UNIVERSIDADE DO ESTADO DO RIO GRANDE DO NORTE
UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Jonathan Lopes da Silva

Contribuições em otimização combinatória para o problema de corte bidimensional guilhotinado não-estagiado

Mossoró - RN
2017

Jonathan Lopes da Silva

Contribuições em otimização combinatória para o problema de corte bidimensional guilhotinado não-estagiado

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação - associação ampla entre a Universidade do Estado do Rio Grande do Norte e a Universidade Federal Rural do Semi-Árido, para a obtenção do título de Mestre em Ciência da Computação.

Orientador: Prof. Dario José Aloise, Dr.

Mossoró - RN
2017

S586c Silva, Jonathan Lopes da.
Contribuições em otimização combinatória para o problema de corte bidimensional guilhotinado não-estagiado / Jonathan Lopes da Silva. - 2017.
128 f. : il.

Orientador: Dario José Aloise.
Dissertação (Mestrado) - Universidade Federal Rural do Semi-árido, Programa de Pós-graduação em Ciência da Computação, 2017.

1. Corte bidimensional guilhotinado. 2. Programação linear. 3. Branch and bound. 4. Metaheurística. 5. Time assíncrono (A-Team). I. Aloise, Dario José, orient. II. Título.

JONATHAN LOPES DA SILVA

Contribuições em otimização combinatória para o problema de corte bidimensional guilhotinado não-estagiado.

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação para a obtenção do título de Mestre em Ciência da Computação.

APROVADA EM: 23 / 08 / 2017

Prof. Dr. Dario José Aloise
Orientador e Presidente

Prof. Dr. Francisco Chagas de Lima Júnior
Universidade do Estado do Rio Grande do Norte

Prof. Dr. Carlos Heitor Pereira Liberalino
Universidade do Estado do Rio Grande do Norte

Prof. Dr. Hugo Alexandre Dantas do Nascimento
Universidade Federal de Goiás - UFG

Dedico este trabalho à minha família e meus amigos que me apoiaram e, por vezes, me carregaram durante o mestrado.

AGRADECIMENTOS

Agradeço a Deus por ter me dado a oportunidade de concluir mais um projeto.

Agradeço a todos os professores do Programa de Pós-graduação em Ciência da Computação da UERN e UFERSA, em especial, ao professor Dr. Dario José Aloise, por ter me aceitado como seu orientando e me dando o suporte necessário para a conclusão deste trabalho, e aos professores Dr. Francisco Chagas de Lima Júnior e Dr. Carlos Heitor Pereira Liberalino pelo apoio e auxílio na implementação de alguns dos algoritmos utilizados na pesquisa.

À minha família e aos meus amigos, em especial à minha mãe, Antonia Ferreira, pelo apoio que me deram durante a realização do mestrado e pela compreensão que tiveram frente as minhas ausências em almoços, jantares e demais eventos que não pude comparecer. Agradeço a excelentíssima senhorita Gracinha (Maria das Graças) pelo carinho e paciência (e brigas também, né!?) gastos com este que talvez não merecesse (mas as brigas eu mereci).

À Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) e ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) pelo apoio financeiro dado durante o desenvolvimento da pesquisa.

Gostaria de deixar aqui registrado meu agradecimento aos meus amigos e companheiros de mestrado Thiago Henrique, Wannyemberg Klaybin, Jocksam Matos e Hallyson Duarte (já mestres, nesta data). Eles me auxiliaram a regressar ao mestrado após tê-lo abandonado por motivos pessoais. Agradeço ainda à Marianny Fidelis, pelo auxílio e experiência compartilhados. Tal conhecimento foi fundamental para que os resultados fossem alcançados. Por fim, e não menos importante, à secretária do mestrado, Rosita Rodrigues, que juntamente ao professor Dr. Dario, me ajudaram a manter o foco na pesquisa com conselhos e repreensões.

"If I have seen further it is by standing on the shoulders of Giants"
Isaac Newton

RESUMO

Os problemas de corte de materiais são recorrentes no cotidiano da indústria, sendo encontrados nas mais diferentes formas. O problema de corte bidimensional guilhotinado é uma dessas formas. Ele surge pelas restrições da ferramenta de corte, tipicamente a guilhotina. Este trabalho apresenta três abordagens para solucionar o problema em questão: uma abordagem matemática, uma abordagem exata computacional e uma abordagem heurística. A abordagem matemática consiste em um modelo de programação linear baseado em listas de itens e montagem do arranjo de corte partindo dos itens, unindo-os dois a dois, tentando maximizar o número de uniões sem ultrapassar as dimensões da placa. A abordagem exata computacional tratá-se de um algoritmo *Branch-and-Bound* modificado para permitir que estados mais promissores possam ser analisados antes, comportando-se como um algoritmo de busca em profundidade com uma pequena etapa em largura, na qual ordena os filhos na árvore de decisão pelo desperdício gerado. Por fim, a abordagem heurística é composta das metaheurísticas GRASP, Busca Tabu, Algoritmo Genético, BRKGA e Religação de Caminhos combinados com uma heurística de montagem baseada nos algoritmos propostos por Nascimento, Longo e Aloise (1999). Essas metaheurísticas foram combinadas em um time assíncrono para alcançar melhores resultados que os já encontrados na literatura. Além de melhorar os resultados conhecidos, a pesquisa também tinha como objetivo apresentar um modelo viável, em número de variáveis, e resultados ótimos para instâncias comumente utilizadas para o problema supracitado e novas opções de obtê-los em instâncias que venham a surgir no futuro. Testes mostraram a competividade dos algoritmos propostos frente aos melhores resultados encontrados, reduzindo inclusive o número total de placas, bem como a capacidade dos métodos exatos propostos de encontrar as soluções ótimas para as instâncias testadas. Cerca de de 25% dos resultados ótimos foram encontrados, passando esse número para 75%, quando considerados os resultados dos algoritmos metaheurísticos que atingiram o limite inferior das instâncias.

Palavras-chave: Corte bidimensional guilhotinado, Programação linear, Branch and Bound, Metaheurística, Time assíncrono (A-Team).

ABSTRACT

The problems of cutting materials are recurrent daily in the industry, being found in the most different forms. The problem of guillotined two-dimensional cutting is one of these forms. It arises from the constraints of a cutting tool, typically a guillotine. The present work prepares three approaches to solve this issue: a mathematical approach, an exact computational approach and a heuristic approach. The mathematical approach consists of a linear programming model based on lists of items and on an assembly of the cut arrangement starting from the items, joining them two by two, trying to maximize the number of joints without exceeding the dimensions of plate. The exact computational approach is a modified Branch-and-Bound algorithm to allow for more promising states to be analyzed first, behaving as a breadth-first search algorithm in which orders the children in the decision tree for the waste generated. Finally, the heuristic approach is composed of the metaheuristics GRASP, Tabu Search, Genetic Algorithm, BRKGA and Path Restoration combined with an assembly heuristic based on the algorithms proposed by Nascimento, Longo e Aloise (1999). These metaheuristics were combined in an asynchronous team to achieve better results than those already found in the literature. In addition, to improving the known results, the research also aimed at presenting a viable model, in number of variables, and optimal results for commonly used instances for the above problem and new options to obtain them in instances that arise in the future. Tests showed the competitiveness of the proposed algorithms against the best results found, reducing even the total number of plates, as well as the capacity of the exact methods proposed to find the optimal solutions for the tested instances. About 25 % of the optimal results were found , passing that number to 75 %, when considering the results of the metaheuristic algorithms that reached the lower limit of the instances.

Key-words: Two-dimensional guillotined cutting, Linear Programming, Branch and Bound, Metaheuristics, Asynchronous team (A-Team)

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Existem muitas demandas na indústria que remetem a tornar mais eficiente o consumo de recursos para manufatura de bens. Podemos citar, por exemplo, o corte de peças de madeira para confecção de utensílios como cadeiras ou mesas. Elas tornam-se relevantes à medida em que as sobras não podem ser reaproveitadas ou que requererem um custo maior para tal. Assim, o mal aproveitamento dos insumos acarretaria em um custo maior da produção.

A questão de como melhor utilizar a matéria-prima pode ser enquadrada como um problema de otimização. Esses problemas visam escolher a "melhor" configuração ou conjunto de parâmetros para atingir algum objetivo (PAPADIMITRIOU; STEIGLITZ, 1998). Para solucioná-los existem inúmeros estudos de diferentes áreas focados em investigar e aplicar diversas técnicas e métodos de resolução como: programação linear, programação dinâmica, *Branch-and-Bound*, heurísticas e metaheurística (GOMES, 2011).

Essa variedade de métodos se deve a um bom motivo: tempo de resposta e qualidade da solução gerada. Os métodos exatos (programação linear, programação dinâmica, *branch and bound* por exemplo) possuem altos tempos de respostas, dependendo da aplicação. Quando tratamos com instâncias pequenas, esses métodos são viáveis. Porém, à medida que seu tamanho cresce, seu tempo de resolução aumenta exponencialmente. O que torna inviável a utilização desses métodos em exemplares de grande porte. Nesses casos, o indicado é a escolha de um método heurístico (ou metaheurístico), que não garante a otimalidade da solução, mas retorna uma solução próxima do ótimo, em um tempo bem menor.

Um grupo desses problemas de otimização encontrados com frequência na indústria são os problemas de corte e empacotamento (*bin-packing problem*). Neles, um dado conjunto L de itens retnagulares devem ser empacotados em identicos contêineres retangulares. Chung, Garey e Johnson (1982) mostraram que apesar da simplicidade, esses problemas se caracterizam como NP-difícil.

Os problemas de corte possuem um particular interesse, visto que um grande número de processos de confecções de bens requerem o corte de um conjunto de pequenos objetos (ou itens, ou peças) a partir de uma limitada quantidade de de tipos de objetos grandes (ou placas, ou contêineres, ou *bins*). Variações desses problemas podem surgir quando restrições são adicionadas. Normalmente essas restrições refletem alguma condição no arranjo dos itens ou limitação tecnológicas (POLYAKOVSKY; M'HALLAH, 2009).

Uma dessas variações causadas pelas limitações tecnológicas é o problema do corte bidimensional guilhotinado. Tal variação deve-se as caracteristicas da máquina

que realizará os cortes nas chapas, a guilhotina. O presente trabalho irá abordar essa variante do problema devido ao grande número de aplicações industriais no qual ele pode ser encontrado. Outro fator motivador para o foco deste trabalho é a importância do tema, tanto econômica, quanto ecológica, visto que a redução do desperdício de matéria-prima pode acarretar aumento de produção e redução do consumo de energia, de água e de outros insumos necessários ao reaproveitamento das sobras.

1.1 CONTEXTUALIZAÇÃO

1.1.1 Problemas de Corte e Empacotamento

Os problemas de corte e de empacotamento podem ser tratados de forma semelhantes devido a inúmeras características em comum que ambos possuem. Neles, há duas listas chamadas de *estoque* e *demanda*. A *lista de estoque* consiste em um infinito número de grandes objetos de dimensões padrão, enquanto a *lista de demanda* consiste em um finito número de itens com dimensões menores. A demanda deve ser totalmente atendida, ou seja, todos os itens devem ser cortados ou empacotados em objetos vindos do estoque. O objetivo é reduzir o número de objetos utilizados (DYCKHOFF, 1990).

Dyckhoff (1990) fez um apanhado dos trabalhos publicados até então e criou uma classificação dos problemas de corte e empacotamento a partir dos critérios de dimensionalidade, avaliação da qualidade, forma das figuras, diversidade, disponibilidade, restrições de padrões, restrições de atribuição, objetivos e estado da informação e variabilidade.

A dimensionalidade discorre a cerca da quantidade de dimensões relevantes para o problema, se unidimensional, bidimensional, tridimensional e multidimensional (quatro ou mais dimensões relevantes). Avaliação da qualidade dita a forma como as soluções são avaliadas, se discreta (inteira) ou contínua (fracionada ou real). A forma das figuras refere-se ao formato, tamanho, orientação e regularidade das formas dos objetos e itens do problema.

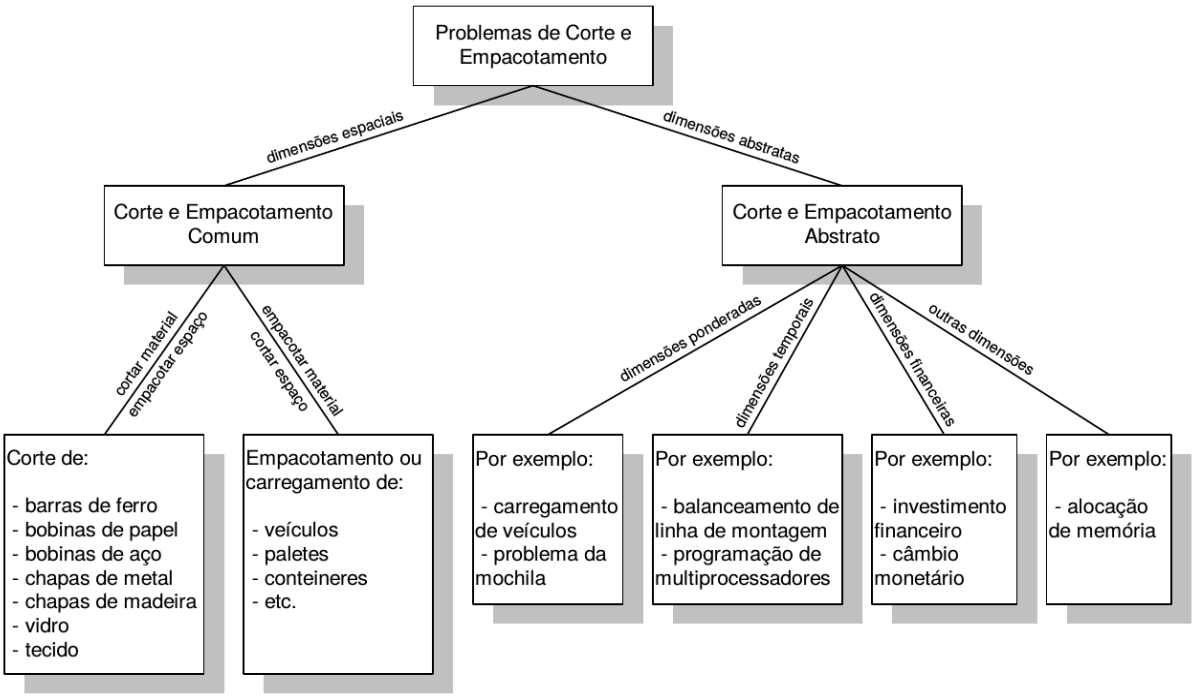
A diversidade diz respeito à quantidade de formas distintas em que as figuras dos itens e objetos podem ser encontrados. A disponibilidade discorre a cerca da quantidade de itens e objetos que podem ser utilizados para solucionar o problema. Restrições de padrões refere-se às limitações impostas aos arranjos dos itens dentro dos objetos maiores (por exemplo, se o corte é livre ou guilhotinado). Restrição de atribuição lida com o tipo de atribuição (se um item pode ou não estar em determinado contêiner), a dinâmica das alocações (como em carregamento de veículos, onde itens são removidos

e acrescentados no trajeto), o número de estágios e o número de cortes, a frequência ou a sequencia de padrões (como no corte de tecido para confecção de roupas, no qual os itens devem ser cortados em enfestos compostos por um certo número de camadas de tecido, todas com o mesmo padrão).

A classificação por objetivo trata de qual é o propósito do problema, se reduzir a quantidade de itens utilizados, a geometria e a utilização dos padrões, a sequencia, a combinação ou o número de padrões. O estado da informação e variabilidade dita se os dados do problema são determinísticos, estocásticos ou incertos, como no caso de alocação de contêineres em navios, onde não se sabe quantos e quais tipos de contêineres serão necessários para carregar a carga do próximo trecho do trajeto do navio.

Os critérios e a tipologia de Dyckhoff (1990) geram a estrutura de problemas de corte e empacotamento apresentada na Figura 1.

Figura 1 – Tipologia proposta por Dyckhoff.



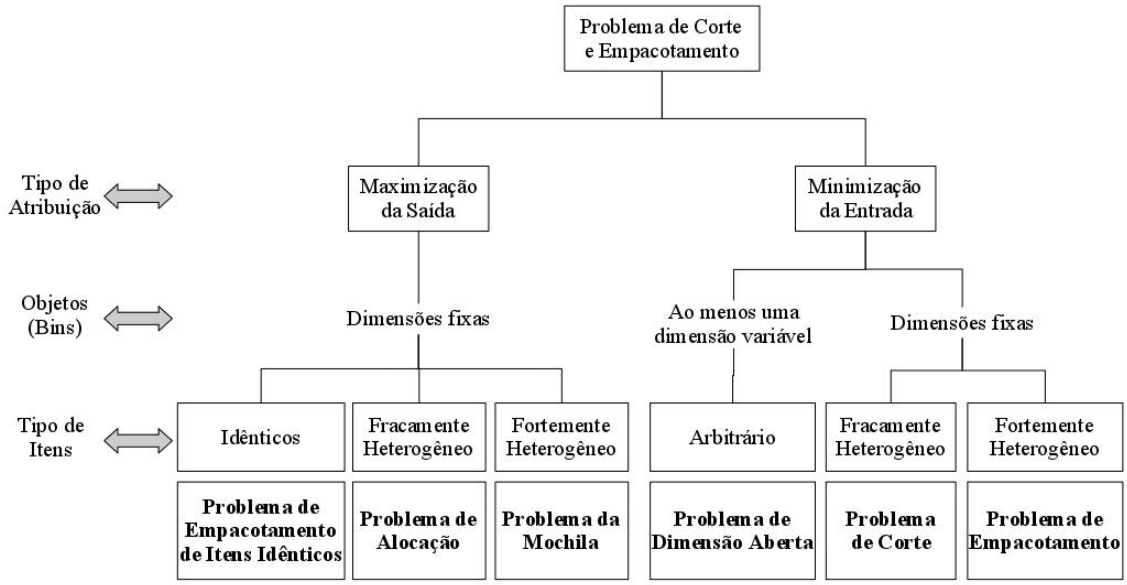
Fonte: (DYCKHOFF, 1990)

O problema de corte bidimensional guilhotinado é classificado, nesta imagem, como corte de chapas de metal, visto suas características.

Com o aumento da tecnologia e o desenvolvimento de ferramentas, algumas dessas definições passaram a fazer pouco sentido e Wäscher, HauBner e Schumann (2007) atualizaram a tipologia dos problemas de corte e empacotamento proposta por Dyckhoff (1990). Essa reestruturação é mostrada na Figura 2.

Olhando com atenção a tipologia proposta por Wäscher, HauBner e Schumann (2007), é possível ver que os problemas de corte e empacotamento possuem diversas variações e essas variações possuem suas próprias origens e características. O problema

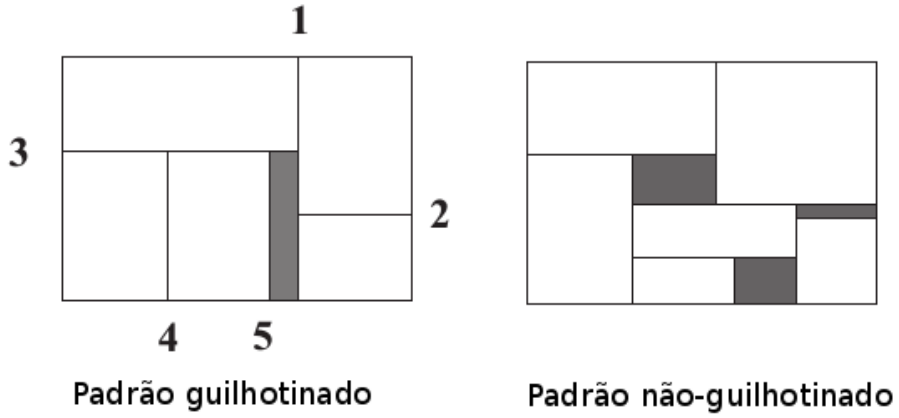
Figura 2 – Tipologia proposta por Wäscher, Haubner e Schumann.



Fonte: (WÄSCHER; HAUBNER; SCHUMANN, 2007)

de corte bidimensional guilhotinado, como o próprio nome diz, trabalha com duas dimensões relevantes, semelhante ao problema de corte e empacotamento bidimensional descrito acima. Sua principal diferença está na limitação do *arranjo de corte dos itens* dentro do objeto ou placa (como passaremos a chamar para o caso bidimensional do problema de corte e empacotamento).

Figura 3 – Padrão de corte guilhotinado ao lado de um padrão não-guilhotinado.



Fonte: (TEMPONI, 2007)

Um padrão guilhotinado ou guilhotinável permite que os itens sejam separados por cortes que se estendem de uma borda a outra da placa ou de um pedaço anteriormente cortado (que passaremos a chamar de retângulos) seguindo uma ordem pré-definida (como pode ser visto na Figura 3). Em contrapartida um padrão não-guilhotinado ou não-guilhotinável não permite a separação dos itens sem que a ferramenta de corte realize curvas ou seja interrompida no decorrer do percurso em qualquer que seja a ordem das operações.

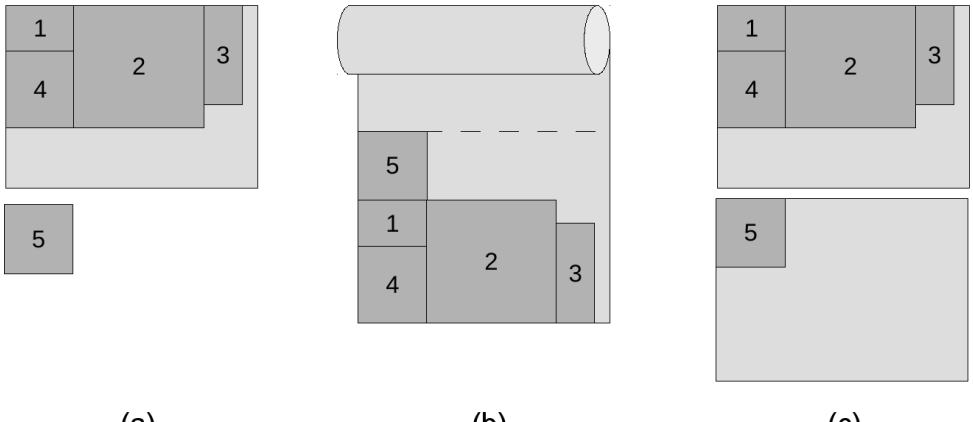
1.1.2 Classificação do problema de Corte Bidimensional Guilhotinado

O problema de corte bidimensional guilhotinado possui algumas variações. Na realidade, a restrição do corte ser guilhotinado pode ser aplicado a diversos problemas e isso colabora para que se crie uma certa ambiguidade em relação aos casos encontrados na literatura.

Furini, Malaguti e Thomopulos (2016) apresentaram três desses problemas que podem receber as restrições de corte guilhotinado e cujas técnicas para resolução apresentam suas próprias características, de modo que não se pode comparar seus resultados diretamente. Eles são o problema da mochila bidimensional (*Two-Dimensional Knapsack Problem* ou **2KP**), o problema de corte com uma dimensão aberta (*Strip Packing Problem* ou **SPP**) e o problema de corte e estoque bidimensional (*Two-Dimensional Bin Packing Problem* ou **2BP**).

Esses três problemas possuem suas versões com restrição de corte guilhotinado de forma que muitos trabalhos que apresentam suas propostas de soluções e resultados para o problema de corte bidimensional guilhotinado podem estar tratando de problemas diferentes.

Figura 4 – Três exemplos de problemas onde cortes guilhotinados podem ser aplicados



Fonte: Baseado em (FURINI; MALAGUTI; THOMOPULOS, 2016)

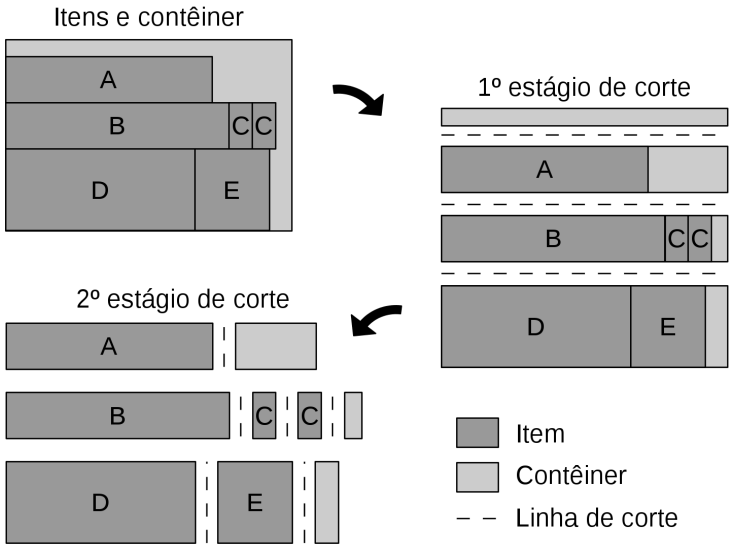
Dentre as diferenças entre esses problemas a mais notável é com relação ao objetivo principal. No problema da mochila, o objetivo é alocar, em um número finito de objetos (normalmente um), os itens de maior valor ou que gerem o maior lucro (Figura 4 (a)); no problema de corte com dimensão aberta, o objetivo é alocar os itens em um objeto que possui uma dimensão infinita (ou aberta) de modo que o arranjo utilize o mínimo possível dessa dimensão (Figura 4 (b)); e no problema de corte e estoque, o objetivo é alocar, em um número infinito de objetos, todos os itens de modo a utilizar o menor número desses objetos (Figura 4 (c)).

Mesmo para o problema baseado no corte e estoque bidimensional há variações. Lodi, Martello e Vigo (1999a) classificam esses problemas como:

- **2BP|O|G:** os itens são orientados (O), ou seja, não podem ser rotacionados e o corte guilhotinado é requerido (G);
- **2BP|R|G:** os itens podem ser rotacionados em 90º (R) e o corte guilhotinado é requerido (G);
- **2BP|O|F:** os itens são orientados (O) e o corte é livre (F);
- **2BP|R|F:** os itens podem ser rotacionados em 90º (R) e o corte é livre (F);

Além dessas variações, existe uma limitação quanto ao número de estágios de corte. Na literatura é possível encontrar, basicamente, dois tipos, os limitados a dois ou três estágios (Figura 5) e os não-estagiados, que não apresentam restrição de número de estágios.

Figura 5 – Estágios de corte.



Fonte: (HOFFMANN et al., 2015)

Este trabalho está focado em solucionar o caso guilhotinado do problema de corte e estoque bidimensional no qual rotações de 90º são permitidas (**2BP|R|G**).

Nessa modalidade, o problema pode ser definido como: dado um conjunto de n itens retangulares que devem ser cortados a partir de um conjunto de um grande conjunto de placas padronizadas em estoque. As placas possuem altura H e largura W. Cada peça $j \in J = (1, ..., n)$ é caracterizada por sua altura $h_j \leq H$ e largura $w_j \leq W$. O problema tenta determinar o padrão de corte que entregue todas os n itens de modo que o número total de placas requeridas do estoque seja minimizado (MARTELLO; VIGO, 1998). Sobre esse problema é aplicado a restrição do corte guilhotinado, ou seja, que o corte vá sempre de um lado ao outro da placa ou de um pedaço já cortado dela (que passaremos a chamar simplesmente de retângulo) sem realizar curvas.

1.2 OBJETIVOS DA PESQUISA

O objetivo principal da pesquisa é criar algoritmos capazes de alcançar bons resultados para o problema de corte bidimensional guilhotinado não-estagiado encontrados na literatura, utilizando Programação Linear, *Branch-and-Bound*, Time assíncrono, GRASP, Busca Tabu, Algoritmos Genéticos, BRKGA, Religação de Caminhos e uma heurística baseada no algoritmo proposto por Nascimento, Longo e Aloise (1999), que chamaremos aqui de HHDHeuristic.

Os objetivos específicos que nortarão a pesquisa são:

- Propor um modelo matemático de programação linear baseado nas caracteristicas relevantes do problema;
- Propor um algoritmo exato baseado no *Branch-and-Bound* para encontrar soluções ótimas para o problema;
- Apresentar uma abordagem que combine algoritmos metaheurísticos GRASP, Busca Tabu, Algoritmos Genéticos, BRKGA e Religação de Caminhos uma a heurística HHDHeuristic para encontrar soluções em tempo hábil;

1.3 ORGANIZAÇÃO DO TRABALHO

Os demais capítulos deste trabalho estão organizados da seguinte forma: o capítulo 2 faz um apanhado dos trabalhos publicados na área e mostra o estado da arte da pesquisa sobre o corte bidimensional guilhotinado; o capítulo 3 propõe um

novo modelo matemático; o capítulo 4 descreve o algoritmo exato baseado no *Branch-and-Bound* e no modelo matemático proposto neste trabalho; o capítulo 5 faz uso de heurísticas e metaheurísticas para tentar alcançar um novo patamar de qualidade de soluções para o problema; o capítulo 6 apresenta a metodologia dos testes (ambiente de testes, instâncias, limites inferiores) e os resultados para cada uma das abordagens e o capítulo 7 traz algumas considerações finais e propostas para trabalhos futuros que por ventura possam vir a enriquecer essa importante área de pesquisa.

2 CORTE BIDIMENSIONAL GUILHOTINADO

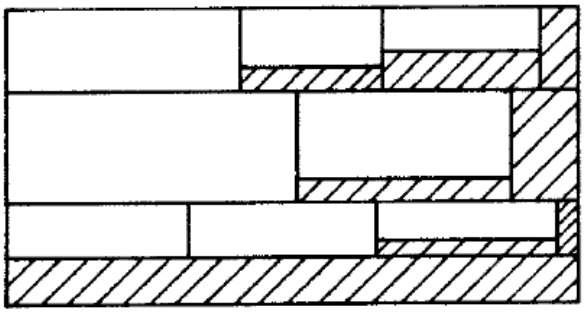
2.1 HISTÓRICO DO PROBLEMA DE CORTE BIDIMENSIONAL GUILHOTINADO

Os problemas de corte e empacotamento existem a muito tempo, nascendo, provavelmente, junto com a manufatura de bens. Entretanto, diversos autores (como Velasco (2005), Cherri et al. (2015), Lodi (1999), Temponi (2007) e Poldi (2003)) concordam que o estudo formal desses problemas datam da década de 1960 com Gilmore e Gomory.

Gilmore e Gomory (1961) apresentaram um método pioneiro para resolução do problema de corte unidimensional. Em seu artigo, os autores utilizaram um subproblema baseado no problema da mochila para gerar colunas viáveis para o método simplex e assim conseguir resolver um problema que era inviável. O algoritmo foi atualizado em 1963 pelos próprios autores.

Gilmore e Gomory (1965) apresentaram um método linear para resolução do problema de corte bidimensional com restrição guilhotinada para dois e três estágios. Esse método também se utiliza do problema da mochila como subproblema para facilitar os cortes. Gilmore e Gomory trataram o problema em duas etapas. Inicialmente cortando as placas em faixas e depois cortando os itens da lista de demanda a partir dessas faixas (como mostra a Figura 6).

Figura 6 – Padrão de corte guilhotinado organizado em faixas.



Fonte: (GILMORE; GOMORY, 1965)

Gilmore e Gomory (1966) apresentaram um algoritmo através do qual é possível gerar padrões de cortes não-estagiados utilizando a abordagem do problema da mochila e uma estrutura em forma de árvore. Em tal estudo, os autores dividiram, em cada estágio de corte, um retângulo em dois retângulos menores. Desse modo conseguiram gerar padrões de cortes não-estagiados por meio da repetição das operações de corte.

Herz (1972) propôs, uma melhoria no algoritmo iterativo de Gilmore e Gomory (1966), utilizando recursividade para determinar o melhor padrão de corte possível

na resolução de problemas bidimensionais irrestritos. Assim ele conseguiu reduzir a quantidade de memória utilizada no procedimento e alcançou os mesmos resultados de Gilmore e Gomory.

Christofides e Whitlock (1977) propuseram um algoritmo heurístico para solucionar o problema de corte bidimensional guilhotinado utilizando busca em árvore, bem como um algoritmo de enumeração que lista todos os possíveis padrões de corte do problema. Esse algoritmo de enumeração baseia-se no algoritmo de *backtrak*, ou seja, utiliza recursividade na escolha dos cortes .

Hinxman (1980) fez uma revisão dos problemas de corte, abordando algumas de suas principais variações. Também apontou uma estreita conexão entre o problema da mochila e os problemas de corte utilizando, inclusive, os métodos de Gilmore e Gomory como exemplo dessa conexão.

Wang (1983) apresentou dois algoritmos que geram padrões guilhotinados através de sucessivas uniões entre retângulos, como mostra a Figura 7, aplicando parâmetros para rejeitar adições indesejadas com a finalidade de reduzir o número de possibilidades. Nos algoritmos, quando dois retângulos R são unidos, um retângulo maior é formado. Caso a união não seja perfeita, uma sobra W (retângulo que não comporta nenhum dos itens de demanda) é acrescida.

Berkey e Wang (1987) apresentaram alguns algoritmos para solucionar o problema de corte e empacotamento, utilizando premissas simples, como o *first-fit*, *next-fit* e algoritmos híbridos.

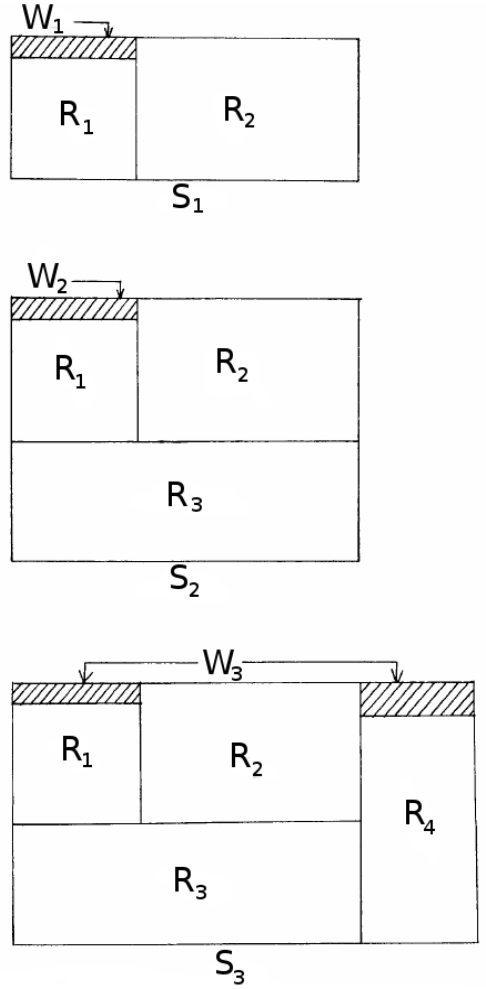
Dyckhoff (1990) fez uma nova revisão da área e propôs uma nova taxonomia dos problemas de corte e empacotamento baseado em dimensionalidade, avaliação da qualidade, forma das figuras, diversidade, disponibilidade, restrições de padrões, restrições de atribuição, objetivos e estado da informação e variabilidade.

Oliveira e Ferreira (1990) introduziram modificações no algoritmo desenvolvido por Wang (1983), melhorando seu desempenho. O Algoritmo de Wang Modificado é o resultado de uma alteração no nível de aspiração, no critério de rejeição de soluções indesejáveis. Dessa forma, o algoritmo avalia a perda associada a cada solução parcial construída (Soma dos W da Figura 7). Caso este valor seja maior que o limite estabelecido, a solução parcial é rejeitada.

Sinuany-Stern e Weiner (1994) consideraram o problema de corte e estoque unidimensional com dois objetivos: minimizar a sobra gerada e acumular a máxima quantidade de sobras no último objeto a ser cortado..

Martello e Vigo (1998) apresentaram um algoritmo exato baseado em árvores e alguns algoritmos heurísticos. Os autores criaram ainda algumas classes de instâncias desse problema, sendo seis delas baseadas no procedimento descrito no artigo de Berkey e Wang (1987) e seis novas classes. Essas classes possuem cinquenta problemas cada, divididas em cinco grupos pelo número total de itens (vinte, quarenta, sessenta, oitenta

Figura 7 – Padrões formados em cada iteração do algoritmo de Wang.



Fonte: (WANG, 1983)

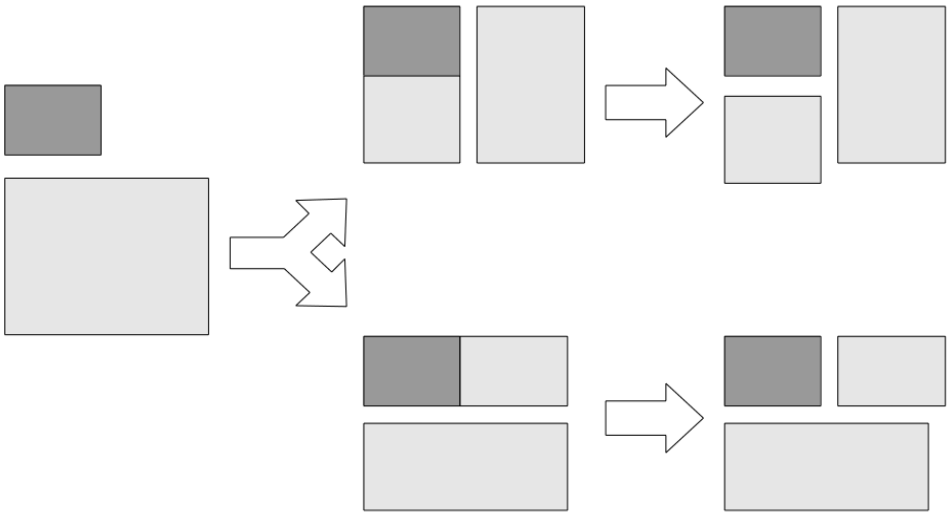
e cem itens).

Nascimento, Longo e Aloise (1999) propuseram uma heurística *O(mn)* para o corte bidimensional guilhotinado, onde *m* é a quantidade de itens diferentes a serem cortados e *n* é a quantidade total de itens. Essa heurística apresenta uma abordagem recursiva na qual uma lista ordenada de itens é transformada em um arranjo de corte, alocando o maior item na menor sobra que lhe comporte. Essa alocação pode gerar duas novas sobras que serão utilizadas na próxima iteração como entrada para o próximo item da lista como pode ser visto na Figura 8.

Lodi (1999) apresentou algumas heurísticas e metaheurísticas para o problema de corte e empacotamento bidimensional. Também faz uma compilação das instâncias de Berkey e Wang (1987) e Martello e Vigo (1998) e gerou um conjunto de dez classe, cada classe com cinco grupos de dez instâncias cada.

Dell'Amico, Martello e Vigo (1999) apresentaram um método pseudo-polinomial para encontrar os limites inferiores para os problemas de corte bidimensional guilhotinado, para quando rotações não são permitidas. Eles também apresentaram algumas

Figura 8 – Padrão de corte guilhotinado.



Fonte: Baseado em (NASCIMENTO; LONGO; ALOISE, 1999)

heurísticas e metaheurísticas para resolver o problema.

Lodi, Martello e Vigo (1999a) apresentam mais algumas heurísticas e metaheurística baseada em busca tabu para o problema, conseguindo bons resultados frente aos limites inferiores encontrados por Dell'Amico, Martello e Vigo (1999).

Dell'Amico, Martello e Vigo (2002) adaptam seu método para encontrar os limites inferiores para o problema de corte bidimensional guilhotinado para a variação que permite rotações.

Velasco (2005) propôs um algoritmo GRASP utilizando listas ordenadas baseadas nas alturas e nas larguras dos itens. Essas listas são utilizadas para formar faixas horizontais e verticais. Dependendo do aproveitamento, uma delas é selecionada e o algoritmo recomeça a partir da sobra. O algoritmo trata o problema com restrição ao número de estágios de corte, de modo que são permitidos apenas três estágios de corte.

Wäscher, HauBner e Schumann (2007) propuseram uma atualização na taxonomia de (DYCKHOFF, 1990), acrescentando os trabalhos mais recentes e adicionando mais detalhes.

Clautiaux, Jouglet e El Hayek (2007) apresentaram um novo método para encontrar os limites inferiores para o problema de corte bidimensional guilhotinado no qual rotações são permitidas (o mesmo que Dell'Amico, Martello e Vigo (2002)).

Messaoud, Chu e Espinouse (2008) propuseram uma formulação para o problema de corte guilhotinado em sua variação com uma dimensão aberta.

Polyakovsky e M'Hallah (2009) apresentaram um método heurístico que acomoda os itens dentro da placa usando como parâmetros as suas áreas. Também apresentaram um algoritmo baseado em agentes para aprimorar os resultados de sua heurística.

Park et al. (2013) desenvolveram uma heurística voltada ao processo de otimização em corte de vidro, destacando o problema de corte bidimensional de dois estágio, com

restrições de rotação de 90º e tipo de corte guilhotinado, utilizando um método de empacotamento baseado em nível (prateleira). Os autores tiveram como objetivo minimizar o custo de produção na indústria de vidro.

Fleszar (2013) abordou o problema de corte e estoque bidimensional com restrição de corte guilhotinado, com ou sem rotação, com aplicações em indústrias de corte de vidro e madeira. O autor propôs três novas heurísticas construtivas. A representação dos padrões de corte (soluções) é na forma de árvores, onde cada árvore representa um padrão de corte em uma placa. Nessa representação todos os nós folhas da árvore representam itens (as sobras são negligenciadas). Todos os outros nós são de dois tipo, *H* representando cortes horizontais ou *V* representando cortes verticais.

Mariano (2014) apresentou uma abordagem baseada em time assíncrono, utilizando o GRASP proposto por Velasco (2005) e o HHDHeuristic proposto por Nascimento, Longo e Aloise (1999). A abordagem conseguiu bons resultados, em face, principalmente ao que as heurísticas GRASP e HHDHeuristic conseguiram isoladamente. Essa abordagem contou ainda com uma heurística de melhoria baseada na desconstrução de subárvores com um percentual alto de sobras e sua reconstrução, tentando retirar elementos da placa com menos itens. Isso foi feito com a intenção de reaproveitar os espaços intermediários deixados nas placas e esvaziar a última placa, reduzindo assim o número de placas necessárias para cortar os itens.

Russo, Sforza e Sterle (2014) propuseram um método baseado em programação dinâmica para solucionar de forma exata o problema de corte guilhotinado irrestrito. Nessa modalidade, há apenas uma grande placa retangular no estoque e deseja-se obter uma certa quantidade de retângulos menores, cada um com suas dimensões e lucro diferente.

Furini, Malaguti e Thomopulos (2016) apresentaram uma formulação linear para o problema de corte bidimensional guilhotinado. Os autores abordaram a mesma variação do problema atacado por Russo, Sforza e Sterle (2014), além de formulações para o *Cutting Stock Problem* e o *Strip Paking Problem*.

2.2 MODELOS MATEMÁTICOS

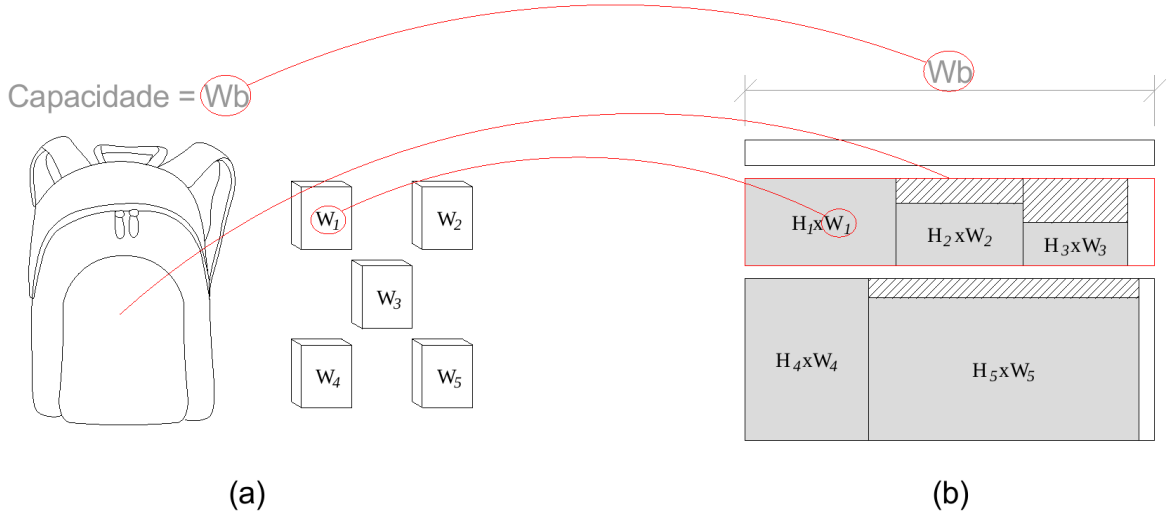
2.2.1 Formulação matemática para o caso restrito a dois ou três estágios de corte

Gilmore e Gomory (1965) apresentaram um modelo matemático com escopo restrito a dois ou três estágios de corte. Esse modelo se utiliza de faixas, nas quais um item é marcado como item de abertura da faixa e todos os demais itens nessa faixa

devem ter altura igual ou inferior a este item. Várias faixas são montadas desta forma e unidas em um contêiner maior.

Essa abordagem é baseada no problema da mochila, onde cada uma das faixas (Figura 9 (b)) se comporta como uma mochila cuja capacidade é a largura do contêiner (Wb) e as larguras dos itens (W_i) passam a representar os pesos dos itens no problema da mochila equivalente (Figura 9 (a)).

Figura 9 – Equivalência entre uma faixa no modelo de Gilmore e Gomory (1965) e o problema da mochila



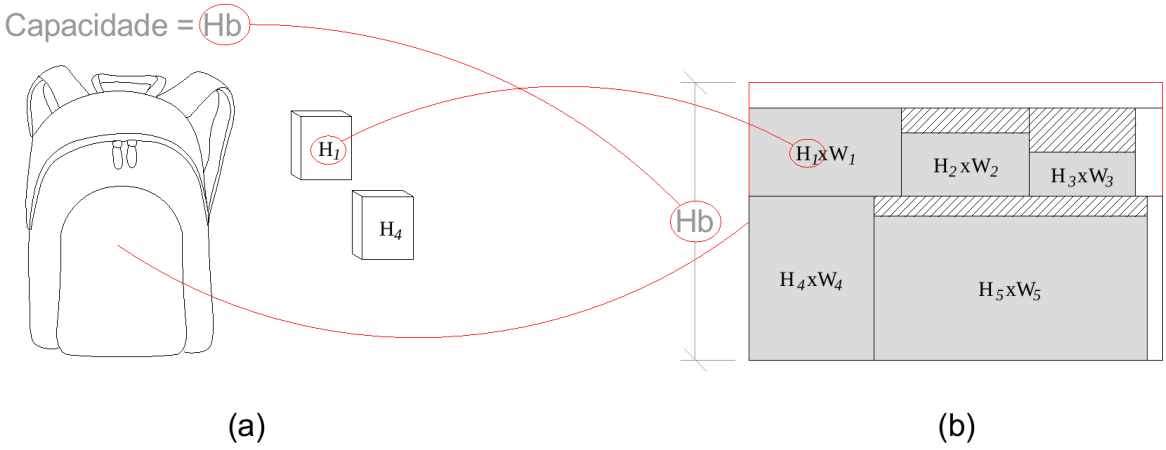
Fonte: Baseado em (GILMORE; GOMORY, 1965)

As várias faixas formadas são agrupadas umas sobre as outras para compor o arranjo de corte (Figura 10 (b)). Como cada faixa é da altura do primeiro item, a soma das alturas de todos os primeiros itens das faixas deve ser menor ou igual à altura do contêiner, ou seja, a capacidade da mochila geral (Figura 10 (a)) é a altura da placa (Hb) e os pesos de cada faixas é a altura de seu primeiro item (H_1 e H_4 na Figura 10). Desse modo, o trabalho dos autores trata o problema de corte bidimensional guilhotinado como um problema da mochila, no qual temos várias mochilas dentro de uma mochila maior ou uma mochila com vários compartimentos.

É importante lembrar que Gilmore e Gomory trataram o problema como um problema da mochila propriamente dito (**2KP**). Nele, cada item tem um valor e o objetivo é colocar os itens cuja soma resulte no maior valor dentro da mochila, ou, usando os termos dos problemas de cortes, recortar o conjunto de itens de valor máximo a partir do contêiner. E essa não é a variação do problema de corte que abordamos neste trabalho. Todavia, é fácil modificar tal formulação para atender à variação aqui abordada.

Uma forma bem simples de utilizar a formulação de Gilmore e Gomory para o problema de corte bidimensional guilhotinado em sua variação do problema de corte e estoque (que passaremos a chamar apenas de corte bidimensional guilhotinado) é

Figura 10 – Equivalência entre uma placa no modelo de Gilmore e Gomory (1965) e o problema da mochila



Fonte: Baseado em (GILMORE; GOMORY, 1965)

retirar o limite da quantidade de mochilas possíveis e tentar reduzir seu valor. Um trabalho que se utiliza disso é a pesquisa de Lodi, Martello e Vigo (2004).

Retirar o limite da quantidade de mochilas da formulação de Gilmore e Gomory possui vantagens e desvantagens. Ela permite que o problema seja trabalhado, de forma linear inteira ou linear mista, com poucas variáveis e restrições em relação à abordagem em árvore ou livre, por exemplo. Todavia, ela não garante que a solução encontrado seja a ótima global, se considerar apenas a restrição de cortes guilhotináveis. Isso acontece porque o modelo trabalha apenas até o segundo estágio de corte, de modo que se o ótimo global depender de um terceiro estágio de corte, a formulação não irá encontrar tal arranjo.

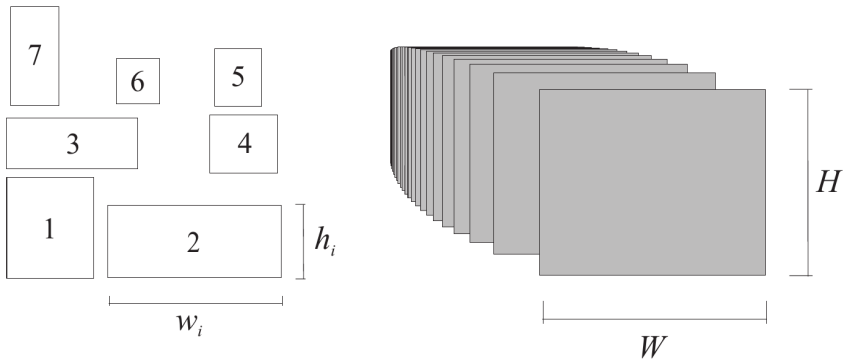
A formulação matemática proposta por Lodi, Martello e Vigo (2004) para o caso restrito a dois ou três estágios de corte se baseia no problema da mochila, onde cada estágio de corte é representado por um conjunto de mochilas. Nele as mochilas que representam o segundo estágio de corte são utilizadas como itens para as mochilas do primeiro estágio de corte.

As informações provenientes da instância são representadas pelas constantes W, H, w_i e h_i, onde W e H representam a largura e a altura da placa ou contêiner, e w_i e h_i representam a largura e a altura de cada item i (conforme visto na Figura 11).

As variáveis de decisão utilizadas são as variáveis binárias $x_{i,j}$, y_i, $z_{k,i}$ e q_k. A variável $x_{i,j}$ recebe 1 se o item i está alocado na faixa j ou 0, caso contrário. A variável y_i assume valor 1 se o item i inicializa a faixa de mesmo índice ou 0, se não. A variável $z_{k,i}$ indica se a faixa i está alocada no contêiner k caso receba valor 1 ou 0, caso não. E a variável q_k recebe 1 para indicar que a faixa k inicializa o contêiner de mesmo índice ou 0, para indicar o contrário.

Desse modo, os autores descrevem o problema de corte bidimensional com

Figura 11 – Instância do problema de corte e empacotamento bidimensional



Fonte: (TEMPONI, 2007)

restrição de dois ou três estágios matematicamente da seguinte forma:

$$min \sum_{k=1}^{n} q_k \tag{2.1}$$

sujeita à:

$$\sum_{i=1}^{j-1} x_{i,j} + y_j = 1, \forall j = 1, ..., n \tag{2.2}$$

$$\sum_{j=i+1}^{n} w_j x_{i,j} \leq (W - w_i) y_i, \forall i = 1, ..., n-1 \tag{2.3}$$

$$\sum_{k=1}^{i-1} z_{k,i} + q_i = y_i, \forall i = 1, ..., n \tag{2.4}$$

$$\sum_{i=k+1}^{n} h_i z_{k,i} \leq (H - h_k) q_k, \forall k = 1, ..., n-1 \tag{2.5}$$

$$y_i \in \{0, 1\}, \forall i = 1, ..., n \tag{2.6}$$

$$x_{i,j} \in \{0, 1\}, \forall i = 1, ..., n-1; j > i \tag{2.7}$$

$$q_k \in \{0, 1\}, \forall k = 1, ..., n \tag{2.8}$$

$$y_{k,i} \in \{0, 1\}, \forall k = 1, ..., n-1; i > k \tag{2.9}$$

Nesse modelo, a função objetivo (2.1) tem por finalidade reduzir o número de contêineres utilizados. Como a variável q_k indica se a faixa k é a primeira do contêiner de mesmo índice, indiretamente ela indica se o contêiner k foi utilizado ou não.

A restrição (2.2) força com que cada item apareça exatamente uma vez, seja inicializando uma faixa ($y_j = 1$) ou em uma faixa inicializada por um item anterior ($\sum_{j=i+1}^{n} x_{i,j} = 1$), mas nunca os dois ao mesmo tempo.

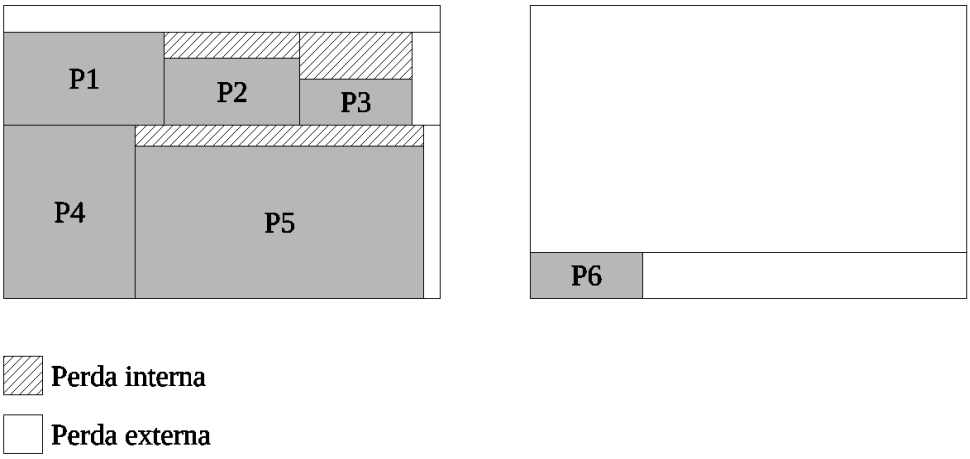
A equação (2.3) restrige a largura de cada faixa para que ela seja menor que a largura da placa menos a largura do item de abertura da faixa. O somatório de $w_j x_{i,j}$ irá acumular a largura de todos os itens presentes na faixa i. Ser menor ou igual a $(W - w_i)y_i$ garante tanto que a soma das larguras dos itens sejam menores que a largura da placa, caso a faixa tenha sido aberta ($y_j = 1$), quanto que nenhum item esteja em uma faixa que não foi aberta ($y_j = 0$). Isso porque, se não há item de abertura da faixa, o somatório deve ser menor ou igual a zero e para tanto, nenhuma variável $x_{i,j}$ pode assumir valor 1 para a faixa j.

Equação (2.4) impõe que cada faixa seja alocada até uma vez. Isso porque pode ser que a faixa não tenha sido aberta ($y_i = 0$) e não deve aparecer. Mas se a faixa tiver sido aberta ($y_i = 1$), ela deve aparecer como primeira faixa do contêiner ($q_k = 1$) ou como faixa subsequênte de outro contêiner ($\sum_{k=1}^{i-1} z_{k,i} = 1$).

A equação (2.5) delimita a altura de cada contêiner k utilizado. O somatório de $h_i z_{k,i}$ guarda a soma das alturas das faixas de cada placa k. A imposição de ser menor ou igual a $(H - h_k)q_k$ força com que o somatório não ultrapasse a altura da placa, se ela tiver sido aberta ($q_k = 1$) ou que não exista faixa em um contêiner que não foi utilizado ($q_k = 0$).

Por fim as restrições (2.6), (2.7), (2.8) e (2.9) definem que valores as variáveis podem assumir.

Figura 12 – Perdas internas provenientes dos cortes.



Fonte: Baseado em (TEMPONI, 2007)

Essa formulação garante a otimalidade da solução desde que não exista itens num terceiro nível de corte. Esses espaços gerados para o terceiro nível de corte não podem ser reaproveitados e formam as perdas internas (dentro das faixas). Essas perdas são apresentadas na Figura 12. Nela é possível perceber ainda a formação das faixas e as perdas externas. As perdas externas podem ser reaproveitadas, caso exista algum item não alocado que caiba dentro dela. As perdas internas, por sua vez, estão num terceiro estágio de corte e este modelo não as contemplam.

O item P6 da Figura 12, por exemplo, poderia ser alocado na sobra interna do item P3, mas, devido às limitações do modelo, ela não é utilizada e dois contêineres são necessários para comportar o conjunto de itens que apenas um poderia fazer.

2.2.2 Formulação matemática para o caso não-estagiado

Da década de 60 para os dias atuais, nenhuma formulação para o corte guilhotinado não estagiado foi encontrada até recentemente quando Messaoud, Chu e Espinouse (2008) apresentaram um modelo para a variação do problema proveniente do problema de uma dimensão aberta ou problema de faixa (*strip packing problem*). Mas, somente Furini, Malaguti e Thomopulos (2016) apresentaram um modelo capaz de tratar o caso derivado do problema de corte e empacotamento (*bin packing*).

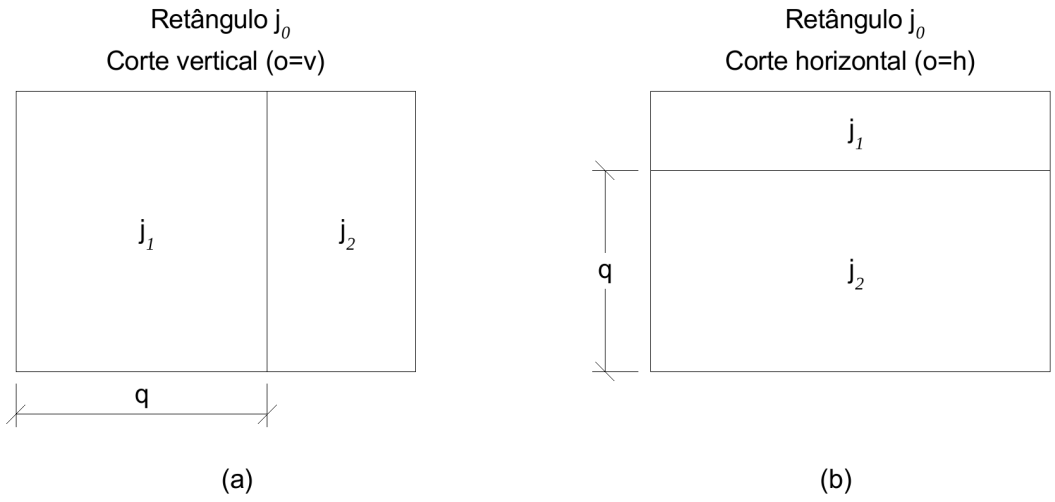
Os autores pegaram a formulação de Dyckhoff (1981) para o corte unidimensional e a ampliaram para o corte bidimensional. Dyckhoff faz uso da ideia de corte e sobra: sempre que um item i de comprimento l_i é cortado, obtêm-se uma sobra de dimensão $L - l_i$. O processo se reinicia com essa sobra como ponto de partida. A expansão de uma para duas dimensões se dá utilizando duas sobras retangulares no lugar do item e sobra unidimensional. As novas sobras realimentam o processo até que os retângulos residuais tenham a dimensão exata de um item (FURINI; MALAGUTI; THOMOPULOS, 2016).

Semelhante ao que ocorre com a formulação de Lodi, Martello e Vigo (2004), duas constantes W e H definem a largura e a altura da placa ou contêiner e constantes w_j e h_j definem a largura e altura de cada item j. A única constante nova é a constante d_j que diz qual a demanda a ser atendida de cada item j. Cada corte nessa formulação é representada por uma posição q, um retângulo j e uma orientação o. A posição q indica a distância entre o canto inferior esquerdo e o corte em cada retângulo j para cada orientação o como pode ser visto na Figura 13.

O conjunto O é definido como o conjunto das orientações possíveis (h e v). O conjunto J armazena todos os retângulos possíveis e um subconjunto $\bar{J}$ indica quais desses retângulos têm as mesmas dimensões de um item, ou seja, o conjunto $\bar{J}$ também representa o conjunto de itens. Além disso $j = 0$ (a placa sem corte) também está contido em $\bar{J}$, neste caso, sua demanda será zerada ($d_0 = 0$). Para cada retângulo $j \in J$, um conjunto de cortes possíveis $Q(j, o)$ é definido de modo que $Q(j, h) \subseteq \{1, ..., h_j - 1\}$ e $Q(j, v) \subseteq \{1, ..., w_j - 1\}$ como visto na Figura 14.

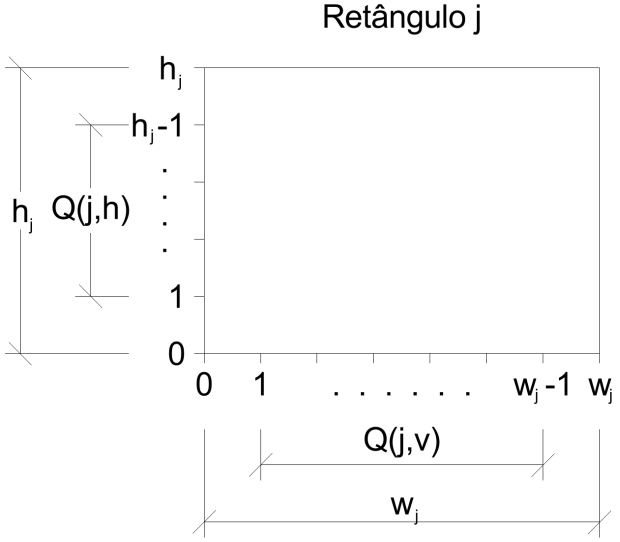
A variável inteira $x^o_{q,j}$ armazenará quantas vezes o retângulo do tipo j foi cortado na posição q com orientação o. O coeficiente $a^o_{q,k,j}$ tem valor 1 quando o retângulo k é obtido pelo corte na posição q do retângulo j com orientação o ou 0 caso contrário.

Figura 13 – Representação do corte q.



Fonte: Baseado em (FURINI; MALAGUTI; THOMOPULOS, 2016)

Figura 14 – Valores válidos do conjunto Q para cada retângulo j.



Fonte: Baseado em (FURINI; MALAGUTI; THOMOPULOS, 2016)

Para manter a linearidade do modelo, esse valor é obtido com pré-processamento na expansão do modelo compacto (na geração das tabelas do modelo de programação linear). A variável inteira y_j é definida para quando $j \in \bar{J}$ e armazena quantos retângulos do tipo j foram obtidos pelo modelo.

O problema de corte bidimensional guilhotinado da seguinte forma:

$$min \sum_{o \in O} \sum_{q \in Q(0,o)} x^o_{q,0} + y_0 \tag{2.10}$$

$$\sum_{k \in J} \sum_{o \in O} \sum_{q \in Q(k,o)} a^o_{q,k,j} x^o_{q,k} - \sum_{o \in O} \sum_{q \in Q(j,o)} x^o_{q,j} - y_j \geq 0, \forall j \in \bar{J}, j \neq 0 \tag{2.11}$$

$$\sum_{k\in J}\sum_{o\in O}\sum_{q\in Q(k,o)} a^o_{q,k,j}x^o_{q,k} - \sum_{o\in O}\sum_{q\in Q(j,o)} x^o_{q,j} \geq 0, \forall j \in J \setminus \bar{J} \tag{2.12}$$

$$y_j = d_j, \forall j \in \bar{J} \tag{2.13}$$

$$x^o_{q,j} \in \mathbb{Z}^+, \forall j \in J, o \in O, q \in Q(j,o) \tag{2.14}$$

$$x^o_{q,j} \in \mathbb{Z}^+, \forall j \in \bar{J} \tag{2.15}$$

A função objetivo está definida em (2.10). Ela tenta minimizar o numero de contêineres utilizados. A restrição (2.11) impõe que o número de partes j que estão em cada retângulo ou itens individuais não exceda o número de retângulos j obtidos através de corte de algum outro retângulo. A restrição (2.12) é equivalente à restrição (2.11) para os retângulos $j \notin \bar{J}$ (consequentemente, para quando as variáveis correspondentes y_j não estejam definidas). A restrição (2.13) impõe que a demanda associada a cada item seja satisfeita e as restrições (2.14) e (2.15) definem o escopo das variáveis.

O principal problema dessa abordagem é a quantidade de variáveis geradas e isso impacta fortemente na complexidade em solucionar o problema. Como dito pelos próprios autores, um exemplo do uso desse modelo para solucionar o caso da mochila, onde poucos contêineres são utilizados (geralmente um) gerou cerca de sessenta e seis milhões de variáveis para 50 itens.

Fazendo um comparativo hipotético, se o problema fosse polinomial e de ordem $O(n^2)$, seriam geradas $4,356 \cdot 10^{15}$ possíveis soluções para testar. Em um sistema computacional que pudesse realizar um milhão de testes por segundo, seriam necessários 4,356 bilhões de segundos, cerca de 138 anos, para resolver o problema com 50 itens. Como o problema descrito na formulação pode gerar até cinquenta mochilas (caso em que cada item ocupe um contêiner), imagina-se que este problema do corte bidimensional guilhotinado possa gerar bem mais variáveis.

3 MODELO MATEMÁTICO

3.1 PROPOSTA DE UMA FORMULAÇÃO MATEMÁTICA LINEAR

Como foi mencionado anteriormente, o problema de corte bidimensional guilhotinado em seu caso não estagiado é difícil de ser modelado pelas abordagens de faixas e de forma de árvore. Neste trabalho, optamos por uma abordagem um pouco diferente. Essa nova abordagem é simplesmente uma mudança na ótica da abordagem em forma de árvore binária.

Essa abordagem nasceu das observações dos trabalhos de Christofides e Whitlock (1977), Wang (1983), Nascimento, Longo e Aloise (1999) e Furini, Malaguti e Thomopulos (2016):

- Cada corte sempre separa um pedaço em dois outros, ou, a partir de outra ótica, cada corte une dois pedaços em um maior;
- Se n itens forem cortados a partir de um único pedaço de material e sem gerar sobras, haverão $n - 1$ cortes;
- Se dois itens retangulares são unidos em um pedaço de material retangular, a sobra gerada terá a largura do item mais baixo e a altura será a diferença entre as alturas dos itens para cortes verticais;
- A mesma análise dos cortes verticais serve para cortes horizontais, invertendo apenas as posições das alturas e larguras no ponto anterior;
- Se nem todos os itens cabem em uma única placa, após um certo número de uniões será gerado um pedaço de material que ocupará a totalidade da placa ou que a nenhum outro item caberá em uma das sobras geradas.

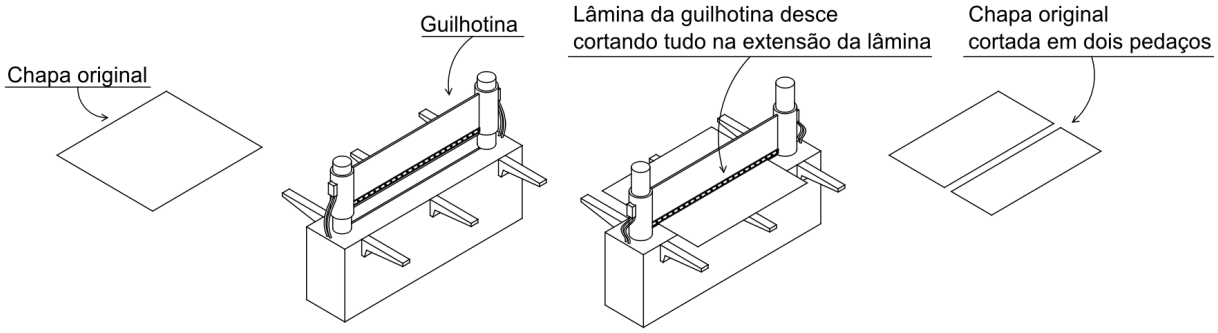
A partir dessas observações, propõe-se uma formulação matemática baseada em matrizes triangulares, ou melhor, em um conjunto de listas cujos tamanhos vão sendo reduzidos em uma unidade a medida que os cortes vão sendo realizados. A lista inicial terá o número retângulos igual ao número de itens a serem cortados. A medida que os cortes ou uniões vão sendo realizadas, dois retângulos darão lugar a apenas um, por isso a redução em um retângulo. Caso não consiga realizar essa união, o retângulo que inviabiliza as uniões deverá ser eliminado, de forma a manter a lista seguinte sempre com um retângulo a menos.

Para descrever tal comportamento matematicamente, inicialmente foi gerada uma formulação não linear. Sobre essa formulação foram aplicadas algumas técnicas matemáticas que apresentaremos na seção 3.1.2 para linearizar algumas restrições e apresentar um modelo matemático linear para o problema de corte bidimensional guilhotinado.

3.1.1 Formulação matemática não linear

A principal característica do corte guilhotinado é que o corte sempre ocorre de um lado ao outro do pedaço de material a ser cortado. Isso se deve as características da ferramenta de corte utilizada. A guilhotina (Figura 15) possui uma grande lâmina que corta toda uma faixa, separando a placa original em dois pedaços.

Figura 15 – Ilustração de uma guilhotina e do processo de corte.



Fonte: Autoria própria

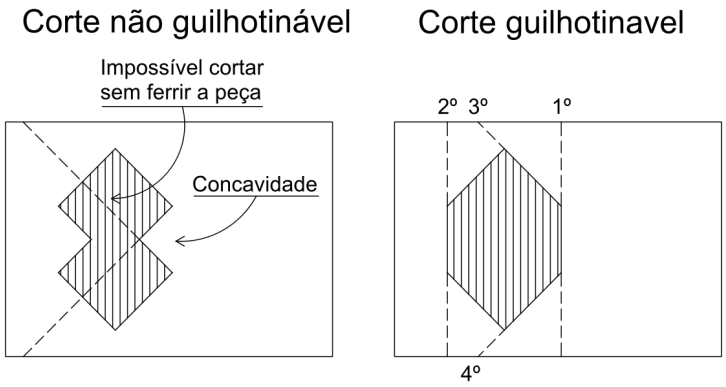
A limitação dos cortes em lista reta irem sempre de um lado ao outro, inviabiliza que sejam realizados cortes de peças com concavidade, mas peças convexas podem ser recortadas do material original através de uma sequência de cortes (Figura 16).

Como forma de facilitar os cortes, os itens a serem cortados das placas originais, serão sempre retangulares. Desse modo, um item que não seja retangular, primeiro deve ser alocado em um retângulo e depois alocado na placa para corte (Figura 17).

É importante lembrar que alocar todos os itens em retângulos não garante que qualquer corte seja guilhotinável. O arranjo deve permitir que um corte atravesse de um lado ao outro da placa ou de um retângulo previamente cortado sem ferir nenhum outro item no processo.

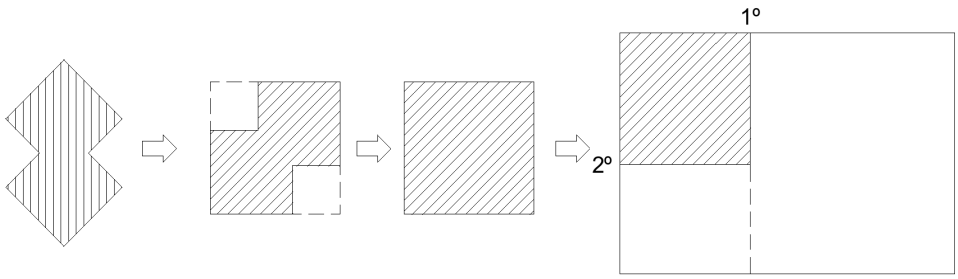
A partir da premissa que todos os itens a serem cortados serão alocados em retângulos, a distribuição desses retângulos na placa ficará sujeita a localização e a rotação dos mesmos. Quanto a rotação, permitir rotações apenas ortogonalmente mostra-se o mais recomendado, pois reduz a quantidade de ângulos possíveis de um número

Figura 16 – Peça não guilhotinável e peça guilhotinável.



Fonte: Autoria própria

Figura 17 – Alocação de peças (guilhotináveis ou não) em retângulos.



Fonte: Autoria própria

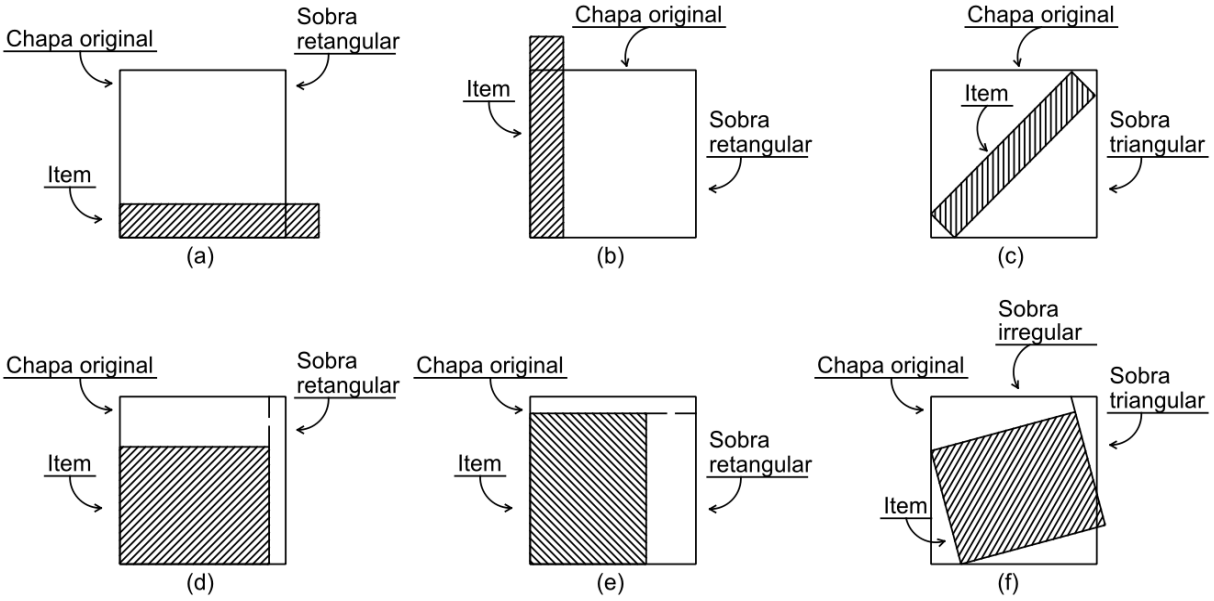
infinito para apenas dois, visto que um retângulo rotacionado em 180º é igual a ele mesmo.

Além disso, exceto em situações bem específicas (como mostrado na Figura 18(c)), as rotações em ângulos não ortogonais trazem apenas prejuízo, tais como exigir tamanhos maiores de placas (Figura 18(f)) e gerar sobras não retangulares (Figura 18(c), (f)), onde é mais difícil alocar itens retangulares dentro e mesmo alocando, sempre haverá desperdício. Dessa forma, apenas as rotações 0º e 90º serão consideradas para o modelo matemático.

O posicionamento de um retângulo dentro da placa também pode ser infinito, todavia serão consideradas apenas as localizações nos cantos. Olhando para o posicionamento dos retângulos dentro da placa, é fácil perceber que nelas as sobras são maiores e que, se um retângulo $j + 1$ cabe em uma sobra com o retângulo j ao centro, ela também caberá em uma das sobras com j posicionado no canto (Figura 19).

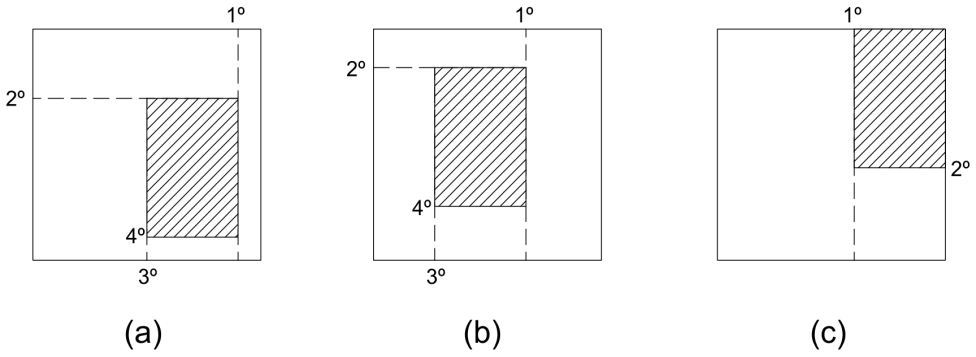
Um questionamento que pode ser levantado é que um determinado conjunto de retângulos só encaixem perfeitamente se um retângulo j estiver no centro. Nesse caso, bastaria refazer a ordem em que os retângulos são recortados e postergar o corte desse retângulo j. Na Figura 20, por exemplo, o arranjo poderia ser conseguido com o corte do retângulo $j + 1$(a), depois o retângulo $j + 3$(b) e então o retângulo j estará posicionado em um dos cantos (c).

Figura 18 – Rotações de itens, possibilidades de encaixes e sobras decorrentes de cada uma.



Fonte: Autoria própria

Figura 19 – Posicionamento de um item dentro de um retângulo.

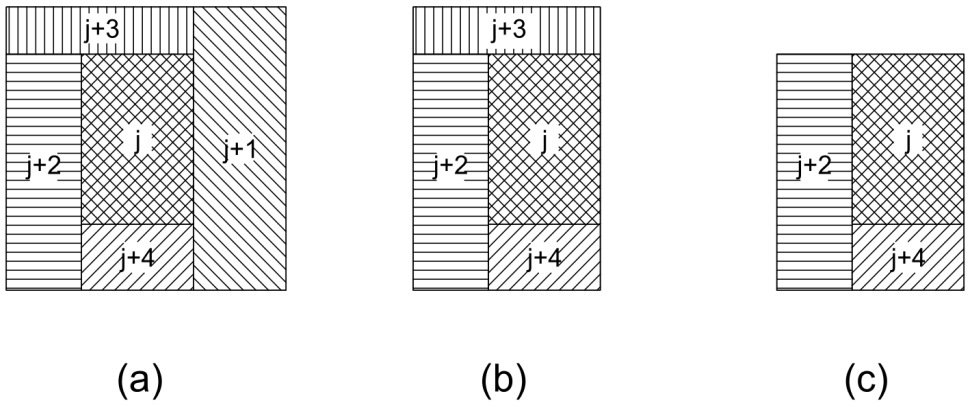


Fonte: Autoria própria

Uma outra característica interessante é que o número de cortes necessários para separar um retângulo diminui para dois, caso seja alocado em um dos cantos da placa. Como mostrado na Figura 19(c). O canto a ser escolhido também não importa, pois, se a ordem dos dois cortes for a mesma (vertical e horizontal ou vice e versa), as sobras terão sempre o mesmo tamanho. Esse comportamento está exemplificado na Figura 21. E, devido a característica recursiva inerente ao problema, são as dimensões das sobras que importam.

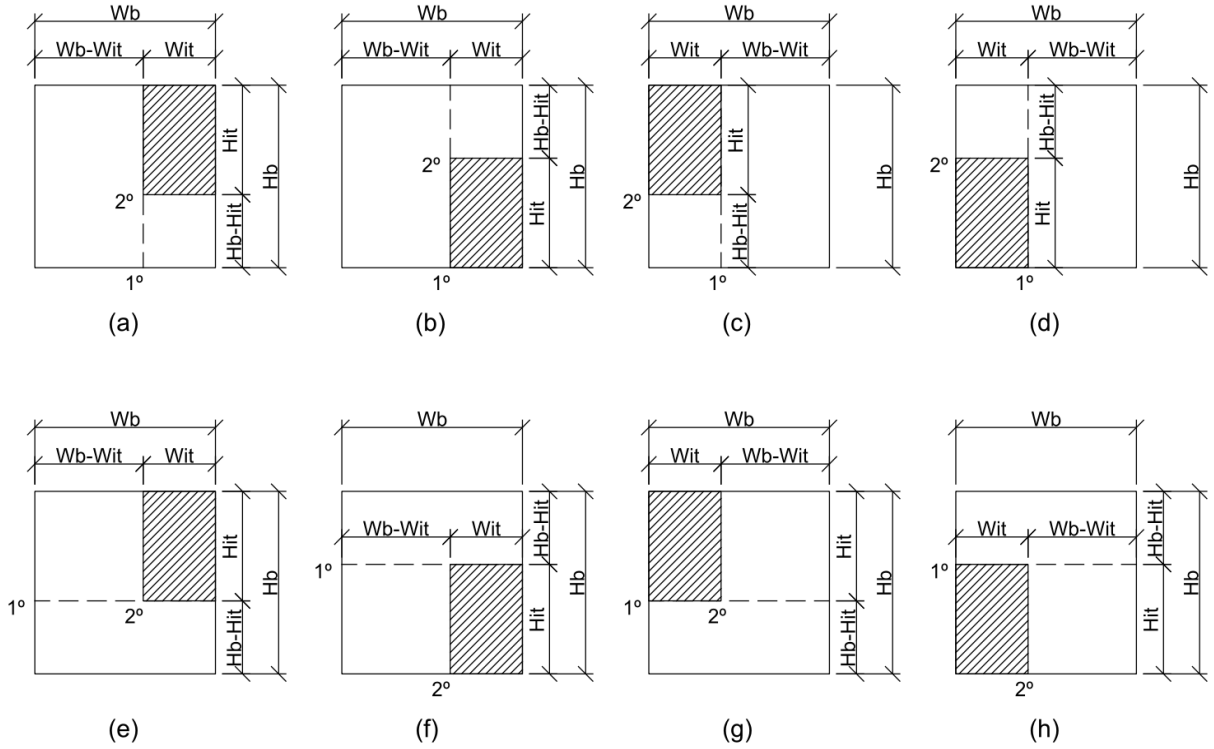
A alocação de uma lista de itens em uma placa segue uma função recursiva, visto que o resultado de um corte podem ser encarados como novas placas a serem cortadas. Dessa forma, todas as características levantadas para o primeiro corte podem ser aplicadas aos cortes seguintes. Na Figura 22 é possível ver que o corte do i é aplicado sobre o retângulo (a) e geram os retângulos (b) e (c). Do mesmo modo o corte $i + 1$ gera,

Figura 20 – Posicionamento centralizado de um item através da ordem dos cortes.



Fonte: Autoria própria

Figura 21 – A posição em qualquer um dos quatro cantos geram sobras com as mesmas dimensões.

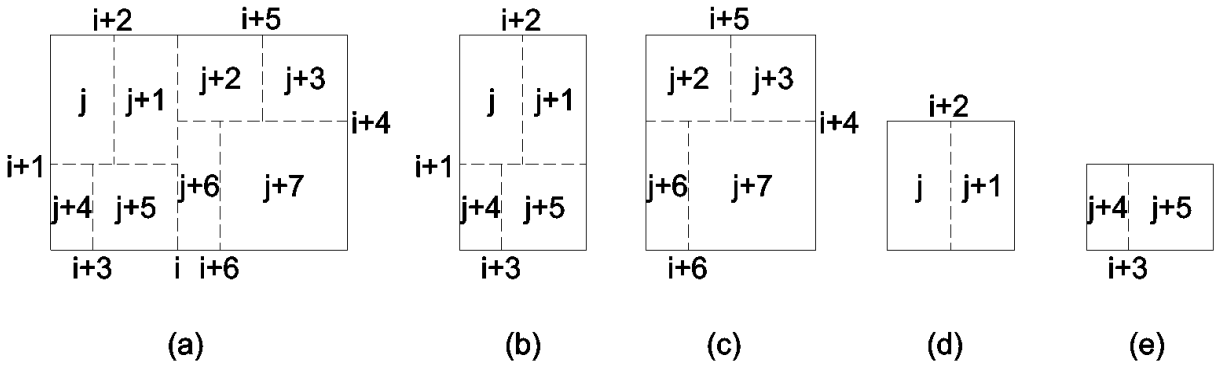


Fonte: Autoria própria

a partir do retângulo (b), os retângulos (d) e (e). Essa função recursiva segue até que dos os retângulos gerados contenham apenas um item dentro.

Aqui há uma ressalva, no exemplo da Figura 22, não há sobras, porem elas devem ser consideradas. Entretanto, não há como saber quantas sobras haverão e nem quais suas dimensões até que sejam realizados os cortes. Uma solução para tal problema pode ser encontrada na construção do arranjo de baixo para cima. Se um corte separa um retângulo em dois menores, então é possível dizer que dois retângulos são unidos por um corte em um maior.

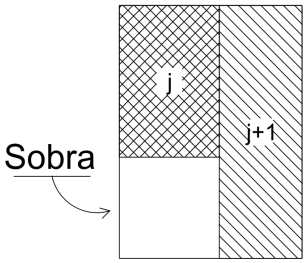
Figura 22 – Recursividade presente nos cortes dos retângulos.



Fonte: Autoria própria

Voltando ao exemplo da Figura 22, do mesmo modo que o corte $i + 1$ separa o retângulo (b) em (d) e (e) é possível dizer que o que une (d) e (e) no retângulo (b) é o corte $i + 1$. Dessa forma, a união de dois itens, resultará em um retângulo que poderá ser unido com outro ou com um item. Quando a união de dois itens, um item e um retângulo ou dois retângulos não resultarem em outro retângulo perfeito, uma sobra será acrescida (Figura 23).

Figura 23 – Sobra gerada para complementar o retângulo.



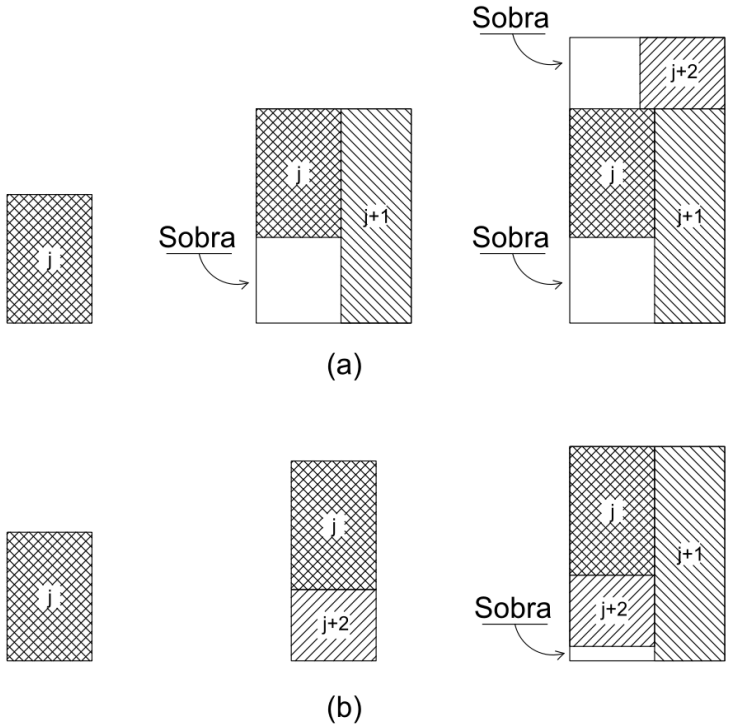
Fonte: Autoria própria

As dimensões das sobras acrescidas desse modo, seguem a mesma lógica mostrado na Figura 21, com a diferença que no local de uma das sobras, temos um outro item. Se esses arranjos serão os menores e mais otimizados, caberá ao algoritmo de buscas definir.

Um questionamento que pode ser levantado aqui é que se a sobra for grande o bastante para caber um outro item dentro, então essa solução não será ótima. Esse questionamento é válido, mas bastaria alterar a ordem das uniões para se chegar a um valor ótimo. Por exemplo, digamos que na Figura 23 houvesse um retângulo $j + 2$ que coubesse dentro da sobra complementar. Neste caso, bastaria que primeiro fosse feita a união do retângulos j e $j + 2$ para depois unir esse retângulo resultante com o retângulo $j + 1$ (Figura 24).

Considerando as características e delimitações do problema que já foram mostradas. Pode-se afirmar que com dois itens, existem apenas oito possibilidades de arranjo

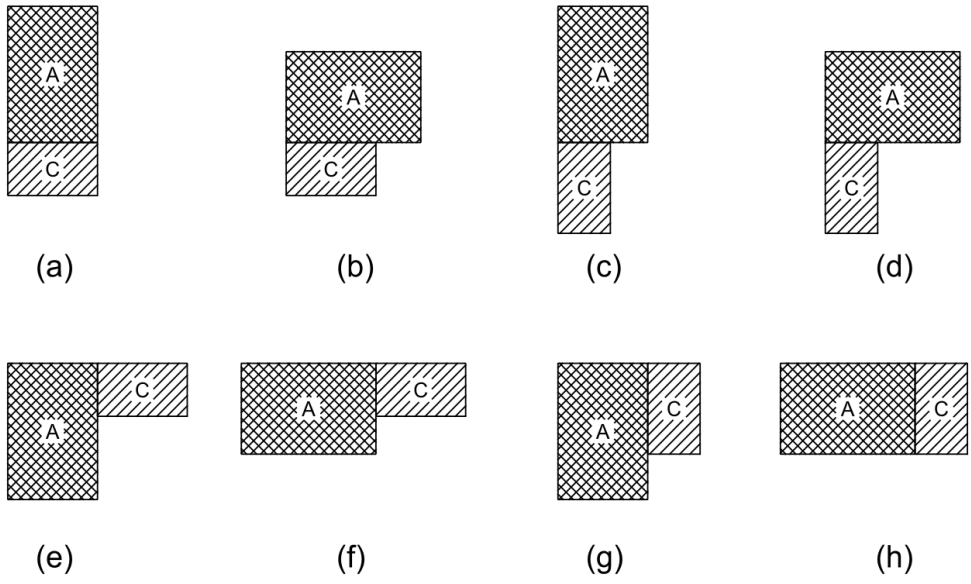
Figura 24 – Reaproveitamento de uma sobra através da alteração da ordem das junções.



Fonte: Autoria própria

entre esses itens. Essas possibilidades estão exemplificadas na Figura 25. Essas oito possibilidades nascem da combinação das possibilidades de rotação de cada item (0º ou 90º para cada item) e orientação do corte (vertical ou horizontal).

Figura 25 – Possibilidades de união entre dois retângulos.



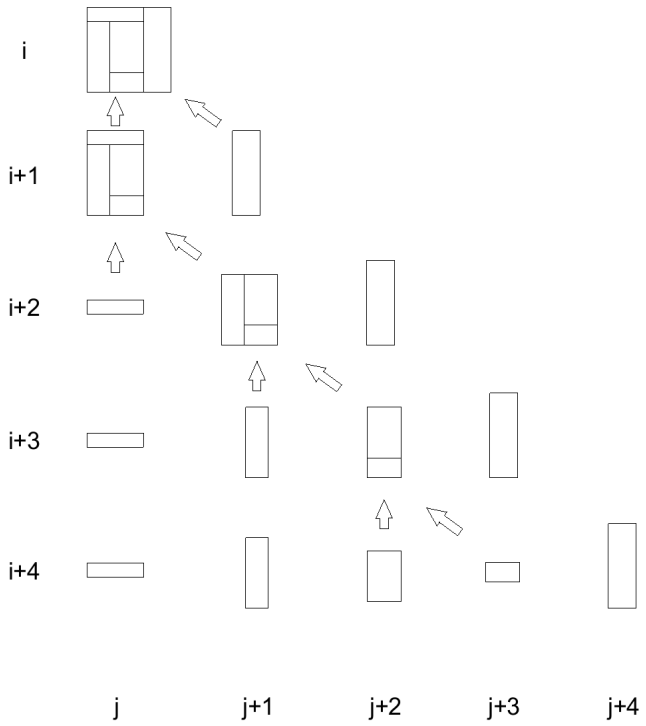
Fonte: Autoria própria

Para cada um desses arranjos, um novo retângulo será formado e acrescido lista de retângulos. Os itens que o compõem serão retirados e assim o processo recursivo

segue até que sobre apenas um, desde que ele não ultrapasse as dimensões da placa original.

Como a construção ocorre de baixo para cima, ou seja, vai unido os itens e não cortanto as placas, haverá uma lista contendo n retângulos inicialmente e, a cada iteração, essa lista irá diminuir em uma unidade, pois dois elementos serão unidos em um para a próxima lista. Olhando a partir dessa ótica, pode-se afirmar que existirão n listas e que seu tamanho irá decaindo de n até 1 formando uma matriz triangular, como mostrado na Figura 26.

Figura 26 – Matriz triangular formada pelas **n** listas de itens.



Fonte: Autoria própria

Caso exista um corte na lista i e coluna j, o retângulo da lista i e coluna j será formado por dois da lista abaixo, sendo um da mesma coluna e outra da coluna a frente (j e $j + 1$, respectivamente). Caso não exista um corte na lista i e coluna j, ele terá as dimensões do retângulo j ou $j + 1$ da lista abaixo ($i - 1$), dependendo da sua posição em relação ao corte da lista i, antes ou depois, respectivamente.

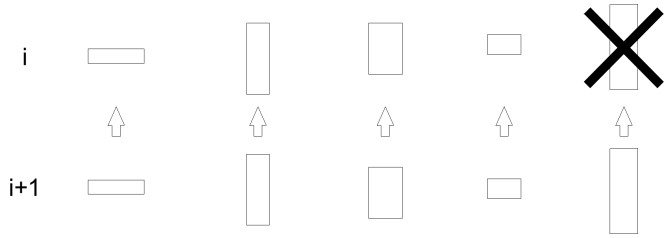
Devido a característica recursiva, o acréscimo de mais um elemento na lista inicial apenas faria com que fossem executados mais passos na montagem do arranjo desses retângulos, isso porque cada lista é a "entrada" para geração da lista acima.

A montagem dos retângulos ocorre apenas com vizinhos e a partir da lista $n - 1$ até a lista 1. De modo que a ordem em que os itens são alocados na lista n é de suma importância e sua alteração pode gerar soluções diferentes. Além disso, é nela que se define se os itens estão rotacionados ou não. A partir da lista $n - 1$ isso se torna

redundante, visto que se rotacionar duas vezes o mesmo item ele volta a posição original (vide Figura 18) e rotacionar dois retângulos unidos em um corte vertical é equivalente a uni-los rotacionados horizontalmente.

Os elementos da lista i que não são resultados de um corte são copiados a partir da lista $i+1$, como pode ser visto nas Figuras 26. Todavia, como o tamanho dela diminui, ha uma distinção para a realização dessa cópia. Todos os retângulos anteriores ao corte são copiados da lista $i+1$ para a lista i mantendo a posição na coluna j. Os que estão após o corte são deslocados para não formar lacunas, ou serem descartados no processo. Devido a isso, caso não exista um corte na lista i o úlimo elemento da lista $i+1$ não será copiado, como mostra a Figura 27.

Figura 27 – Descarte do último elemento de uma lista



Fonte: Autoria própria

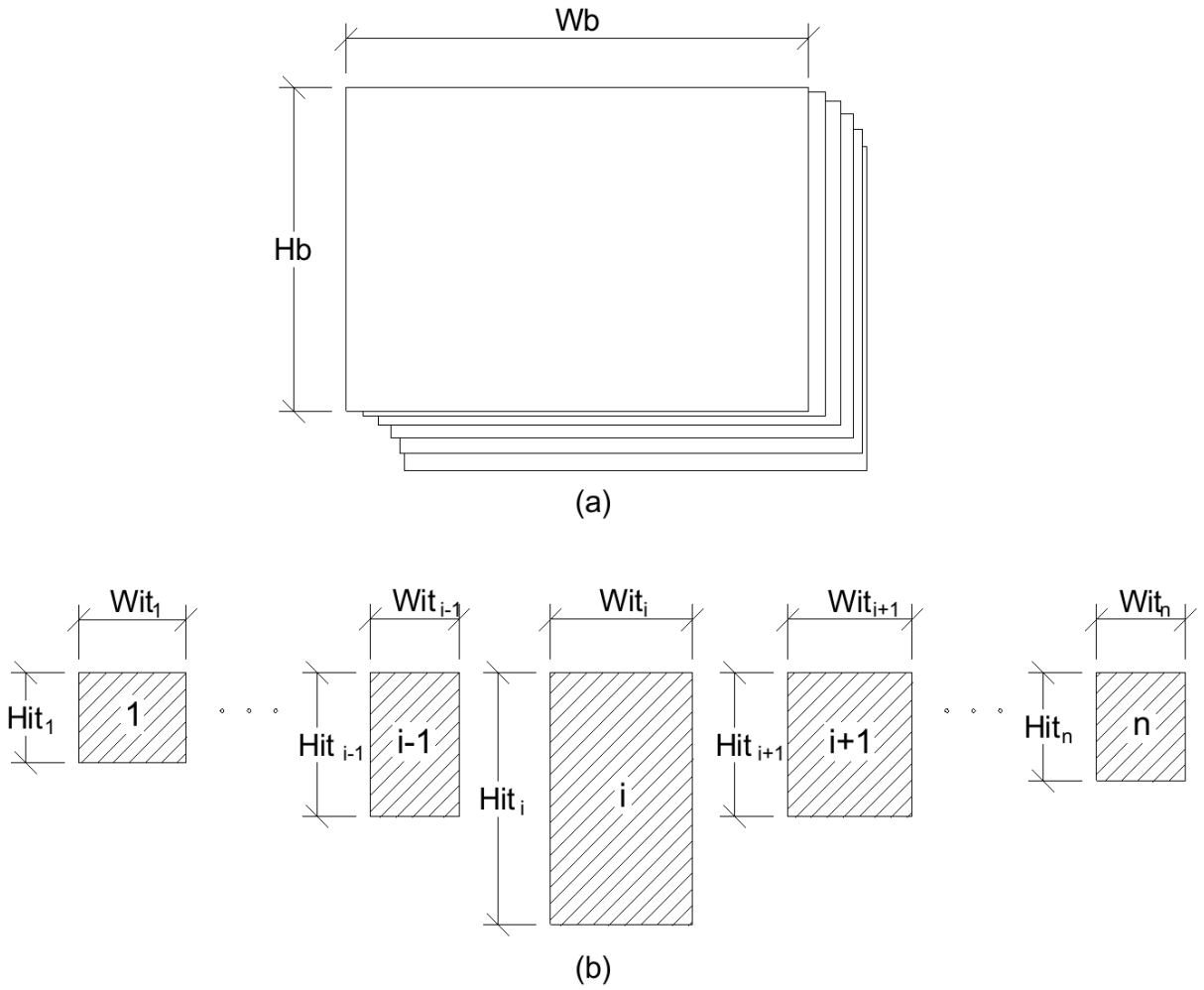
Um questionamento que pode ser feito a respeito dessa característica do modelo é que, se o retângulo a ser descartado puder ser alocado a uma das sobras, então essa solução não será ótima. Para entender como o modelo lida com essa situação, primeiro é preciso entender porque em uma lista não poderia haver um corte. Isso só ocorrerá se a união de quaisquer dois retângulos da lista inferior resultar em um retângulo que ultrapasse algum das dimensões da placa.

Olhando por essa ótica, é como se o retângulo $(j + (j + 1))$ da Figura 20 não pudesse ser unido ao retângulo $j + 2$, pois ultrapassaria alguma das dimensões da placa original. Todavia, como pode ser visto na mesma figura, há outra sequencia de uniões, na qual existe a possibilidade de utiliza-lo em uma sobra anterior. Logo, se o seu descarte gerar uma solução não ótima, isso implica que deve haver uma sequencia diferente onde esse ele é aproveitado em uma sobra, de modo a gerar uma solução ótima.

Semelhante aos modelos anteriores, duas constantes, Wb e Hb, representam a largura e a altura da placa. As constantes Wit_i e Hit_i representam a largura e a altura de cada item i. Tais valores são provenientes da instância a ser utilizada (Figura 28).

As variáveis de decisão propriamente ditas são as variáveis $x_{i,j}$, $y_{i,j}$, r_j e v_i. A variável $x_{i,j}$ indica a posição dos itens na lista inicial, caso um elemento $x_{i,j}$ tenha valor 1, isso significa que o item i está alocado na posição j da lista n, do contrário, $x_{i,j}$ receberá 0. A variável $y_{i,j}$ indica a posição dos cortes, caso uma variável $y_{i,j}$ receba valor 1, isso significa que o elemento j da lista i é formado pelos elementos j e $j+1$ da lista $i+1$. A variável r_j indica as rotações dos itens, quando uma variável r_j tem valor 1, significa

Figura 28 – Alturas e larguras dos itens e objetos.



Fonte: Baseado em (TEMPONI, 2007)

que o item da posição j na lista n está rotacionada de 90º, caso receba valor 0, esse item não está rotacionado. O vetor V_{n-1} indica a orientação do corte em cada linha i da matriz $Y_{n-1,n-1}$, um elemento v_i desse vetor recebe 1 caso o corte seja vertical e 0 caso seja horizontal.

Além das variáveis de decisão, variáveis auxiliares, cujo valor deriva das variáveis de decisão e das constantes, são usadas no modelo para garantir sua integridade. As variáveis $w_{i.j}$ e $h_{i,j}$ recebem os valores das alturas e larguras de cada retângulo j da lista i (vide Figura 26). A variável $a_{i,j}$ situa os elementos da lista i em relação ao corte, caso tenha valor 0, significa que o elemento j está antes do corte, se 1, ele está depois.

Além delas, as variáveis $wst_{s,i,j}$ e $hst_{s,i,j}$ também recebem a largura e a altura desses retângulos, mas para cada situação s as dimensões dos elementos das listas de 1 a $n-1$ podem ser formadas. A situação 1 é quando é um corte ($y_{i,j} = 1$) e ele é vertical ($v_i = 1$), a situação 2 é quando é um corte ($y_{i,j} = 1$) e ele é horizontal ($v_i = 0$), a situação 3 é quando não é um corte ($y_{i,j} = 0$) e ele está depois do corte da lista atual ($a_{i,j} = 1$) e a

situação 3 é quando não é um corte ($y_{i,j} = 1$) e ele esta antes do corte ($a_{i,j} = 0$).

Dessa forma o problema pode ser definido matematicamente da seguinte forma:

$$max \sum_{i=1}^{n-1} \sum_{j=1}^{i} y_{i,j} \tag{3.1}$$

Sujeito à:

$$\sum_{j=1}^{n-1} y_{i,j} \leq 1, \forall i < n \tag{3.2}$$

$$h_{n,j} = (1 - r_j) \cdot \sum_{i=1}^{n} (x_{i,j} \cdot Hit_i) + r_j \cdot \sum_{i=1}^{n} (x_{i,j} \cdot Wit_i), \forall j \tag{3.3}$$

$$w_{n,j} = (1 - r_j) \cdot \sum_{i=1}^{n} (x_{i,j} \cdot Wit_i) + r_j \cdot \sum_{i=1}^{n} (x_{i,j} \cdot Hit_i), \forall j \tag{3.4}$$

$$\sum_{j=1}^{n} x_{i,j} = 1, \forall i \tag{3.5}$$

$$\sum_{i=1}^{n} x_{i,j} = 1, \forall j \tag{3.6}$$

$$a_{i,j} = \sum_{k=1}^{j} y_{i,j}, \forall i < n, j \leq i \tag{3.7}$$

$$hst_{1,i,j} = y_{i,j} \cdot v_i \cdot max(h_{i+1,j}, h_{i+1,j+1}), \forall i < n, j \leq i \tag{3.8}$$

$$hst_{2,i,j} = y_{i,j} \cdot (1 - v_i) \cdot (h_{i+1,j} + h_{i+1,j+1}), \forall i < n, j \leq i \tag{3.9}$$

$$hst_{3,i,j} = (1 - y_{i,j}) \cdot (1 - a_{i,j}) \cdot h_{i+1,j+1}, \forall i < n, j \leq i \tag{3.10}$$

$$hst_{4,i,j} = (1 - y_{i,j}) \cdot a_{i,j} \cdot h_{i+1,j}, \forall i < n, j \leq i \tag{3.11}$$

$$h_{i,j} = hst_{1,i,j} + hst_{2,i,j} + hst_{3,i,j} + hst_{4,i,j}, \forall i < n, j \leq i \tag{3.12}$$

$$wst_{1,i,j} = y_{i,j} \cdot v_i \cdot (w_{i+1,j} + w_{i+1,j+1}), \forall i < n, j \leq i \tag{3.13}$$

$$wst_{2,i,j} = y_{i,j} \cdot (1 - v_i) \cdot max(w_{i+1,j}, w_{i+1,j+1}), \forall i < n, j \leq i \tag{3.14}$$

$$wst_{3,i,j} = (1 - y_{i,j}) \cdot (1 - a_{i,j}) \cdot w_{i+1,j+1}, \forall i < n, j \leq i \tag{3.15}$$

$$wst_{4,i,j} = (1 - y_{i,j}) \cdot a_{i,j} \cdot w_{i+1,j}, \forall i < n, j \leq i \tag{3.16}$$

$$w_{i,j} = wst_{1,i,j} + wst_{2,i,j} + wst_{3,i,j} + wst_{4,i,j}, \forall i < n, j \leq i \tag{3.17}$$

$$0 < h_{i,j} \leq Hb, \forall i, j \leq i \tag{3.18}$$

$$0 < w_{i,j} \leq Wb, \forall i, j \leq i \tag{3.19}$$

$$hst_{s,i,j}, wst_{s,i,j}, h_{i,j}, w_{i,j} \in R^{+}, \forall s \in \{1,2,3,4\}, i, j \leq i \tag{3.20}$$

$$x_{i,j}, r_j \in \{0,1\}, \forall i, j \leq i \tag{3.21}$$

$$y_{i,j}, a_{i,j}, v_i \in \{0,1\}, \forall i < n, j \leq i \tag{3.22}$$

A função objetivo está definida na equação (3.1). Como o objetivo do problema é encontrar a menor quantidade de placas necessários para comportar todos os itens, quanto mais cortes existirem, menos placas farão parte da solução. Isso ocorre porque cada corte une dois retângulos da lista imediatamente abaixo, reduzindo o número deles na lista acima.

A restrição (3.2) define que pode haver, no máximo, um corte em cada lista. As restrições (3.3) e (3.4) definem a altura ($h_{n,j}$) e a largura ($w_{n,j}$) da lista n. Nessa lista em específico, todos os retângulos possuem as dimensões dos itens ali alocados. Os somatórios de ($x_{i,j} \cdot Hit_i$) e ($x_{i,j} \cdot Wit_i$) nessas equações definem a altura e a largura de cada retângulo j dessa lista. As multiplicações desses somatórios por r_j e $(1 - r_j)$ definem se a altura do retângulo receberá a largura ou altura do item ali alocada, se rotacionado ou não.

As restrições (3.5) e (3.6) existem para garantir que cada item apareça somente uma vez e que cada posição tenha somente um item respectivamente. A restrição (3.7) garante que a variável $a_{i,j}$ receba 0 ou 1, caso esteja o elemento j esteja antes ou depois, respectivamente. Isso acontece porque se estiver depois existirá um $y_{i,k} = 1$ de modo que o somatório será 1, caso contrário o resultado desse somatório será 0.

Um detalhe é que se não houver corte na lista atual todas as variáveis $a_{i,j}$ dela terão valor 0, assim todos os seus elementos são copias da lista $i + 1$ mantendo a posição da coluna j. Como há um elemento a menos na lista atual, um da lista inferior é descartado (Figura 27).

As restrições (3.8), (3.9), (3.10) e (3.11) indicam as quatro situações em que a altura de um retângulo pode ser formada. A restrição (3.8) é a situação em que há um corte na posição j da lista i e esse corte é vertical. Nessa situação a altura do retângulo é a maior altura dentre os seus componentes. A restrição (3.9) representa a situação na qual há um corte, mas ele é horizontal. Neste caso a altura do retângulo é a soma das alturas dos retângulos componentes.

As restrições (3.10) e (3.11) são para o caso onde não há corte naquela posição. Quando o retângulo está antes do corte da lista (3.10) o ele recebe a altura do retângulo diretamente abaixo (posição j da lista $i + 1$). Quando está depois do corte da lista (3.11)

é feito um deslocamento dos retângulos para não gerar lacunas, assim o retângulo atual recebe o retângulo abaixo e a frente (da posição $j + 1$ da lista $i + 1$).

Os resultados dessas quatro situações ficam armazenadas nas variáveis $hst_{1,i,j}$, $hst_{2,i,j}$, $hst_{3,i,j}$ e $hst_{4,i,j}$. Essas quatro situações são excludentes, se uma for verdadeira, as demais serão falsas e as multiplicações as tornam nulas, a soma das quatro situações resulta apenas na única situação válida e será armazenada na variável $h_{i,j}$. Essa soma ocorre na restrição (3.12).

As restrições (3.13), (3.14), (3.15), (3.16) e (3.17) funcionam de forma análoga as restrições (3.8), (3.9), (3.10), (3.11) e (3.12), porém para a largura do retângulo. Semelhantemente, as variáveis $wst_{1,i,j}$, $wst_{2,i,j}$, $wst_{3,i,j}$, $wst_{4,i,j}$ e $w_{i,j}$ equivalem as variáveis $hst_{1,i,j}$, $hst_{2,i,j}$, $hst_{3,i,j}$, $hst_{4,i,j}$ e $h_{i,j}$, porém, para a largura.

Vale salientar que essas restrições poderiam ter sido condensadas em duas restrições apenas (uma para a altura e outra para a largura) se as restrições (3.8), (3.9), (3.10) e (3.11) fossem substituídas na restrição (3.12) e as restrições (3.13), (3.14), (3.15) e (3.16) fossem substituídas na restrição (3.17). Essa separação foi feita com o intuito de facilitar o entendimento do modelo matemático e, na sessão seguinte, simplificar a linearização desse modelo.

As restrições (3.18) e (3.19) forçam que os retângulos tenham dimensões positivas e não ultrapassem as medidas das placas, Hb para altura e Wb para largura (como mostra a Figura 28) e as restrições (3.20), (3.21) e (3.22) indicam quais variáveis são reais positivas e quais são binárias.

3.1.2 Processo de linearização

Essas equações expressam o comportamento do problema de corte bidimensional guilhotinado sob a forma não linear. Apesar da forma não linear permitir a compreensão do modelo, os métodos de resolução de sistemas não lineares são complexos e, muitas vezes, inviáveis para aplicações práticas. Dessa forma, para podermos aplicar métodos de resolução lineares, como o método simplex, algumas adaptações devem ser feitas.

A primeira delas é desfazer a multiplicação entre variáveis que ocorre na restrição (3.3). Inicialmente os somatórios serão separados e atribuídos a outra variável. Dentro do somatório a multiplicação é entre uma constante e uma variável, o que é permitido em sistemas lineares.

$$hsta_{n,j} = \sum_{i=1}^{n} (x_{i,j} \cdot Hit_i), \forall j \tag{3.23}$$

$$wsta_{n,j} = \sum_{i=1}^{n} (x_{i,j} \cdot Wit_i), \forall j \tag{3.24}$$

$$hst_{1,n,j} = (1 - r_j) \cdot hsta_{n,j}, \forall j \tag{3.25}$$

$$hst_{2,n,j} = r_j \cdot wsta_{n,j}, \forall j \tag{3.26}$$

$$h_{n,j} = hst_{1,n,j} + hst_{2,n,j}, \forall j \tag{3.27}$$

A substituição não promove grandes mudanças, mas permite transformar a multiplicação, que antes era de uma variável com um somatório de variáveis em uma multiplicação entre duas variáveis apenas, sendo uma delas binária (r_j) e a outra real ($hsta_{n,j}$ na restrição (3.23) e $wsta_{n,j}$ na restrição (3.24)).

O fato de ser uma multiplicação entre uma variável binária e uma linear e que essa variável linear possui um limite superior (as dimensões da placa), pode-se usar restrições extras para substituir a multiplicação. Esse método consiste em limitar o valor máximo e o mínimo de uma variável *mult* com base nos valores da variável linear e da variável binária multiplicada pelo limite superior da variável linear.

$$mult = b \cdot l \tag{3.28}$$

$$mult \leq b \cdot M \tag{3.29}$$

$$mult \leq l \tag{3.30}$$

$$mult \geq l - M + M \cdot b \tag{3.31}$$

$$mult \geq 0 \tag{3.32}$$

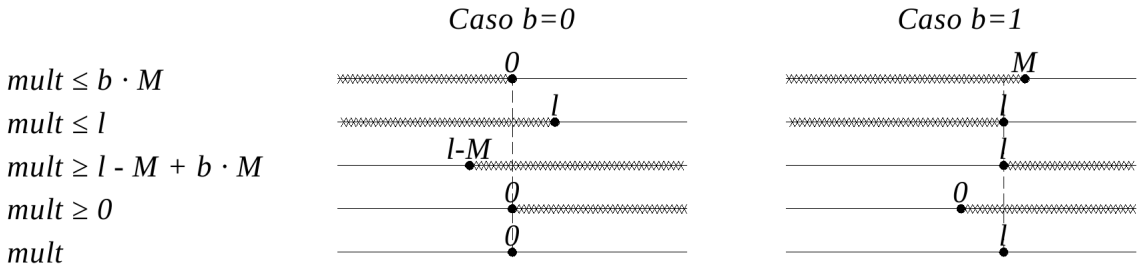
A multiplicação entre uma variável binária e uma linear não-nula positiva na equação (3.28) pode ser substituída pelas inequações (3.29), (3.30), (3.31) e (3.32) desde que exista uma constante M de modo que a variável linear l seja menor ou igual a constante M. Desse modo, o resultado da multiplicação dependerá do valor que a variável binária b assuma, ou seja 0 ou 1.

Substituindo b por 0 na inequação (3.29), teremos que o resultado *mult* deverá que ser menor ou igual a 0. Como a variável l é maior que 0, ser menor ou igual a 0 (restrição (3.29)) respeita a inequação (3.28). Substituindo b por 0 na inequação (3.31), teremos que o resultado *mult* deverá que ser maior ou igual a $l - M$. Como M é uma constante maior ou igual a l, esse resultado será no máximo 0 e *mult* terá que ser maior que 0 (caso $l = M$) ou que um número negativo (caso $l < M$). Desse modo, ser maior ou igual a um número negativo ou nulo respeita a inequação (3.32), que diz que deve ser maior ou igual a 0. Como *mult* deve ser menor ou igual a 0 e maior ou igual a 0. O único resultado que atenda a essas restrições é 0.

Substituindo b por 1 na inequação (3.29), teremos que o resultado *mult* deverá ser menor ou igual a M. Como a variável l é menor ou igual a M, ser menor ou igual a l

(inequação (3.30)) respeita a inequação (3.29). Substituindo b por 1 na inequação (3.30), teremos que o resultado *mult* deverá que ser maior ou igual a $l - M + M$, ou seja, maior ou igual a l. Como l é uma variável maior que 0, ser maior ou igual a l (restrição (3.31)) respeita a inequação a inequação (3.32), que diz que deve ser maior ou igual a 0. Como *mult* deve ser menor ou igual a l e maior ou igual a l. O único resultado que atenda a essas restrições é l. Tal comportamento está expresso na Figura 29.

Figura 29 – Multiplicação de uma variável binária e uma linear expressa como um sistema de inequações.



Fonte: Autoria própria

Aplicando o método na equação (3.25) e (3.26) teremos o seguinte sistema de inequações.

$$hst_{1,n,j} \leq Hb - Hb \cdot r_j, \forall j \tag{3.33}$$

$$hst_{1,n,j} \leq hsta_{n,j}, \forall j \tag{3.34}$$

$$hst_{1,n,j} \geq hsta_{n,j} - Hb \cdot r_j, \forall j \tag{3.35}$$

$$hst_{1,n,j} \geq 0, \forall j \tag{3.36}$$

$$hst_{2,n,j} \leq HB \cdot r_j, \forall j \tag{3.37}$$

$$hst_{2,n,j} \leq wsta_{n,j}, \forall j \tag{3.38}$$

$$hst_{2,n,j} \geq wsta_{n,j} - Hb + Hb \cdot r_j, \forall j \tag{3.39}$$

$$hst_{2,n,j} \geq 0, \forall j \tag{3.40}$$

O mesmo método pode ser aplicado à equação (3.4), inicialmente separando os somatórios e substituindo as multiplicações de variáveis pelo seguinte sistema de inequações.

$$wst_{1,n,j} \leq Wb - Wb \cdot r_j, \forall j \tag{3.41}$$

$$wst_{1,n,j} \leq wsta_{n,j}, \forall j \tag{3.42}$$

$$wst_{1,n,j} \geq wsta_{n,j} - Wb \cdot r_j, \forall j \tag{3.43}$$

$$wst_{1,n,j} \geq 0, \forall j \tag{3.44}$$

$$wst_{2,n,j} \leq Wb \cdot r_j, \forall j \tag{3.45}$$

$$wst_{2,n,j} \leq wsta_{n,j}, \forall j \tag{3.46}$$

$$wst_{2,n,j} \geq wsta_{n,j} - Wb + Wb \cdot r_j, \forall j \tag{3.47}$$

$$wst_{2,n,j} \geq 0, \forall j \tag{3.48}$$

$$w_{n,j} = wst_{1,n,j} + wst_{2,n,j}, \forall j \tag{3.49}$$

A próxima restrição que precisa ser linearizada é a restrição (3.8). Nela, há, além da multiplicação de variáveis, a função *max*, que é uma função não linear.

Para transformar essa função em uma função linear, primeiro ela será transformada em uma multiplicação de variáveis. Basicamente, uma variável binária *hbga* será usada para informar quem é a maior alturas dos retângulos da lista abaixo, se $h_{i+1,j}$ ou $h_{i+1,j+1}$. Além disso, restrições extras serão adicionadas para garantir que a soma dessas duas variáveis multiplicadas, uma por *hbga* e outra por $1 - hbga$ sempre sejam maiores que as duas variáveis. Desse modo, a função $max(h_{i+1,j}, h_{i+1,j+1})$ será substituída pelas sistema de inequações.

$$hstp_{i,j} = (1 - hbga_{i,j}) \cdot h_{i+1,j} + hbga_{i,j} \cdot h_{i+1,j+1}, \forall i < n, j \leq i \tag{3.50}$$

$$hstp_{i,j} \geq h_{i+1,j}, \forall i < n, j \leq i \tag{3.51}$$

$$hstp_{i,j} \geq h_{i+1,j+1}, \forall i < n, j \leq i \tag{3.52}$$

Feito isso, o método descrito anteriormente pode ser aplicado sobre as duas multiplicações de variáveis que ocorrem na equação (3.50) de modo que ela será substituída pelo seguinte conjunto de restrições.

$$hsta_{i,j} \leq 2 \cdot Hb - 2 \cdot Hb \cdot hbga_{i,j}, \forall i < n, j \leq i \tag{3.53}$$

$$hsta_{i,j} \leq h_{i+1,j}, \forall i < n, j \leq i \tag{3.54}$$

$$hsta_{i,j} \geq 0, \forall i < n, j \leq i \tag{3.55}$$

$$hsta_{i,j} \geq h_{i+1,j} - 2 \cdot Hb \cdot hbga_{i,j}, \forall i < n, j \leq i \tag{3.56}$$

$$hstb_{i,j} \leq 2 \cdot Hb \cdot hbga_{i,j}, \forall i < n, j \leq i \tag{3.57}$$

$$hstb_{i,j} \leq h_{i+1,j+1}, \forall i < n, j \leq i \tag{3.58}$$

$$hstb_{i,j} \geq 0, \forall i < n, j \leq i \tag{3.59}$$

$$hstb_{i,j} \geq h_{i+1,j} - 2 \cdot Hb + 2 \cdot Hb \cdot hbga_{i,j}, \forall i < n, j \leq i \tag{3.60}$$

$$hstp_{i,j} = hsta_{i,j} + hstb_{i,j}, \forall i < n, j \leq i \tag{3.61}$$

É importante notar que, neste caso, $2 \cdot Hb$ foi utilizado no lugar de M e não apenas Hb, como foi na linearização anterior. Isso foi feito porque há quatro possibilidades de junção ou cópia dos retângulos da lista inferior $(i + 1)$ para a lista atual (i) e se delas se mostrar inviável, todas elas seriam descartadas nesse momento, mesmo que uma delas fosse viável. Desse modo, essa folga foi acrescida para permitir tamanhos além do tamanho da placa nas variáveis auxiliares, porém nas variáveis principais das alturas e larguras dos retângulos, a restrição de ser menor ou igual as respectivas dimensões da placa permanecem.

Analogamente, a mesma coisa pode ser feita com a restrição (3.14), gerando o seguinte conjunto de inequações.

$$wsta_{i,j} \leq 2 \cdot Wb - 2 \cdot Wb \cdot wbga_{i,j}, \forall i < n, j \leq i \tag{3.62}$$

$$wsta_{i,j} \leq w_{i+1,j}, \forall i < n, j \leq i \tag{3.63}$$

$$wsta_{i,j} \geq 0, \forall i < n, j \leq i \tag{3.64}$$

$$wsta_{i,j} \geq w_{i+1,j} - 2 \cdot Wb \cdot wbga_{i,j}, \forall i < n, j \leq i \tag{3.65}$$

$$wstb_{i,j} \leq 2 \cdot Wb \cdot wbga_{i,j}, \forall i < n, j \leq i \tag{3.66}$$

$$wstb_{i,j} \leq w_{i+1,j+1}, \forall i < n, j \leq i \tag{3.67}$$

$$wstb_{i,j} \geq 0, \forall i < n, j \leq i \tag{3.68}$$

$$wstb_{i,j} \geq w_{i+1,j} - 2 \cdot Wb + 2 \cdot Wb \cdot wbga_{i,j}, \forall i < n, j \leq i \tag{3.69}$$

$$wstp_{i,j} = wsta_{i,j} + wstb_{i,j}, \forall i < n, j \leq i \tag{3.70}$$

$$wstp_{i,j} \geq w_{i+1,j}, \forall i < n, j \leq i \tag{3.71}$$

$$wstp_{i,j} \geq w_{i+1,j+1}, \forall i < n, j \leq i \tag{3.72}$$

Assim, as restrições (3.8) e (3.14) serão reescritas como

$$hst_{1,i,j} = y_{i,j} \cdot v_i \cdot hstp_{i,j}, \forall i < n, j \leq i \tag{3.73}$$

$$wst_{2,i,j} = y_{i,j} \cdot (1 - v_i) \cdot wstp_{i,j} \forall i < n, j \leq i \tag{3.74}$$

Visto que $hstp_{i,j}$ e $wtsp_{i,j}$ são as variáveis que recebem o resultado da função *max*, convertida no sistema de inequações.

Por fim, as multiplicações das restrições (3.9), (3.10), (3.11), (3.13), (3.15) e (3.16), além das equações (3.73) e (3.74) que substituem as restrições (3.8) e (3.14) precisam ser linearizadas. O método é exatamente o mesmo da linearização da multiplicação de uma variável binária e uma linear. As únicas diferenças estão na quantidade de inequações, visto que uma inequação a mais será criada para a segunda variável binária e inequação (3.31), onde haverão duas variáveis binárias e a constante M será multiplicada por 2.

O conjunto de inequações para substituir a multiplicação entre duas variáveis binárias e uma real positiva ficaria da seguinte forma.

$$mult = b_1 \cdot b_2 \cdot l \tag{3.75}$$

$$mult \leq b_1 \cdot M \tag{3.76}$$

$$mult \leq b_1 \cdot M \tag{3.77}$$

$$mult \leq l \tag{3.78}$$

$$mult \geq l - 2 \cdot M + M \cdot b_1 + M \cdot b_2 \tag{3.79}$$

$$mult \geq 0 \tag{3.80}$$

Assim, as restrições (3.9), (3.10), (3.11), (3.13), (3.15), (3.16), (3.73) e (3.74) seriam reescritas da seguinte forma. Primeiro a restrição (3.73) que substitui a (3.8).

$$hst_{1,i,j} \leq 2 \cdot Hb \cdot y_{i,j}, \forall i < n, j \leq i \tag{3.81}$$

$$hst_{1,i,j} \leq 2 \cdot Hb \cdot v_i, \forall i < n, j \leq i \tag{3.82}$$

$$hst_{1,i,j} \leq hstp_{i,j}, \forall i < n, j \leq i \tag{3.83}$$

$$hst_{1,i,j} \geq hstp_{i,j} - 4 \cdot Hb + 2 \cdot Hb \cdot y_{i,j} + 2 \cdot Hb \cdot v_i, \forall i < n, j \leq i \tag{3.84}$$

$$hst_{1,i,j} \geq 0, \forall i < n, j \leq i \tag{3.85}$$

A restrição (3.9) ficará da seguinte forma.

$$hst_{2,i,j} \leq 2 \cdot Hb \cdot y_{i,j}, \forall i < n, j \leq i \tag{3.86}$$

$$hst_{2,i,j} \leq 2 \cdot Hb - 2 \cdot Hb \cdot v_i, \forall i < n, j \leq i \tag{3.87}$$

$$hst_{2,i,j} \leq h_{i+1,j} + h_{i+1,j+1}, \forall i < n, j \leq i \tag{3.88}$$

$$hst_{2,i,j} \geq (h_{i+1,j} + h_{i+1,j+1}) - 2 \cdot Hb + 2 \cdot Hb \cdot y_{i,j} - 2 \cdot Hb \cdot v_i, \forall i < n, j \leq i \tag{3.89}$$

$$hst_{2,i,j} \geq 0, \forall i < n, j \leq i \tag{3.90}$$

A restrição (3.10) ficará da seguinte forma.

$$hst_{3,i,j} \leq 2 \cdot Hb - 2 \cdot Hb \cdot y_{i,j}, \forall i < n, j \leq i \tag{3.91}$$

$$hst_{3,i,j} \leq 2 \cdot Hb - 2 \cdot Hb \cdot a_{i,j}, \forall i < n, j \leq i \tag{3.92}$$

$$hst_{3,i,j} \leq h_{i+1,j}, \forall i < n, j \leq i \tag{3.93}$$

$$hst_{3,i,j} \geq h_{i+1,j} - 2 \cdot Hb \cdot y_{i,j} - 2 \cdot Hb \cdot v_i, \forall i < n, j \leq i \tag{3.94}$$

$$hst_{3,i,j} \geq 0, \forall i < n, j \leq i \tag{3.95}$$

A restrição (3.11) ficará da seguinte forma.

$$hst_{4,i,j} \leq 2 \cdot Hb - 2 \cdot Hb \cdot y_{i,j}, \forall i < n, j \leq i \tag{3.96}$$

$$hst_{4,i,j} \leq 2 \cdot Hb \cdot a_{i,j}, \forall i < n, j \leq i \tag{3.97}$$

$$hst_{4,i,j} \leq h_{i+1,j+1}, \forall i < n, j \leq i \tag{3.98}$$

$$hst_{4,i,j} \geq h_{i+1,j+1} + 2 \cdot Hb - 2 \cdot Hb \cdot y_{i,j} + 2 \cdot Hb \cdot v_i, \forall i < n, j \leq i \tag{3.99}$$

$$hst_{4,i,j} \geq 0, \forall i < n, j \leq i \tag{3.100}$$

A restrição (3.13) ficará da seguinte forma.

$$wst_{1,i,j} \leq 2 \cdot Wb \cdot y_{i,j}, \forall i < n, j \leq i \tag{3.101}$$

$$wst_{1,i,j} \leq 2 \cdot Wb \cdot v_i, \forall i < n, j \leq i \tag{3.102}$$

$$wst_{1,i,j} \leq w_{i+1,j} + w_{i+1,j+1}, \forall i < n, j \leq i \tag{3.103}$$

$$wst_{1,i,j} \geq (w_{i+1,j} + w_{i+1,j+1}) - 4 \cdot Wb + 2 \cdot Wb \cdot y_{i,j} + 2 \cdot Wb \cdot v_i, \forall i < n, j \leq i \tag{3.104}$$

$$wst_{1,i,j} \geq 0, \forall i < n, j \leq i \tag{3.105}$$

A restrição (3.74) que substitui a (3.14) ficará da seguinte forma

$$wst_{2,i,j} \leq 2 \cdot Wb \cdot y_{i,j}, \forall i < n, j \leq i \tag{3.106}$$

$$wst_{2,i,j} \leq 2 \cdot Wb - 2 \cdot Wb \cdot v_i, \forall i < n, j \leq i \tag{3.107}$$

$$wst_{2,i,j} \leq wstp_{i,j}, \forall i < n, j \leq i \tag{3.108}$$

$$wst_{2,i,j} \geq wstp_{i,j} - 2 \cdot Wb + 2 \cdot Wb \cdot y_{i,j} - 2 \cdot Wb \cdot v_i, \forall i < n, j \leq i \tag{3.109}$$

$$wst_{2,i,j} \geq 0, \forall i < n, j \leq i \tag{3.110}$$

A restrição (3.15) ficará da seguinte forma.

$$wst_{3,i,j} \leq 2 \cdot Wb - 2 \cdot Wb \cdot y_{i,j}, \forall i < n, j \leq i \tag{3.111}$$

$$wst_{3,i,j} \leq 2 \cdot Wb - 2 \cdot Wb \cdot a_{i,j}, \forall i < n, j \leq i \tag{3.112}$$

$$wst_{3,i,j} \leq w_{i+1,j}, \forall i < n, j \leq i \tag{3.113}$$

$$wst_{3,i,j} \geq w_{i+1,j} - 2 \cdot Wb \cdot y_{i,j} - 2 \cdot Wb \cdot v_i, \forall i < n, j \leq i \tag{3.114}$$

$$wst_{3,i,j} \geq 0, \forall i < n, j \leq i \tag{3.115}$$

A restrição (3.16) ficará da seguinte forma.

$$wst_{4,i,j} \leq 2 \cdot Wb - 2 \cdot Wb \cdot y_{i,j}, \forall i < n, j \leq i \tag{3.116}$$

$$wst_{4,i,j} \leq 2 \cdot Wb \cdot a_{i,j}, \forall i < n, j \leq i \tag{3.117}$$

$$wst_{4,i,j} \leq w_{i+1,j+1}, \forall i < n, j \leq i \tag{3.118}$$

$$wst_{4,i,j} \geq w_{i+1,j+1} + 2 \cdot Wb - 2 \cdot Wb \cdot y_{i,j} + 2 \cdot Wb \cdot v_i, \forall i < n, j \leq i \tag{3.119}$$

$$wst_{4,i,j} \geq 0, \forall i < n, j \leq i \tag{3.120}$$

3.1.3 O modelo matemático linear

Como mencionado anteriormente, o problema de corte bidimensional guilhotinado em sua variante do problema de corte e estoque foi abordado como um problema compostos por várias listas, nas quais, dois retângulos da lista atual serão unidos em um retângulo da lista seguinte ou, caso a união não seja possível, o último retângulo da lista atual não integrará a lista seguinte. Desse modo, a primeira lista conterá todos os itens que compões o problema e as listas seguintes terão sempre um retângulo a menos até que reste apenas um retângulo.

Cada união bem sucedida reduzirá o número total de placas necessárias para acomodar todos os itens, visto que elas implicam que os dois retângulos unidos na lista atual cabem em um única placa. Cada retângulo da primeira lista corresponderá aos itens do problema e a ordem em que esses itens estão dispostos nessa lista afetará a união ou a eliminação dos retângulos, tanto da lista inicial quanto das seguintes.

Cada retângulo eliminado em cada lista ou ocupará uma placa, de modo a não caber mais nenhum retângulo (caso a solução seja ótima), ou existirá uma outra ordem de disposição dos itens na lista inicial em que esse retângulo eliminado seja uma placa completa.

A primeira lista conterá os itens do problema, assim não existirão cortes nessa lista. A partir da segunda lista haverá, no máximo, um corte de modo que o retângulo dessa lista poderá ser uma dessas quatro situações:

1. O retângulo é um corte vertical, logo a altura será a maior das alturas dos dois retângulos abaixo e a largura será a soma das larguras deles;
2. O retângulo é um corte horizontal, logo a altura será a soma das alturas dos dois retângulos abaixo e a largura será a maior das larguras deles;
3. O retângulo está antes da união (se não há união, ele é considerado antecessor), logo ele é o mesmo retângulo da lista anterior
4. O retângulo está depois da união, logo ele é o retângulo da posição seguinte da lista anterior

Dada essas características, podemos definir o modelo matemático linear do problema de corte bidimensional guilhotinado da seguinte forma:

$$max \sum_{i=1}^{n-1} \sum_{j=1}^{i} y_{i,j} \tag{3.121}$$

Sujeito à:

$$\sum_{j=1}^{n-1} y_{i,j} \leq 1, \forall i < n \tag{3.122}$$

$$\sum_{j=1}^{n} x_{i,j} = 1, \forall i \tag{3.123}$$

$$\sum_{i=1}^{n} x_{i,j} = 1, \forall j \tag{3.124}$$

$$hsta_{n,j} = \sum_{i=1}^{n} (x_{i,j} \cdot Hit_i), \forall j \tag{3.125}$$

$$wsta_{n,j} = \sum_{i=1}^{n} (x_{i,j} \cdot Wit_i), \forall j \tag{3.126}$$

$$hst_{1,n,j} \leq Hb - Hb \cdot r_j, \forall j \tag{3.127}$$

$$hst_{1,n,j} \leq hsta_{n,j}, \forall j \tag{3.128}$$

$$hst_{1,n,j} \geq hsta_{n,j} - Hb \cdot r_j, \forall j \tag{3.129}$$

$$hst_{1,n,j} \geq 0, \forall j \tag{3.130}$$

$$hst_{2,n,j} \leq HB \cdot r_j, \forall j \tag{3.131}$$

$$hst_{2,n,j} \leq wsta_{n,j}, \forall j \tag{3.132}$$

$$hst_{2,n,j} \geq wsta_{n,j} - Hb + Hb \cdot r_j, \forall j \tag{3.133}$$

$$hst_{2,n,j} \geq 0, \forall j \tag{3.134}$$

$$h_{n,j} = hst_{1,n,j} + hst_{2,n,j}, \forall j \tag{3.135}$$

$$wst_{1,n,j} \leq Wb - Wb \cdot r_j, \forall j \tag{3.136}$$

$$wst_{1,n,j} \leq wsta_{n,j}, \forall j \tag{3.137}$$

$$wst_{1,n,j} \geq wsta_{n,j} - Wb \cdot r_j, \forall j \tag{3.138}$$

$$wst_{1,n,j} \geq 0, \forall j \tag{3.139}$$

$$wst_{2,n,j} \leq Wb \cdot r_j, \forall j \tag{3.140}$$

$$wst_{2,n,j} \leq wsta_{n,j}, \forall j \tag{3.141}$$

$$wst_{2,n,j} \geq wsta_{n,j} - Wb + Wb \cdot r_j, \forall j \tag{3.142}$$

$$wst_{2,n,j} \geq 0, \forall j \tag{3.143}$$

$$w_{n,j} = wst_{1,n,j} + wst_{2,n,j}, \forall j \tag{3.144}$$

$$a_{i,j} = \sum_{k=1}^{j} y_{i,j}, \forall i < n, j \leq i \tag{3.145}$$

$$hsta_{i,j} \leq 2 \cdot Hb - 2 \cdot Hb \cdot hbga_{i,j}, \forall i < n, j \leq i \tag{3.146}$$

$$hsta_{i,j} \leq h_{i+1,j}, \forall i < n, j \leq i \tag{3.147}$$

$$hsta_{i,j} \geq 0, \forall i < n, j \leq i \tag{3.148}$$

$$hsta_{i,j} \geq h_{i+1,j} - 2 \cdot Hb \cdot hbga_{i,j}, \forall i < n, j \leq i \tag{3.149}$$

$$hstb_{i,j} \leq 2 \cdot Hb \cdot hbga_{i,j}, \forall i < n, j \leq i \tag{3.150}$$

$$hstb_{i,j} \leq h_{i+1,j+1}, \forall i < n, j \leq i \tag{3.151}$$

$$hstb_{i,j} \geq 0, \forall i < n, j \leq i \tag{3.152}$$

$$hstb_{i,j} \geq h_{i+1,j} - 2 \cdot Hb + 2 \cdot Hb \cdot hbga_{i,j}, \forall i < n, j \leq i \tag{3.153}$$

$$hstp_{i,j} = hsta_{i,j} + hstb_{i,j}, \forall i < n, j \leq i \tag{3.154}$$

$$hstp_{i,j} \geq h_{i+1,j}, \forall i < n, j \leq i \tag{3.155}$$

$$hstp_{i,j} \geq h_{i+1,j+1}, \forall i < n, j \leq i \tag{3.156}$$

$$wsta_{i,j} \leq 2 \cdot Wb - 2 \cdot Wb \cdot wbga_{i,j}, \forall i < n, j \leq i \tag{3.157}$$

$$wsta_{i,j} \leq w_{i+1,j}, \forall i < n, j \leq i \tag{3.158}$$

$$wsta_{i,j} \geq 0, \forall i < n, j \leq i \tag{3.159}$$

$$wsta_{i,j} \geq w_{i+1,j} - 2 \cdot Wb \cdot wbga_{i,j}, \forall i < n, j \leq i \tag{3.160}$$

$$wstb_{i,j} \leq 2 \cdot Wb \cdot wbga_{i,j}, \forall i < n, j \leq i \tag{3.161}$$

$$wstb_{i,j} \leq w_{i+1,j+1}, \forall i < n, j \leq i \tag{3.162}$$

$$wstb_{i,j} \geq 0, \forall i < n, j \leq i \tag{3.163}$$

$$wstb_{i,j} \geq w_{i+1,j} - 2 \cdot Wb + 2 \cdot Wb \cdot wbga_{i,j}, \forall i < n, j \leq i \tag{3.164}$$

$$wstp_{i,j} = wsta_{i,j} + wstb_{i,j}, \forall i < n, j \leq i \tag{3.165}$$

$$wstp_{i,j} \geq w_{i+1,j}, \forall i < n, j \leq i \tag{3.166}$$

$$wstp_{i,j} \geq w_{i+1,j+1}, \forall i < n, j \leq i \tag{3.167}$$

$$hst_{1,i,j} \leq 2 \cdot Hb \cdot y_{i,j}, \forall i < n, j \leq i \tag{3.168}$$

$$hst_{1,i,j} \leq 2 \cdot Hb \cdot v_i, \forall i < n, j \leq i \tag{3.169}$$

$$hst_{1,i,j} \leq hstp_{i,j}, \forall i < n, j \leq i \tag{3.170}$$

$$hst_{1,i,j} \geq hstp_{i,j} - 4 \cdot Hb + 2 \cdot Hb \cdot y_{i,j} + 2 \cdot Hb \cdot v_i, \forall i < n, j \leq i \tag{3.171}$$

$$hst_{1,i,j} \geq 0, \forall i < n, j \leq i \tag{3.172}$$

$$hst_{2,i,j} \leq 2 \cdot Hb \cdot y_{i,j}, \forall i < n, j \leq i \tag{3.173}$$

$$hst_{2,i,j} \leq 2 \cdot Hb - 2 \cdot Hb \cdot v_i, \forall i < n, j \leq i \tag{3.174}$$

$$hst_{2,i,j} \leq h_{i+1,j} + h_{i+1,j+1}, \forall i < n, j \leq i \tag{3.175}$$

$$hst_{2,i,j} \geq (h_{i+1,j} + h_{i+1,j+1}) - 2 \cdot Hb + 2 \cdot Hb \cdot y_{i,j} - 2 \cdot Hb \cdot v_i, \forall i < n, j \leq i \tag{3.176}$$

$$hst_{2,i,j} \geq 0, \forall i < n, j \leq i \tag{3.177}$$

$$hst_{3,i,j} \leq 2 \cdot Hb - 2 \cdot Hb \cdot y_{i,j}, \forall i < n, j \leq i \tag{3.178}$$

$$hst_{3,i,j} \leq 2 \cdot Hb - 2 \cdot Hb \cdot a_{i,j}, \forall i < n, j \leq i \tag{3.179}$$

$$hst_{3,i,j} \leq h_{i+1,j}, \forall i < n, j \leq i \tag{3.180}$$

$$hst_{3,i,j} \geq h_{i+1,j} - 2 \cdot Hb \cdot y_{i,j} - 2 \cdot Hb \cdot v_i, \forall i < n, j \leq i \tag{3.181}$$

$$hst_{3,i,j} \geq 0, \forall i < n, j \leq i \tag{3.182}$$

$$hst_{4,i,j} \leq 2 \cdot Hb - 2 \cdot Hb \cdot y_{i,j}, \forall i < n, j \leq i \tag{3.183}$$

$$hst_{4,i,j} \leq 2 \cdot Hb \cdot a_{i,j}, \forall i < n, j \leq i \tag{3.184}$$

$$hst_{4,i,j} \leq h_{i+1,j+1}, \forall i < n, j \leq i \tag{3.185}$$

$$hst_{4,i,j} \geq h_{i+1,j+1} + 2 \cdot Hb - 2 \cdot Hb \cdot y_{i,j} + 2 \cdot Hb \cdot v_i, \forall i < n, j \leq i \tag{3.186}$$

$$hst_{4,i,j} \geq 0, \forall i < n, j \leq i \tag{3.187}$$

$$h_{i,j} = hst_{1,i,j} + hst_{2,i,j} + hst_{3,i,j} + hst_{4,i,j}, \forall i < n, j \leq i \tag{3.188}$$

$$wst_{1,i,j} \leq 2 \cdot Wb \cdot y_{i,j}, \forall i < n, j \leq i \tag{3.189}$$

$$wst_{1,i,j} \leq 2 \cdot Wb \cdot v_i, \forall i < n, j \leq i \tag{3.190}$$

$$wst_{1,i,j} \leq w_{i+1,j} + w_{i+1,j+1}, \forall i < n, j \leq i \tag{3.191}$$

$$wst_{1,i,j} \geq (w_{i+1,j} + w_{i+1,j+1}) - 4 \cdot Wb + 2 \cdot Wb \cdot y_{i,j} + 2 \cdot Wb \cdot v_i, \forall i < n, j \leq i \tag{3.192}$$

$$wst_{1,i,j} \geq 0, \forall i < n, j \leq i \tag{3.193}$$

$$wst_{2,i,j} \leq 2 \cdot Wb \cdot y_{i,j}, \forall i < n, j \leq i \tag{3.194}$$

$$wst_{2,i,j} \leq 2 \cdot Wb - 2 \cdot Wb \cdot v_i, \forall i < n, j \leq i \tag{3.195}$$

$$wst_{2,i,j} \leq wstp_{i,j}, \forall i < n, j \leq i \tag{3.196}$$

$$wst_{2,i,j} \geq wstp_{i,j} - 2 \cdot Wb + 2 \cdot Wb \cdot y_{i,j} - 2 \cdot Wb \cdot v_i, \forall i < n, j \leq i \tag{3.197}$$

$$wst_{2,i,j} \geq 0, \forall i < n, j \leq i \tag{3.198}$$

$$wst_{3,i,j} \leq 2 \cdot Wb - 2 \cdot Wb \cdot y_{i,j}, \forall i < n, j \leq i \tag{3.199}$$

$$wst_{3,i,j} \leq 2 \cdot Wb - 2 \cdot Wb \cdot a_{i,j}, \forall i < n, j \leq i \tag{3.200}$$

$$wst_{3,i,j} \leq w_{i+1,j}, \forall i < n, j \leq i \tag{3.201}$$

$$wst_{3,i,j} \geq w_{i+1,j} - 2 \cdot Wb \cdot y_{i,j} - 2 \cdot Wb \cdot v_i, \forall i < n, j \leq i \tag{3.202}$$

$$wst_{3,i,j} \geq 0, \forall i < n, j \leq i \tag{3.203}$$

$$wst_{4,i,j} \leq 2 \cdot Wb - 2 \cdot Wb \cdot y_{i,j}, \forall i < n, j \leq i \tag{3.204}$$

$$wst_{4,i,j} \leq 2 \cdot Wb \cdot a_{i,j}, \forall i < n, j \leq i \tag{3.205}$$

$$wst_{4,i,j} \leq w_{i+1,j+1}, \forall i < n, j \leq i \tag{3.206}$$

$$wst_{4,i,j} \geq w_{i+1,j+1} + 2 \cdot Wb - 2 \cdot Wb \cdot y_{i,j} + 2 \cdot Wb \cdot v_i, \forall i < n, j \leq i \tag{3.207}$$

$$wst_{4,i,j} \geq 0, \forall i < n, j \leq i \tag{3.208}$$

$$w_{i,j} = wst_{1,i,j} + wst_{2,i,j} + wst_{3,i,j} + wst_{4,i,j}, \forall i < n, j \leq i \tag{3.209}$$

$$0 < h_{i,j} \leq Hb, \forall i, j \leq i \tag{3.210}$$

$$0 < w_{i,j} \leq Wb, \forall i, j \leq i \tag{3.211}$$

$$h_{i,j}, hst_{s,i,j}, hsta_{i,j} \in R^+, \forall s \in \{1,2\}, i = n, j \leq i \tag{3.212}$$

$$w_{i,j}, wst_{s,i,j}, wsta_{i,j} \in R^+, \forall s \in \{1,2\}, i = n, j \leq i \tag{3.213}$$

$$h_{i,j}, hst_{s,i,j}, hsta_{i,j}, hstb_{i,j}, hstp_{i,j} \in R^+, \forall s \in \{1,2,3,4\}, i \leq n, j \leq i \tag{3.214}$$

$$w_{i,j}, wst_{s,i,j}, wsta_{i,j}, wstb_{i,j}, wstp_{i,j} \in R^+, \forall s \in \{1,2,3,4\}, i \leq n, j \leq i \tag{3.215}$$

$$x_{i,j}, r_j \in \{0,1\}, \forall i, j \leq i \tag{3.216}$$

$$y_{i,j}, a_{i,j}, v_i \in \{0,1\}, \forall i < n, j \leq i \tag{3.217}$$

A equação (3.121) tentará maximizar o número de cortes que estão sendo representadas pela variável $y_{i,j}$, como só há cortes a partir da segunda lista, o índice i não atingirá o número total de retângulos (n) e o índice j não ultrapassará o índice i, formando assim uma matriz triangular.

Como em cada lista ocorrerá, no máximo, um corte, a restrição (3.122) forçará com que a soma de todos os cortes ($y_{i,j} = 1$) existentes em cada lista seja no máximo 1, para todas as listas do problema, exceto a lista n ($\forall i < n$).

A ordem dos itens dispostos na primeira lista (lista n) será definida pela variável $x_{i,j}$ de modo que se o item i estiver na posição j da lista n, então $x_{i,j} = 1$, senão $x_{i,j} = 0$. A restrição (3.123) garante que um item i apareça apenas em uma posição j e a restrição (3.124) garante que cada posição j tenha apenas um item i.

A altura e a largura do retângulo j da lista n será a altura e a largura do item alocado àquela posição, todavia, se este item estiver rotacionado, a altura e a largura se inverterão. Então uma altura e larguras temporárias ($hsta_{n,j}$ e $wsta_{n,j}$, respectivamente) armazenarão a altura e a largura dos itens ali alocados antes da aplicação das rotações. A restrição (3.125) define que a altura temporária $hsta_{n,j}$ será o somatório das alturas dos itens (Hit_i) multiplicadas pelas variáveis de posição ($x_{i,j}$). Como a variável de posição só será 1 se o item i estiver na posição j, as alturas das demais posições serão zeradas, mantendo apenas a altura do item alocado àquela posição. A restrição (3.126) faz a mesma coisa, porém para a largura.

Com as alturas e larguras temporárias definidas, as dimensões do retângulo j da lista n ficam dependendo apenas se o item ali alocado está rotacionado ou não. A variável r_j define isso, se seu valor for 0, o item não está rotacionado, se for 1, o item está rotacionado.

Para realizar essa decisão na forma de um "*if...else*", uma multiplicação por $1 - r_j$ ou r_j será realizada em cada situação. Caso a primeira situação (item não rotacionado ou $r_j = 0$) seja verdadeira, a altura será $hsta_{n,j} \cdot (1 - r_j)$, caso a segunda situação (item rotacionado ou $r_j = 1$) a altura será $wsta_{n,j} \cdot r_j$. As restrições (3.127), (3.128), (3.129) e (3.130) realizam a primeira multiplicação na forma de inequações lineares e armazena seu resultado em $hst_{1,n,j}$, enquanto as restrições (3.131), (3.132), (3.133) e (3.134) realizam a segunda multiplicação e armazena seu resultado em $hst_{2,n,j}$. Os valores $1 - r_j$ e r_j são expressões binárias excludentes, de forma que se uma for verdadeira a outra será falsa e seu valor zerado. Assim, ($h_{n,j}$), que representa a altura do retângulo j da lista n será simplesmente a soma das duas situações ($hst_{1,n,j}$ e $hst_{2,n,j}$), representada na restrição (3.135).

O mesmo ocorre com a largura, a primeira situação está descrita nas restrições (3.136), (3.137), (3.138) e (3.139) e representada pela variável $wst_{1,n,j}$. A segunda situação está descrita nas restrições (3.140), (3.141), (3.142) e (3.143) e a variável representa essa situação é $wst_{2,n,j}$. Semelhante ao que acontece com a altura, basta somar as larguras nas duas situações (pois são excludentes) e teremos a largura do retângulo j da lista n (variável $w_{n,j}$). Fato denotado na restrição (3.144).

Após a primeira lista, todas as demais podem assumir até quatro valores provenientes de quatro situações distintas: se é um corte vertical ou horizontal, se não é corte e está antes do corte da lista ou depois do corte da lista. Ser um corte está definido na variável $y_{i,j}$, ser vertical ou horizontal cabe a variável v_i e estar antes ou depois do corte da lista atual está definido na variável $a_{i,j}$. As duas primeiras variáveis são

variáveis de decisão, todavia a variável $a_{i,j}$ é uma variável derivada da variável $y_{i,j}$. A forma de gerar o valor dessa variável está descrita na restrição (3.145), se todos os $y_{i,j}$ da lista atual, até o retângulo atual, forem 0, significa que não há nenhum corte antes e a variável $a_{i,j}$ será igual a 0. Se um $y_{i,j}$ for 1, então há um corte antes e a variável $a_{i,j}$ terá valor 1.

Assim como foi feito com a variável r_j para saber se o retângulo j da lista n estava rotacionado ou não, multiplicações por $y_{i,j}$ e $1 - y_{i,j}$ definirão se há ou não um corte naquele ponto. Havendo um corte, multiplicações por v_i e $1 - v_i$ definirão se o corte é vertical ou não. Não havendo uma união, multiplicações por $a_{i,j}$ e $1 - a_{i,j}$ definirão se o retângulo da lista atual está antes ou depois do corte da lista.

Todavia, antes de calcular as quatro situações é necessário calcular a função *max* para saber qual é a maior dimensão dos dois retângulos da lista seguinte que poderão ser unidos. Esse cálculo é realizado tanto para a altura (quando o corte é vertical) quanto para a largura (quando o corte é horizontal). Quando o corte for vertical, uma variável binária $hbga_{i,j}$, indicando que o retângulo seguinte (retângulo "b") é maior que o retângulo da coluna atual (retângulo "a"), será utilizada. Seu uso se baseará na lógica do "*if...else*" com as multiplicações por $hbga_{i,j}$ e $1 - hbga_{i,j}$ para os casos onde o segundo é maior que o primeiro ou o onde o segundo é maior que o primeiro, respectivamente.

Essas multiplicações são representadas pelas restrições (3.146), (3.147), (3.148) e (3.149) para o caso do primeiro retângulo ser o maior e pelas restrições (3.150), (3.151), (3.152) e (3.153) para o caso do maior ser o segundo retângulo. As variáveis $hsta_{i,j}$ e $hstb_{i,j}$ receberão esse valores respectivamente. Como as expressões são excludentes, a soma dessas variáveis terá a maior altura apenas. Essa soma é guardada na variável $hstp_{i,j}$ na restrição (3.154) e então é comparada com os retângulos cortados (os retângulos j e $j + 1$ da lista $i + 1$) nas restrições (3.155) e (3.156), caso não seja verdadeira, então a variável $hbga_{i,j}$ deve ter seu valor alterado.

A mesma lógica pode ser aplicada a largura, onde as restrições (3.157), (3.158), (3.159) e (3.160) e a variável $wsta_{i,j}$ representam o retângulo "a". As restrições (3.161), (3.162), (3.163) e (3.164) e a variável $wstb_{i,j}$ representam o retângulo "b". As restrições (3.165), (3.166) e (3.167), juntamente com a variável binária $wbga_{i,j}$ garantirão que na variável $wstp_{i,j}$ será armazenada apenas a maior largura entre os dois retângulos cortados.

Com os resultados da função *max* armazenados nas variáveis $hstp_{i,j}$ e $wstp_{i,j}$ para altura e largura é possível calcular as quatro situações descritas no início da sessão. Iniciando pela altura, a primeira situação é quando há um corte e ele é vertical ($y_{i,j} = 1$ e $v_i = 1$). Nessa situação a altura será a maior altura dos itens cortados (ou seja, $hstp_{i,j}$) e as multiplicações entre as variáveis está definida nas restrições (3.168), (3.169), (3.170), (3.171) e (3.172). Seu resultado é armazenado na variável $hst_{1,i,j}$.

A segunda situação é quando há um corte e ele é horizontal ($y_{i,j} = 1$ e $v_i = 0$).

Nesse caso a altura será a soma das alturas dos itens cortados e a multiplicação entre as variáveis está definida nas restrições (3.173), (3.174), (3.175), (3.176) e (3.177). Seu resultado é armazenado na variável $hst_{2,i,j}$.

A terceira situação é quando não há corte e este retângulo está antes do corte da lista atual ($y_{i,j} = 0$ e $a_{i,j} = 0$). Assim, a altura será a altura do retângulo diretamente abaixo e a multiplicação entre as variáveis está definida nas restrições (3.178), (3.179), (3.180), (3.181) e (3.182). Seu resultado é armazenado na variável $hst_{3,i,j}$.

Por fim, a quarta situação é quando não há corte e o retângulo está depois do corte da lista atual ($y_{i,j} = 0$ e $a_{i,j} = 1$). Desse modo, a altura será a altura do retângulo seguinte ao que está abaixo e a multiplicação entre as variáveis está definida nas restrições (3.183), (3.184), (3.185), (3.186) e (3.187). Seu resultado é armazenado na variável $hst_{4,i,j}$.

A altura final nesse caso será a soma dessas quatro situações, visto quando uma for verdadeira, todas as demais serão falsas e sempre haverá uma situação verdadeira. Essa altura será armazenada na variável $h_{i,j}$ e está descrita na restrição (3.188).

Para a largura, a primeira situação está descrita nas restrições (3.189), (3.190), (3.191), (3.192) e (3.193) e armazenada na variável $wst_{1,i,j}$. A segunda situação está descrita nas restrições (3.194), (3.195), (3.196), (3.197) e (3.198) e armazenada na variável $wst_{2,i,j}$. a terceira situação está descrita nas restrições (3.199), (3.200), (3.201), (3.202) e (3.203) e armazenada na variável $wst_{3,i,j}$. E, por fim, a quarta situação está descrita nas restrições (3.204), (3.205), (3.206), (3.207) e (3.208) e armazenada na variável $wst_{4,i,j}$. Sendo todas essas situações unidas na variável $w_{i,j}$ na restrição (3.209).

As restrições (3.210) e (3.211) definem que as alturas e larguras de nenhum dos retângulos pode ser 0 ou ultrapassar as dimensões das placas (*Hb* para altura e *Wb* para largura). As restrições (3.212), (3.213), (3.214), (3.215), (3.216) e (3.217) definem quais os escopos de cada variável.

4 UMA ABORDAGEM EXATA COMPUTACIONAL

Apesar do modelo matemático gerar e garantir a otimalidade das soluções, um algoritmo exato específico para o problema torna o processo, por vezes mais simplificado. Isso porque há características do problema que precisam ser adaptadas ao formalismo matemático e que poderiam ser trabalhadas de outra maneira pelas ferramentas computacionais.

No modelo matemático proposto nesse trabalho, há diversas multiplicações por variáveis binárias e essas foram colocadas como uma forma de tomada de decisão. Essa tomada de decisão matemática poderia ser substituida por simples comparações *"if...else"* das línguagens computacionais e boa parte das restrições geradas para garantir a integridade do modelo sem ferir a linearidade do mesmo poderiam ser substituídas, por exemplo.

A simples substituição das ferramentas matemáticas por ferramentas computacionais não garante que soluções sejam encontradas mais rapidamente, porém, devido à quantidade de multiplicações de variáveis binárias substituindo *"if...else"* (inclusive aninhadas), a função não linear *max* e a matriz $X_{n,n}$ que representa a ordem e que, na computação, poderia ser somente um vetor, existem boas razões para acreditar que uma solução computacional poderia ter resultados em menos tempo que uma solução matemática.

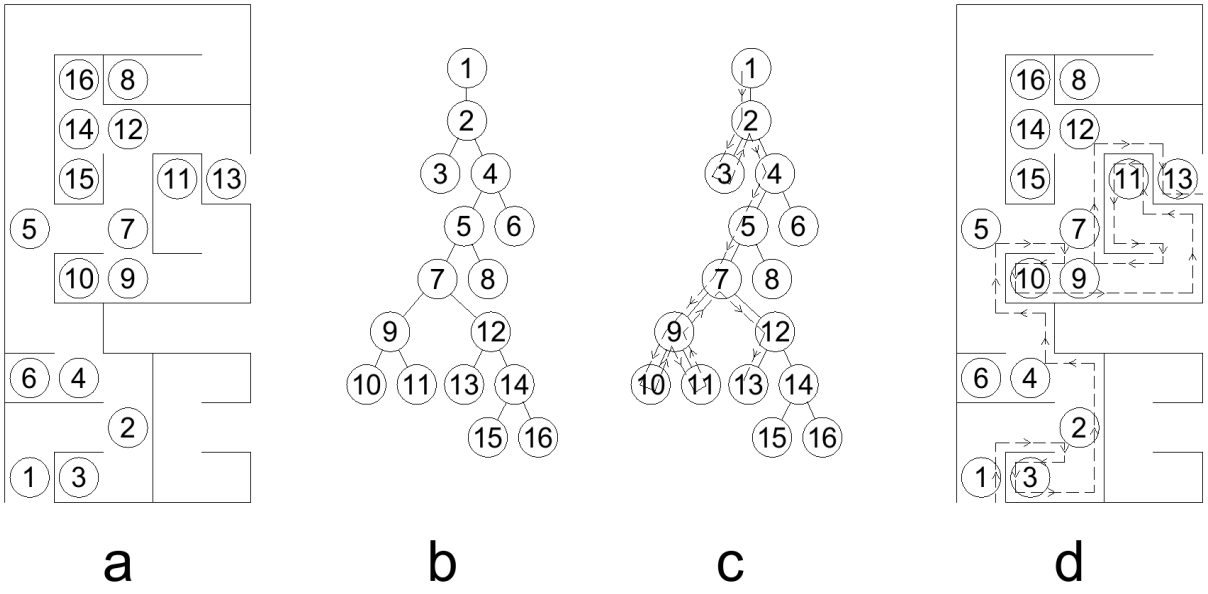
4.1 BUSCA EM PROFUNDIDADE

O primeiro algoritmo exato a ser testado é dos mais antigos, todavia ainda um importante método, o algoritmo de busca em profundidade. É difícil precisar quem propôs o algoritmo, até mesmo pelas datas de estudo do mesmo, entretanto alguns autores como Luo, Yu e Yuan (2016), Young e Etzkorn (2016) e Sadaphule e Shaikh (2016) acreditam ser do matemático francês Charles Pierre Trémaux no século 19, devido às suas investigações sobre uma versão desse método como estratégia para encontrar saídas de labirintos.

A Figura 30 exemplifica a busca em profundidade para encontrar saídas de labirintos. Nela um labirinto (a) é modelado como um grafo (b), sobre esse grafo é aplicado a busca em profundidade (c) e o resultado é aplicado no labirinto original (d).

A busca em profundidade funciona passando sempre ao primeiro vértice adjacente ao vértice atual, indo o mais profundo possível e quando esgotadas as possibilidades de descida daquele vértice, o algoritmo retrocede ao vértice anterior. Esse processo

Figura 30 – Busca em profundidade aplicada a resolução de labirinto.

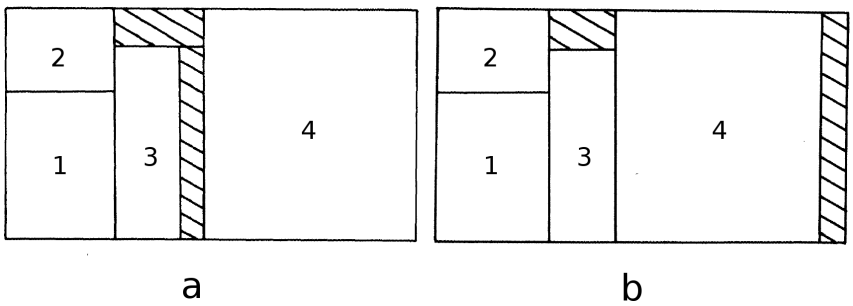


Fonte: Autoria própria

se repete até que o vértice objetivo seja alcançado ou que o algoritmo retorne ao vértice inicial indicando que o objetivo não existe.

Para implementar a busca em profundidade para o problema de corte bidimensional guilhotinado foi usada a abordagem baseada em listas. Essa abordagem possibilita que algumas das redundâncias descritas por Christofides e Whitlock (1977) sejam evitadas, como a questão da normalização e localização dos cortes (Figura 31). Isso acontece porque os arranjos são montados de baixo para cima através de sucessivas uniões, de forma que não se formam espaços entre eles (Figura 31 (a)). A redundância da ordem dos itens dentro de uma união ($a|b = b|a$) é resolvida com a ordenação das operações de uniões entre os itens. Nessa regra, o primeiro item é unido sempre com um item posterior, garantindo que apareça $a|b$, mas não permite $b|a$.

Figura 31 – Normalizacção dos cortes para evitar redundância na geração dos padrões.

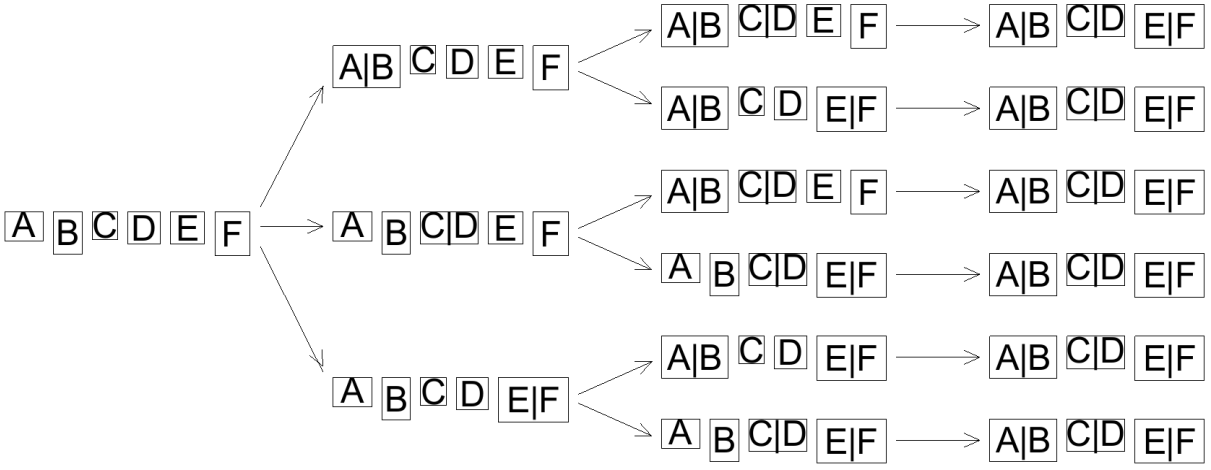


Fonte: (CHRISTOFIDES; WHITLOCK, 1977)

Inicialmente não foi feito tratamento de estados duplicados gerados pela ordem de uniões que não interferem umas nas outras, como por exemplo, unir os itens A e B, C

e D, E e F. Independente da ordem em que são feitas essas uniões, a lista gerada será (A|B), (C|D) e (E|F), todavia, na árvore de decisão, cada união dessas é uma ramificação e várias ramificações irão gerar o mesmo estado, o que é redundante e prejudica a eficiência do algoritmo.

Figura 32 – Redundância causada por diferentes ordens de uniões independentes.



Fonte: Autoria própria

Na Figura 32 é possível ver a situação descrita anteriormente, nela é possível ver que três ramificações, representando quando a união (A|B), (C|D) ou (E|F) ocorre primeiro. Em cada uma delas, mais duas ramificações, explicitando a ordem das duas outras junções. Como pode ser visto ao final, seis estados nessa árvore de decisão são iguais e isso prejudica a busca pela solução ótima. Entretanto a busca em profundidade não é o objetivo deste trabalho e essa falha não afetará muito o algoritmo pretendido.

Semelhante à formulação matemática, o algoritmo de busca em profundidade (algoritmo 1) tentará juntar o maior número de itens possíveis, fazendo junções a partir da lista completa de itens. A cada etapa do algoritmo, dois itens são selecionados (linhas 6 e 7). Uma nova lista de itens é criada sem esses itens (linha 8) e uma pequena lista de uniões possíveis é criada (linha 9) a partir das possibilidades de união entre os itens selecionados. Para cada uma dessas uniões, o algoritmo é chamado recursivamente com a nova lista acrescida dessa união (linha 11). O algoritmo executa até não haver mais uniões possíveis. A variável *nContêineres* é uma variável estática que inicia com o valor total da lista e vai sendo atualizada sempre que uma lista menor for alcançada (linhas 2 e 3) e se uma lista com apenas um item for gerada e como o algoritmo não tentará juntar apenas um item (condição textbfse da linha 5), se isso ocorrer, a variável receberá valor um (linha 16).

As uniões são descritas no algoritmo 2. O algoritmo inicia com uma lista vazia (linha 2) que irá ser preenchida com as possibilidades de união. Para evitar redundâncias, o algoritmo de união rotaciona apenas os itens isolados. Assim, sempre que uma união

Algorithm 1 buscaProfundidade

```
1:  função buscaProfundidade(listaItens)
2:      se listaItens.tamanho < nContêineres então
3:          nContêineres ← listaItens.tamanho          ▷ nContêineres é variável estática
4:      fim se
5:      se listaItens.tamanho > 1 então
6:          para i ← 1 até N − 1 faça
7:              para j ← i + 1 até N faça
8:                  novaListaItens ← listaItens − item[i] − item[j]
9:                  novosItens ← uniõesPossíveis(item[i],item[j])
10:                 para k ← 1 até novosItens.tamanho faça
11:                     buscaProfundidade(novaListaItens + novosItens[k])
12:                 fim para
13:             fim para
14:         fim para
15:     senão
16:         nContêineres ← 1
17:     fim se
18: fim função
```

for feita, o item resultante é marcado para que não possa ser rotacionado no futuro (linha 3). Até oito uniões podem ser feitas, duas com os itens não rotacionados (vertical e horizontal nas linhas 4 e 5), duas se o primeiro item puder ser rotacionado (linhas 7 e 8), duas se o segundo item puder ser rotacionado (linhas 11 e 12) e mais duas se ambos os itens puderem ser rotacionados (linhas 15 e 16). As possibilidades de união que ultrapassarem alguma das dimensões do contêiner são retiradas da lista de possibilidades (linha 20) e essa lista de possibilidades é retornada ao algoritmo 1.

Considerando o algoritmo 1 é fácil perceber que a complexidade assintótica do algoritmo é exponencial, visto que o algoritmo chama a si mesmo passando como entrada uma lista com tamanho $N-1$. Todavia, numa análise mais profunda, não é difícil perceber que a complexidade é maior, pois em cada iteração há dois laços *for* aninhados, sendo um de 1 a $N-1$ e outro do próximo elemento do primeiro *for* até N $\left(\frac{(N-1)\cdot(N-2)}{2}\right)$. Elevando a complexidade do algoritmo para $\frac{(N-1)\cdot(N-2)}{2}!$. O algoritmo 2, apesar de variar conforme a quantidade de rotações permitidas, apenas 2, 4 e 8 possibilidades são permitidas. Como no pior caso, é um valor constante de 8 possibilidades, a ordem final seria $\left(\frac{8\cdot(N-1)\cdot(N-2)}{2}\right)!$ ou $O((4N^2 - 12N + 8)!)$. Simplificando para a ordem mais relevante ficaria $O((N^2)!)$

Apesar da ordem elevada de complexidade, se comparado à quantidade de variáveis do modelo matemático e lembrar que sua complexidade é 2^{nVar} para variáveis binárias e o número de variáveis cresce em ordem quadrática (vide sessão **??**), tem-se uma ordem $O(2^{N^2})$. O que torna o algoritmo e a formulação compatíveis em complexidade. Ficando o desempenho de cada um a cargo das capacidades de eliminar soluções

Algorithm 2 uniõesPossíveis

```
1:  função uniõesPossíveis(itemA, itemB)
2:      novosItens[] ← ∅
3:      itemNovo.permiteRotação ← falso
4:      novosItens ← novosItens + uniãoVertical(itemA, itemB)
5:      novosItens ← novosItens + uniãoHorizontal(itemA, itemB)
6:      se itemA.permiteRotação então
7:          novosItens ← novosItens + uniãoVertical(rot(itemA), itemB)
8:          novosItens ← novosItens + uniãoHorizontal(rot(itemA), itemB)
9:      fim se
10:     se itemB.permiteRotação então
11:         novosItens ← novosItens + uniãoVertical(itemA, rot(itemB))
12:         novosItens ← novosItens + uniãoHorizontal(itemA, rot(itemB))
13:     fim se
14:     se itemA.permiteRotação E itemB.permiteRotação então
15:         novosItens ← novosItens + uniãoVertical(rot(itemA), rot(itemB))
16:         novosItens ← novosItens + uniãoHorizontal(rot(itemA), rot(itemB))
17:     fim se
18:     para i ← novosItens.tamanho até 1 faça
19:         se novosItens[i] ≥ contêiner então
20:             novosItens ← novosItens − novosItens[i]      ▷ Remove itens inválidos
21:         fim se
22:     fim paradevolve novosItens
23: fim função
```

Tabela 1 – Resultado da busca em profundidade para a instância com 4 itens

Instância de 4 itens e contêineres 3x3			
Nº contêineres	Desperdício	LB	tempo
4	27.000	1	0.00013 s
3	18.000	1	0.00015 s
2	9.000	1	0.00016 s
1	0.000	1	0.00020 s
Quantidade de contêineres:			1
Tempo transcorrido:			0.000204 s

inconvenientes dos métodos.

Os resultados do algoritmo de busca em profundidade foram consideravelmente mais rápidos que o modelo matemático, menos de um segundo para as mesmas instâncias, como pode ser visto nas tabelas 1,2 e 3. Mas, apesar da redução do tempo, ainda ficou um tempo considerável para instâncias maiores, além do fato que a busca em profundidade pura não ser a melhor opção para problemas de otimização, visto a existência de métodos que aceleram a busca.

Tabela 2 – Resultado da busca em profundidade para a primeira instância com 5 itens

Instância de 5 itens e contêineres 20x15			
Nº contêineres	Desperdício	LB	tempo
5	1200.000	1	0.00013 s
4	900.000	1	0.00015 s
3	600.000	1	0.00016 s
2	300.000	1	0.00016 s
1	0.000	1	0.00025 s
Quantidade de contêineres:			1
Tempo transcorrido:			0.000258 s

Tabela 3 – Resultado da busca em profundidade para a segunda instância com 5 itens

Instância de 5 itens e contêineres 15x10			
Nº contêineres	Desperdício	LB	tempo
5	450.000	2	0.00014 s
4	300.000	2	0.00016 s
3	150.000	2	0.00017 s
2	0.000	2	0.00018 s
Quantidade de contêineres:			2
Tempo transcorrido:			0.000188 s

4.2 GREEDY BRANCH AND BOUND

A abordagem escolhida para acelerar as buscas do algoritmo é usando *Branch and Bound*. Apresentado ao mundo por Land e Doig (1960), o algoritmo *Branch and Bound* consiste em ramificar, a partir de um ponto inicial, uma árvore de decisões e limitanto seu crescimento por limites superiores e inferiores que seus nós podem assumir. Quando uma solução intermediária em uma ramificação ultrapassa o limite superior, ela é descartada, pois ela não poderá atingir o valor ótimo.

No presente trabalho, o ponto de partida é a lista completa de itens e as ramificações são as possíveis junções. O desperdício de cada lista foi usado como limitante das ramificações. Basicamente, foi acrescida à busca em profundidade da sessão anterior, um limiar de corte para evitar "descer" em ramos da árvore de decisão que não podem conter soluções melhores que as já encontradas (algoritmo 3, linhas 2 e 3).

A lógica por trás desse limitante é a seguinte, as perdas totais de um arranjo é a soma da área dos contêineres necessários para comportar a lista completa de itens, menos a área desses itens. A cada iteração, dois itens são unidos e um contêiner a menos é necessário. Todavia, chega um momento no qual um item gerado por sucessivas uniões não pode mais ser unido a nenhum outro item, por menor que seja o segundo.

Nesse momento, esse item é descartado e suas perdas não poderão mais ser reduzidas. Se em algum momento esse desperdício ultrapassar o do melhor arranjo já encontrado, essa solução parcial não poderá resultar em soluções melhores que as já encontradas e a ramificação é interrompida.

Algorithm 3 Branch and Bound

```
1: função buscaProfundidade(listaItens, perdas)
2:     se perdas ≤ limiar então
3:         devolve listaItens.tamanho
4:     fim se
5:     se listaItens.tamanho < nContêineres então
6:         nContêineres ← listaItens.tamanho        ▷ nContêineres é variável estática
7:         limiar ← perdas + listaItens.perdas      ▷ limiar é variável estática
8:     fim se
9:     se listaItens.tamanho > 1 então
10:        para i ← 1 até N − 1 faça
11:            para j ← i + 1 até N faça
12:                novaListaItens ← listaItens − item[i] − item[j]
13:                menorItem ← buscaMenorItem(novaListaItens)
14:                novosItens ← uniõesPossíveis(item[i],item[j])
15:                para k ← 1 até novosItens.tamanho faça
16:                    se união(novosItens[k], menorItem) ≤ contêiner então
17:                        buscaProfundidade(novaListaItens + novosItens[k], perdas)
18:                    senão
19:                        novasPerdas ← perdas + novosItens[k].perdas
20:                        buscaProfundidade(novaListaItens, novasPerdas)
21:                    fim se
22:                fim para
23:            fim para
24:        fim para
25:    senão
26:        nContêineres ← 1
27:    fim se
28: fim função
```

Essa estratégia não acelerou o tempo de execução de maneira relevante no cenário testado (tabela 4), provavelmente porque nenhum boa solução foi encontrada nas primeiras ramificações. Nada impede, por exemplo, que em uma instância diferente ou mesmo com os itens ordenados de forma diferente, o resultado possa ser radicalmente melhor, todavia não há informações de qual seriam essas condições, para as quais soluções melhores possam ser encontradas.

Essa indefinição levou a segunda atualização do algoritmo: colocar gula. A ideia não é nova, algo semelhante foi utilizada em Yeh (2008). A lógica da abordagem apresentada neste trabalho baseia-se nas sobras internas de cada junção. Na sessão 3.1.1 deste trabalho é explicada a formação da sobra complementar (Figura 23). Essa

Tabela 4 – comparativo da busca em profundidade com *branch and bound* para a instância com 9 itens

Instância de 9 itens e contêineres 30x20

Nº contêineres	Desperdício	LB	tempo	
			B. em prof	*Branch and bound*
9	4800.000	1	0.00019 s	0.00019 s
8	4200.000	1	0.00021 s	0.00021 s
7	3600.000	1	0.00022 s	0.00022 s
6	3000.000	1	0.00023 s	0.00023 s
5	2400.000	1	0.00023 s	0.00023 s
4	1800.000	1	0.00024 s	0.00024 s
3	1200.000	1	0.00025 s	0.00025 s
2	600.000	1	0.00026 s	0.00026 s
1	0.000	1	1082.22192 s	1081.30190 s
Quantidade de contêineres:			1	1
Tempo transcorrido:			1082.222046 s	1081.302012 s

sobra complementar não é reaproveitada nas junções futuras, mesmo que caiba um item dentro, isso porque, se o aproveitamento dessa sobra estiver na solução ótima, haverá outra ordem de junções onde ela não é formada (Figura 24).

E é sobre essa ordem de junções que trabalha o *Greedy Branch and Bound*. Ao invés de descer na árvore de decisão assim que uma ramificação é encontrada, o algoritmo adia a decisão até que todos os ramos sejam analisados. Esses ramos são ordenados pela área dessas sobras complementares de modo que as junções que gerem menos sobras sejam preferidas em relação as demais. Essa estratégia tenta reduzir o limiar de corte mais rapidamente, forçando o descarte prematuro de soluções parciais que não levem a uma solução ótima.

O algoritmo *Greedy Branch and Bound* (algoritmo 4) inicia com a lista completa de itens e realiza as uniões possíveis. Diferente do que ocorre com o *Branch and Bound* simples, quando uma união é testada, o algoritmo não é chamado recursivamente de imediato. Neste caso, as uniões são armazenadas em uma lista de uniões ordenada pela área da sobra complementar gerada em cada união (linha 17). A gula é aplicada justamente nessa ordem, pois o algoritmo irá chamar a próxima recursão com as uniões que geram as menores sobras complementares em ordem crescente (linhas 21 a 28).

É importante lembrar que essas árvores são hipotéticas e não necessariamente representa uma árvore de decisão de uma instância real. Pelas características levantadas do problema, cada junção gera oito possibilidades e com apenas três itens na primeira lista, a primeira ramificação já teria dezesseis possibilidades e mais quatro em cada filho (um dos novos itens não rotaciona). Desse modo, apesar ser mostrado um ganho de apenas duas visitas, com instâncias do problema, esse ganho pode ser bem maior (como pode ser visto na tabela 5).

Algorithm 4 Greedy Branch and Bound

```
1:  função buscaProfundidade(listaItens, perdas)
2:      se perdas ≤ limiar então
3:          devolve listaItens.tamanho
4:      fim se
5:      se listaItens.tamanho < nContêineres então
6:          nContêineres ← listaItens.tamanho          ▷ nContêineres é variável estática
7:          limiar ← perdas + listaItens.perdas        ▷ limiar é variável estática
8:      fim se
9:      se listaItens.tamanho > 1 então
10:         novosItensOrdenado[] ← ∅
11:         para i ← 1 até N − 1 faça
12:             para j ← i + 1 até N faça
13:                 novaListaItens ← listaItens − item[i] − item[j]
14:                 menorItem ← buscaMenorItem(novaListaItens)
15:                 novosItens ← uniõesPossíveis(item[i],item[j])
16:                 para k ← 1 até novosItens.tamanho faça
17:                     InsereOrdenado(novosItensOrdenado, novosItens[k])
18:                 fim para
19:             fim para
20:         fim para
21:         para k ← 1 até ParOrdenado.tamanho faça
22:             se união(novosItensOrdenado[k], menorItem) ≤ contêiner então
23:                 buscaProfundidade(novaListaItens + novosItensOrdenado[k], perdas)
24:             senão
25:                 novasPerdas ← perdas + novosItensOrdenado[k].perdas
26:                 buscaProfundidade(novaListaItens, novasPerdas)
27:             fim se
28:         fim para
29:     senão
30:         nContêineres ← 1
31:     fim se
32: fim função
```

Como é possível ver na tabela 5, o algoritmo *Greedy Branch and Bound* pode ser bem mais eficiente que o *Branch and Bound* simples. A aplicação da gula permite que o limiar de corte seja rapidamente reduzido e que o ótimo global seja localizado mais cedo na busca. Mesmo que a melhor solução não contenha a união com a primeira menor sobra, a probabilidade que ela contenha alguma das primeiras menores sobras é alta, uma vez que a soma das áreas das sobras complementares não pode ser reduzida em iterações seguintes.

Por esse mesmo motivo é possível afirmar que quanto mais profundo na árvore de decisão, menores serão as chances de uma união conter uma sobra alta, porque se a solução ótima possuir S unidades de área em sobras complementares e a primeira união possuir s_1 unidades de área, a soma das sobras seguintes não poderão exceder

Tabela 5 – comparativo da busca em profundidade, *branch and bound* e *Greedy Branch and Bound* para a instância com 9 itens

Instância de 9 itens e contêineres 30x20

Nº contêineres	Desperdício	LB	tempo		
			B. em prof	*Branch and bound*	*Greedy BnB*
9	4800.000	1	0.00019 s	0.00019 s	0.00033 s
8	4200.000	1	0.00021 s	0.00021 s	0.00052 s
7	3600.000	1	0.00022 s	0.00022 s	0.00063 s
6	3000.000	1	0.00023 s	0.00023 s	0.00069 s
5	2400.000	1	0.00023 s	0.00023 s	0.00073 s
4	1800.000	1	0.00024 s	0.00024 s	0.00075 s
3	1200.000	1	0.00025 s	0.00025 s	0.00077 s
2	600.000	1	0.00026 s	0.00026 s	0.00078 s
1	0.000	1	1082.22192 s	1081.30190 s	13.02939 s
Quantidade de contêineres:			1	1	1
Tempo transcorrido:			1082.222046 s	1081.302012 s	13.029607 s

$S - s_1$ unidades de área. Como S e s_1 são estritamente positivas ou nulas (não há área negativa), $S \geq S - s_1$. A mesma coisa ocorrerá com as sobras seguintes de modo que $s_1 \leq S$, $s_2 \leq S - s_1$, $s_3 \leq S - (s_1 + s_2)$ até $s_{n-1} \leq S - \sum_{i=1}^{n-2}(s_i)$.

Essa característica também irá reduzir o impacto da redundância causada pelas operações (Figura 32). Como as melhores soluções são jogadas para as primeiras ramificações, muitas redundâncias com sobras maiores serão removidas, pois o limiar de corte (limite superior) não irá permiti-las. Redundâncias com sobras menores também podem ser removida, desde que o nó que as originou esteja na solução ótima. Neste caso, o algoritmo irá parar sem retornar ao nó pai e, consequentemente, sem gerar a redundância. Olhando a Figura 32 e imaginando que a união (C|D) faça parte do ótimo, quando o menor resultado for alcançado (limite inferior), o algoritmo para, evitando que as uniões (A|B) e (E|F) e as redundâncias geradas por elas sejam testadas.

Para os casos onde o nó pai não esteja presente na solução ótima, infelizmente as redundâncias geradas pela ordem das operações irão existir. Entretanto, como explicado anteriormente, a gula tende a colocar as melhores soluções nas primeiras ramificações e isso reduz o impacto no tempo de execução.

Existe a possibilidade que a técnica de Programação Dinâmica resolva esse problema, uma vez que ela trabalha sobre ramificações semelhantes evitando descidas em subárvores idênticas as já visitadas e conhecidas. Como uma redundância causada pelas operações se caracteriza pela geração de listas idênticas em ramos diferentes da árvore de decisão e a partir de listas idênticas sempre são geradas subárvores idênticas é fácil imaginar que a técnica poderia resolver o problema. Entretanto, não há como garantir que a redução das redundâncias compense a sobrecarga de processamento que as comparações com listas visitadas irá causar.

5 UMA ABORDAGEM HEURÍSTICA

Soluções por métodos exatos, sejam matemáticos ou computacionais, tendem a demorar muito, principalmente para instâncias grandes. Para sanar essa lacuna, foram criadas classes de algoritmos que, mesmo não garantindo o ótimo, tendem a se aproximar dele. Esses são os métodos heurísticos.

Heurística vem do grego *heuriskein* que significa "encontrar". Elas podem ser definidas como procedimentos de busca por solução através de tentativa e erro ou por regras definidas empiricamente (OXFORD, 2017). Essas regras são comumente construídas através de experiências anteriores com o problema ou com problemas semelhantes.

Os métodos heurísticos tendem a encontrar soluções em menor tempo que os métodos exatos pela sua complexidade computacional. É de conhecimento comum que mesmo os métodos exatos mais eficientes possuem complexidade assintótica na ordem de $O(n!)$ ou $O(2^n)$, enquanto métodos heurísticos tendem a ordens mais baixas, como $O(n^2)$ ou $O(n^3)$, o que os torna mais indicados em casos aonde o tempo de resposta é mais importante que a corretude da solução ou em casos aonde esse n tende a valores "elevados" e o tempo de resposta dos algoritmos conhecidos se torna impraticável.

Há diversos métodos heurísticos para o problema de corte bidimensional guilhotinado, tais como FBS (*Finite best-strip*) proposta por Berkey e Wang (1987), FC (*Floor Ceiling*) proposto por Lodi, Martello e Vigo (1999b), GBL (*Guillotine bottom left*) proposto por Polyakovsky e M'Hallah (2009) e CFIH (*critical-fit insertion heuristic*) proposto por Fleszar (2013).

Todas essas heurísticas mostraram-se boas em seus resultados, todavia, seus criadores optaram pela junção das mesmas com alguma metaheurística (ou hiperheurística no caso da GBL) para aprimorar seus resultados. O presente trabalho não apresenta uma heurística nova, o foco foi na junção com metaheurísticas conhecidas, algumas formas de hibridização e um tíme assíncrono para tentar melhorar ainda mais os resultados das metaheurísticas com a heurística.

A heurística escolhida foi a *HHDHeuristic*, proposta por Nascimento, Longo e Aloise (1999) e sua escolha se deu pelos resultados obtidos no trabalho de Mariano (2014), que mostrou sua boa sinergia com o GRASP na forma de um time assíncrono.

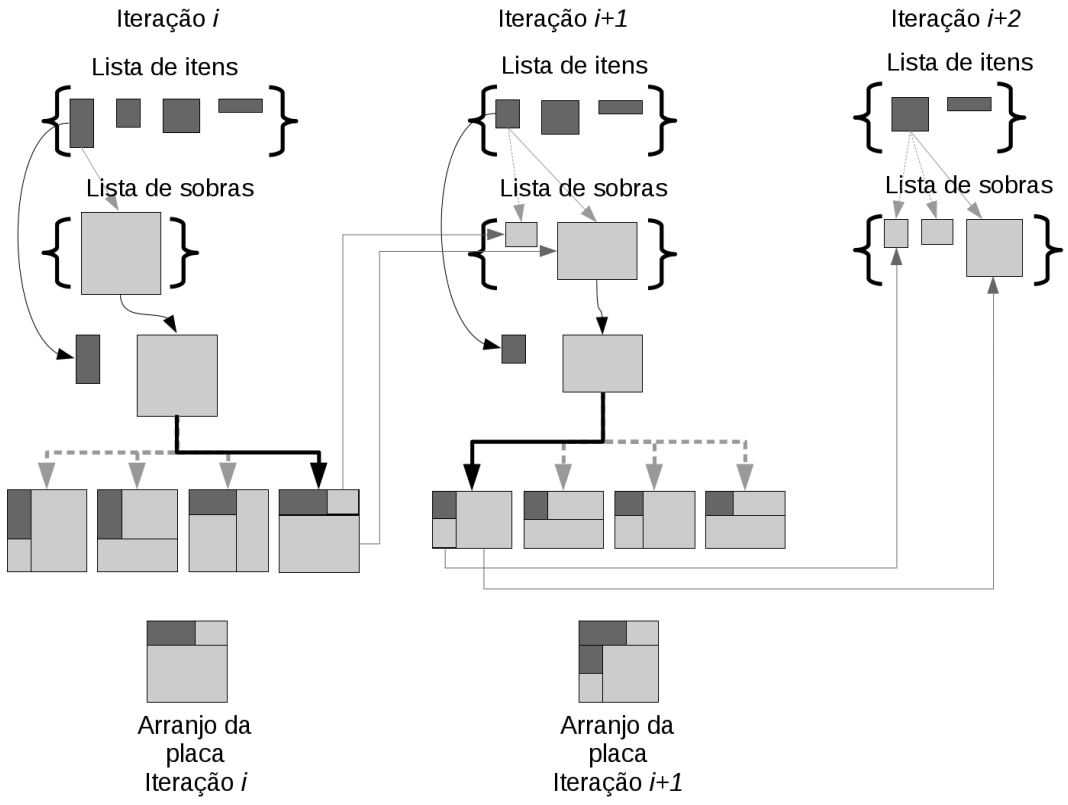
5.1 HHDHEURISTIC

O *HHDHeuristic* é uma heurística de montagem que utiliza a recursividade dos cortes da placa para gerar padrões. Ele começa elegendo a ordem em que os itens serão recortados e inserindo os itens nas placas a partir dessa ordem. Além da lista de itens, há uma lista de sobras e a cada nova inserção, uma sobra é removida dessa lista e até duas novas sobras podem ser inseridas nela.

Existem quatro ou duas possibilidades de corte, se as rotações são ou não permitidas, respectivamente. O algoritmo escolhe uma dessas possibilidades e as sobras resultantes são acrescidas a lista de sobras.

No algoritmo original, as sobras são fixadas, de modo que todos os itens são testados até que o um item caiba na sobra e isso pode fazer com que a ordem dos mesmos não tenha tanta importância, visto que um menor toma a frente de um item maior, caso ele seja maior que a sobra atual. Por isso uma pequena modificação será feita no algoritmo, ao invés de fixar a sobra e percorrer a lista de itens, o item será fixado e o algoritmo irá testar todas as sobras existente, conforme mostrado na Figura 33.

Figura 33 – Comportamento do algoritmo *HHDHeuristic*

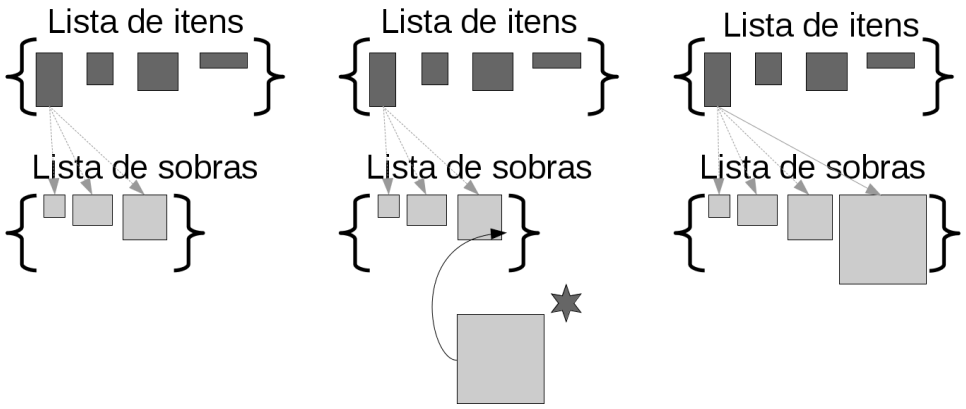


Fonte: Baseado em (NASCIMENTO; LONGO; ALOISE, 1999)

Caso o item atual não encaixe em nenhum sobra ou a lista de sobras esteja vazia, uma nova placa é aberta e acrescida a lista de sobras. E o algoritmo recomeça a partir

dessa nova lista de itens e sobras, mas com o número total de placas acrescido em uma unidade (Figura 34).

Figura 34 – Inserção de um novo contêiner



Fonte: Baseado em (NASCIMENTO; LONGO; ALOISE, 1999)

O algoritmo original possuí três variações em relação a escolha entre as possibilidades de corte: maximizar a área da menor sobra não nula; maximizar a área da maior sobra não nula; e maximizar a menor dimensão das sobras geradas. Todavia, apenas uma será utilizada nesse trabalho, a que maximiza a área da maior sobra. Essa escolha se deve ao fato que foi a que apresentou bons resultados ao ser combinada com o GRASP no trabalho de Mariano (2014).

O algoritmo *HHDHeuristic* (algoritmo 5) inicia com uma lista de itens, originalmente essa lista deveria ser ordenada pela área dos itens, da maior área para a menor área, mas neste trabalho quem irá ordenar e a forma como será ordenada será definida pelos demais algoritmos aqui propostos. A quantidade de contêineres necessários inicia com zero (linha 2). Para cada item na lista de itens, primeiro verifica se não é maior que o contêiner (linhas 4 a 6), se não extrapolar, o algoritmo busca a primeira sobra em que o item caiba, ou seja, que o item seja menor que a sobra (linhas 7 a 10).

Quando achar a sobra que comporte o item atual, retira essa sobra da lista de sobras (linha 12) e gera até duas novas sobras como mostrada na Figura 33 (linha 13). Caso acabe a lista de sobras e não achar uma que comporte o item atual, gera as novas sobras com um contêiner como se fosse uma sobra (linha 15) e acrescenta em um o número de contêineres utilizados (linha 16). Essas novas sobras são adicionadas a lista de sobras, ordenadas da menor para a maior área (linhas 18 e 19). Ao final, retorna a quantidade de contêineres gerados, juntamente com o que falta para a última (e maior) sobra ocupar um contêiner inteiro (linha 21).

Esse percentual do que a última sobra falta para ocupar o contêiner inteiro visa reduzir as sobras anteriores, concentrando todas em uma única sobra e tentando

Algorithm 5 HHDHeuristic

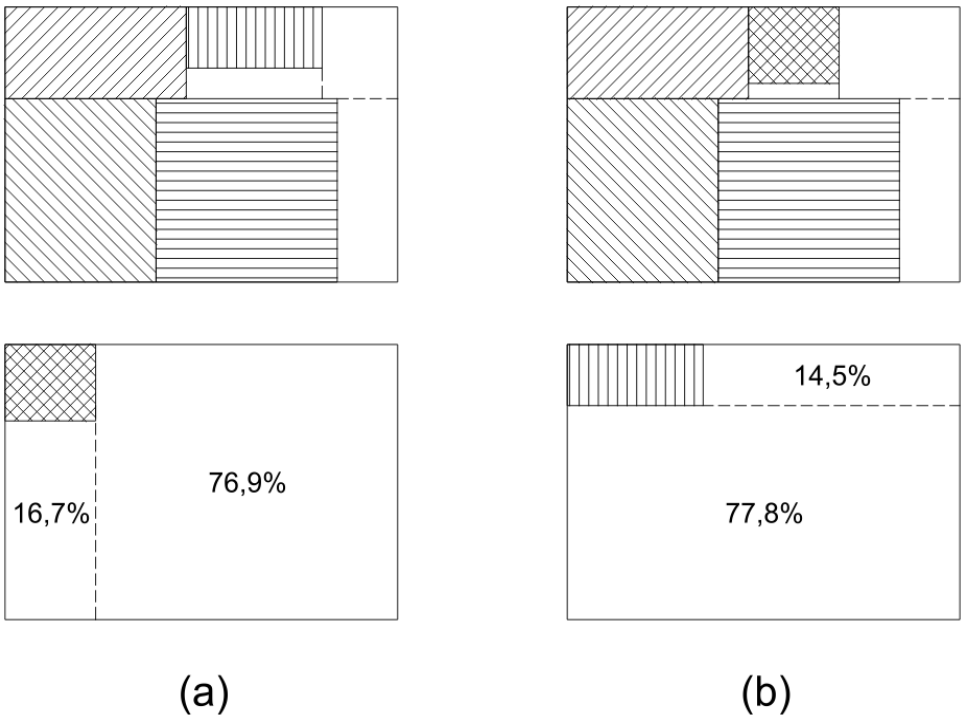
```
1:  função AVALIA(listaItens)
2:      nContêineres ← 0
3:      para i ← 1 até listaItens.tamanho faça
4:          se listaItens[i] > Contêiner então
5:              devolve Erro                ▷ Um item não pode ser maior que o contêiner
6:          fim se
7:          j ← 1
8:          enquanto itens[i] > sobras[j] E j ≤ listaSobras.tamanho faça
9:              j ← j + 1
10:         fim enquanto
11:         se j ≤ listaSobras.tamanho então
12:             listaSobras ← listaSobras - sobras[j]
13:             novasSobras ← uniãoMaiorSobra(item[i],sobra[j])
14:         senão
15:             novasSobras ← uniãoMaiorSobra(item[i],Contêiner)
16:             nContêineres ← nContêineres +1
17:         fim se
18:         adicionaOrdenando(listaSobras, novasSobras.menorSobra)
19:         adicionaOrdenando(listaSobras, novasSobras.maiorSobra)
20:     fim para
21:     devolve nContêineres + (contêiner - listaSobras.últimaSobra) / contêiner
22: fim função
```

otimizar os arranjos dos contêineres intermediários, pois quanto menos e menores forem as sobras dos contêineres intermediários, melhor será o aproveitamento dos mesmos.

Essa forma de avaliação é baseada nos trabalho de Mariano (2014), aonde a parte inteira do número seria a quantidade de placas utilizadas e uma parte fracionada indicaria o percentual de ocupação da placa menos utilizada. Foi feita apenas a substituição o percentual de ocupação pela área da maior sobra, estratégia semelhante a utilizada por Sinuany-Stern e Weiner (1994), mas para corte bidimensional.

A Figura 35 exemplifica a forma de avaliação e diferencia o método proposto nesse trabalho do método de Mariano (2014). Em ambos os casos, a parte inteira seria dois, pois é a quantidade de placas utilizadas. A parte fracionada iria diferir um pouco, no método proposto por Mariano (2014) a parte fracionada seria cem porcento menos o percentual de ocupação de todas as sobras da placa menos utilizada, ou seja, $1 - 0.167 - 0.769 = 0.064$ para (a) e $1 - 0.145 - 0.778 = 0.077$ para (b). No método proposto nesse trabalho, apenas a maior sobra seria considerada, então ficaria $1 - 0.769 = 0.231$ para (a) e $1 - 0.778 = 0.222$ para (b). Dessa forma, o algoritmo aqui proposto considera (b) uma solução melhor do que (a), enquanto deles considerariam (a) melhor do que (b).

Figura 35 – Melhores soluções pelo critério de avaliação do contêiner menos utilizado (a) e da maior sobra (b)



Fonte: Baseado em (MARIANO, 2014)

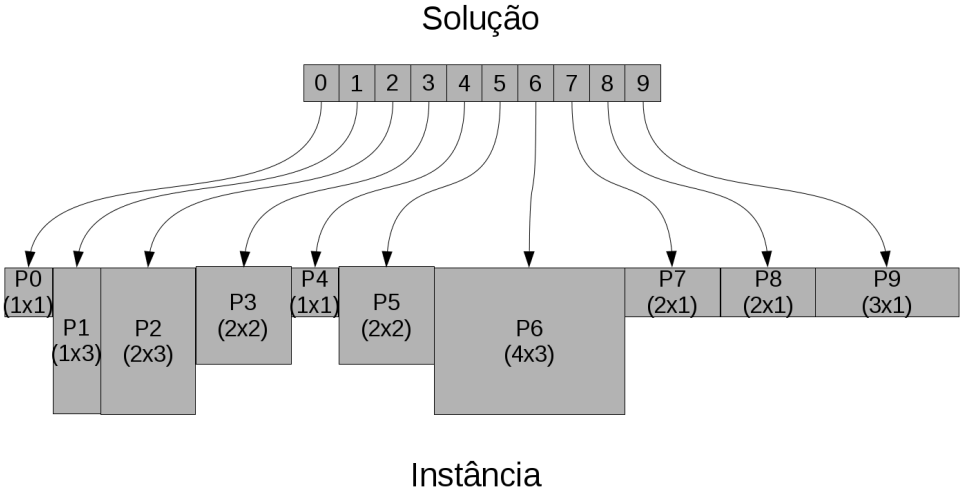
5.2 METAHEURÍSTICAS

Conforme comentado na início do capítulo, o *HHDHeuristic* seria combinado com diversas metaheurísticas. Esses algoritmos de otimização foram escolhidos pelos resultados obtidos anteriormente, bem como testar novas possibilidades. Para tanto, foi feito uso de uma estratégia bem conhecida dos algoritmos evolutivos, em especial o RKGA e BRKGA: a codificação e decodificação da solução.

Uma estrutura genérica é usada para que os algoritmos possam trabalhar e somente na função de avaliação é que essa estrutura genérica é convertida em uma instância do problema e avaliada. A estrutura genérica escolhida foi um vetor de inteiros, de modo que uma solução seria a ordem em que os itens estão dispostos nele (Figura 36).

Essa estratégia permite que os diversos algoritmos possam utilizar a mesma estrutura de dados e que um decodificador comum fosse utilizado para transformar essa estrutura genérica em uma solução do problema de Corte Bidimensional Guilhotinado. Os algoritmos trabalham sobre a estrutura genérica e a cada nova solução encontrada, um decodificador ou heurística de montagem realiza a conversão para que essa solução possa ser avaliada (Figura 37). Permitindo assim que os algoritmos possam aplicar sobre ela as suas políticas de busca, intensificação e diversificação sem se preocupar com as

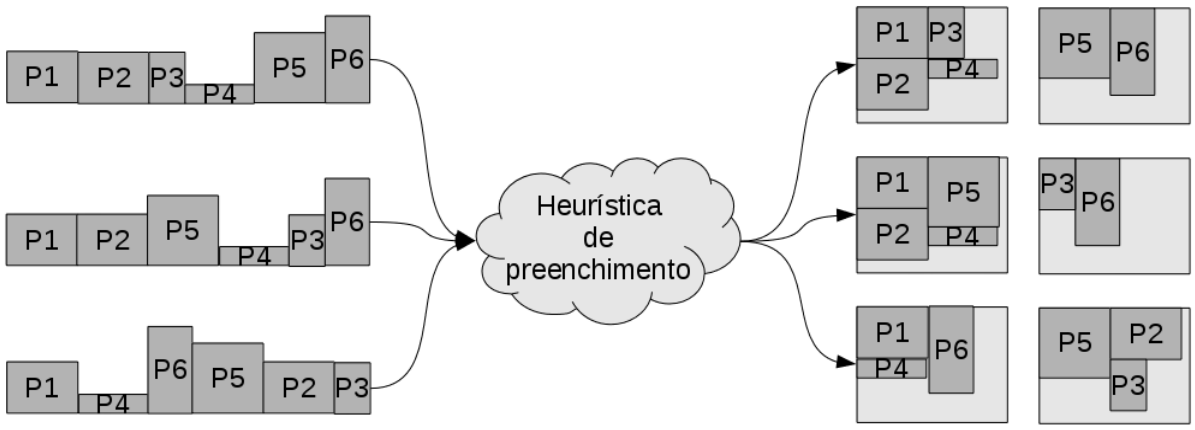
Figura 36 – Representação de uma solução na forma de um vetor



Fonte: Autoria própria

peculiaridades do problema.

Figura 37 – Heurística de preenchimento que transforma uma solução genérica em um arranjo

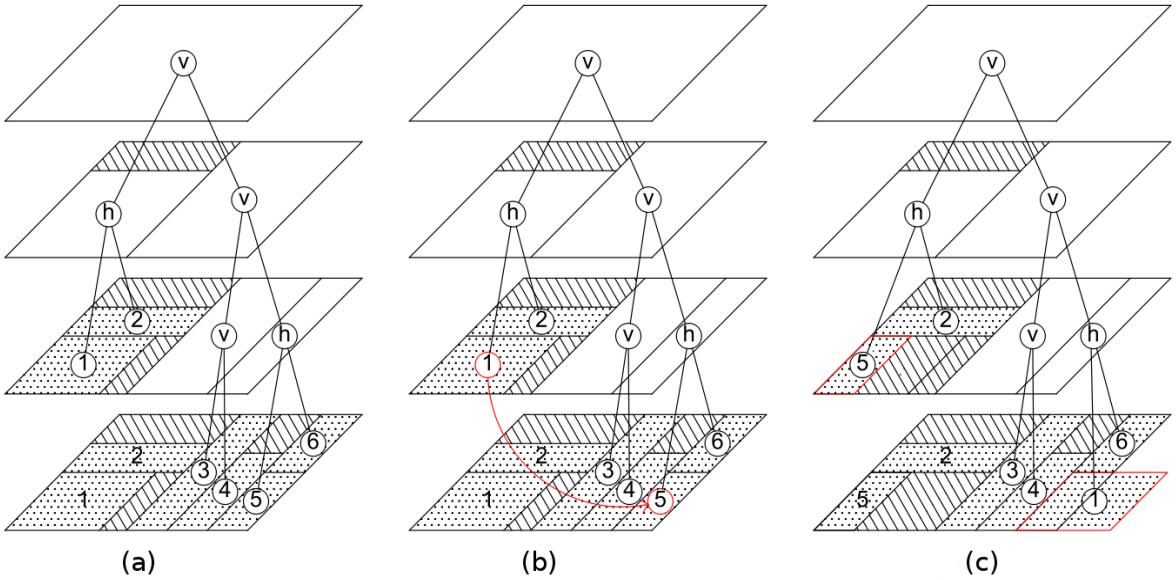


Fonte: Autoria própria

É interessante notar que, apesar da árvore binária conseguir representar bem o arranjo dos itens dentro de uma placa no problema de Corte Bidimensional Guilhotinado, ela não é tão interessante para as fases de busca e melhoria uma vez que a troca de posição entre duas folhas dessa árvore pode resultar em estados inválidos como mostrado na Figura 38.

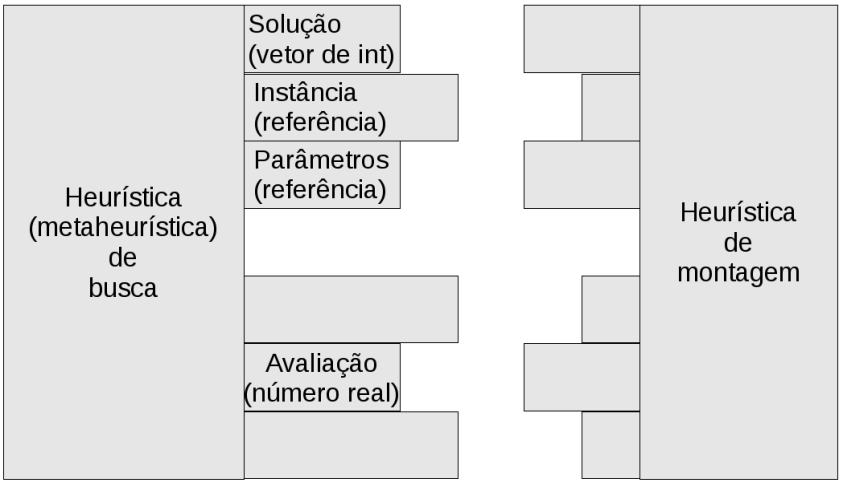
Como dito anteriormente, o *HHDHeuristic* servirá como heurística de montagem ou decodificador para a função de avaliação dos demais algoritmos, de modo que as metaheurísticas escolhidas irão encontrar diferentes ordens de itens e enviar ao *HHDHeuristic*, este monta os arranjos e retorna ao algoritmo que o chamou apenas um valor real indicando a qualidade daquela solução (Figura 39).

Figura 38 – Troca 2-opt em árvore binária



Fonte: Baseado em (KRÖGER, 1995)

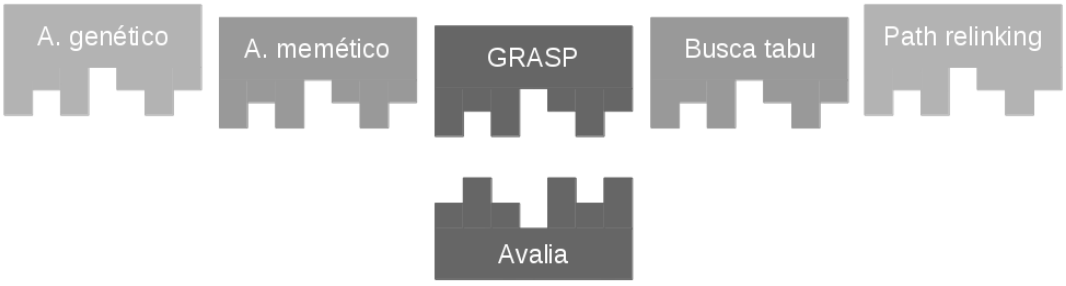
Figura 39 – Interface de comunicação entre heurística de busca e de montagem



Fonte: Autoria própria

Essa estratégia simplifica a implementação das metaheurísticas que fazem parte dos agentes. Isso porque os algoritmos não precisam de estruturas de dados muito complexas e, consequentemente, a manipulação delas também exigirá pouco esforço. Toda a complexidade é passada para o decodificador, que está embutido na função de avaliação. Desse modo, a implementação dos algoritmos se deu de forma genéricos e, na hora de avaliar, uma função contendo as especificações do problema é chamada. Isso permite que a função de avaliação seja reaproveitada em diversos algoritmos, desde que as entradas e saídas sejam compatíveis (Figura 40).

Figura 40 – Reaproveitamento da heurística de montagem como função de avaliação em diversos algoritmos de buscas diferentes



Fonte: Autoria própria

5.2.1 GRASP

Como o *HHDHeuristic* será usado como heurística de montagem, o primeiro algoritmo proposto nesse trabalho é o GRASP (sigla para *Greedy Randomized Adaptive Search Procedure*). Apresentado ao mundo por Feo e Resende (1995), o GRASP é uma das metaheurísticas mais conhecidas e utilizadas para solucionar problemas de diversas naturezas. Ela se utiliza da premissa de que boas soluções são compostas por indivíduos mais aptos, mas não necessariamente os melhores. Assim ele tenta combinar os indivíduos mais aptos, dando margens para que indivíduos próximos aos mais aptos possam compor a solução.

O GRASP é um algoritmo que combina uma fase construtiva randomizada com uma fase de melhoria (PARREÑO et al., 2010). Ele começa com uma fase inicial construtiva, onde os elementos que comporão a solução são ordenados a partir do quão mais aptos eles aparentam ser. A partir dessa lista ordenada, uma lista menor, chamada lista restrita de candidatos, é selecionada e então sorteado um elemento que fará parte da solução. Esse processo se segue até que todos os elementos da solução tenham sido sorteados.

Com uma solução inicial em mãos, o algoritmo começa uma fase de buscas na vizinhança com o intuito de achar soluções melhores. E essa busca se segue até que nenhuma solução melhor seja encontrada na vizinhança. Finalizada a fase de buscas o algoritmo retorna a fase construtiva com o intuito de, a partir de uma nova solução inicial, achar outras soluções, tentando assim, fugir de ótimos locais através de múltiplos reinícios.

O tamanho da lista restrita de candidatos depende de um parâmetro α. Isso pode ocorrer de duas formas: baseada na cardinalidade, onde o tamanho dessa lista será $1 + \alpha(N - 1)$, no qual N é o número total de candidatos disponíveis; ou baseada na qualidade dos elementos, no qual farão parte da lista restrita de candidatos todos os candidatos cuja atratividade esteja até certo valor, onde esse valor é definido como

$c^{min} + \alpha(c^{min} - c^{max})$, onde c é uma função de custo com c^{min} como custo mínimo e c^{max} como custo máximo dentre todos os candidatos a compor a solução (ALMEIDA, 2014).

O GRASP foi um dos algoritmos escolhidos pelo seu bom desempenho em trabalhos anteriores na área (como (VELASCO, 2005), (ALVAREZ-VALDES et al., 2007) e (MARIANO, 2014)) e pelos resultados obtidos por (NASCIMENTO; LONGO; ALOISE, 1999) que mostram que alocar itens maiores inicialmente é mais vantajoso, mas não garante a otimalidade da solução. Assim, a área dos itens torna-se um bom parâmetro para se aplicar gulosidade.

Na implementação do GRASP presente neste trabalho, foi utilizada a abordagem baseada na cardinalidade. Os candidatos a compor a solução são ordenados pela área dos itens e os p primeiros elementos dessa lista são colocados na lista restrita de candidatos. Após isso, um candidato é sorteado e o elemento seguinte da lista ordenada de itens passa a integrar a lista de restrita de candidatos (como mostra a Figura 41). Esse processo se repete até que a solução esteja montada.

Uma vez montada uma solução inicial, segue-se a fase de buscas. Foram utilizadas buscas com vizinhança de trocas 2-opt (trocar a posição de dois elementos), com a política de descida rápida, onde a escolha do próxima solução só ocorre depois de passar por toda a vizinhança. A vizinhança tem abrangência $O(n^2)$, onde há a troca de todos os pares de dois elementos.

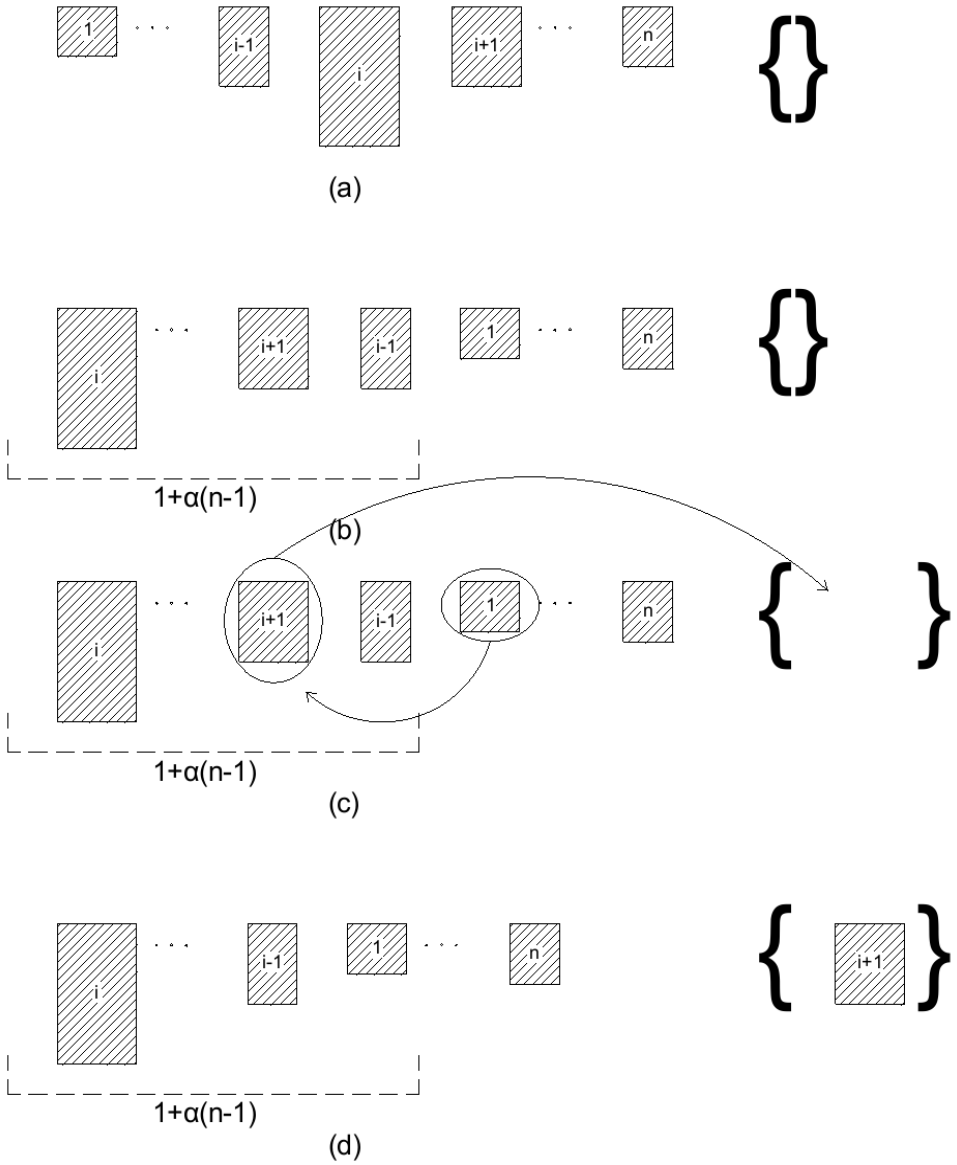
O algoritmo GRASP (algoritmo 6) inicia o laço principal, que o manterá executando. Enquanto um critério de parada não for atingido (linha 2), ele alterna entre gerar uma solução e busca em sua vizinhança por soluções melhores. Inicia uma solução vazia (linha 3) e ordena os itens em uma lista de candidatos (linha 4).

A partir da lista de candidatos, o algoritmo seleciona os p primeiros elementos (linha 6), sorteia um elemento dessa lista restrita de candidatos (linha 7), acrescenta esse elemento a solução e o remove da lista de candidatos (linhas 8 e 9). Esse processo se repete até que a lista de candidatos esteja vazia (linha 5). Como descrito anteriormente, o valor p depende do parâmetro α, nesse caso, pelo critério da cardinalidade.

Gerada uma solução (linhas 3 a 10), o algoritmo entra em um laço, realizando buscas locais até atingir um ótimo local. Para tanto, o algoritmo inicia uma variável que indicará se houve melhora ou não (linha 11) e entra em um laço enquanto o algoritmo achar um valor melhor na vizinhança (linha 12). Nesse laço, são selecionados dois elementos da lista de itens (linhas 15 e 16), esses elementos são trocados e a nova lista de itens é guardado como uma nova solução (linha 17), caso essa solução seja melhor que a solução atual (linha 18), ela é guardada como melhor solução encontrada e o parâmetro de melhora é atualizado para evitar que o laço termine prematuramente (linhas 19 e 20). Ao final de todas as trocas possíveis, a melhor solução é selecionada como a solução atual e o laço da busca local se reinicia.

Ao final da busca local, a melhor solução da iteração é comparada com a melhor

Figura 41 – Formação de uma solução a partir da lista de itens pelo algoritmo GRASP



Fonte: Baseado em (FEO; RESENDE, 1995)

de todas as iterações até o momento (linha 26) e caso seja melhor, ela é armazenada (linha 27) e o laço principal do algoritmo é reiniciado, gerando uma nova solução e realizando buscas locais sobre ela.

5.2.2 Busca tabu

Apresentado ao mundo por Glover (1986), o algoritmo de busca tabu é um processo iterativo onde, a cada iteração, o algoritmo move da solução s' para a melhor solução encontrada na vizinhança. Para prevenir que a busca visite soluções já visitadas

Algorithm 6 GRASP

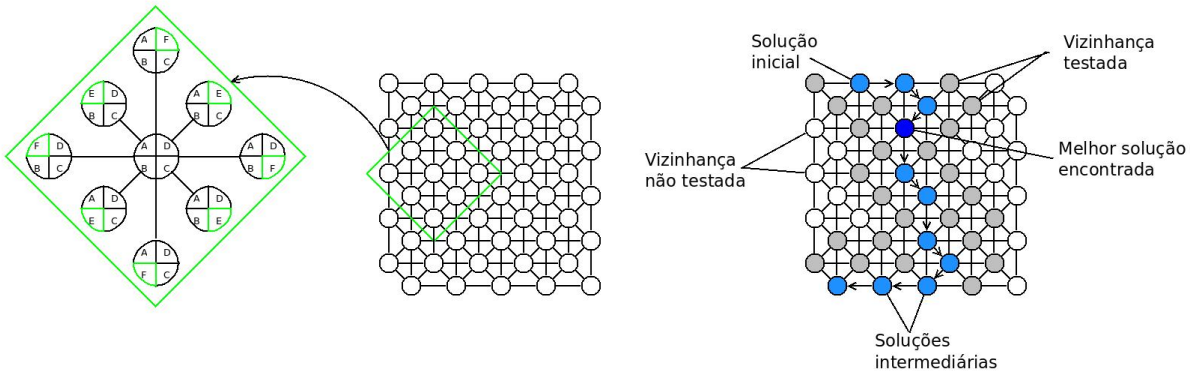
```
1: função GRASP(arquivoInstância, opções[])
2:     enquanto critériosDeParada faça
3:         soluçãoAtual ← ∅
4:         listaDeCandidatos ← ordena(instância.listaItens)
5:         enquanto listaDeCandidatos ≠ ∅ faça
6:             listaRestritaDeCandidatos ← primeirosItens(α, listaDeCandidatos)
7:             elementoSorteado ← listaRestritaDeCandidatos.itemAleatório
8:             soluçãoAtual ← soluçãoAtual + elementoSorteado
9:             listaDeCandidatos ← listaDeCandidatos − elementoSorteado
10:        fim enquanto
11:        houveMelhora ← sim
12:        enquanto houveMelhora faça
13:            houveMelhora ← não
14:            melhorSolução ← soluçãoAtual
15:            para i ← 1 até soluçãoAtual.tamanho −1 faça
16:                para j ← i + 1 até soluçãoAtual.tamanho faça
17:                    novaSolução ← troca2opt(soluçãoAtual, i, j)
18:                    se avalia(novaSolução) < avalia(soluçãoAtual) então
19:                        melhorSolução ← novaSolução
20:                        houveMelhora ← sim
21:                    fim se
22:                fim para
23:            fim para
24:            soluçãoAtual ← melhorSolução
25:        fim enquanto
26:        se avalia(melhorSolução) < avalia(melhorSoluçãoGeral) então
27:            melhorSoluçãoGeral ← melhorSolução
28:        fim se
29:    fim enquanto
30: fim função
```

e escape de ótimos locais, movimentos recentes são ditos tabus por um dado número de iterações. Sua escolha parte do trabalho de uma das propostas de trabalhos futuros de do trabalho de Velasco (2005), além dos resultados de Lodi, Martello e Vigo (1999a) para uma variação do problema.

O algoritmo começa com a geração de uma solução inicial e, a partir dela, realizar uma busca em sua vizinhança por soluções melhores. Cada movimento dentro do espaço de buscas é guardado numa lista chamada lista tabu para evitar que o algoritmo retorne a um ponto já visitado e assim, escape de ótimos locais. Para evitar que a lista tabu cresça indefinidamente, algum critério de aspiração deve ser criado para que movimentos ditos tabus o deixem de ser.

A Figura 42 ilustra esse Comportamento. Nela é possível perceber como o algoritmo de busca tabu se movimenta pelo espaço de buscas e como ele pode ser levado a pegar soluções de qualidade inferior com o intuito de escapar de ótimos locais.

Figura 42 – Movimentação da busca tabu no espaço de busca



Fonte: Baseado em (GLOVER, 1986)

O algoritmo de busca tabu foi escolhido exatamente por essa capacidade de escapar de ótimos locais ao evitar retorno a soluções já visitadas. Essa característica contrasta com o GRASP convencional, pois a única forma de fugir dessas soluções é por reinício e se essa nova solução inicial pertencer à mesma vizinhança, ele chegará aos mesmo resultados. A ideia central dessa escolha é que a Busca tabu consiga escapar de ótimos locais que o GRASP venha a ficar preso.

O algoritmo de Busca Tabu (algoritmo 7) inicia uma lista contendo os movimentos proibidos sem elementos (linha 2) e uma solução inicial é criada randomicamente (linha 3) e acrescentada a lista tabu (linha 4). Após isso o laço principal do algoritmo é iniciado mantendo a execução até que algum critério de parada seja alcançado (linha 5).

Uma variável é usada para sinalizar que a primeira solução válida foi alcançada (linha 6). Dois itens são escolhidos (linhas 7 e 8) e realizada a troca entre eles, gerando assim uma nova solução (linha 9). Com uma nova solução em mãos, o algoritmo verifica se a mesma não está na lista tabu (linha 10), se não estiver então a melhor solução da rodada é atualizada para a primeira solução válida (linha 12), ou seja, que não estava na lista tabu e a variável que indica se a primeira execução válida é atualizada. Vale salientar que esta primeira solução válida não é comparada com nenhuma outra, ou seja, se ela for pior que as já encontradas, ela tomará seu lugar, caso seja a única válida.

Após encontrar a primeira solução válida, todas as demais devem ser comparadas com a melhor encontrada na iteração atual (linha 15), caso seja melhor, a melhor solução da iteração é atualizada (linha 16). Ao final de todos os pares, se não houver encontrado nenhuma solução válida, uma nova solução randômica é criada (linhas 22 e 23) e o algoritmo reinicia o laço principal. Caso tenha achado ao menos uma solução válida, ela toma o lugar da solução atual para a próxima iteração (linha 25) e se esta for melhor que a melhor geral, então a melhor da iteração a substituirá melhor geral (linhas 26 e 27). Ao final, uma função de aspiração é executada para ver que soluções deixarão de ser tabu (linha 30).

Algorithm 7 Busca Tabu

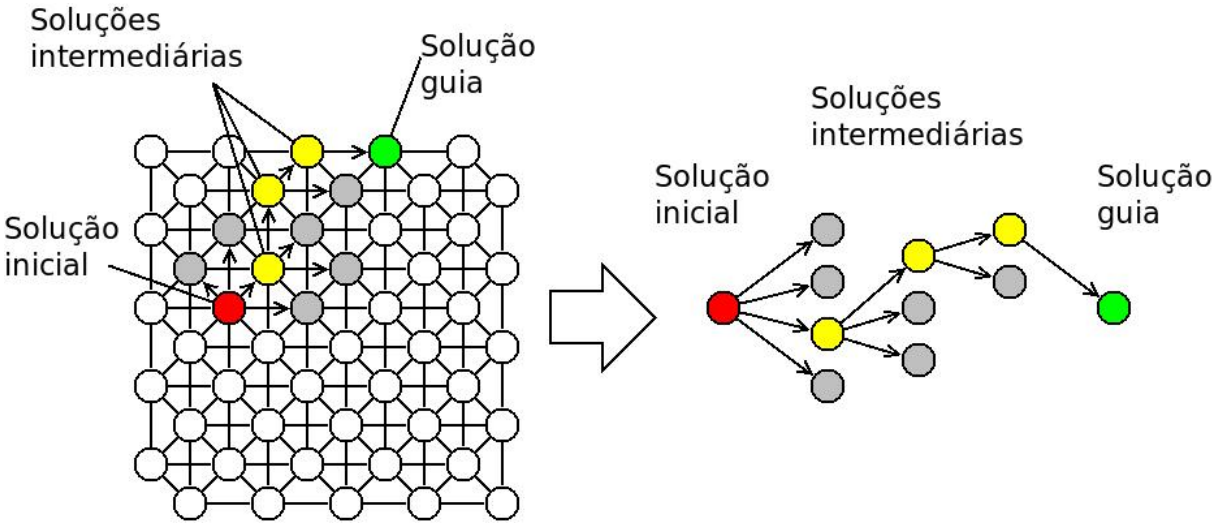
```
1:  função BuscaTabu(arquivoInstância, opções[])
2:      listaTabu ← ∅
3:      soluçãoAtual ← novaSoluçãoRandômica()
4:      listaTabu ← listaTabu + soluçãoAtual
5:      enquanto critériosDeParada faça
6:          primeiraExecuçãoVálida ← sim
7:          para i ← 1 até soluçãoAtual.tamanho −1 faça
8:              para j ← i + 1 até soluçãoAtual.tamanho faça
9:                  novaSolução ← troca2opt(soluçãoAtual, i, j)
10:                 se novaSolução ∉ listaTabu então
11:                     se primeiraExecuçãoVálida então
12:                         melhorSolução ← novaSolução
13:                         primeiraExecuçãoVálida ← não
14:                     senão
15:                         se avalia(novaSolução) < avalia(melhorSolução) então
16:                             melhorSolução ← novaSolução
17:                         fim se
18:                     fim se
19:                 fim se
20:             fim para
21:         fim para
22:         se primeiraExecuçãoVálida então
23:             soluçãoAtual ← novaSoluçãoRandômica()
24:         senão
25:             soluçãoAtual ← melhorSolução
26:             se avalia(melhorSolução) < avalia(melhorSoluçãoGeral) então
27:                 melhorSoluçãoGeral ← melhorSolução
28:             fim se
29:         fim se
30:         funçãoDeAspiração(listaTabu)
31:     fim enquanto
32: fim função
```

5.2.3 Religação de caminhos

Segundo autores como Alvarez-Valdes et al. (2007) e Laguna e Marti (1999), o algoritmo de religação de caminhos foi proposto, por Glover, como uma estratégia de intensificação e diversificação no contexto da Busca Tabu. Essa estratégia gera novas soluções pela exploração de trajetórias que conectam soluções de alta qualidade.

Começando por uma solução, chamada solução inicial e caminhando pelos espaços de vizinhança que levam até outra solução, chamada solução guia. Essa movimentação pelos espaços de vizinhança acontece pela introdução gradativa de atributos da solução guia na solução inicial (ALVAREZ-VALDES et al., 2007).

Figura 43 – Caminho gerado pela movimentação que o algoritmo de religação de caminhos faz no espaço de buscas



Fonte: Baseado em (ALVAREZ-VALDES et al., 2007)

Ele substitui componentes da solução inicial por componentes da solução guia, um elemento por vez, com o intuito que uma solução intermediária, com partes da solução inicial e partes da solução guia, seja melhor que ambas as soluções. Isso parte do pressuposto que a solução ótima global pode conter parte das soluções ótimas locais.

O algoritmo de religação de caminhos foi escolhido por, além de ter sido criado como aprimoramento à Busca Tabu, existirem diversos trabalhos na literatura onde o GRASP é utilizado em conjunto com ele (como os trabalhos de Parejo et al. (2014), Campos et al. (2014), Martí et al. (2015) e Ferone, Festa e Resende (2016)). Dessa forma, ele aparenta ser um excelente agoritmo a compor um time composto por GRASP e Busca Tabu.

O algoritmo de religação de caminhos (algoritmo 8) inicia com duas soluções, a solução inicial e a solução guia (linhas 2 e 3). A melhor solução geral é a melhor entre a solução inicial e a solução guia (linhas 4 a 8). Após escolher a melhor solução geral, o algoritmo pega elemento a elemento da solução inicial (linha 9), remove e acrescenta um elemento da solução guia (linhas 10 e 14), corrige qualquer irregularidade gerada pela troca (linhas 11 e 15) e marca qual o elemento que foi selecionado (linhas 12 e 18). Esse elemento selecionado é armazenado para que, quando o algoritmo passar por todas as trocas possíveis ele possa ser removido da solução guia (linha 21) e não seja inserido na solução inicial mais de uma vez.

Ao final da iteração, a melhor solução gerada pela troca do elementos selecionados toma o lugar da solução inicial (linha 22) e da melhor solução geral, caso sua avaliação seja melhor (linhas 23 a 25) e o algoritmo reinicia o laço principal, tentando trocar o próximo elemento da solução inicial por outro da solução guia. Ao terminar todas as

Algorithm 8 Religação de caminhos

```
1:  função rCaminhos(solução1, solução2)
2:      soluçãoInicial ← solução1
3:      soluçãoGuia ← solução1
4:      se avalia(solução1) < avalia(solução2) então
5:          melhorSoluçãoGeral ← solução1
6:      senão
7:          melhorSoluçãoGeral ← solução2
8:      fim se
9:      para i ← 1 até soluçãoInicial.tamanho faça
10:         novaSolução ← soluçãoInicial − soluçãoInicial[i] + soluçãoGuia[1]
11:         melhorSolução ← corrigeEstadosInválidos(novaSolução)
12:         elementoSelecionado ← 1
13:         para j ← 2 até soluçãoGuia.tamanho faça
14:             novaSolução ← solução1 − soluçãoInicial[i] + soluçãoGuia[1]
15:             novaSolução ← corrigeEstadosInválidos(novaSolução)
16:             se avalia(novaSolução) < avalia(melhorSolução) então
17:                 melhorSolução ← novaSolução
18:                 elementoSelecionado ← j
19:             fim se
20:         fim para
21:         soluçãoGuia ← soluçãoGuia - soluçãoGuia[elementoSelecionado]
22:         soluçãoInicial ← melhorSolução
23:         se avalia(melhorSolução) < avalia(melhorSoluçãoGeral) então
24:             melhorSoluçãoGeral ← melhorSolução
25:         fim se
26:     fim para
27:     devolve melhorSoluçãoGeral
28: fim função
```

iterações, o algoritmo retorna a melhor solução geral, seja a solução inicial, guia ou outra composta por partes de ambas.

É interessante notar que, neste caso, o algoritmo não chama uma instância. Isso porque o algoritmo de religação de caminhos não trabalha só. Ele deve ser combinado com outro algoritmo para que possa trabalhar. Desse modo, um outro algoritmo irá carregar a instância, gerar soluções e repassar essas soluções ao algoritmo de religação de caminhos para que este possa tentar combina-las e gerar soluções melhores.

5.2.4 Algoritmo genético e BRKGA

Baseado no processo evolutivo das espécies proposto por Darwin, os algoritmos genéticos vem sendo estudados desde a década de 1950, porém não há um consenso

sobre quem efetivamente criou o método computacional. As principais pesquisas históricas dessa área são as pesquisas de Rechenberg (1973) e Holland (1975).

O método segue o princípio da seleção natural de Charles Darwin. Nele a solução é codificada como um conjunto de parâmetros (genes) unidos na forma de uma sequência (cromossomo) e representam um indivíduo. Esses indivíduos são recombinados na fase de reprodução para gerar uma nova população. Os pais são escolhidos aleatoriamente utilizando algum artifício que favoreça a escolha de indivíduos mais bem avaliados. Uma vez selecionados, os cromossomos dos pais são recombinados utilizando um processo conhecido como cruzamento (*crossover*). Mutações são utilizadas em alguns indivíduos para garantir a diversidade da população (GONCALVES; RESENDE, 2012).

Neste trabalho a codificação é a estrutura genérica explicada anteriormente e um indivíduo seria uma sequencia com todos os itens em uma ordem específica. A seleção dos pais se dá pela classificação dos mesmos dentro da população, neste caso, um dos progenitores sempre viria da população elite e o outro da população não elite. O elitismo, neste caso, se dá pelo fato que há menos indivíduos da população elite que na população não elite.

O algoritmo genético (algoritmo 9) inicia criando uma nova população de soluções de forma aleatória e classifica-a da melhor para a pior avaliação (linhas 2 a 5). Feito isso se inicia o laço principal do algoritmo (linha 6) que irá salvar a população elite de uma iteração para a seguinte (linhas 7 a 10), adicionar indivíduos inteiramente novos e classificando-os na nova população (linhas 11 a 14), gerar uma lista de pares que irão cruzar (linha 15) e gerar os filhos a partir dessa lista, mantendo a classificação atualizada (linhas 16 a 19). Ao final, a população atual e a nova população são invertidas (linhas 20 a 22) juntamente com suas classificações (linhas 23 a 25) para que na iteração seguinte a população atual seja sobrescrita por novos indivíduos gerados pelo processo descrito anteriormente.

É interessante salientar que não foi utilizado nenhum operador de mutação, em seu lugar, novos indivíduos gerados aleatoriamente são acrescentados a população em cada iteração para evitar sua homogeneidade e garantir que o algoritmo continue evoluindo em iterações mais avançadas.

O operador de cruzamento utilizado é o cruzamento por um ponto. Neste, uma posição de corte é aleatoriamente escolhida entre os cromossomos pais e então a parte esquerda de cada cromossomo é recombinada com a parte direita do outro (OLIVEIRA, 2010). Esse comportamento é mostrado na Figura 44.

Um segundo algoritmo genético desenvolvido foi o BRKGA. *Biased Random-Key Genetic Algorithm* (Algoritmo Genético com Chaves Aleatórias Viciadas) ou simplesmente BRKGA é uma variação de algoritmo genético que se utiliza da ideia de chaves aleatórias proposta por Bean (1994). Foi apresentado por Goncalves e Beirão (1999) e Goncalves e Almeida (2002).

Algorithm 9 Algoritmo genético

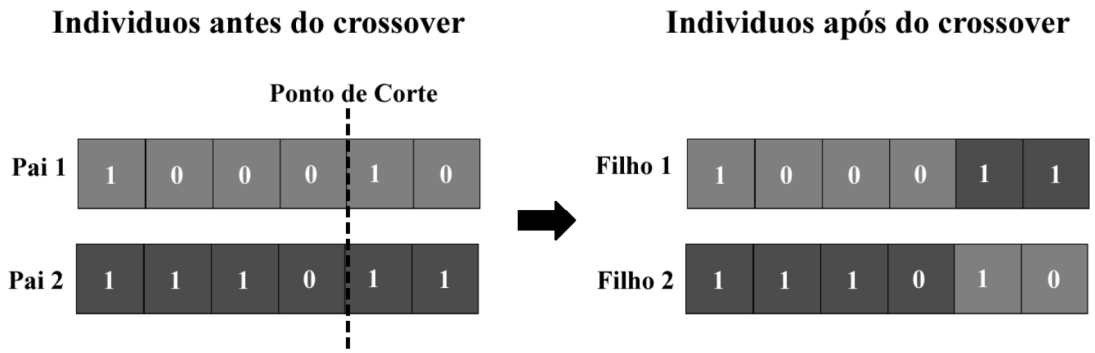
```
1: função AGenetico(arquivoInstância, opções[])
2:     para i ← 1 até populaçãoAtual.tamanho faça
3:         populaçãoAtual[i] ← novaSoluçãoRandômica()
4:         ordenaçãoPorInserção(classificaçãoAtual, populaçãoAtual[i])
5:     fim para
6:     enquanto critériosDeParada faça
7:         para i ← i até tamanhoPopulaçãoElite faça
8:             novaPopulação ← populaçãoAtual[classificaçãoAtual[i]]
9:             novaClassificação[i] ← classificaçãoAtual[i]
10:        fim para
11:        para j ← 1 até tamanhoPopulaçãoImigrante faça
12:            novaPopulação[j + i] ← novaSoluçãoRandômica()
13:            ordenaçãoPorInserção(novaClassificação, novaPopulação[j + i])
14:        fim para
15:        listaPais ← organizaPais(classificaçãoAtual)
16:        para k ← 1 até tamanhoPopulaçãoDeFilhos faça
17:            novaPopulação[k + j + i] ← cruzamento(listaPais, populaçãoAtual)
18:            ordenaçãoPorInserção(novaClassificação, novaPopulação[k + j + i])
19:        fim para
20:        auxTroca ← populaçãoAtual
21:        populaçãoAtual ← novaPopulação
22:        novaPopulação ← auxTroca
23:        auxTroca ← classificaçãoAtual
24:        classificaçãoAtual ← novaClassificação
25:        novaClassificação ← auxTroca
26:    fim enquanto
27: fim função
```

Semelhante ao algoritmo genético tradicional, ele codifica uma solução em cromossomos, com a diferença que os genes possuem valores fracionados que variam entre zero a um e toda o processamento da heurística ocorre independente do problema e, durante a avaliação, um decodificador é colocado para transformar o cromossomo numa solução adequada ao problema (Figura 45).

Os pais no BRKGA são sorteados sempre com um indivíduo da população elite e outro da população não elite. No *crossover*, cada gene é sorteado independente, porém com probabilidade p_e do gene vir do indivíduo da população elite e $1 - p_e$ do indivíduo da população não elite (GONCALVES; RESENDE, 2013). Assim, além do elitismo na escolha dos pais, há uma tendência do algoritmo escolher mais genes do progenitor elite que do não elite.

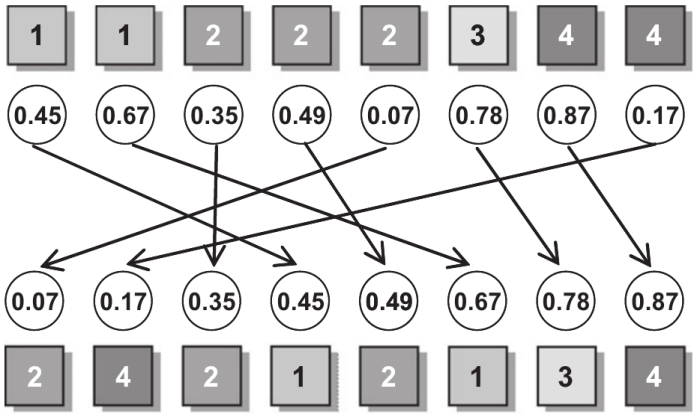
Apesar de serem dois algoritmos distintos, o pseudo código permanece o mesmo do algoritmo genético. Isso porque a essência do algoritmo BRKGA é a mesma do algoritmo genético. As principais diferenças estão na codificação no intervalo de zero a um e no sorteio viciado de onde virá cada gene que comporá a solução filha.

Figura 44 – Cruzamento de um ponto.



Fonte: (OLIVEIRA, 2010)

Figura 45 – Exemplo de decodificação dos genes (círculos) em sequência de elementos do problema original (quadrados).



Fonte: (GONCALVES; RESENDE, 2013)

5.2.5 Algoritmos Híbridos

Alguns algoritmos híbridos foram feitos pela combinação entre os já citados. De início todos os algoritmos são híbridos da metaheurística escolhida com a heurística *HHDHeuristic*, isso porque todas as metaheurísticas utilizam uma das variações do algoritmo proposto por Nascimento, Longo e Aloise (1999) para transformar uma lista ordenada de itens em um arranjo, portanto isso será omitido.

A primeira e mais simples hibridização proposta nesse trabalho é colocar a fase construtiva do GRASP nos demais algoritmos propostos. Essa hibridização gera mais dois algoritmos: A busca tabu com GRASP e o Algoritmo Genético com GRASP. Na prática, a solução inicial da Busca Tabu e a população inicial do Algoritmo Genético são gerados utilizando os critérios de gulosidade e aleatoriedade do GRASP.

A segunda forma de hibridização proposta é a aplicação da Religação de Caminhos dentro do Algoritmo Genético. Isso foi feito substituindo o cruzamento tradicional

pela Religação de Caminhos. Como a Religação de Caminhos é um algoritmo de ordem $O(n^2)$ (desprezando a função de avaliação), substituir totalmente o cruzamento mostrou-se muito custoso em termos de tempo de computação. Então foi aplicada a cada determinado número de cruzamentos.

Desse modo os algoritmos testados são:

- GRASP
- Busca Tabu
- Busca Tabu + GRASP
- Algoritmo Genético
- Algoritmo Genético + GRASP
- Algoritmo Genético + Religação de Caminhos
- BRKGA

5.3 TIME ASSÍNCRONO

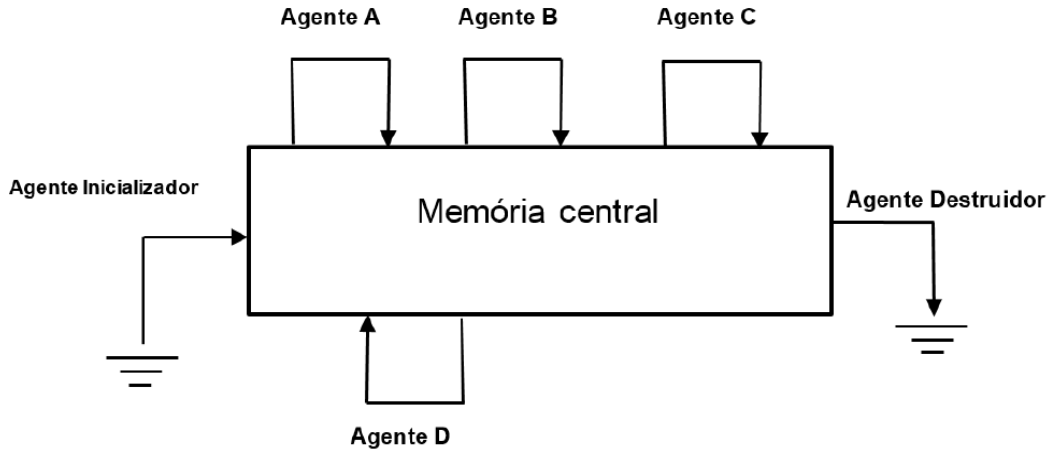
Um time assíncrono pode ser definido como uma arquitetura de inteligência computacional que consiste em múltiplos métodos de resolução (chamados agentes) trabalhando juntos em um problema comum (WALDHERR; KNUST, 2014). Essa arquitetura tem por base um conjunto de agentes autônomos e um conjunto de memórias interconectados para formar uma rede cíclica. Resultados ou soluções testadas são acumulados nas memórias para formar populações. Essas populações variam com o tempo. Novos membros são continuamente adicionados por agentes construtores, enquanto membros antigos podem ser apagados por agentes de destruição (TALUKDAR et al., 1998).

Um time assíncrono possui três características marcantes: os agentes possuem autonomia para tomar suas próprias decisões sobre seleção da entrada, escalonamento e política de alocação de recursos; agentes podem ler e escrever na memória compartilhada sem a necessidade de coordenação entre eles; agentes recuperam, alteram e salvam informações continuamente na memória compartilhada (FILHO; SANTOS; MENESES, 2012).

Tanto a abordagem de Polyakovsky e M'Hallah (2009), quanto de Mariano (2014) alcançaram bons resultados e levantaram a possibildiade de combinar metaheurísticas amplamente utilizadas com um sistema multiagente.

Na abordagem de times assíncronos, existem basicamente três tipos de agentes: os agentes de inicialização (ou de construção), que são responsáveis pela criação das soluções iniciais e colocar na memória compartilhada; agentes de melhoria, responsáveis pelo aprimoramento das soluções presentes na memória compartilhada; agentes de destruição, responsáveis pelo descarte das soluções antigas (MARIANO, 2014). Esse esquema é mostrado na Figura 46.

Figura 46 – Arquitetura baseada em agentes e memória compartilhada.



Fonte: (MARIANO, 2014)

Para solucionar uma tarefa, uma implementação do time assíncrono usa uma população de soluções que são melhoradas por agentes de otimização, no qual os agentes representam diferentes algoritmos de otimização. Os agentes trabalham independentemente, em paralelo, e cooperam indiretamente usando a memória comum que contem a população de soluções (JEDRZEJOWICZ; WIERZBOWSKA, 2014).

Desse modo, um time não precisa de um sistema muito complexo de gerenciamento, o que facilita tarefas como inserção ou retirada de um agente, recuperação de falhas dos nós ou de comunicação entre os mesmos. Desde que o conjunto de memórias compartilhadas estejam operando, o time assíncrono pode funcionar sem grandes problemas em caso de falha de alguns de seus componentes.

O time de agentes utilizará um conjunto de heurísticas e metaheurísticas para solucionar o problema de Corte Bidimensional Guilhotinado. Essa característica se assemelha muito ao conceito de hiper-heurística. A ideia central de uma hiper-heurística é usar membros de um conjunto de heurística conhecidas e razoavelmente compreendida para transformar o estado de um problema. Tentando associar cada heurística com os estados do problema enquanto ele amadurece e aplicar diferentes heurísticas em diferentes partes ou fase do processo de solucionar o problema (BURKE et al., 2003).

A ideia de implementação do time assíncrono, com o uso de diferentes heurísticas, visa a obtenção de soluções melhores com a troca de heurísticas. Fazendo, dessa forma, com que soluções que foram trabalhadas com determinada heurística ou metaheurística

seja passada a outro algoritmo para que este consiga melhorar ou superar as dificuldades de outros algoritmos, como ótimos locais no GRASP ou o excesso de homogeneidade em algoritmos genéticos.

Todavia, o time assíncrono carrega características, como a independência dos agentes e pouca coordenação entre eles, que fogem um pouco ao conceito de hiper-heurística. Além disso, os estados do problema não são monitorados e, dessa forma, não há como selecionar as heurísticas que serão aplicadas e cada fase.

Outra característica que diferencia uma hiper-heurística de um time assíncrono é qua hiper-heurística escolherá que heurística ou metaheurística será utilizada em cada momento, enquanto no time todas as heurísticas são utilizadas o tempo todo visando que a combinação delas tenham uma boa sinergia e gerem boas soluções. De fato, a escolha de uma arquitetura baseada em time assíncrono em detrimento à estrutura de uma hiper-heurística se dá pelo desconhecimento dos estados do problema e quais as melhores heurísticas a serem aplicadas em cada um deles. Além da baixa complexidade no gerenciamento dos agentes, o que permite adicionar ou remover agentes com facilidade.

5.3.1 Estrutura do time assíncrono

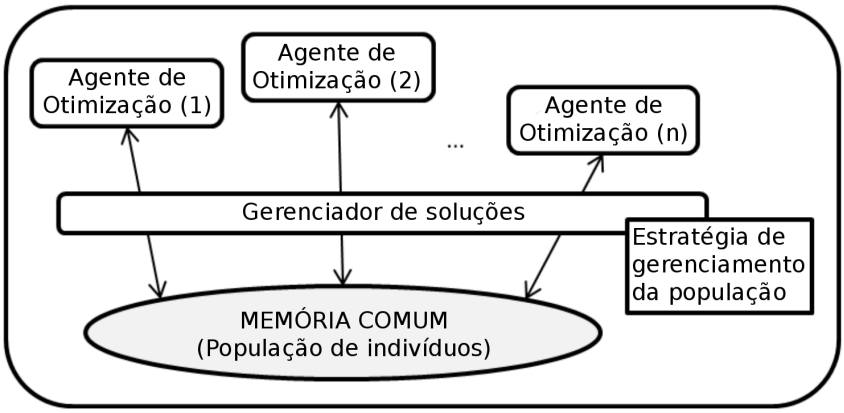
A implementação do time assíncrono utilizado nesse trabalho foi pensada na simplicidade de alteração, acréscimo e retirada de agentes. A modularização dos componentes integrantes foi o foco no desenvolvimento e isso se reflete na escolha da estrutura que irá integra-los e quais seriam esses componentes.

A primeira escolha foi pela separação dos algoritmos heurísticos e metaheurísticos do código do time. Para tanto uma camada de abstração foi implementada fazendo com que os algoritmos de otimização fossem implementados isoladamente e um segundo componente seria responsável por integrar esses algoritmos na forma do time, ligando-os ao conjunto de memórias compartilhadas (Figura 47).

Essa estratégia permite que os algoritmos executem de forma independente do time e que qualquer algoritmo possa ser integrado ao time, desde que suas entradas e saídas de dados sejam compatíveis. Na prática, isso significa que o componente do time responsável pela integração do algoritmo ao time faz todo o processo de gerenciamento de soluções e passa ao algoritmo apenas as informações necessárias para o mesmo executar e quando este retorna uma solução, ele envia essa solução para a memória compartilhada.

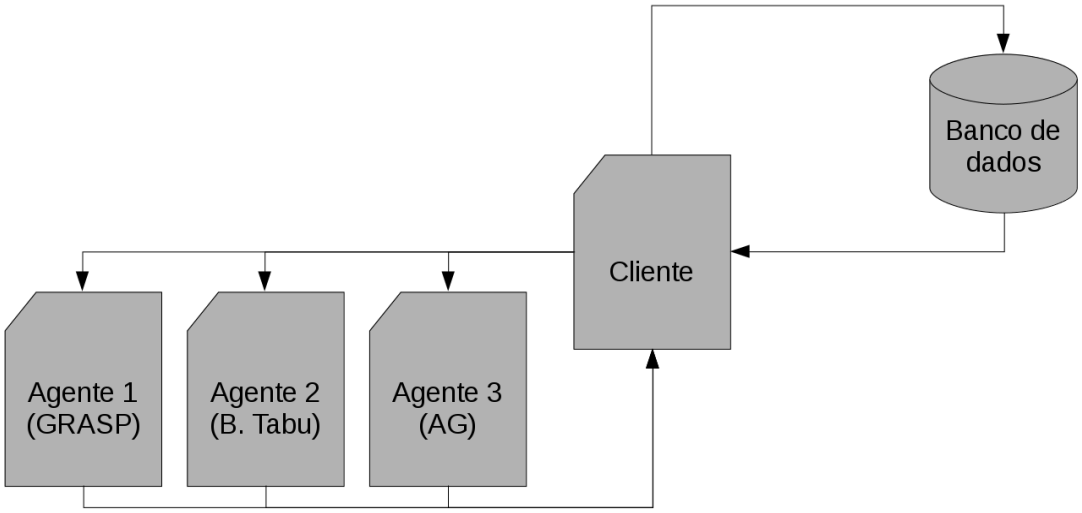
O componente do time assíncrono, nomeado cliente, acessa o conjunto de memórias e pega as informações necessárias para a execução do algoritmo, em seguida

Figura 47 – Estratégia de gerenciamento da população de soluções.



Fonte: (BARBUCHA, 2014)

Figura 48 – Conexão entre os algoritmos e a memória comum (banco de dados) promovia pelo cliente.



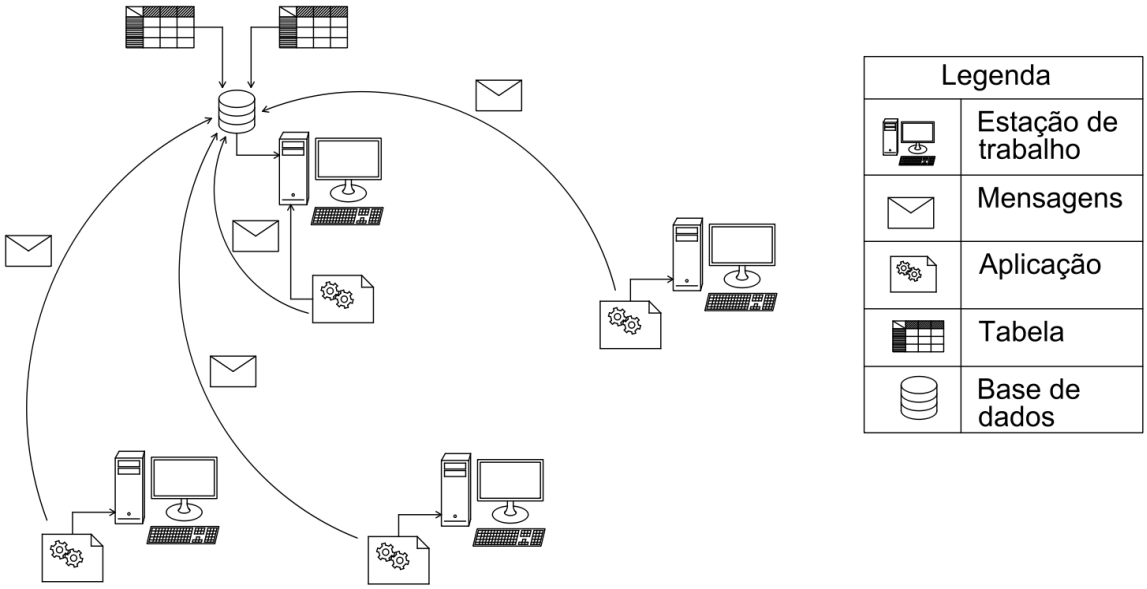
Fonte: Autoria própria

executa o algoritmo de otimização com as configurações definidas, pega o resultado dessa execução e envia novamente ao conjunto de memórias, juntamente com outras informações (Figura 48).

Como conjunto de memórias comum, foi escolhido um banco de dados, neste caso o PostreSQL. Usar um banco de dados garante a consistência dos dados, o controle de acesso e a possibilidade de acesso remoto, reduzindo o trabalho de implementação do time. Nesse sentido o PostreSQL foi escolhido por ser uma ferramenta livre e gratuita com ampla documentação e uma grande comunidade de mantenedores. A estrutura do time com a base de dados é mostrada na Figura 49.

A sobrecarga do sistema causada pelo envio e recebimento das mensagens, processamento das consultas no banco e atrasos na rede é pequena em relação ao tempo total de processamento de cada algoritmo, uma vez que as mensagens são de pequeno tamanho e os testes foram executados apenas em redes locais.

Figura 49 – Estrutura cliente servidor do time assíncrono.



Fonte: Autoria própria

6 RESULTADOS

6.1 METODOLOGIA

6.1.1 Ambiente de testes

Os testes foram executados em uma máquina equipada com processador intel core 2 quad Q8200 4x2,3GHz, 4GB de memória ram DDR3 1066MHz e sistema Kubuntu linux 14.04 64bits. Para realizar os testes do modelo foi utilizado o *solver* de código aberto chamado *lp_solve* em sua versão 5.5 disponível no site <http://lpsolve.sourceforge.net/5.5/>. Os algoritmos heurísticos e exato utilizaram a mesma configuração de hardware, com exeção do Time Assíncrono, que executou em rede.

Por demandar de recursos de rede, o Time Assíncrono executou em um dos laboratórios do mestrado, quatro máquinas foram utilizadas. A primeira, equipada com processador Pentium T4200 2x2,20GHz, 4GB de memória ram DDR2 800MHz e sistema linux 14.04 foi utilizada como servidor de banco de dados e nenhum agente foi executado nela. As demais máquinas possuiam processador Fusion A4-3400 2x2,7GHz, 4GB de memória ram DDR2 800MHz e sistema linux 16.10 e a rede cabeada em topologia estrela com switch D-Link 10/100. O banco de dados escolhido foi o PostgreSQL 9.3 (versão do repositório do Ubuntu 14.04).

6.1.2 Instâncias

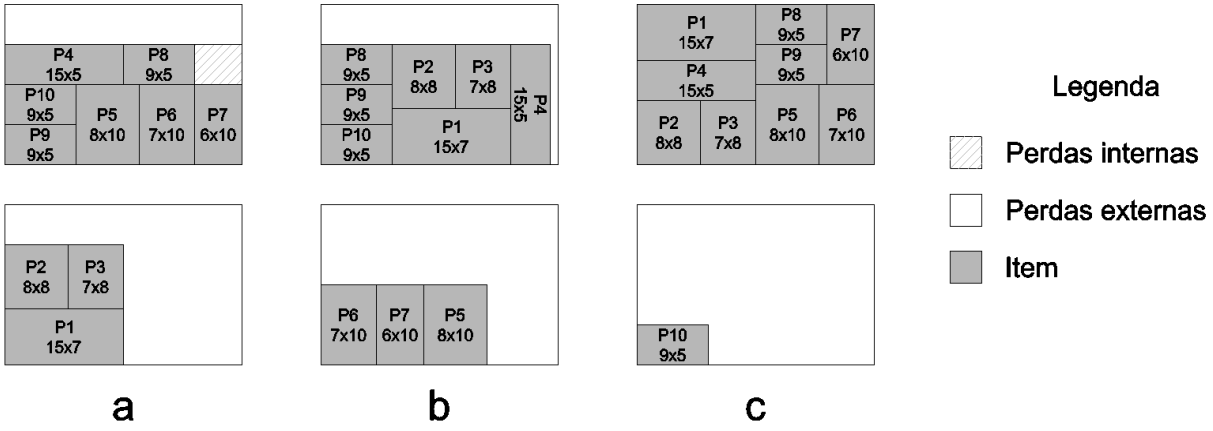
Devido ao tempo de execução dos testes preliminares do modelo matemático, utilizar instâncias de tamanhos moderados seria inviável, por isso foram criadas instâncias de testes com 4 e 5 itens. Para os demais algoritmos foram escolhidas as instâncias de Berkey, Wang, Martello e Vigo, compiladas no trabalho de Lodi, Martello e Vigo (1999a) e disponível em <http://or.dei.unibo.it/library/two-dimensional-bin-packing-problem>.

O conjunto de instâncias possui dez classe com cinquenta instâncias. Cada classe é dividida em cinco grupos contendo vinte, quarenta, sessenta, oitenta e cem itens, totalizando quinhentas instâncias. As seis primeiras classes foram geradas randomicamente utilizando o método proposto por Berkey e Wang (1987) enquanto as outras quatro classes foram criadas usando o método proposto por Martello e Vigo (1998). No trabalho

de Lodi, Martello e Vigo (1999a) há uma explicação detalhada da geração, bem como dos parâmetros utilizados em cada método.

A ideia inicial era considerar como soluções ótimas, aquelas que atingissem o limite inferior, todavia, dois problemas foram detectados. O primeiro é que algumas instâncias demoravam muito para atingir o limite inferior, enquanto outras atingiam muito rapidamente. O segundo problema foi que o ótimo em relação ao número de contêineres pode não ser o ótimo em relação a aproveitamento, como visto na Figura 50.

Figura 50 – Soluções ótimas segundo quantidade de contêineres (a), aproveitamento de sobras internas (b) e ideal (c).



Fonte: Autoria própria

Na Figura 50 é possível ver que a solução ótima do problema pode ter mais de uma face e os critérios para considerar uma solução ótima podem ser mais complexos do que apenas a quantidade de contêineres necessários para atender à lista de pedidos. A primeira solução foi obtida através do *Greedy Branch and Bound* usando o limite inferior como critério de parada, tal solução foi alcançada em $0,00086$ s.

A segunda solução foi obtida ao acrescentar o critério de que a sobra interna teria que ser reduzida, essa solução demorou um pouco mais, cerca de 250 s. A terceira poderia ser obtida ao tentar maximizar as sobras no último contêiner ou maximizar o tamanho de uma única sobra, mas não há garantias que em instâncias maiores, que demandem mais contêineres, os arranjos intermediários seriam otimizados também. Entretanto a otimização dos contêineres intermediários foge ao escopo do tema deste trabalho e ficará como proposta para trabalhos futuros.

6.1.3 Limites inferiores

Dell'Amico, Martello e Vigo (2002) apresentaram um método pseudo polinomial para obter limites inferiores para o problema. Esse método transforma os itens retangulares em quadrados através de sucessivos cortes. Esses quadrados são então unidos e o número de placas necessárias para comporta todos os itens passa a ser o limie inferior. Clautiaux, Jouglet e El Hayek (2007) aprimoraram esse método e geraram novos limites inferiores para o problema. Fleszar (2013) calculou esses limites inferiores e os disponibilizou no site <https://staff.aub.edu.lb/~kf09/research.htm>.

6.2 RESULTADOS COMPUTACIONAIS

6.2.1 Modelo matemático

Apesar do grande número de restrições em sua forma compacta, o número de variáveis é bem reduzido em relação à formulação não estagiada baseada em árvore, por exemplo. Fazendo uma análise do número total de variáveis temos:

- O vetor R_{n-1} contendo $n-1$ variáveis
- O vetor V_n contendo n variáveis
- As matrizes triangulares $Y_{n-1,n-1}$, $A_{n-1,n-1}$, $HST3_{n-1,n-1}$, $HST4_{n-1,n-1}$, $HSTB_{n-1,n-1}$, $HSTP_{n-1,n-1}$, $WST3_{n-1,n-1}$, $WST4_{n-1,n-1}$, $WSTB_{n-1,n-1}$, $WSTP_{n-1,n-1}$ contendo $\frac{(n-1)\cdot(n-2)}{2}$ variáveis cada;
- As matrizes triangulares $H_{n,n}$, $HST1_{n,n}$, $HST2_{n,n}$, $HSTA_{n,n}$, $W_{n,n}$, $WST1_{n,n}$, $WST2_{n,n}$, $WSTA_{n,n}$ contendo $\frac{n\cdot(n-1)}{2}$ variáveis cada;
- A matriz $X_{n,n}$ contendo n^2 variáveis;

Desse modo, a quantidade de variáveis desse modelo está definida assim:

$$nVar = n^2 + 8 \cdot \left(\frac{n\cdot(n-1)}{2}\right) + 10 \cdot \left(\frac{(n-1)\cdot(n-2)}{2}\right) + n + n - 1$$
$$nVar = n^2 + 8 \cdot \left(\frac{n^2-n}{2}\right) + 10 \cdot \left(\frac{n^2-3n+2}{2}\right) + n + n - 1$$
$$nVar = n^2 + \left(\frac{8n^2-8n}{2}\right) + \left(\frac{10n^2-30n+20}{2}\right) + n + n - 1$$
$$nVar = n^2 + 4n^2 - 4n + 5n^2 - 15n + 10 + n + n - 1$$
$$nVar = 10n^2 - 17n + 9$$

O número de restrições seguem o seguinte raciocínio:

- Uma restrição de ordem $n - 1$;
- Vinte e duas restrições de ordem n;
- Sessenta e cinco restrições de ordem $\frac{(n-1)\cdot(n-2)}{2}$
- Duas restrições de ordem $\frac{n\cdot(n-1)}{2}$

Desse modo, a quantidade total de restrições desse modelo fica definida assim:
$nRes = 2 \cdot \left(\frac{n(n-1)}{2}\right) + 65 \cdot \left(\frac{(n-1)\cdot(n-2)}{2}\right) + 22n + n - 1$
$nRes = 2 \cdot \left(\frac{n^2-n}{2}\right) + 65 \cdot \left(\frac{n^2-3n+2}{2}\right) + 23n - 1$
$nRes = n^2 - n + \frac{65n^2}{2} - \frac{195n}{2} + 65 + 23n - 1$
$nRes = \frac{2n^2+65n^2}{2} - \frac{195n-44n}{2} + 64$
$nRes = \frac{67n^2}{2} - \frac{151n}{2} + 64$

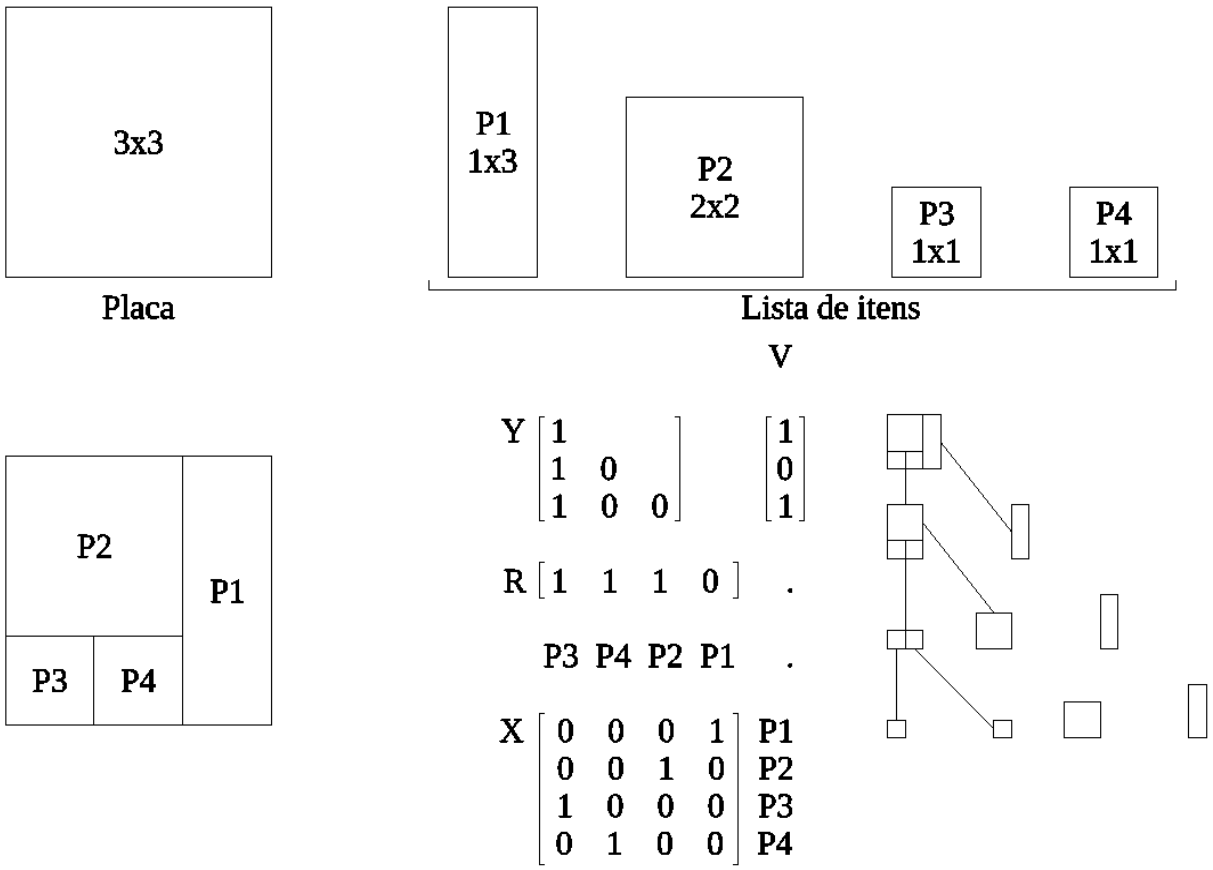
Assim, para um $n = 50$ seriam geradas cerca de 24.159 variáveis ou colunas para o método simplex. Além dessas, também seriam geradas as variáveis de folga ou excesso, provavelmente uma para cada inequação (80.039 no total). Assim, ficariam 104.198 variáveis e 80.039 restrições, valores altos, mas ainda assim, bem menores que os 66 milhões de variáveis encontrados por Furini, Malaguti e Thomopulos (2016). Além disso, a taxa de crescimento tanto das restrições, quanto das variáveis fica na ordem de $O(n^2)$. Sendo esse um dos maiores entraves para as formulações desse tipo (VELASCO, 2005).

A Figura 51 mostra como ficou o arranjo de corte definido pelo modelo para quatro itens. Nele é possível ver como a matriz de variáveis $x_{i,j}$ ordena os itens. E a ordem dos cortes ou uniões dos retângulos definida na matriz de variáveis $y_{i,j}$. O vetor de variáveis v_i define se os cortes de cada linha são verticais ou não (ou seja, horizontais), enquanto o vetor de variáveis r_j define se os retângulos da linha n estão rotacionados ou não. O tempo de execução do solver ficou em torno dos 5 segundos para a instância com 4 itens.

A Figura 52 mostra como ficou o arranjo de corte definido pelo modelo para cinco itens. É interessante notar que nem todos os cortes ocorreram na primeira coluna, como aconteceu no exemplo anterior. Aqui também é possível ver as rotações dos itens sendo aplicadas realmente. No exemplo anterior também foram aplicadas rotações, mas foram de itens quadrados, o que as torna irrelevante, deste caso as rotações foram aplicadas em itens cuja altura e a largura são diferentes e que, de outra forma, não seria possível a utilização de apenas uma placa.

A Figura 53 mostra como ficou o arranjo de corte definido pelo modelo para cinco placas, todavia com a placa tendo a metade do tamanho do exemplo da Figura 52. Nele é possível ver como o modelo trata essa situação, marcando o último corte como 0,

Figura 51 – Resultado do modelo com quatro itens.

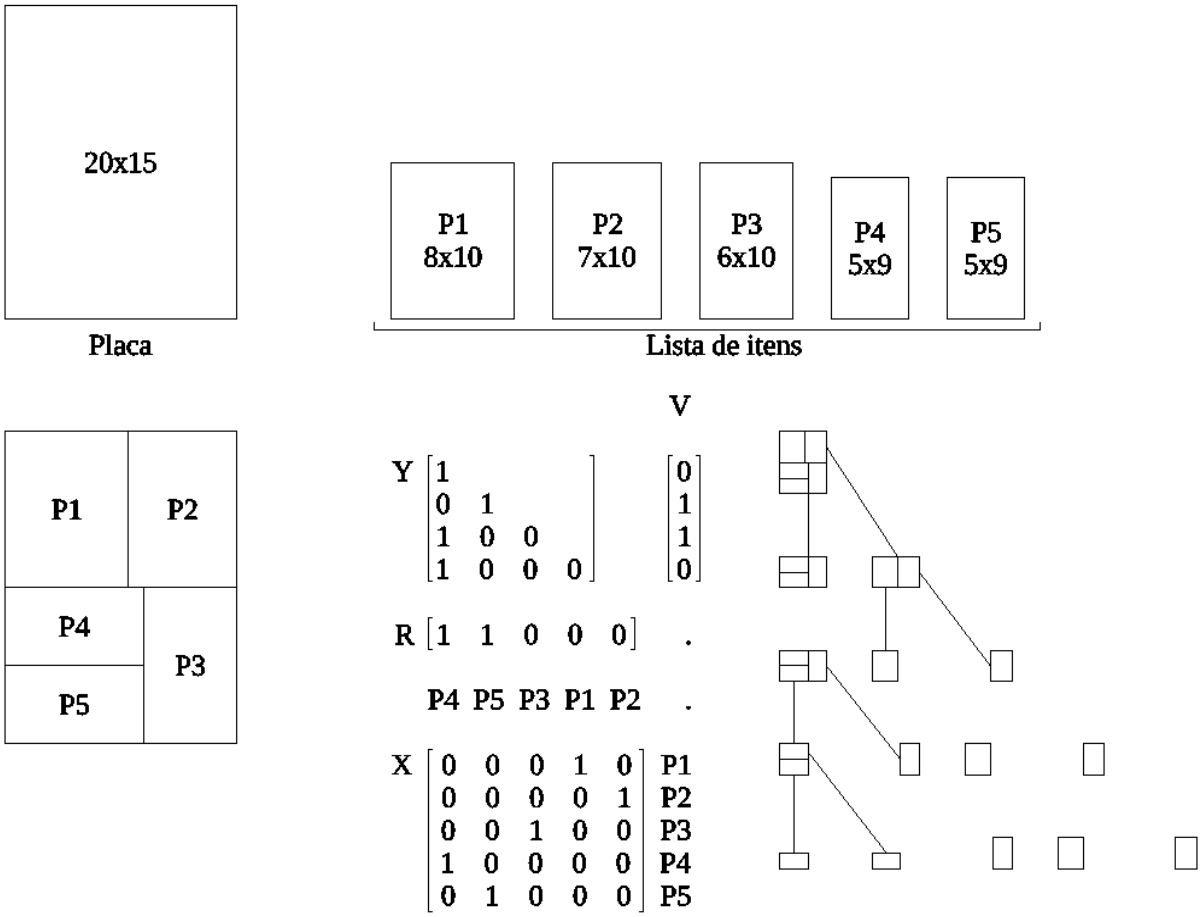


Fonte: Autoria própria

informando que não é possível unir os dois retângulos abaixo. Também é possível ver o descarte deste retângulo. É provável que, com mais itens, aconteceriam mais descartes e a situação que acontece na lista 1 da matriz Y (nenhuma variável com valor 1) se repita mais vezes. O tempo de execução do solver ficou em torno dos 1260 segundos para as instâncias com 5 itens e esse crescimento abrupto desencorajou a utilização do modelo para instâncias maiores.

Nas três figuras, sobretudo as duas últimas, é possível perceber que o número de placas ou contêineres necessários é dependente da quantidade de cortes. Como o número de contêineres cai uma unidade a cada corte e podem haver até um contêiner por item (caso extremo onde cada item ocupe um contêiner), é possível afirmar que o número de placas é simplesmente o número de itens menos o número de cortes gerados.

Figura 52 – Resultado do modelo com cinco itens.



Fonte: Autoria própria

6.2.2 Greedy Branch and Bound

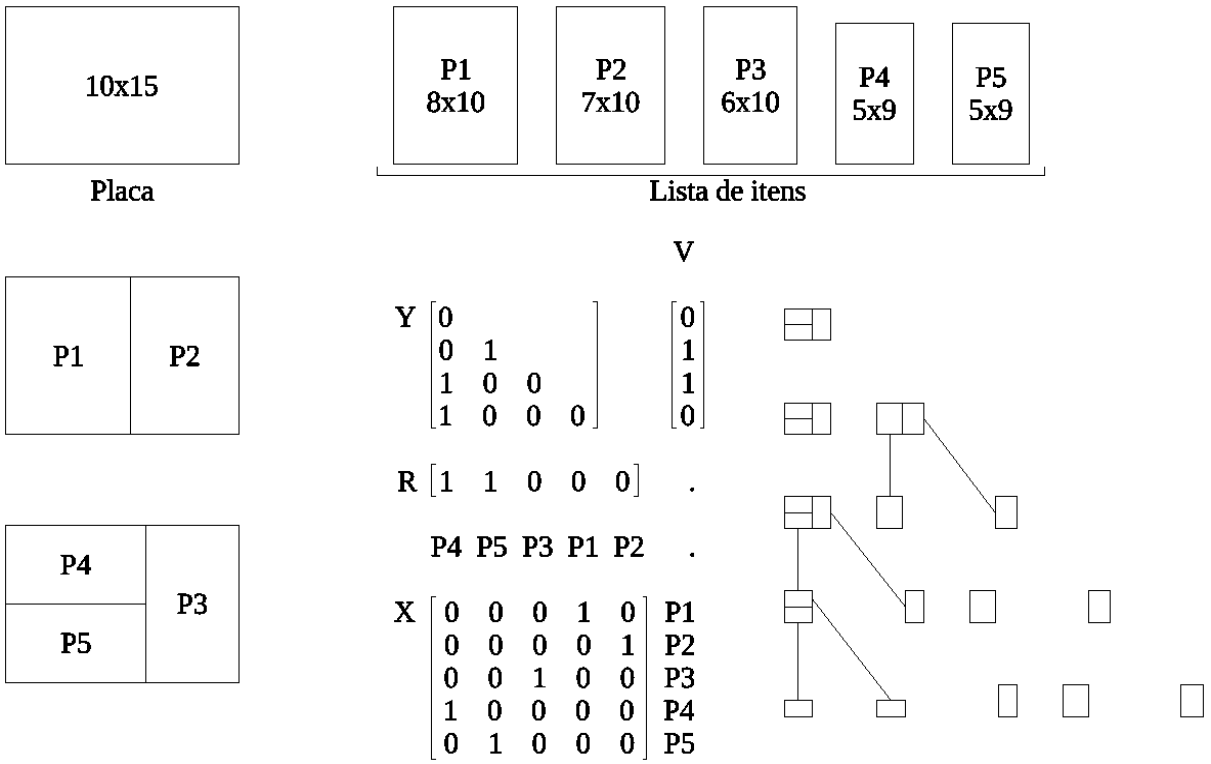
Devido as características expostas do problema, o algoritmo proposto nesse trabalho não usará limites inferiores como critérios de paradas, visando assim localizar arranjos com melhores distribuições dos itens. Como não foram usados limites inferiores como critério de parada, os tempos de execução ficariam muito altos, assim foram fixados em dez minutos. Os resultados estão listados na tabela 6.

Tabela 6 – Resultados do algoritmo de *Greedy Branch and Bound* para as instâncias de Berkey, Wang, Martello e Vigo

Classe	Grupo	Instância										total
		1	2	3	4	5	6	7	8	9	10	
1	20	8	5	8	6	6	10	7	7	8	8	73

continua na próxima página

Figura 53 – Resultado do modelo com cinco itens e que ocupam duas placas.



Fonte: Autoria própria

continuação da página anterior

Classe	Grupo	Instância										total
		1	2	3	4	5	6	7	8	9	10	
	40	11	13	18	17	18	15	14	18	12	15	151
	60	25	23	24	26	22	21	19	25	23	28	236
	80	30	32	31	32	30	33	34	34	35	30	321
	100	37	42	35	37	42	43	32	40	37	47	392
2	20	1	1	1	1	1	2	1	1	1	1	11
	40	2	2	2	2	2	2	2	2	2	2	20
	60	3	3	3	3	3	3	2	3	3	3	29
	80	3	4	3	4	4	4	4	4	4	4	38
	100	4	5	4	4	5	5	4	5	4	5	45
3	20	7	4	6	5	5	7	6	4	6	7	57
	40	8	11	13	11	14	11	10	14	10	9	111
	60	19	17	20	21	16	16	15	22	16	24	186
	80	23	23	25	25	23	29	27	26	28	24	253
	100	25	30	25	27	30	35	27	32	28	37	296
4	20	1	1	1	1	1	1	1	1	1	1	10
	40	2	2	2	2	2	2	2	2	2	2	20

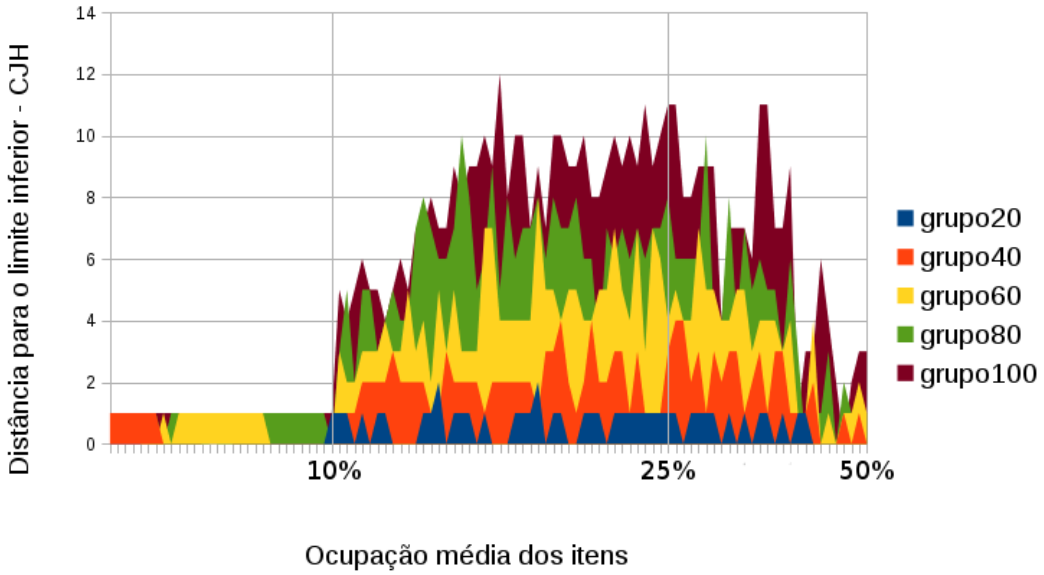
continua na próxima página

continuação da página anterior												
Classe	Grupo	Instância										total
		1	2	3	4	5	6	7	8	9	10	
	60	3	3	3	3	3	3	2	3	3	3	29
	80	3	3	4	4	4	4	4	4	4	4	38
	100	4	5	4	4	5	5	4	5	4	5	45
5	20	8	4	7	5	6	8	6	5	7	8	64
	40	9	13	15	14	15	13	11	17	11	12	130
	60	24	23	21	23	20	19	17	25	20	27	219
	80	29	27	27	30	29	31	36	29	34	28	300
	100	33	38	35	35	37	40	33	37	35	42	365
6	20	1	1	1	1	1	1	1	1	1	1	10
	40	2	2	2	2	2	2	2	2	2	2	20
	60	3	2	3	2	2	2	2	3	3	3	25
	80	3	3	3	3	3	4	4	4	4	3	34
	100	4	4	3	4	4	5	4	4	4	5	41
7	20	6	5	5	7	6	5	4	6	6	4	54
	40	11	15	11	15	10	13	14	12	9	14	124
	60	22	17	20	18	18	16	17	19	17	22	186
	80	27	28	24	27	26	28	29	27	27	28	271
	100	33	34	30	31	31	36	33	35	30	38	331
8	20	6	6	6	7	5	6	5	6	7	5	59
	40	13	14	13	12	10	13	12	12	10	14	123
	60	19	20	19	17	16	18	16	21	19	20	185
	80	25	26	23	26	30	27	26	26	25	29	263
	100	30	33	29	38	34	32	33	34	32	37	332
9	20	19	14	14	16	16	14	9	14	15	13	144
	40	26	32	29	32	27	29	25	26	22	33	281
	60	48	45	47	44	42	38	44	47	46	46	447
	80	58	60	58	53	62	64	60	58	52	61	586
	100	73	70	71	81	66	71	69	75	69	75	720
10	20	6	3	5	5	4	4	5	3	5	3	43
	40	9	9	11	7	7	6	9	9	9	10	86
	60	14	14	14	10	10	14	13	12	11	10	122
	80	17	15	13	18	17	16	18	13	17	18	162
	100	19	20	20	22	23	18	17	23	20	21	203
Total												8291

Apenas uma das instâncias parou por chegar ao fim da execução do algoritmo, a instância 4 da classe 9 com 20 itens, todas as demais pararam por atingir o tempo

limite. Todavia, cento e vinte e cinco delas atingiram o limite inferior para a quantidade de contêineres e duzentas e sessenta ficaram a até um contêiner do limite inferior. Foi verificada uma estreita relação entre a área percentual de cada item em relação ao contêiner e seu distanciamento do limite inferior.

Figura 54 – Distância entre os resultados e o limite inferior em relação a área média que os itens ocupam em cada instância.



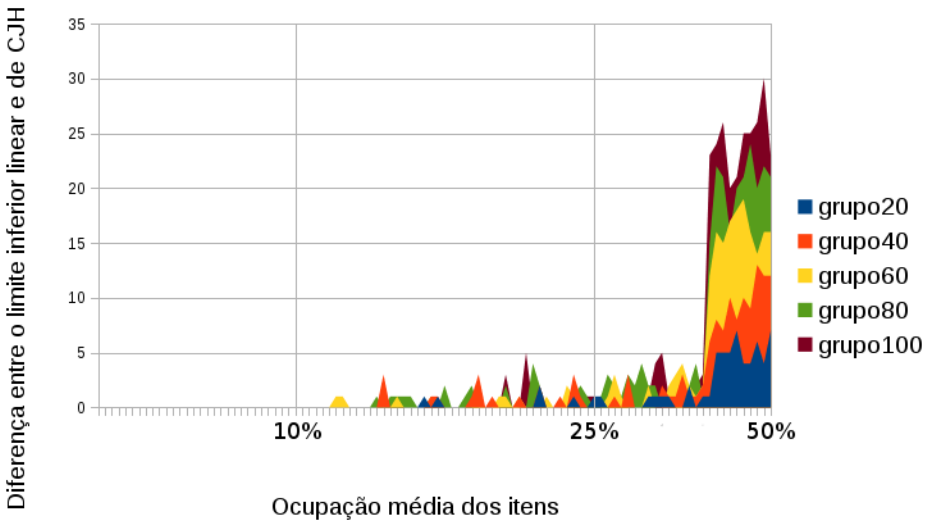
Fonte: Autoria própria

O gráfico da Figura 54 mostra que, no conjunto de instâncias testado, quando os itens ocupavam em média até aproximadamente 10% da área total do contêiner, o algoritmo ficou a um do limite inferior executando por cerca de dez minutos. A partir dos 10% até mais ou menos 25% há um crescimento da distância entre o alcançado e o limite inferior e uma nova redução a partir desse valor. Para tentar entender o motivo desse comportamento, foi feita uma comparação entre a área dos itens e o aproveitamento do limite inferior.

Pelo gráfico da Figura 55 é possível notar que quanto maior a área do item em relação ao contêiner, maior também será a distância entre o limite inferior e o limite linear. Esse limite linear é o desperdício mínimo, visto que é apenas a soma das áreas dos itens dividido pela área do contêiner. A partir desses gráficos, pode-se inferir que a velocidade de convergência do algoritmo depende inversamente da área dos itens, mas diretamente do desperdício final da solução ótima. Seguindo essa hipótese, o valor de 25% seria o limite para o qual o desperdício ultrapassa a importância da área do item para definição da velocidade de convergência.

Outra informação que é possível retirar desse gráfico é que, como esperado, a quantidade de itens influi na qualidade das soluções encontradas no tempo delimitado. A partir do formato serrilhado do gráfico pode-se inferir ainda que a área média dos

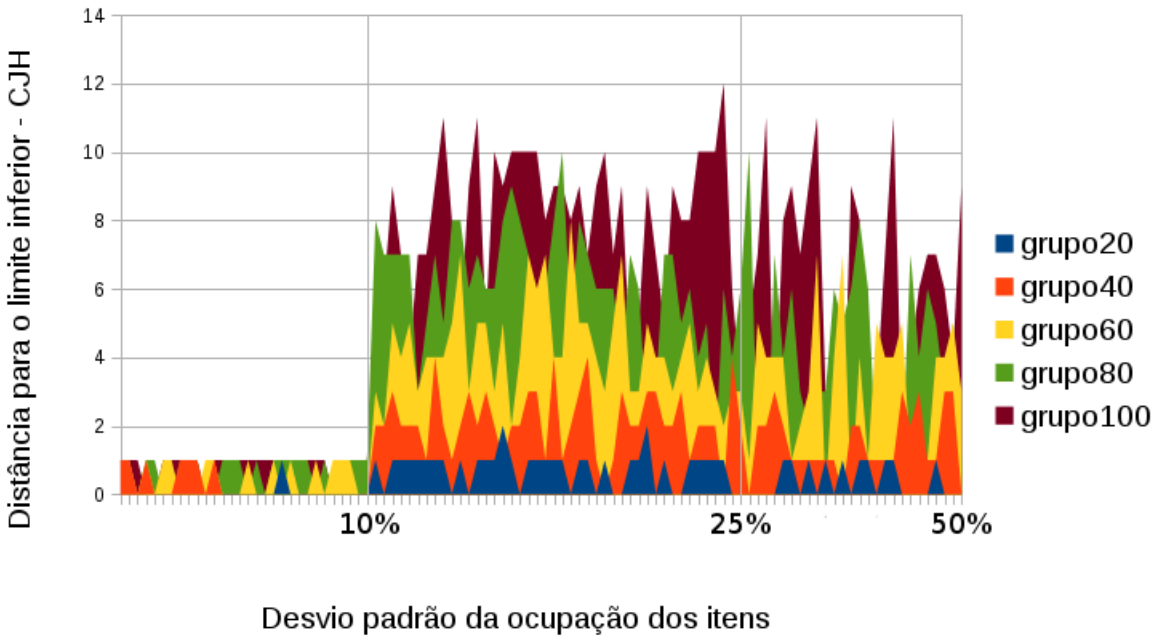
Figura 55 – Distância entre o limite inferior e o limite inferior linear (menor desperdício possível) em relação a área média que os itens ocupam em cada instância.



Fonte: Autoria própria

itens e seu desperdício final não são os únicos fatores relevantes na velocidade de convergência do algoritmo.

Figura 56 – Distância entre os resultados e o limite inferior em relação ao desvio padrão das áreas dos itens de cada instância.



Fonte: Autoria própria

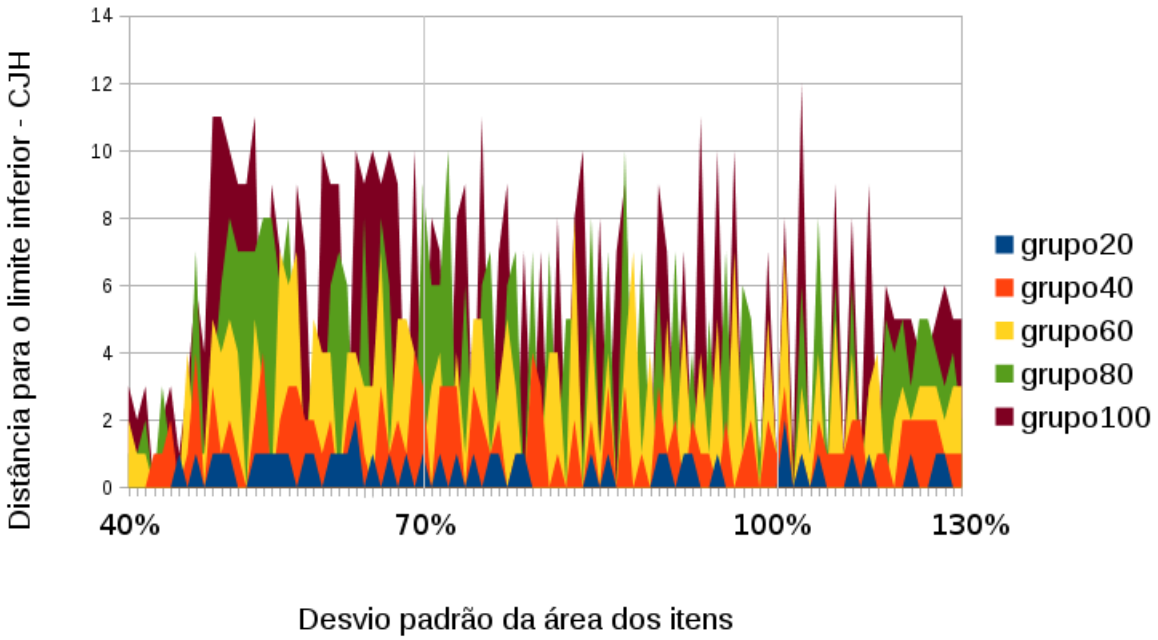
O gráfico da Figura 56 mostra a relação entre a distância do limite inferior e a heterogeneidade dos itens das instâncias. Nesse gráfico é possível notar essa correlação, porém mais fraca, denotada pela irregularidade na curva de crescimento.

Aqui novamente vê-se a relação da quantidade de itens e a distância entre a solução encontrada e o limite inferior. Mas, semelhante ao que ocorre com a área, essa relação só se evidencia a partir de certo valor de desvio (algo próximo a 9%). Esse crescimento se mantém até cerca de 15%, mas a média dos valores cai lentamente após isso.

Vale salientar que os valores do eixo X dos gráficos são referentes ao percentual da área do contêiner. Assim, 10% implica dizer um décimo da área do contêiner e não um décimo da área média dos itens. Levando isso em consideração, um desvio padrão de 25% em uma instância com área média dos itens de 50% do contêiner, implica dizer que os itens variam, em média, de 25% a 75% da área do contêiner e não de 37,5% a 62,5% (como seria se fosse pela área média dos itens).

Uma comparação pela distribuição pela variação da área relativa à média dos itens é apresentada na Figura 57. A partir desse gráfico não se pode chegar a conclusões mais claras, visto uma baixa correlação entre as variáveis, de modo que pode-se afirmar que o tamanho dos itens em relação ao tamanho do contêiner é mais importante que a relação do tamanho dos itens com a média dos mesmos.

Figura 57 – Distância entre os resultados e o limite inferior em relação a homogeneidade das áreas dos itens de cada instância.



Fonte: Autoria própria

A quantidade total de contêineres obtidos pelo algoritmo de *Greedy Branch and Bound* para atender todas as quinhentas instâncias desse conjunto foi de 8291 e o limite inferior calculado por utilizando o método de Clautiaux, Jouglet e El Hayek (2007) é de 6892 contêineres para atendê-las. Essa diferença ainda é grande (cerca de 20%), mas vale salientar que esse é um método exato e que esse comportamento era previsível, pois o

algoritmo foi interrompido em apenas 10 minutos, tempo insuficiente considerando a complexidade encontrada.

Os resultados em relação ao limites inferiores podem ser encontrados na tabela 7. Nessa tabela é possível ver que a maior concentração de resultados que atingiram o limite inferior estão nas classes 2, 4 e 6, classes compostas por itens de pequeno tamanho tamanho em relação ao tamanho do contêiner. Outra classe que também concentra uma boa quantidade de soluções que atingem o limite inferior é a classe 9, que é composta por itens de tamanho relativamente grandes em relação ao tamanho do contêiner.

Tabela 7 – Resultados do algoritmo de *Greedy Branch and Bound* em relação ao limite inferior para as instâncias de Berkey, Wang, Martello e Vigo

Classe	Grupo	Instância										total
		1	2	3	4	5	6	7	8	9	10	
1	20	1,14	**1,00**	1,14	1,20	**1,00**	1,11	1,17	1,17	1,14	**1,00**	1,11
	40	1,22	1,18	1,20	1,21	1,20	1,07	1,27	1,06	1,09	1,36	1,18
	60	1,14	1,28	1,14	1,18	1,29	1,24	1,27	1,19	1,28	1,17	1,21
	80	1,25	1,28	1,15	1,23	1,15	1,18	1,10	1,17	1,21	1,20	1,19
	100	1,32	1,35	1,30	1,23	1,35	1,19	1,14	1,21	1,19	1,24	1,25
2	20	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	2,00	**1,00**	**1,00**	**1,00**	**1,00**	1,10
	40	2,00	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	1,05
	60	**1,00**	1,50	**1,00**	**1,00**	1,50	1,50	**1,00**	**1,00**	1,50	**1,00**	1,16
	80	**1,00**	1,33	**1,00**	1,33	1,33	1,33	1,33	1,33	**1,00**	1,33	1,23
	100	**1,00**	1,25	1,33	**1,00**	1,25	1,25	**1,00**	1,25	**1,00**	1,25	1,15
3	20	1,40	1,33	1,20	1,25	1,25	1,17	1,50	**1,00**	1,20	**1,00**	1,21
	40	1,33	1,38	1,18	1,22	1,17	1,10	1,25	1,08	1,43	1,29	1,22
	60	1,27	1,42	1,54	1,50	1,45	1,33	1,36	1,57	1,23	1,41	1,41
	80	1,44	1,35	1,47	1,39	1,44	1,53	1,42	1,24	1,33	1,33	1,39
	100	1,39	1,43	1,39	1,35	1,43	1,35	1,42	1,45	1,33	1,28	1,38
4	20	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**
	40	2,00	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	1,05
	60	**1,00**	1,50	1,50	1,50	1,50	1,50	**1,00**	**1,00**	1,50	**1,00**	1,26
	80	**1,00**	**1,00**	1,33	1,33	1,33	1,33	1,33	1,33	1,33	1,33	1,27
	100	1,33	1,25	1,33	**1,00**	1,25	1,25	1,33	1,25	**1,00**	1,25	1,22
5	20	1,14	**1,00**	**1,00**	**1,00**	1,20	**1,00**	1,20	**1,00**	1,17	1,14	1,08
	40	1,29	1,30	1,07	1,17	1,15	1,08	1,10	1,06	1,22	1,20	1,15
	60	1,20	1,44	1,24	1,21	1,33	1,27	1,31	1,39	1,33	1,17	1,28

continua na próxima página

continuação da página anterior												
Classe	Grupo	Instância										total
		1	2	3	4	5	6	7	8	9	10	
	80	1,32	1,29	1,13	1,30	1,32	1,24	1,38	1,16	1,31	1,27	1,27
	100	1,43	1,41	1,52	1,40	1,37	1,21	1,38	1,28	1,35	1,20	1,34
6	20	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**
	40	2,00	2,00	**1,00**	**1,00**	**1,00**	2,00	**1,00**	**1,00**	2,00	2,00	1,33
	60	1,50	**1,00**	1,50	**1,00**	**1,00**	**1,00**	**1,00**	1,50	1,50	**1,00**	1,19
	80	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	1,33	1,33	1,33	1,33	**1,00**	1,13
	100	1,33	1,33	**1,00**	1,33	1,33	1,25	1,33	1,33	1,33	1,25	1,28
7	20	1,20	1,25	1,25	1,17	1,20	**1,00**	**1,00**	1,20	1,20	**1,00**	1,15
	40	1,22	1,36	1,22	1,25	1,11	1,30	1,40	1,09	1,29	1,27	1,25
	60	1,47	1,31	1,33	1,38	1,38	1,23	1,31	1,27	1,31	1,29	1,33
	80	1,50	1,27	1,33	1,35	1,30	1,40	1,32	1,42	1,35	1,33	1,36
	100	1,38	1,42	1,43	1,35	1,41	1,44	1,43	1,40	1,30	1,31	1,38
8	20	1,20	1,20	1,20	1,17	**1,00**	1,20	1,25	1,20	1,17	1,25	1,18
	40	1,30	1,27	1,44	1,09	1,25	1,30	1,33	1,20	1,25	1,27	1,27
	60	1,19	1,33	1,27	1,31	1,23	1,38	1,23	1,40	1,36	1,33	1,30
	80	1,32	1,30	1,28	1,44	1,25	1,42	1,37	1,30	1,32	1,38	1,34
	100	1,30	1,43	1,38	1,41	1,36	1,39	1,43	1,36	1,33	1,32	1,37
9	20	**1,00**	1,08	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	1,01
	40	1,08	**1,00**	1,04	1,03	**1,00**	**1,00**	1,04	**1,00**	1,05	**1,00**	1,02
	60	1,04	1,02	1,02	**1,00**	1,02	1,03	1,10	**1,00**	1,02	1,02	1,03
	80	**1,00**	1,05	1,02	1,02	1,02	1,03	1,02	**1,00**	1,06	1,02	1,02
	100	1,03	1,09	1,04	1,04	1,02	**1,00**	1,05	1,03	1,06	1,04	1,04
10	20	**1,00**	**1,00**	1,25	1,25	**1,00**	**1,00**	**1,00**	1,50	**1,00**	1,50	1,10
	40	1,13	1,29	1,22	1,17	1,17	1,20	1,29	1,29	1,29	1,25	1,23
	60	1,27	1,27	1,27	1,43	1,25	1,17	1,30	1,33	1,38	1,25	1,28
	80	1,42	1,50	1,18	1,38	1,31	1,23	1,29	1,30	1,42	1,29	1,33
	100	1,36	1,33	1,33	1,29	1,35	1,38	1,31	1,28	1,25	1,40	1,33
Total												1,20

6.2.3 Heurísticas

Os testes realizados nesta subsessão seguem os mesmo moldes dos executados na subsessão anterior. Com a diferença que, por se tratar de algoritmos heurísticos, os algoritmos foram executados ao menos trinta vezes e feita a moda estatística dos resultados.

Como os algoritmos podem executar sem estarem atrelados ao time, cada algoritmo foi executado independentemente e depois executados dentro do time. O primeiro, em ordem, foi o GRASP. As configurações utilizadas foram $\alpha = 0,2$ e 40 iterações. Essa configuração foi definida para todas as instâncias, através dos resultados preliminares obtidos com um conjunto pequeno de instâncias.

O GRASP alcançou 7084 contêineres na moda dos resultados para o conjunto de instâncias testados. Com esse resultado, ele estaria no mesmo patamar que os melhores resultados obtidos por Fleszar (2013) e a 21 contêineres para os resultados de Polyakovsky e M'Hallah (2009), que foi é o melhor encontrado em até o momento pela revisão bibliográfica deste trabalho. Todavia, seu melhor resultado foi de 7049, o que baixa em 14 o valor do melhor encontrado na literatura. O tempo médio do GRASP foi de 6,2 segundos para estagnar, na média, na iteração 12, mas atingiu 58,2 segundos em instâncias grandes.

Aqui vale relembrar que dentro da função objetivo há uma heurística de preenchimento, de modo que o número baixo de iterações é alcançado pelo fato da heurística arranjar os itens dentro dos contêineres seguindo suas políticas de otimização. Em muitos casos, essas políticas da heurística são eficientes o suficiente para que com pouco ou nenhum esforço da metaheurística se alcance bons resultados.

A Busca Tabu mostrou-se o algoritmo mais inconstante do todos. Com o segundo melhor resultado entre os melhores resultados e o segundo pior entre os piores resultados, a Busca Tabu demostra que a solução inicial é muito importante para a qualidade final, com uma solução inicial randômica, a Busca Tabu abriu um distância muito grande entre seu melhor e seu pior resutado. Os parâmetros utilizados nos testes foram 200 iterações e tamanho da população igual ao dobro do número de itens de cada instância. A Bsuca Tabu obteve 7131 como melhor resultado, mas a moda ficou com 7230.

A Busca Tabu com a fase construtiva do GRASP deveria ser bem melhor que sua contrapartida, mas não foi bem isso que aconteceu. Apesar da fase construtiva do GRASP permitir boas soluções iniciais, o desenrolar da Busca Tabu não conseguiu cumprir com as expectativas. Talvez os múltiplos reinícios do GRASP tenha feito falta na Busca Tabu. Além das configurações da Busca Tabu simples, a Busca Tabu com GRASP contou ainda com o $\alpha = 0,2$. Nesse cenário, obteve 7127 como melhor resultado, mas a moda ficou com 7211.

O Algoritmo Genético mostrou-se relativamente rápido (média de 0,57 segundos com pior caso de 2,2 contra média de 6,2 e pior caso de 58,2 do GRASP para estagnar), porém pouco eficiente em relação aos resultados do GRASP, ficando a moda com 7247 e como melhor resultado 7196 contêineres. Ele executou por 300 iterações, mas estagnava em torno da iteração 73, seus demais parâmetros foram uma população de 300 indivíduos, uma população elite de 60 indivíduos e uma população imigrante (indivíduos novos em cada iteração) de 30 indivíduos.

Combinar o Algoritmo Genético com a fase construtiva do GRASP se mostrou interessante em termos de resultados. Os tempos permaneceram quase os mesmos do genético convencional. Porém os resultados se mostraram consideravelmente melhores, 7161 do genético com GRASP contra 7247 do convencional. Sendo o melhor resultado encontrado com 7131 contêineres.

Esse resultado era esperado, uma vez que a variação da heurística *HHDHeuristic* escolhida foi inicialmente pensada para uso com uma lista de itens ordenados pela área e a gulosidade do GRASP se baseia na nesse mesmo conceito.

A combinação do Algoritmo Genético com a Religação de Caminhos mostrou-se ainda mais interessante, em termos de resultados. mas desastrosa em relação ao tempo. A Religação de Caminhos foi utilizada duas vezes em cada iteração, mas seu peso no tempo total foi alto, a média de tempo foi de 15,85 segundos (contra 6,2 do GRASP) e seus resultados não chegaram a bater os do GRASP, 7115 contra 7084 e seu melhor resultado ficou em 7090 (contra 7049 do GRASP).

O último dos algoritmos individuais é o BRKGA. Com relação as configurações, ele utiliza as mesmas utilizadas no Algoritmo Genético com a diferença que, neste caso, há a tendência para o cruzamento característico do BRKGA (afinal, sem a tendência na hora do sorteio, ele não seria "viciado"). A tendência escolhida, a partir de testes preliminares, foi de 0,7 ou 70% de chances do gene vir do progenitor elite. O BRKGA conseguiu melhorar os resultados do Algoritmo Genético convencional, mas ficou aquém das combinações do Genético com o GRASP e com a Religação de Caminhos. Foram 7172 contra 7247 do Algoritmo Genético, 7161 do Genético com GRASP e 7115 do Genético com Religação de Caminhos. Mas, ainda assim, aquém dos 7084 do GRASP. Em relação ao tempo, foram em média 0,95 segundos de execução até estagnar ou 97 iterações.

O time assíncrono executou os algoritmos com as mesmas configurações descritas anteriormente, exceto pelo número de iterações. Como a ideia é que os algoritmos vão melhorando as soluções presentes na memória comum, não faz sentido deixá-los executar por longos períodos. Assim, os algoritmos executavam por curtos períodos e enviavam suas soluções à memória compartilhada.

Cada algoritmo foi usado tanto para gerar soluções novas (agentes de inicialização) como para melhorar soluções existentes na memória compartilhada, com exceção do GRASP e BRKGA. Por ser um algoritmo sem memória, o GRASP não toma proveito da existência de uma solução prévia com bons resultados, diferente do Algoritmo Genético ou Busca Tabu. O BRKGA até poderia utilizar-se de soluções prévias, porem isso demandaria o acréscimo de uma fase de "codificação" para transformar uma solução padrão no formato característico do BRKGA (normalizado no intervalo de 0 a 1) e como já haviam outros algoritmos que poderiam promover melhorias, foi tomada a decisão de deixá-lo apenas como algoritmo de inicialização.

Tabela 8 – Configuração de cada agente de inicialização e melhoria

Configuração de cada agente

	Tipo	Entrada	Iterações
GRASP	Inicialização	-	5
Busca Tabu	Inicialização	-	20
Busca Tabu + GRASP	Inicialização	-	20
Genético	Inicialização	-	50
Genético + GRASP	Inicialização	-	50
Genético + R. Caminhos	Inicialização	-	20
BRKGA	Inicialização	-	50
Busca Tabu	Melhoria	Melhor atual	10
Genético	Melhoria	10 melhores	20
Genético + GRASP	Melhoria	10 melhores	20
R. Caminhos	Melhoria	10 melhores	1
Cruzamento Genético	Melhoria	10 melhores	1

Tabela 9 – Comparativo entre os algoritmos integrantes do time assíncrono e o próprio time

Comparativo entre os algoritmos isolados e do time

	Melhor	Pior	Média	Moda	Tempo (s)	Nº soluções	Iterações
GRASP	7049	7108	7083,475	7084	6,221	76688,8	11,83
Busca Tabu	7131	7278	7210,470	7230	-	-	-
Busca Tabu + GRASP	7127	7278	7208,326	7211	-	-	-
Genético	7196	7282	7245,548	7247	0,576	11865,7	72,66
Genético + GRASP	7131	7185	7163,158	7161	0,562	10649,1	65,01
Genético + R. Caminhos	7090	7139	7114,145	7115	15,851	123409,5	88,93
BRKGA	7140	7190	7169,082	7172	0,959	15775,7	96,742
Time assíncrono	7045	7117	7085,138	7085	15,627	-	-

Os algoritmos de melhoria foram todos os demais, com o acréscimo de mais dois algoritmos. O primeiro é a Religação de Caminhos isolada. Basicamente pega dez soluções da memória compartilhada, sorteia duas delas e tenta combiná-las. O segundo algoritmo foi o cruzamento do Algoritmo Genético, onde foi feita a mesma coisa que a Religação de Caminhos. As configurações de cada algoritmo está disposta na tabela 8.

A tabela 9 compara os algoritmos isoladamente e dentro do time. É importante notar que o time tem a possibilidade de executar tanto em rede, quanto em várias *threads*, o que inviabiliza a contabilização do número de iterações e soluções testadas, além de tornar irrelevante o comparativo de tempo.

Os melhores resultados estão dispostos na tabela 10. Nela é possível ver quais foram os melhores resultados para cada instância. A tabela 11 mostra um comparativo desses resultados com o limite inferior de Clautiaux, Jouglet e El Hayek (2007).

Tabela 10 – Melhoress resultados do Time Assíncrono para as instâncias de Berkey, Wang, Martello e Vigo

Classe	Grupo	Instância										total
		1	2	3	4	5	6	7	8	9	10	
1	20	7	5	7	5	6	9	6	6	7	8	66
	40	9	11	15	14	15	14	11	17	11	11	128
	60	22	18	21	22	17	17	15	21	18	24	195
	80	24	25	27	26	26	28	31	29	29	25	270
	100	28	31	27	30	31	36	28	33	31	38	313
2	20	1	1	1	1	1	1	1	1	1	1	10
	40	1	2	2	2	2	2	2	2	2	2	19
	60	3	2	3	3	2	2	2	3	2	3	25
	80	3	3	3	3	3	3	3	3	4	3	31
	100	4	4	3	4	4	4	4	4	4	4	39
3	20	5	3	5	4	4	6	4	4	5	7	47
	40	6	8	11	10	12	10	8	13	8	8	94
	60	16	12	13	14	12	12	11	14	13	17	134
	80	16	17	17	18	17	19	20	21	21	18	184
	100	19	22	19	20	22	26	20	22	22	29	221
4	20	1	1	1	1	1	1	1	1	1	1	10
	40	1	2	2	2	2	2	2	2	2	2	19
	60	3	2	3	3	2	2	2	3	2	3	25
	80	3	3	3	3	3	3	3	3	4	3	31
	100	3	4	3	4	4	4	3	4	4	4	37
5	20	7	4	7	5	5	8	5	5	6	7	59
	40	8	10	14	12	13	12	10	16	9	10	114
	60	20	16	18	19	15	15	14	18	16	23	174
	80	22	22	24	23	22	25	26	26	27	22	239
	100	24	28	24	26	28	33	25	29	27	35	279
6	20	1	1	1	1	1	1	1	1	1	1	10
	40	1	2	2	2	2	2	2	2	1	1	17
	60	2	2	2	2	2	2	2	2	2	3	21
	80	3	3	3	3	3	3	3	3	3	3	30
	100	3	4	3	3	3	4	3	3	3	4	33
7	20	5	5	5	6	6	5	4	6	6	4	52
	40	9	12	9	13	9	11	10	11	8	12	104
	60	16	13	16	14	13	13	14	16	14	18	147

continua na próxima página

continuação da página anterior

Classe	Grupo	Instância										total
		1	2	3	4	5	6	7	8	9	10	
	80	20	23	19	21	22	21	23	21	22	23	215
	100	26	25	23	25	23	28	25	27	25	31	258
8	20	6	6	5	6	5	6	4	5	6	4	53
	40	11	12	10	11	8	11	10	10	9	12	104
	60	16	16	16	14	13	14	13	16	15	17	150
	80	20	21	19	19	25	21	21	21	21	23	211
	100	25	25	22	29	27	25	25	27	26	30	261
9	20	19	13	14	16	16	14	9	14	15	13	143
	40	24	32	28	31	27	29	24	26	21	33	275
	60	46	44	46	44	41	37	40	47	45	45	435
	80	58	57	57	52	61	62	59	58	49	60	573
	100	71	64	68	78	65	71	66	73	65	72	693
10	20	6	3	4	4	4	4	5	3	5	3	41
	40	8	8	9	6	6	6	7	7	8	8	73
	60	11	12	11	8	8	12	10	10	9	8	99
	80	13	11	11	14	14	13	14	10	12	14	126
	100	15	16	16	17	18	13	14	18	16	15	158
Total												7045

Tabela 11 – Melhores resultados do Time Assíncrono em relação ao limite inferior para as instâncias de Berkey, Wang, Martello e Vigo

Classe	Grupo	Instância										total
		1	2	3	4	5	6	7	8	9	10	
1	20	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**
	40	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**
	60	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**
	80	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**
	100	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**
2	20	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**
	40	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**
	60	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**
	80	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**
	100	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**

continua na próxima página

continuação da página anterior

Classe	Grupo	Instância										total
		1	2	3	4	5	6	7	8	9	10	
3	20	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**
	40	**1,00**	**1,00**	**1,00**	1,11	**1,00**	**1,00**	**1,00**	**1,00**	1,14	1,14	1,04
	60	1,07	**1,00**	**1,00**	**1,00**	1,09	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	1,02
	80	**1,00**	**1,00**	**1,00**	**1,00**	1,06	**1,00**	1,05	**1,00**	**1,00**	**1,00**	1,01
	100	1,06	1,05	1,06	**1,00**	1,05	**1,00**	1,05	**1,00**	1,05	**1,00**	1,03
4	20	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**
	40	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**
	60	**1,00**	**1,00**	1,50	1,50	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	1,10
	80	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	1,33	**1,00**	1,03
	100	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**
5	20	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**
	40	1,14	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	1,01
	60	**1,00**	**1,00**	1,06	**1,00**	**1,00**	**1,00**	1,08	**1,00**	1,07	**1,00**	1,02
	80	**1,00**	1,05	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	1,04	1,04	**1,00**	1,01
	100	1,04	1,04	1,04	1,04	1,04	**1,00**	1,04	**1,00**	1,04	**1,00**	1,03
6	20	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**
	40	**1,00**	2,00	**1,00**	**1,00**	**1,00**	2,00	**1,00**	**1,00**	**1,00**	**1,00**	1,20
	60	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**
	80	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**
	100	**1,00**	1,33	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	1,03
7	20	**1,00**	1,25	1,25	**1,00**	1,20	**1,00**	**1,00**	1,20	1,20	**1,00**	1,11
	40	**1,00**	1,09	**1,00**	1,08	**1,00**	1,10	**1,00**	**1,00**	1,14	1,09	1,05
	60	1,07	**1,00**	1,07	1,08	**1,00**	**1,00**	1,08	1,07	1,08	1,06	1,05
	80	1,11	1,05	1,06	1,05	1,10	1,05	1,05	1,11	1,10	1,10	1,08
	100	1,08	1,04	1,10	1,09	1,05	1,12	1,09	1,08	1,09	1,07	1,08
8	20	1,20	1,20	**1,00**	**1,00**	**1,00**	1,20	**1,00**	**1,00**	**1,00**	**1,00**	1,06
	40	1,10	1,09	1,11	**1,00**	**1,00**	1,10	1,11	**1,00**	1,13	1,09	1,07
	60	**1,00**	1,07	1,07	1,08	**1,00**	1,08	**1,00**	1,07	1,07	1,13	1,06
	80	1,05	1,05	1,06	1,06	1,04	1,11	1,11	1,05	1,11	1,10	1,07
	100	1,09	1,09	1,05	1,07	1,08	1,09	1,09	1,08	1,08	1,07	1,08
9	20	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**
	40	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**
	60	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**
	80	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**
	100	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**

continua na próxima página

continuação da página anterior

Classe	Grupo	Instância										total
		1	2	3	4	5	6	7	8	9	10	
10	20	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	1,50	**1,00**	1,50	1,10
	40	**1,00**	1,14	**1,00**	**1,00**	**1,00**	1,20	**1,00**	**1,00**	1,14	**1,00**	1,05
	60	**1,00**	1,09	**1,00**	1,14	**1,00**	**1,00**	**1,00**	1,11	1,13	**1,00**	1,05
	80	1,08	1,10	**1,00**	1,08	1,08	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	1,03
	100	1,07	1,07	1,07	**1,00**	1,06	**1,00**	1,08	**1,00**	**1,00**	**1,00**	1,03
Total												1,02

Alguns desses resultados merecem uma atenção especial, eles mostram tanto a força, quanto a fragilidade da utilização de uma abordagem por codificação. O primeiro desses resultados, na realidade é um conjunto de resultados. A classe de instância 9 (a que possui os maiores itens) obteve seus melhores resultados na primeira iteração, ou melhor, antes dela. Isso porque a heurística de montagem *HHDHeuristic* montou o melhor resultado sem o auxílio das metaheurísticas. Esse comportamento se repetiu, independente se a solução era gerada aleatoriamente (Busca Tabu, Algoritmo Genético, BRKGA) ou com gulosidade (GRASP ou combinados com ele). Outras instâncias também obtiveram esse comportamento, mas mais espaçadamente, apenas a classe 9 apresentou esse comportamento em sua totalidade.

O segundo resultado que merece destaque é o resultado para a instância 6 do grupo de 40 itens da classe 6. Em primeiro lugar, essa classe apresentou os menores desvios padrões, com exceção de uma instância (instância 8 do grupo de 100 itens), todos os algoritmos ficaram com o desvio padrão zerados. Todavia isso não significa que alcançaram o ótimo. O resultado da instância 6 do grupo de 40 itens deixa isso claro (Figura 58).

Como pode ser visto, o item "P12" poderia ser facilmente alocado ao lado dos itens "P24" e "P16", se a ordem do corte fosse primeiro vertical e depois horizontal, mas o algoritmo *HHDHeuristic* opta por realizar primeiro o corte horizontal e depois o vertical pois gerará a maior sobra no momento do corte do item "P24". Esse comportamento impediu que o melhor resultado fosse obtido e nenhuma metaheurística conseguiu contornar esse problema com o limite de 30 execuções com as configurações padronizadas.

Apenas o GRASP conseguiu contornar isso (Figura 59), porém com um número muito maior de iterações, estagnando por volta de 850 iterações contra 12 das outras instâncias. É provável que a Busca Tabu alcançasse esse resultado com muitas execuções (lêia-se reinícios), todavia não foi alcançado nos testes realizados. O resultado da Figura 59, apensar de válido, não está sendo contabilizado nos resultados oficiais, isto porque as configurações que levaram a ele são muito diferentes das configurações escolhidas para as demais instâncias.

Tabela 12 – Comparativo feito por Mariano (2014) com o acréscimo dos resultados do presente trabalho

Comparativo entre alguns algoritmos para a classe 10

	Quantidade de itens					
Método	20	40	60	80	100	Ano
FBS2	42	75	103	131	163	1999
FC2	41	74	101	130	163	1999
Busca Tabu-Lodi	43	75	104	130	166	2002
GLS	42	74	102	130	162	2003
HBP	42	74	102	130	160	2003
WA	43	74	102	129	159	2009
GRASP-Velasco	42	74	100	129	159	2010
MS-LGFi	42	74	101	129	160	2012
EA-LGFi	42	74	101	128	160	2012
FFIH	42	73	101	130	161	2013
BFIH	42	73	101	130	160	2013
CFIH	43	73	101	129	159	2013
FFIH+J4	42	73	101	130	161	2013
BFIH+J4	42	73	101	129	160	2013
CFIH+J4	42	73	101	129	159	2013
Time assíncrono-Mariano	41	73	100	127	159	2014
Time assíncrono-Silva	41	73	99	126	158	2017
Limite inferior-CJH	39	70	95	122	153	2007

Com relação à qualidade das soluções frente à área que os itens das instâncias ocupam, o time apresentou as mesmas características que o algoritmo de *Branch and Bound* com gulosidade. A distância para o limite inferior cresce de 10% até o item ocupar cerca de 25% da área do contêiner e desce após esse valor. O que leva ao questionamento se essa não é uma característica constante do problema, quando mais os itens se aproximam de 25% da área, mais difícil o problema se torna. Claro, há a necessidade de testar diferentes heurísticas de montagens para que se possa afirmar com convicção.

A tabela 12 apresenta os resultados para a classe 10. Isso porque o autor optou por utilizar apenas essa classe para comparar com os demais algoritmos. Fazendo um comparativo rápido, é fácil perceber que houve uma melhoria em relação aos resultados obtidos pela outra abordagem que utiliza time assíncrono. Uma hipótese para isso está na utilização do *HHDHeuristic* dentro dos algoritmos e não como um algoritmo a parte, uma vez que os resultados obtidos pela maioria dos algoritmos propostos aqui ficou aquém dos resultados obtidos pelo GRASP, algoritmo que também está presente no outro time assíncrono.

A tabela 13 apresenta um comparativo dos melhores resultados encontrados na revisão bibliográfica para o conjunto de instâncias, juntamente com suas heurísticas

Tabela 13 – Comparativo entre alguns modelos de soluções com os do presente trabalho

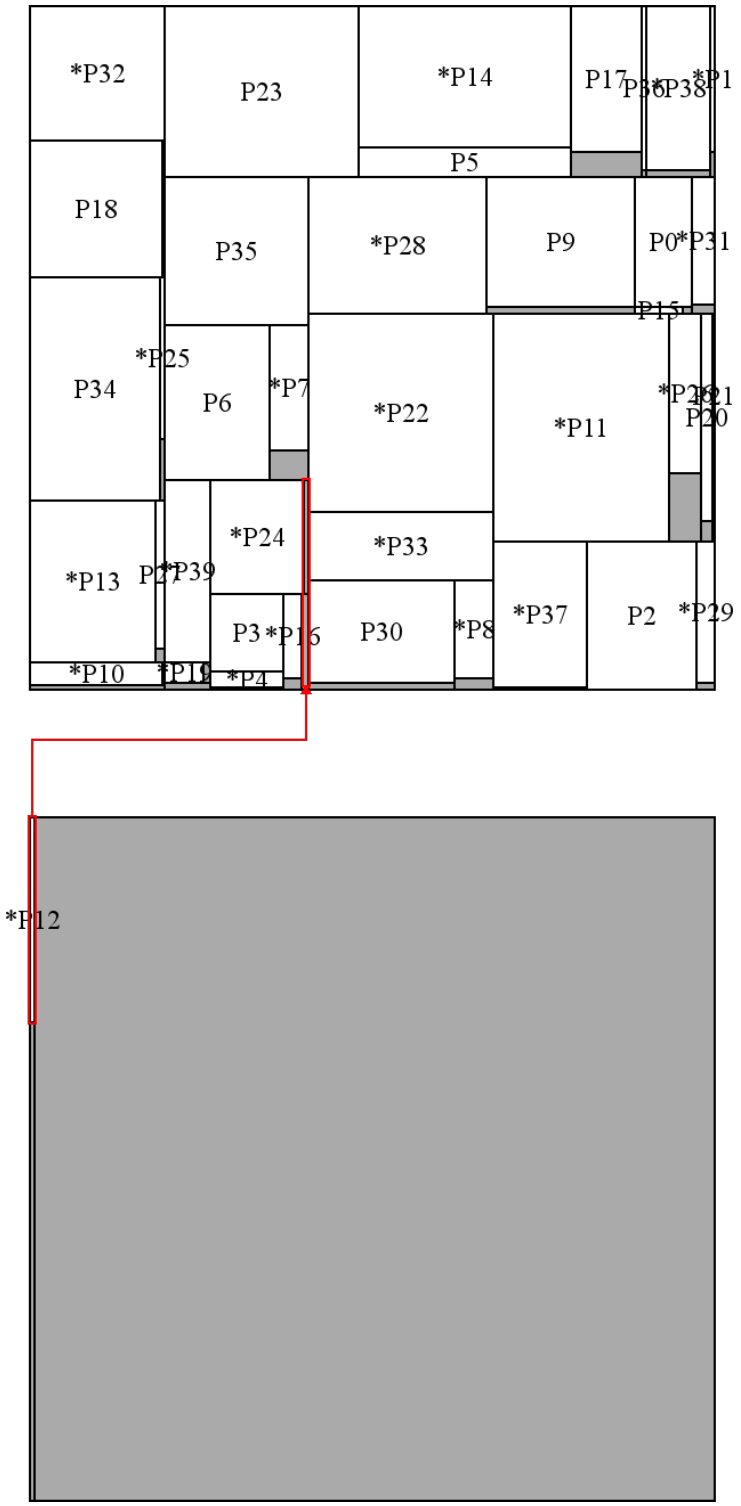
Comparativo entre alguns algoritmos para todas as instâncias

Método	Tipo	Contêineres	% lim. inferior	Ano
KP	H	7297	5,876%	1999
TSKP	H M	7101	3,032%	1999
GLS	H	7367	6,982%	2003
A-B	H Hp	7063	2,481%	2009
CH	H	7161	3,903%	2011
CHBP	H A	7064	2,495%	2011
CFIH	H	7100	3,018%	2013
CFIH+J4	H A	7080	2,728%	2013
HHDHeuristic	H	7349	6,631%	1999
GRASP+*HHDHeuristic*	H M	7049	2,278%	2017
Time assíncrono	H M A-team	7045	2,220%	2017
Lower Bound	pseudo-polinomial	6892	0,000%	2007

H → Heurística
A → Aprimoramento
M → Metaheurística
Hp → Hiperheurística

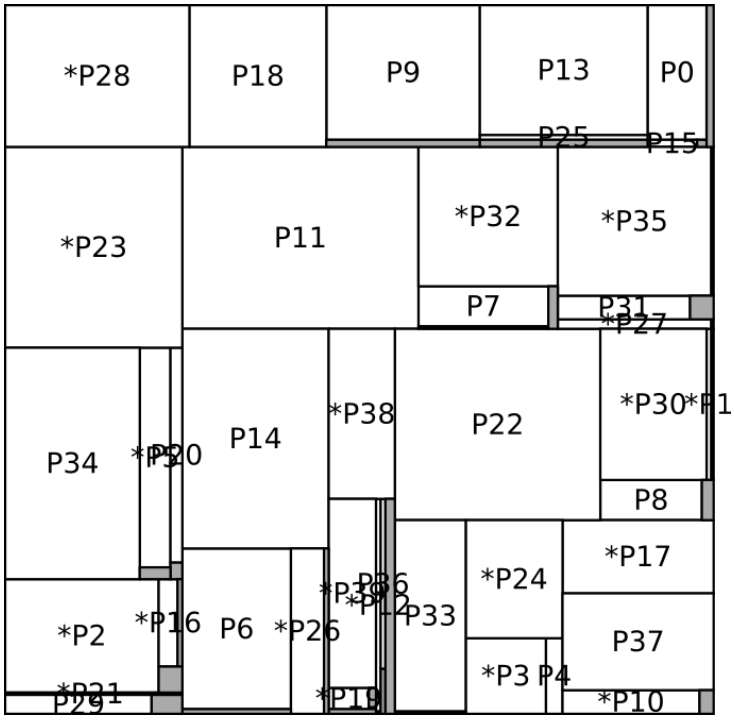
bases. É interessante notar o ganho quando uma heurística é aplicada em conjunto com uma meta-heurística. Apenas a heurística de CFIH e seu aprimoramento CFIH+J4 de Fleszar (2013) apresentam uma pequena distância (20 contêineres).

Figura 58 – Arranjo da instância 6 do grupo de 40 itens da classe 6.



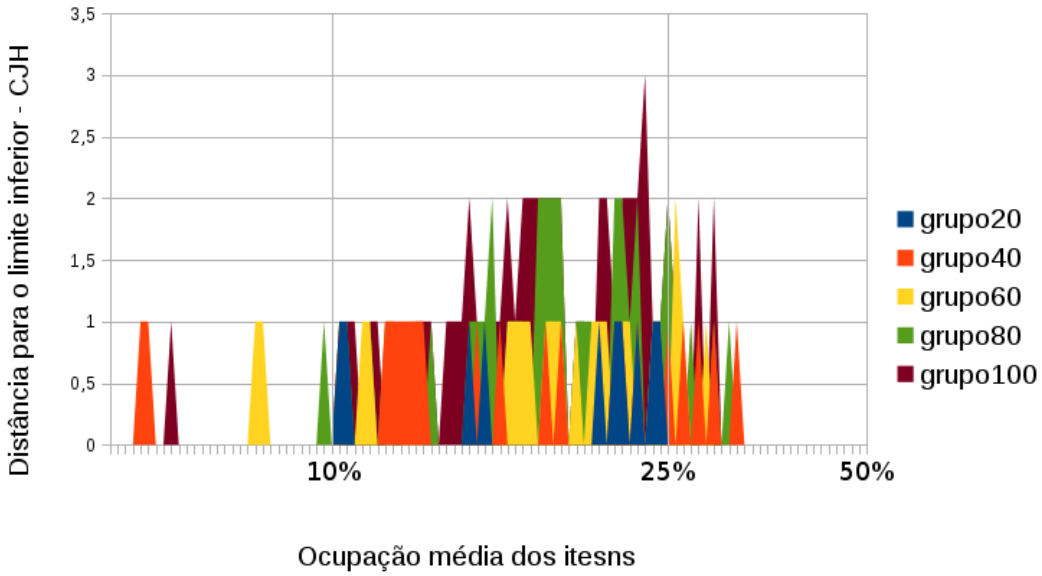
Fonte: Autoria própria

Figura 59 – Arranjo da instância 6 do grupo de 40 itens da classe 6 com limite de 3000 iterações.



Fonte: Autoria própria

Figura 60 – Distância entre os resultados e o limite inferior em relação a área média que os itens ocupam em cada instância.



Fonte: Autoria própria

7 CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

Os resultados dos algoritmo mostram que o objetivo principal deste trabalho foi alcançado, uma vez que reduzimos em 19 placas o melhor resultado presente na literatura. E é provável que esse resultado seja melhorado com mais iterações ou configurações expecíficas para cada tipo de instância, visto o ocorrido com a instância 6 do grupo de 40 itens da classe 6, o que reduz ainda mais o melhor resultado encontrado por nossos algoritmos.

As conclusões principais ao término deste trabalho é que a dificuldade do problema aparenta crescer com a aproximação da área dos itens a 25% da área do contêiner. Itens pequenos demais (abaixo de 10% da área do contêiner) geram padrões ótimos ou bem próximos a ele com certa facilidade, mesmo para algoritmos exatos. A mesma coisa ocorre quando os itens são grandes demais, com áreas próximas a 50% da área do contêiner.

Acreditamos que áreas maiores inviabilizem grande quantidade de arranjos, possibilitando que a busca exaustiva percorra um pequeno espaço de soluções viáveis. No caso de itens pequenos, isso provavelmente ocorre porque existem muitos arranjos ótimos, ou seja, arranjos que a quantidade de contêineres necessários para contemplar todos os itens do pedido é a mínima. Assim, o algoritmo consegue encontrar um desses padrões com mais facilidade.

É importante notar que todas as instâncias de Berkey, Wang, Martello e Vigo foram geradas aleatoriamente e que nenhuma delas aparenta ter uma solução sem sobras. Talvez isso tenha colaborado para os resultados obtidos nesse trabalho. Podemos citar, por exemplo, as instâncias de 9 e 10 itens criadas para ajustar os algoritmos exatos, onde a primeira (sem sobras) tomou 13,029607 segundos enquanto a mesma instância com um item a mais (que gera sobras) levou 0,00086 segundos.

O exemplo dessas instâncias de 9 e 10 itens levanta a hipótese que instâncias com sobras possibilitam mais soluções ótimas, com relação ao número de contêineres. Com relação ao aproveitamento de cada contêiner, essa hipótese pode ser descartada.

Outra conclusão importante a que se chega pelos resultados é que a utilização de heurísticas de montagens (como o *HHDHeuristic* ou GLS) associada a metaheurísticas baseada em busca (como GRASP e Busca Tabu) possuem grande potencial de encontrar boas soluções em pouco tempo, todavia a solução inicial mostrou-se muito importante. Os melhores resultados obtidos na literatura, por exemplo, pertencem a trabalhos que optaram por essa abordagem (tais como Polyakovsky e M'Hallah (2009), Lodi, Martello e Vigo (2004) e Mariano (2014)).

Todavia, a utilização de uma heurística de montagem também pode limitar

os resultados, como nos casos da instância 6 do grupo de 40 itens da classe 6 e da instância 8 do grupo de 100 itens da classe 6, onde os critérios de otimização heurísticos dificultaram a localização da solução ótima. Na segunda instância, as metaheurísticas baseadas em buscas locais (GRASP e Busca Tabu) conseguiram superar essa fragilidade da heurística, mas no primeiro caso não foram capazes de alcançar o resultado ótimo.

Como proposta para trabalhos futuros acreditamos que a utilização de mais de uma heurística de montagem venha a ser interessante, visto que as limitações de uma podem ser superadas pelas outras. Essa característica é muito bem aproveitada no trabalho de Mariano (2014), onde, além da heurística *HHDHeuristic*, foram utilizadas heurísticas de montagem em faixas e desmontagem e remontagem em níveis da árvore binária. A utilização de diferentes heurísticas também poderia vir a confirmar ou negar a hipótese do aumento da complexidade do problema quando a área média dos itens se aproxima de 25% da área do contêiner.

Como dito anteriormente, alguns estados redundantes podem ser removidos pelo uso da estratégia de programação dinâmica dentro do *Greedy Branch and Bound* e seria interessante testa-la para ver se o ganho na remoção desses estados compensa a sobrecarga que a verificação das listas da programação dinâmica vai causar.

Outra proposta com relação ao algoritmo exato. Existe a possibilidade que separar os itens em subconjuntos possa aprimorar os resultados. Isso porque o acréscimo de um item pode fazer com que o tempo de resolução cresça muito. Tratando o problema em duas etapas, inicialmente como um problema de particionamento de conjuntos e aplicar as junções depois, pode melhorar o desempenho, uma vez que o problema de particionamento de conjuntos de forma mais simples que o problema de corte bidimensional guilhotinado, devido as suas restrições espaciais.

Com relação ao modelo matemático, seria interessante a redução do modelo. Mesmo que o mesmo tenha um crescimento do número de restrições e variáveis na ordem de $O(n^2)$, o número total ainda é muito alto para sistemas computacionais resolverem, de modo que o modelo não foi testado a fundo. A ideia era que as três abordagens se utilizassem das instâncias de Lodi, Martello e Vigo (1999a), mas o modelo se mostrou lento, mesmo com as instâncias de pequeno porte.

REFERÊNCIAS

ALMEIDA, I. I. de. *Metaheurística híbrida utilizando GRASP reativo e aprendizagem por reforço: Uma aplicação na segurança pública*. Dissertação (Mestrado) — Universidade Do Estado do Rio Grande Do Norte, 2014. Citado na página 81.

ALVAREZ-VALDES, R. et al. GRASP and path relinking for the two-dimensional two-stage cutting-stock problem. *INFORMS Journal on Computing*, INFORMS, v. 19, n. 2, p. 261–272, 2007. Citado 3 vezes nas páginas 81, 85 e 86.

BARBUCHA, D. Team of A-Teams Approach for Vehicle Routing Problem with Time Windows. In: *Nature Inspired Cooperative Strategies for Optimization (NICSO 2013)*. [S.l.]: Springer, 2014. p. 273–286. Citado na página 94.

BEAN, J. C. Genetic algorithms and random keys for sequencing and optimization. *ORSA journal on computing*, INFORMS, v. 6, n. 2, p. 154–160, 1994. Citado na página 88.

BERKEY, J. O.; WANG, P. Y. Two-dimensional finite bin-packing algorithms. *Journal of the operational research society*, Springer, v. 38, n. 5, p. 423–429, 1987. Citado 4 vezes nas páginas 24, 25, 73 e 96.

BURKE, E. et al. Hyper-heuristics: An emerging direction in modern search technology. In: *Handbook of metaheuristics*. [S.l.]: Springer, 2003. p. 457–474. Citado na página 92.

CAMPOS, V. et al. GRASP with path relinking for the orienteering problem. *Journal of the Operational Research Society*, Springer, v. 65, n. 12, p. 1800–1813, 2014. Citado na página 86.

CHERRI, A. et al. Modelo Matemático e Heurística para o Problema de Corte com Sobras Aproveitáveis e Vendas de Retalhos. *XLVII SBPO-Simpósio Brasileiro de Pesquisa Operacional*, 2015. Citado na página 23.

CHRISTOFIDES, N.; WHITLOCK, C. An algorithm for two-dimensional cutting problems. *Operations Research*, INFORMS, v. 25, n. 1, p. 30–44, 1977. Citado 3 vezes nas páginas 24, 35 e 64.

CHUNG, F. R.; GAREY, M. R.; JOHNSON, D. S. On packing two-dimensional bins. *SIAM Journal on Algebraic Discrete Methods*, SIAM, v. 3, n. 1, p. 66–76, 1982. Citado na página 15.

CLAUTIAUX, F.; JOUGLET, A.; El Hayek, J. A new lower bound for the non-oriented two-dimensional bin-packing problem. *Operations Research Letters*, Elsevier, v. 35, n. 3, p. 365–373, 2007. Citado 4 vezes nas páginas 26, 98, 106 e 111.

DELL'AMICO, M.; MARTELLO, S.; VIGO, D. *An exact algorithm for non-oriented twodimensional bin packing problems*. [S.l.]: preparation, 1999. Citado 2 vezes nas páginas 25 e 26.

DELL'AMICO, M.; MARTELLO, S.; VIGO, D. A lower bound for the non-oriented two-dimensional bin packing problem. *Discrete Applied Mathematics*, Elsevier, v. 118, n. 1, p. 13–24, 2002. Citado 2 vezes nas páginas 26 e 98.

DYCKHOFF, H. A new linear programming approach to the cutting stock problem. *Operations Research*, INFORMS, v. 29, n. 6, p. 1092–1104, 1981. Citado na página 32.

DYCKHOFF, H. A typology of cutting and packing problems. *European Journal of Operational Research*, Elsevier, v. 44, n. 2, p. 145–159, 1990. Citado 4 vezes nas páginas 16, 17, 24 e 26.

FEO, T. A.; RESENDE, M. G. Greedy randomized adaptive search procedures. *Journal of global optimization*, Springer, v. 6, n. 2, p. 109–133, 1995. Citado 2 vezes nas páginas 80 e 82.

FERONE, D.; FESTA, P.; RESENDE, M. G. Hybridizations of GRASP with path relinking for the far from most string problem. *International Transactions in Operational Research*, Wiley Online Library, v. 23, n. 3, p. 481–506, 2016. Citado na página 86.

FILHO, M. A. B.; SANTOS, M. O.; MENESES, C. N. Asynchronous teams for joint lot-sizing and scheduling problem in flow shops. *International Journal of Production Research*, Taylor & Francis, v. 50, n. 20, p. 5809–5822, 2012. Citado na página 91.

FLESZAR, K. Three insertion heuristics and a justification improvement heuristic for two-dimensional bin packing with guillotine cuts. *Computers & Operations Research*, Elsevier, v. 40, n. 1, p. 463–474, 2013. Citado 5 vezes nas páginas 27, 73, 98, 109 e 118.

FURINI, F.; MALAGUTI, E.; THOMOPULOS, D. Modeling two-dimensional guillotine cutting problems via integer programming. *INFORMS Journal on Computing*, INFORMS, v. 28, n. 4, p. 736–751, 2016. Citado 6 vezes nas páginas 19, 27, 32, 33, 35 e 99.

GILMORE, P.; GOMORY, R. The theory and computation of knapsack functions. *Operations Research*, INFORMS, v. 14, n. 6, p. 1045–1074, 1966. Citado na página 23.

GILMORE, P.; GOMORY, R. E. Multistage cutting stock problems of two and more dimensions. *Operations research*, INFORMS, v. 13, n. 1, p. 94–120, 1965. Citado 5 vezes nas páginas 9, 23, 27, 28 e 29.

GILMORE, P. C.; GOMORY, R. E. A linear programming approach to the cutting-stock problem. *Operations research*, INFORMS, v. 9, n. 6, p. 849–859, 1961. Citado na página 23.

GLOVER, F. Future paths for integer programming and links to artificial intelligence. *Computers & operations research*, Elsevier, v. 13, n. 5, p. 533–549, 1986. Citado 2 vezes nas páginas 82 e 84.

GOMES, J. A. R. Problema de corte bidimensional: Aplicação a um caso real. Instituto Superior de Economia e Gestão, 2011. Citado na página 15.

GONCALVES, J.; RESENDE, M. G. A biased random key genetic algorithm for 2D and 3D bin packing problems. *International Journal of Production Economics*, Elsevier, v. 145, n. 2, p. 500–510, 2013. Citado 2 vezes nas páginas 89 e 90.

GONCALVES, J. F.; ALMEIDA, J. R. de. A hybrid genetic algorithm for assembly line balancing. *Journal of Heuristics*, Springer, v. 8, n. 6, p. 629–642, 2002. Citado na página 88.

GONCALVES, J. F.; BEIRÃO, N. Um algoritmo genético baseado em chaves aleatórias para sequenciamento de operações. *Revista Associação Portuguesa de Desenvolvimento e Investigação Operacional*, v. 19, n. 2, p. 123–137, 1999. Citado na página 88.

GONCALVES, J. F.; RESENDE, M. G. A parallel multi-population biased random-key genetic algorithm for a container loading problem. *Computers & Operations Research*, Elsevier, v. 39, n. 2, p. 179–190, 2012. Citado na página 88.

HERZ, J. Recursive computational procedure for two-dimensional stock cutting. *IBM Journal of Research and Development*, IBM, v. 16, n. 5, p. 462–469, 1972. Citado na página 23.

HINXMAN, A. The trim-loss and assortment problems: A survey. *European Journal of Operational Research*, Elsevier, v. 5, n. 1, p. 8–18, 1980. Citado na página 24.

HOFFMANN, F. M. et al. Otimização de padrões de cortes bidimensionais guilhotinados restritos. *Revista ESPACIOS| Vol. 36 (Nº 09) Año 2015*, 2015. Citado na página 20.

HOLLAND, J. H. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. [S.l.]: MIT Press, 1975. (Bradford book). ISBN 9780585038445. Citado na página 88.

JEDRZEJOWICZ, P.; WIERZBOWSKA, I. Impact of Migration Topologies on Performance of Teams of Agents. In: *Recent Advances in Knowledge-based Paradigms and Applications*. [S.l.]: Springer, 2014. p. 115–128. Citado na página 92.

KRÖGER, B. Guillotineable bin packing: A genetic approach . *European Journal of Operational Research*, Elsevier, v. 84, p. 645–661, 1995. Citado na página 79.

LAGUNA, M.; MARTI, R. GRASP and path relinking for 2-layer straight line crossing minimization. *INFORMS Journal on Computing*, INFORMS, v. 11, n. 1, p. 44–52, 1999. Citado na página 85.

LAND, A. H.; DOIG, A. G. An automatic method of solving discrete programming problems. *Econometrica: Journal of the Econometric Society*, JSTOR, p. 497–520, 1960. Citado na página 68.

LODI, A. Algorithms for two-dimensional bin packing and assignment problems. *Doktorarbeit, DEIS, Universita di Bologna*, Citeseer, 1999. Citado 2 vezes nas páginas 23 e 25.

LODI, A.; MARTELLO, S.; VIGO, D. Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems. *INFORMS Journal on Computing*, INFORMS, v. 11, n. 4, p. 345–357, 1999. Citado 6 vezes nas páginas 20, 26, 83, 96, 97 e 122.

LODI, A.; MARTELLO, S.; VIGO, D. Neighborhood search algorithm for the guillotine non-oriented two-dimensional bin packing problem. In: *Meta-Heuristics*. [S.l.]: Springer, 1999. p. 125–139. Citado na página 73.

LODI, A.; MARTELLO, S.; VIGO, D. Models and bounds for two-dimensional level packing problems. *Journal of Combinatorial Optimization*, v. 8, p. 363–379, 2004. Citado 3 vezes nas páginas 29, 32 e 121.

LUO, Z.; YU, X.; YUAN, C. A new approach of constructing decision tree based on shortest path methods. In: IEEE. *Audio, Language and Image Processing (ICALIP), 2016 International Conference on*. [S.l.], 2016. p. 630–634. Citado na página 63.

MARIANO, M. F. d. S. *Uma metaheurística A-Teams aplicada ao problema do corte bidimensional guilhotinado*. Dissertação (Mestrado) — Universidade Do Estado do Rio Grande Do Norte, 2014. Citado 12 vezes nas páginas 12, 27, 73, 75, 76, 77, 81, 91, 92, 117, 121 e 122.

MARTELLO, S.; VIGO, D. Exact solution of the two-dimensional finite bin packing problem. *Management science*, INFORMS, v. 44, n. 3, p. 388–399, 1998. Citado 4 vezes nas páginas 21, 24, 25 e 96.

MARTÍ, R. et al. Multiobjective GRASP with path relinking. *European Journal of Operational Research*, Elsevier, v. 240, n. 1, p. 54–71, 2015. Citado na página 86.

MESSAOUD, S. B.; CHU, C.; ESPINOUSE, M.-L. Characterization and modelling of guillotine constraints. *European Journal of Operational Research*, Elsevier, v. 191, n. 1, p. 112–126, 2008. Citado 2 vezes nas páginas 26 e 32.

NASCIMENTO, H. do; LONGO, H.; ALOISE, D. Uma Heurística O(mn) para o Corte Bidimensional Guilhotinado. *XXXI SBPO-Simpósio Brasileiro de Pesquisa Operacional*, p. 1197–1206, 1999. Citado 12 vezes nas páginas 7, 8, 21, 25, 26, 27, 35, 73, 74, 75, 81 e 90.

OLIVEIRA, J.; FERREIRA, J. An improved version of Wang's algorithm for two-dimensional cutting problems. *European Journal of Operational Research*, Elsevier, v. 44, n. 2, p. 256–266, 1990. Citado na página 24.

OLIVEIRA, W. de. *Algoritmo evolutivo paralelo para o problema de atribuição de localidades a anéis em redes sonet/sdh*. Dissertação (Mestrado) — Universidade Federal do Rio Grande do Norte, 2010. Citado 2 vezes nas páginas 88 e 90.

OXFORD. *heuristic - definition of heuristic in English*. 2017. Disponível em: <https://en.oxforddictionaries.com/definition/heuristic>. Citado na página 73.

PAPADIMITRIOU, C. H.; STEIGLITZ, K. *Combinatorial Optimization, Algorithms and Complexity*. Mineola, New York: Dover Publications, Inc., 1998. Citado na página 15.

PAREJO, J. A. et al. QoS-aware web services composition using GRASP with Path Relinking. *Expert Systems with Applications*, Elsevier, v. 41, n. 9, p. 4211–4223, 2014. Citado na página 86.

PARK, K. T. et al. Developing a heuristics for glass cutting process optimization: A case of two-dimensional two-stage guillotine cutting with multiple stock sizes. *Korean Journal of Chemical Engineering*, Springer, p. 1–8, 2013. Citado na página 26.

PARREÑO, F. et al. A hybrid GRASP/VND algorithm for two-and three-dimensional bin packing. *Annals of Operations Research*, Springer, v. 179, n. 1, p. 203–220, 2010. Citado na página 80.

POLDI, K. C. *Algumas extensões do problema de corte de estoque*. Dissertação (Mestrado) — ICMC/USP, 2003. Citado na página 23.

POLYAKOVSKY, S.; M'HALLAH, R. An agent-based approach to the two-dimensional guillotine bin packing problem. *European Journal of Operational Research*, Elsevier, v. 192, n. 3, p. 767–781, 2009. Citado 6 vezes nas páginas 15, 26, 73, 91, 109 e 121.

RECHENBERG, I. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog, 1973. (Problemata, 15). ISBN 9783772803734. Disponível em: <https://books.google.com.br/books?id=-WAQAQAAMAAJ>. Citado na página 88.

RUSSO, M.; SFORZA, A.; STERLE, C. An exact dynamic programming algorithm for large-scale unconstrained two-dimensional guillotine cutting problems. *Computers & Operations Research*, Elsevier, v. 50, p. 97–114, 2014. Citado na página 27.

SADAPHULE, M. V.; SHAIKH, N. F. An application of database query translation into spreadsheets. In: IEEE. *Electrical, Electronics, and Optimization Techniques (ICEEOT), International Conference on*. [S.l.], 2016. p. 2861–2866. Citado na página 63.

SINUANY-STERN, Z.; WEINER, I. The one dimensional cutting stock problem using two objectives. *Journal of the Operational Research Society*, Springer, v. 45, n. 2, p. 231–236, 1994. Citado 2 vezes nas páginas 24 e 76.

TALUKDAR, S. et al. Asynchronous teams: Cooperation schemes for autonomous agents. *Journal of Heuristics*, Springer, v. 4, n. 4, p. 295–321, 1998. Citado na página 91.

TEMPONI, E. C. C. *Uma Proposta de Resolução do Problema de Corte Bidimensional via Abordagem Metaheurística* . Dissertação (Mestrado) — CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS, 2007. Citado 5 vezes nas páginas 18, 23, 30, 31 e 44.

VELASCO, A. S. *GRASP para o Problema de Corte Bidimensional Guilhotinado e Restrito*. Dissertação (Mestrado) — Universidade Estadual do Norte Fluminense - UENF, 2005. Citado 6 vezes nas páginas 23, 26, 27, 81, 83 e 99.

WALDHERR, S.; KNUST, S. Two-stage scheduling in shelf-board production: a case study. *International Journal of Production Research*, Taylor & Francis, v. 52, n. 13, p. 4078–4092, 2014. Citado na página 91.

WANG, P. Y. Two algorithms for constrained two-dimensional cutting stock problems. *Operations Research*, INFORMS, v. 31, n. 3, p. 573–586, 1983. Citado 3 vezes nas páginas 24, 25 e 35.

WÄSCHER, G.; HAUBNER, H.; SCHUMANN, H. An improved typology of cutting and packing problems. *European Journal of Operational Research*, v. 183, n. 3, p. 1109–1130, 2007. ISSN 0377-2217. Disponível em: <http://www.sciencedirect.com/science/article/pii/S037722170600292X>. Citado 3 vezes nas páginas 17, 18 e 26.

YEH, W.-C. A greedy branch-and-bound inclusion-exclusion algorithm for calculating the exact multi-state network reliability. *IEEE Transactions on Reliability*, IEEE, v. 57, n. 1, p. 88–93, 2008. Citado na página 69.

YOUNG, J. M.; ETZKORN, L. Comparing An Object Oriented Runtime Complexity Metric To Depth First Search Complexity with Mobile Agents in a Mobile Autonomous Environment. In: THE STEERING COMMITTEE OF THE WORLD CONGRESS IN COMPUTER SCIENCE, COMPUTER ENGINEERING AND APPLIED COMPUTING (WORLDCOMP). *Proceedings of the International Conference on Software Engineering Research and Practice (SERP)*. [S.l.], 2016. p. 181. Citado na página 63.