, A. C.; Arenales, M. N. & Yanasse, H. H. (2009). The one-dimensional cutting stock problem with usable leftover–a heuristic approach. *European Journal of Operational Research*, 196(3):897--908.

Cui, Y.; Song, X.; Chen, Y. & Cui, Y. (2017). New model and heuristic solution approach for one-dimensional cutting stock problem with usable leftovers. *Journal of the Operational Research Society*, 68(3):269--280.

Davis, L. (1991). *Handbook of genetic algorithms*. CumInCAD.

Dolatabadi, M.; Lodi, A. & Monaci, M. (2012). Exact algorithms for the two-dimensional guillotine knapsack. *Computers & Operations Research*, 39(1):48--53.

Dusberger, F. & Raidl, G. R. (2014). A variable neighborhood search using very large neighborhood structures for the 3-staged 2-dimensional cutting stock problem. In *International Workshop on Hybrid Metaheuristics*, pp. 85--99. Springer.

Dusberger, F. & Raidl, G. R. (2015). Solving the 3-staged 2-dimensional cutting stock problem by dynamic programming and variable neighborhood search. *Electronic Notes in Discrete Mathematics*, 47:133--140.

Dyckhoff, H. (1981). A new linear programming approach to the cutting stock problem. *Operations Research*, 29(6):1092--1104.

Dyckhoff, H. & Finke, U. (1992). *Cutting and packing in production and distribution: A typology and bibliography.* Springer Science & Business Media.

Eiben, A. E. & Smith, J. E. (2003). *Introduction to evolutionary computing.* Springer.

Freire, J.; Queiroz, M.; Modesto, L.; Carvalho, M. & Melo, R. (2019). Uma heurística baseada em programação lógica por restrições para um problema restrito de corte bidimensional guilhotinado. In *LI Simpósio Brasileiro De Pesquisa Operacional*, Limeira, SP - Brasil. Galoá.

Furini, F.; Malaguti, E. & Thomopulos, D. (2016). Modeling two-dimensional guillotine cutting problems via integer programming. *INFORMS Journal on Computing*, 28:736--751.

Gonçalves, J. F.; Mendes, J. J. M. & Resende, M. G. C. (2005). A hybrid genetic algorithm for the job shop scheduling problem. *European Journal of Operational Research*, 167:77--95.

Gonçalves, J. F. & Resende, M. G. (2013). A biased random key genetic algorithm for 2d and 3d bin packing problems. *International Journal of Production Economics*, 145(2):500--510.

Gonçalves, J. F. & Resende, M. G. C. (2011). Biased random-key genetic algorithms for combinatorial optimization. *Journal of Heuristics*, 17:487--525.

Hifi, M.; M'Hallah, R. & Saadi, T. (2008). Algorithms for the constrained two-staged two-dimensional cutting problem. *INFORMS Journal on Computing*, 20(2):212--221.

Hifi, M. & Roucairol, C. (2001). Approximate and exact algorithms for constrained (un) weighted two-dimensional two-staged cutting stock problems. *Journal of combinatorial optimization*, 5(4):465--494.

Jin, M.; Ge, P. & Ren, P. (2015). A new heuristic algorithm for two-dimensional defective stock guillotine cutting stock problem with multiple stock sizes. *Tehnicki vjesnik/Technical Gazette*, 22(5).

Lai, K. K. & Chan, J. (1997). Developing a simulated annealing algorithm for the cutting stock problem. *Computers & Industrial Engineering*, 32:115–127.

Lodi, A.; Martello, S. & Monaci, M. (2002a). Two-dimensional packing problems: A survey. *European journal of operational research*, 141(2):241--252.

Lodi, A.; Martello, S. & Vigo, D. (2002b). Recent advances on two-dimensional bin packing problems. *Discrete Applied Mathematics*, 123(1-3):379--396.

Lydia, T. & Quentin, V. (2018). Challenge roadef / euro 2018 cutting optimization problem description. `https://www.roadef.org/challenge/2018/files/Challenge_ROADEF_EURO_SG_Description.pdf`.

Martello, S. & Vigo, D. (1998). Exact solution of the two-dimensional finite bin packing problem. *Management science*, 44(3):388--399.

Matsumoto, M. & Nishimura, T. (1998). Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 8(1):3--30.

Messaoud, S. B.; Chu, C. & Espinouse, M.-L. (2008). Characterization and modelling of guillotine constraints. *European Journal of Operational Research*, 191(1):112--126.

Noronha, T. F.; Resende, M. G. C. & Ribeiro, C. C. (2011). A biased random-key genetic algorithm for routing and wavelength assignment. *Journal of Global Optimization*, 50:503--518.

Puchinger, J.; Raidl, G. R. & Koller, G. (2004). Solving a real-world glass cutting problem. In *European Conference on Evolutionary Computation in Combinatorial Optimization*, pp. 165--176. Springer.

Spears, W. & deJong, K. (1991). On the virtues of parameterized uniform crossover. In Belew, R. & Booker, L., editores, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pp. 230--236, San Mateo. Morgan Kaufman.

Suliman, S. (2006). A sequential heuristic procedure for the two-dimensional cutting-stock problem. *International Journal of Production Economics*, 99(1-2):177--185.

Vassiliadis, V. S. (2005). Two-dimensional stock cutting and rectangle packing: binary tree model representation for local search optimization methods. *Journal of food engineering*, 70(3):257--268.

Whitwell, G. (2004). *Novel heuristic and metaheuristic approaches to cutting and packing*. Tese de doutorado, University of Nottingham.

Yanasse, H. H. & Morabito, R. (2006). Linear models for 1-group two-dimensional guillotine cutting problems. *International Journal of Production Research*, 44(17):3471--3491.