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

MARIANNY FIDELIS DE SOUSA MARIANO

UMA METAHEURÍSTICA A-TEAMS APLICADA AO PROBLEMA DO CORTE BIDIMENSIONAL GUILHOTINADO

MOSSORÓ – RN
2014

MARIANNY FIDELIS DE SOUSA MARIANO

UMA METAHEURÍSTICA A-TEAMS APLICADA AO PROBLEMA DO CORTE BIDIMENSIONAL GUILHOTINADO

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 Semiárido, para a obtenção do título de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Dario José Aloise – UERN.

Coorientador: Prof. Dr. Hugo A. D. do Nascimento – UFG.

MOSSORÓ – RN

2014

Catalogação da Publicação na Fonte.
Universidade do Estado do Rio Grande do Norte.

Mariano, Marianny Fidelis de Sousa
Uma metaheurística a-teams aplicada ao problema do corte bidimensional guilhotinado. / Marianny Fidelis de Sousa Mariano. – Mossoró, RN, 2014.

97 f.
Orientador(a): Prof. Dr. Dario José Aloise

Dissertação (Mestre em Ciência da Computação). Universidade Federal Rural do Semi-Árido. Universidade do Estado do Rio Grande do Norte. Programa de Pós-Graduação em Ciência da Computação.

1. *Times Assíncronos. 2. Problema do corte bidimensional guilhotinado. 3.Metaheurística - Otimização combinatória.*. I. Aloise, Dario José . II. Universidade do Estado do Rio Grande do Norte. III.Título.

UERN/BC CDD 004

Bibliotecária: Jocelania Marinho Maia de Oliveira CRB 15 / 319

MARIANNY FIDELIS DE SOUSA MARIANO

UMA METAHEURÍSTICA A-TEAMS APLICADA AO PROBLEMA DO CORTE BIDIMENSIONAL GUILHOTINADO

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

BANCA EXAMINADORA:

_____

Prof. Dr. Dario José Aloise

(Orientador)

Universidade do Estado do Rio Grande do Norte – UERN

_____

Prof. Dr. Hugo Alexandre Dantas do Nascimento

(Coorientador)

Universidade Federal de Goiás – UFG

_____

Prof. Dr. Francisco Dantas de Medeiros Neto

(Membro Interno)

Universidade do Estado do Rio Grande do Norte – UERN

_____

Prof. Dr. Luiz Amorim Carlos (UFRN)

(Membro externo)

Universidade Federal do Rio Grande do Norte – UFRN

A meus pais,
FRANCISCO EDSON MARIANO FILHO e
EUGÊNIA FIDELIS DE SOUSA MARIANO

AGRADECIMENTOS

A realização deste trabalho só foi possível graças ao Pai Celestial por estar sempre presente em minha vida, nos melhores e mais difíceis momentos dessa caminhada.

Agradeço aos meus pais, Mariano e Eugênia, pela confiança depositada em mim e apoio, aos meus irmãos, Patrick, Gabriel e Daniel e em especial ao meu noivo Júnior, pela paciência, apoio e pelas horas de distração proporcionadas durante os momentos de preocupação.

Agradeço aos meus primos, em especial, Lian, Juarez e Jobson, pelo grande apoio e pelas horas de diversão. Um agradecimento especial, também ao meu amigo Marlos, pelo apoio, ajuda e pelos bons momentos durante toda essa caminhada desde a graduação.

Agradeço as minhas tias Maiza Fidelis, Hulguete Fidelis, Eliane Fidelis e seu esposo Ednaldo pelo apoio desde o início dessa jornada e em especial a minha tia Natália Segundo, que apesar de não está mais conosco, sempre me deu força e foi uma grande inspiração durante essa caminha, desde a graduação.

Agradeço à Universidade do Estado do Rio Grande do Norte e à Universidade Federal Rural do Semi-Árido, pelo apoio institucional, por oferecer uma estrutura e os meios necessários para que fosse possível desempenhar todas as atividades requeridas pelo Programa de Pós-Graduação em Ciência da Computação (PPgCC). Agradeço à Capes, pelo apoio financeiro e bolsa concedida.

Agradeço a todos os professores do MCC, por proporcionarem a abertura de novos horizontes por meio do conhecimento transmitido nas diferentes disciplinas. À secretária Rosita, por sempre atender minhas solicitações.

Agradeço ao Prof. Dr. Dario José Aloise, pela orientação, apoio, confiança, paciência e por estar sempre presente; pelo conhecimento transmitido, pelo incentivo constante, tempo e disponibilidade prestados a mim.

Agradeço ao Prof. Dr. Hugo A. D. do Nascimento, que mesmo atarefado com suas pesquisas e à distância, sempre disponibilizou um tempo para transmitir conhecimentos e orientação que foram de fundamental importância para o sucesso deste trabalho.

Agradeço a todos pela compreensão e que fizeram parte da minha vida durante o tempo do mestrado.

"E, se algum de vós tem falta de sabedoria, peça-a a Deus, que a todos dá liberalmente, e o não lança em rosto, e ser-lhe-á dada.
Peça-a, porém, com fé, não duvidando; pois o que duvida é semelhante à onda do mar, que é levada pelo vento de uma para outra parte".

Tiago 1:5-6.

RESUMO

Existem vários métodos heurísticos que buscam resolver o problema de corte guilhotinado bidimensional utilizando estratégias e características intrínsecas e diferenciadas na busca por soluções de boa qualidade. O presente trabalho fundamenta-se em uma abordagem metaheurística conhecida como Time Assíncrono ou (A-Team, do inglês *Assynchronous Team*), a qual é constituída por um conjunto de agentes trabalhando de maneira autônoma e comunicando-se assincronamente por meio de uma memória compartilhada. Nessa metaheurística, cada agente pode incorporar uma estratégia heurística em particular para a resolução do problema em questão. Diante desse contexto, este trabalho se propôs investigar e implementar um A-Team aplicado ao Problema do Corte Bidimensional Guilhotinado combinando uma heurística de característica construtiva e uma metaheurística GRASP, além de adaptações para atender características de melhoria com o propósito de obter sinergia, ou seja, alcançar resultados, superiores aos encontrados pelas heurísticas isoladamente. Testes computacionais foram realizados sobre uma classe de instâncias conhecida da literatura e a abordagem se mostrou competitiva e promissora quando comparada aos métodos estudados.

Palavras-Chave: Times Assíncronos, problema do corte bidimensional guilhotinado, metaheurística, otimização combinatória.

ABSTRACT

There are many heuristic methods to solving the two-dimensional guillotine cutting problem based on strategies and differentiated features with aim of producing good quality solutions. This research is based on approach of Asynchronous Team or A-Team, which assumes a collection of software agents working autonomously and communicating asynchronously through a shared memory (cooperate and they to solve an optimization problem in question). Each agent represents a particular problem solving method. In this paper, we propose and describe the implementation of a strategy in which a constructive heuristic, a GRASP metaheuristic and guided local search heuristics are incorporated into an A-Team framework. Computational results have been presented and analyzed on a set of benchmark instances. They are equal or superior to those found by the heuristics separately (synergetic effect). Our experiments and results show that (approach) agent-based paradigms can be considered as a promising and competitive alternative to previous methods already known in the literature.

Keywords: Assynchronous Team, Bi-Dimensional Guillotine Cutting Problem, Optimization, Metaheuristic, Combinatorial Optimization.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS E ABREVIAÇÕES

ACS – *Ant Colony System*

AFFDH – *Adaptative First Fit Decreasing Height*

A-Team – *Assynchronous Team*

BF – *Best Fit*

BFD – *Best-Fit Decreasing*

BFDH – *Best Fit Decreasing Height*

BFIH – *Best-Fit Insertion Heuristic*

CFIH – *Critical-Fit Insertion Heuristic*

EA-LGFi – *Evolutionary Algorithm LGFi*

FAV – *Função de Avaliação*

FBS – *Finite Best Strip*

FBL – *Finite Bottom Left*

FC – *Floor Ceiling*

FF – *First Fit*

FFF – *Finite First Fit*

FFD – *First Fit Decreasing*

FFDH – *First Fit Decreasing Height*

FFIH – *First-Fit Insertion Heuristic*

FNF – *Finite Next Fit*

GA – *Genetic Algorithm*

GLS – *Guided Local Search*

GP – *Genetic Programming*

GRASP – *Greedy Randomized Adaptive Search Procedure*

HFF – *Hybrid First Fit*

HNF – *Hybrid Next Fit*

ILS – *Iterated Local Search*

LRC – *Lista Restrita de Candidatos*

MFFD – *Modified First Fit Decreasing*

MS-LGFi – *Multi-Start LGFi*

NF – *Next Fit*

NFDH – *Next Fit Decreasing Height*

PCBG – *Problema de Corte Bidimensional Guilhotinado*

PCE – *Problemas de Corte e Empacotamento*

PE – *Problemas de Empacotamento*

PSO – *Particle Swarm Optimization*

SA – *Simulated Annealing*

SFFDH – *Split First Fit Decreasing Height*

VND – *Variable Neighborhood Descente*

VNS – *Variable Neighborhood Search*

PE – *Problemas de Empacotamento*

PCE – *Problemas de Corte e Empacotamento*

WA – *Weight Annealing*

Sumário

1. INTRODUÇÃO

Os problemas de corte e empacotamento costumam ser de fácil entendimento. Entretanto, sua aparente simplicidade geralmente esconde a natureza complexa em termos computacionais que o caracterizam como NP-difícil. O interesse por tais problemas é, em parte, explicado por sua grande aplicabilidade prática, especialmente na indústria, mas também por seus desafios teóricos computacionais. É importante notar que melhorias no processo de corte (por exemplo, diminuição do desperdício de materiais utilizados como matéria prima) podem apresentar ganhos significativos e representar uma vantagem decisiva na competição entre empresas.

Por esse motivo, um número crescente de pesquisadores de diferentes áreas, como computação, engenharia, economia e matemática, estão se dedicando ao estudo de tais problemas, investigando e aplicando diversas técnicas e métodos de resolução como: programação linear, programação dinâmica, *branch-and-bound*, heurísticas e metaheurísticas que contribuem de maneira singular para a composição de soluções de boa qualidade (GOMES, 2011).

Assim, no problema de corte bidimensional, deseja-se atender uma demanda de pedidos de um determinado material maximizando a área de utilização de chapas disponíveis. A restrição do corte guilhotinado significa que a cada corte deve ser feito de uma ponta a outra da placa, sempre dividindo o pedaço em questão em outros dois novos pedaços. Essa restrição aumenta ainda mais a complexidade do problema.

1.1 OBJETIVOS

Diante deste contexto, o presente trabalho fundamenta-se em uma abordagem denominada *Assynchronous Team* (A-Team), o qual é constituída por um conjunto de agentes trabalhando de maneira autônoma e comunicando-se assincronamente por meio de memória(s) compartilhada(s). Cada agente está em um *loop* fechado e incorpora métodos de resolução particular para o problema em estudo. Com isso, agentes podem agir como coletor, processador e fornecedor de informações.

Essa abordagem possibilita a implementação de um conjunto de agentes com características e estruturas internas particulares e diferentes, no intuito de trabalhar de

forma cooperativa. O objetivo geral do trabalho é demonstrar que a abordagem A-Team pode ser aplicada ao problema de corte guilhotinado, de modo a produzir resultados superiores ao que os agentes executando isoladamente conseguem alcançar. São objetivos específicos desta dissertação:

- escolher uma heurística construtiva e uma metaheurística para o problema do corte guilhotinado bidimensional;
- propor heurísticas de melhorias com base em adaptações da heurística construtiva e da metaheurística escolhidas.
- propor uma abordagem de times assíncronos aplicado ao Problema do Corte Bidimensional Guilhotinado combinando heurísticas de características construtivas e de melhoria;
- avaliar e validar a abordagem com instâncias da literatura e comparando os resultados obtidos com aqueles já conhecidos.

1.2 ESTRUTURA DO DOCUMENTO

O trabalho está estruturado em 6 capítulos como seguem. No Capítulo 1, é apresentada uma introdução definindo os pontos principais abordados nesta dissertação.

No Capítulo 2, é feita uma introdução sobre o Problema do Corte Bidimensional Guilhotinado, seus conceitos e características, bem como uma revisão bibliográfica e a apresentação de métodos heurísticos e metaheurísticos aplicados ao problema.

O Capítulo 3 apresenta os times assíncronos, sua definição, composição, e características. No Capítulo 4, são apresentados detalhes da abordagem proposta, os agentes construtores e de melhoria implementados, a memória central, bem como a representação das soluções que formam o time. No Capítulo 5, são descritos os testes realizados com configurações diferentes do time e os resultados obtidos, assim como os pontos relevantes observados.

Por fim, as conclusões e sugestões para trabalhos futuros estão dispostas no Capítulo 6.

2. O PROBLEMA DE CORTE E EMPACOTAMENTO

No caso dos Problemas de Empacotamento (PE), o principal objetivo consiste em empacotar um conjunto N de itens menores (por exemplo, peças, retângulos) de largura l_i e comprimento c_i dentro de um conjunto de objetos maiores (por exemplo, placas e folhas) com largura L e comprimento C. Esse problema também pode ser visualizado como problema de corte, de forma que os itens agora devem ser recortados e não alocados ou empacotados. Assim, a solução para o problema de corte é a partição dos itens em um conjunto de padrões, onde cada padrão representa um arranjo de itens que devem ser escolhidos e agrupados de forma a respeitar as dimensões do objeto ao qual foram atribuídos sem haver sobreposição (TEMPONI, 2007). Objetos e itens podem ser definidos em uma, duas ou três dimensões, variando de pouco a muito heterogêneos, regulares e irregulares.

Em consequência de sua natureza genérica os Problemas de Corte e Empacotamento (PCE) modelam e possuem uma variedade de aplicações em situações reais, principalmente no âmbito industrial. Apesar de sua aparente simplicidade, a natureza desses problemas é altamente combinatória, que os caracterizam como NP-Difícil, e têm sido objeto de estudos por diversos pesquisadores.

Problemas de corte e empacotamento são encontrados em indústrias com diferentes características, restrições e objetivos. As indústrias de vidro, madeira, papel estão principalmente preocupados com o corte de figuras regulares, ao passo que as indústrias de construção naval, têxteis e de couro preocupam-se com itens irregulares (HOPPER e TURTON, 2000). As diversas aplicações práticas na indústria vão desde a disposição de artigos em jornais, carregamento de materiais em *containers* e caminhões, ao corte de placas de madeiras e vidros, assim como em aplicações abstratas, envolvendo dimensões não espaciais, como peso e tempo (GOMES, 2011).

Dyckhoff (1990) e Wäscher *et al*. (2006) propuseram uma tipologia no intuito de padronizar e facilitar a classificação das variações dos Problemas de Corte e Empacotamento, o que é apresenta na próxima seção.

2.1 TIPOLOGIA DOS PROBLEMAS DE CORTE E EMPACOTAMENTO

Dyckhoff (1990) propõe quatro critérios de classificação para esta classe de

problemas:

1) Dimensionalidade:
    a) Unidimensional;
    b) Bidimensional;
    c) Tridimensional;
    d) N-dimensional.
2) Tipo de atribuição:
    a) Todos os objetos são utilizados para alocar parte dos itens;
    b) Todos os itens são alocados para utilizar parte dos objetos.
3) Tipo de objetos:
    a) Um único objeto;
    b) Objetos idênticos;
    c) Objetos diferentes.
4) Tipo de itens:
    a) Poucos itens de formas diferentes;
    b) Muitos itens muito heterogêneos;
    c) Muitos itens pouco heterogêneos;
    d) Itens iguais.

A tipologia proposta por (DYCKHOFF, 1990), a princípio foi considerada uma boa forma de classificar e organizar os problemas PCE, porém tornou-se ineficiente para contemplar as novas variações dos problemas de corte e empacotamento, o que motivou a pesquisa por uma nova tipologia.

Wäscher *et al*. (2006) *apud* (TEMPONI, 2007) apresentaram uma nova tipologia com base em Dyckhoff, considerando cinco critérios:

1) Dimensionalidade – conforme esse critério, se divide em:
    a) Unidimensional;
    b) Bidimensional;
    c) Tridimensional.
2) Tipos de Atribuição:
    a) Maximizar a saída: conforme esse critério os objetos disponíveis não são suficientes para a alocação de todos os itens, sendo fundamental a maximização da utilização dos objetos.
    b) Minimizar a entrada: neste critério, os objetos disponíveis são suficientes

para a alocação de todos os itens, buscando a minimização de um valor, por exemplo, desperdício de material.

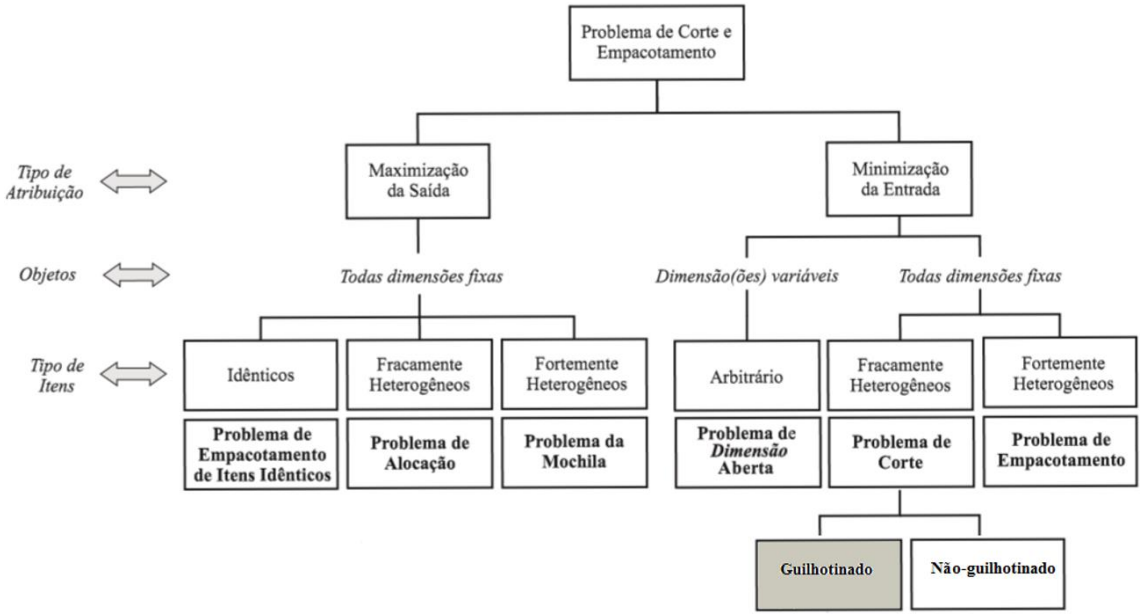
3) Tipo dos itens:
   a) Idênticos;
   b) Pouco heterogêneos;
   c) Muito heterogêneos.
4) Tipo dos objetos:
   a) Um único objeto:
      i) Com dimensões fixas;
      ii) Com dimensões variáveis.
   b) Vários objetos
      i) Idênticos;
      ii) Pouco heterogêneos;
      iii) Muito heterogêneos.
5) Forma dos itens:
   i) Regulares;
   ii) Irregulares.

Conforme Wäscher *et al.* (2006) *apud* (TEMPONI, 2007), a estrutura da classe de problemas de corte e empacotamento define-se em três etapas, sendo a primeira, a identificação dos critérios de atribuição e tipo dos itens; na segunda etapa, define-se o tipo de objetos; e, finalmente, na terceira, os critérios de dimensionalidade e a forma dos itens, definindo a estrutura geral do problema a ser trabalhado. A Figura 1 apresenta a tipologia dos problemas de corte e empacotamento segundo a classificação (WÄSCHER *et al.*, 2006). Desse modo, o autor divide os problemas de corte e empacotamento em cinco subproblemas:

1. **Problema de seleção de objetos** – os objetos (bin, container, placas ou chapas) possuem atributos como: dimensões e custos diferentes;
2. **Problema de seleção de itens** – determinados itens possuem finalidades diferentes, e possuem restrições em relação aos outros;
3. **Problema de agrupamento de itens** – determinado conjunto de itens não pode ficar próximo a outro;
4. **Problema de alocação de itens em objetos** – determinados itens tem a restrição de alocação somente em objetos específicos;

5. **Problema de *layout*** – quando os itens devem ser dispostos nos objetos, respeitando as restrições geométricas.

Figura 1 – Tipologia dos Problemas de Corte e Empacotamento.



FONTE: ADAPTADO DE WÄSCHER *ET AL.* (2006) APUD (TEMPONI, 2007)

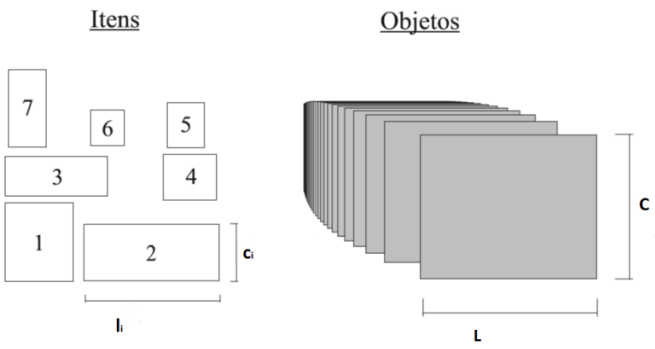
2.2 O PROBLEMA DO CORTE BIDIMENSIONAL

2.2.1 Conceitos e Definições

O problema de corte bidimensional caracteriza-se pelo corte de peças retangulares maiores, geralmente de tamanho padrão, para obtenção de peças retangulares menores com a finalidade de atender uma demanda específica. O processo de corte normalmente implica na perda de matéria prima o que influencia diretamente na competitividade e no lucro da empresa (CHUNG, GAREY e JOHNSON, 1982).

Define-se o problema de corte bidimensional como a necessidade de cortar um número finito de objetos retangulares maiores de largura e comprimento (L, C) armazenados em estoque, a fim de obter itens menores com dimensões (l_i, c_i), de modo a atender uma demanda d_i, para $i = 1, 2, 3, ..., n$, e respeitando $l_i \leq L$ *e* $c_i \leq C$, conforme a Figura 2.

Figura 2 – Exemplo de objetos e demanda de itens.



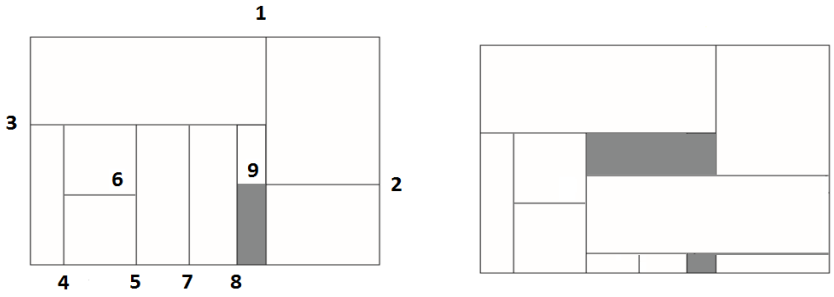
FONTE: TEMPONI (2007)

É preciso atender a um conjunto de outras restrições que caracterizam o problema. Essas restrições estão relacionadas às tipologias apresentadas na Seção 2.2 deste mesmo capítulo. Por exemplo, com relação ao número de peças existentes em um padrão de corte, o problema de corte pode ser restrito ou irrestrito. Dessa forma, seja um padrão de corte determinado por um possível arranjo de peças em uma placa, este é dito restrito se há uma limitação na quantidade de peças obtidas, ou seja, o número de determinada peça a ser obtida com os cortes efetuados nas placas deve satisfazer uma demanda. Assim, para o problema dito restrito há imposição de um limite superior à quantidade de uma determinada peça, em um padrão, acrescentando dificuldades à resolução do problema. Se não houver restrições de limitação associadas ao número de peças, o problema é irrestrito (VELASCO, 2005).

Outra restrição diz respeito a rotação ou não sofrida pelos itens. A escolha por rotação de itens torna o problema ainda mais complexo, uma vez que são consideradas as rotações possíveis antes da alocação do item.

Uma terceira restrição e extremamente relevante diz respeito a extensão do corte em determinada direção, que caracteriza o processo de corte como guilhotinado ou não guilhotinado. No corte guilhotinado é realizado um corte de uma ponta a outra do objeto (placa) produzindo, a cada corte realizado, dois novos retângulos, os quais serão utilizados no processo repetidamente até a obtenção de todas as peças desejadas (SANCHEZ, 2008). Um exemplo é ilustrado na Figura 3.

Figura 3 – Exemplo de padrões de corte guilhotinado e não-guilhotinado



a) Padrão guilhotinado b) Padrão não-guilhotinado

FONTE: AUTORIA PRÓPRIA

Em resumo, o problema abordado neste trabalho e suas características mais representativas são:

- os objetos possuem largura e altura fixa;
- os itens têm dimensões diferenciadas;
- todos os itens e objetos possuem forma retangular;
- os itens podem ser rotacionados em 90°;
- os cortes são do tipo guilhotinado; e
- o principal objetivo é minimizar o desperdício dos objetos utilizados.

2.2.2 Revisão da literatura

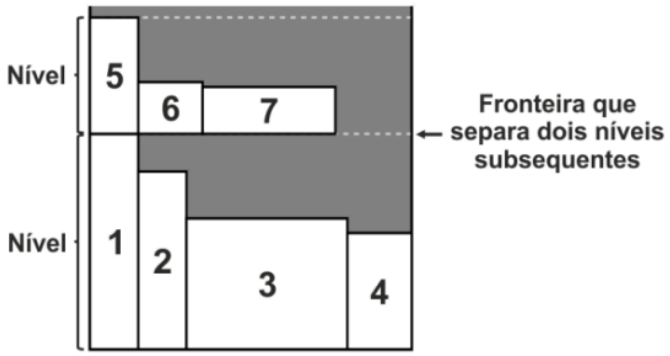
Os problemas de corte se caracterizam como problemas de otimização combinatória e estão preocupados em encontrar bons arranjos de vários itens de tamanhos menores em regiões de objetos maiores. O principal propósito no processo de corte é maximizar o aproveitamento do material utilizado e, por conseguinte, minimizar a área de desperdício. Isto é de um interesse especial para indústrias envolvidas em processos de produção em massa, visto que pequenas melhorias na disposição dos itens podem resultar em uma poupança de material significativa nos custos de produção (GOMES, 2011).

Nos últimos anos uma gama de algoritmos foram propostos com objetivo de resolver variantes dos problemas de corte e empacotamento e diversos artigos de revisão bibliográfica foram escritos. No que diz respeito às heurísticas para o problema de corte e empacotamento bidimensional guilhotinado, estas podem ser diferenciadas entre as de uma fase e duas fases e podem ou não ser orientadas a níveis (LODI, MARTELLO e

VIGO, 2004).

Algoritmos de uma fase encaixam os itens diretamente dentro das placas (ou *bins*). Enquanto algoritmos de duas fases primeiro alocam os itens dentro de níveis com mesma largura *L* do objeto e altura infinita, considerando o problema *strip packing* e em seguida tais níveis são alocados dentro de placas com altura *A*, do objeto, definida. Segundo Hopper e Turton (2000) heurísticas que utilizam estratégias orientadas por níveis são aquelas cujos os itens são alocados em diferentes níveis (fileiras) do objeto, justificado à esquerda. Níveis são retângulos com altura variável e com larguras definidas pela largura do objeto. O processo de corte e/ou empacotamento é construído por uma sequência de níveis, de forma que a altura do nível é determinada pelo item mais alto alocado Hopper e Turton (2000).

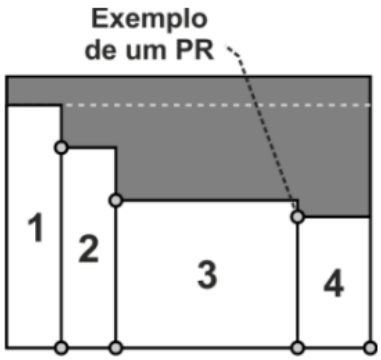
Figura 4 – Definição de dois níveis e da fronteira que os separa



FONTE: GOMES (2011)

Complementar ao conceito de níveis, outros dois conceitos são importantes para a compreensão das heurísticas e metaheurísticas aqui apresentadas, são eles: ponto de referência (PR) e *gap*. Como citado em Gomes (2011), um ponto de referência (PR) é representado por uma coordenada *x* e *y* e indica o ponto mais abaixo e à esquerda do objeto ou nível, no qual é possível alocar um item sem sobrepor os demais já inseridos. Quando um item é escolhido para compor a solução (ou seja, inserido na placa) o ponto inferior esquerdo do item e do PR coincidem. Em seguido o vértice inferior esquerdo do PR desloca-se para o vértice inferior da direita desse item. Se não for possível alocar o novo o PR é novamente atualizado, incrementando, sempre que possível, a abcissa (eixo x) antes da ordenada (eixo y). A Figura 5 ilustra tal conceito ao longo de um empacotamento.

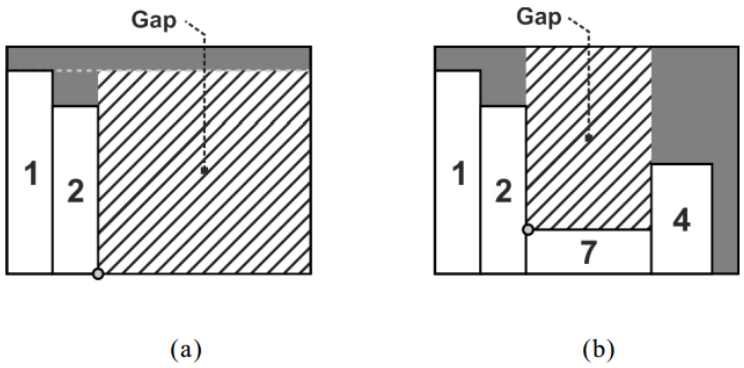
Figura 5 – Ilustração de um ponto de referência



FONTE: GOMES (2011)

No que diz respeito a *gap*, este é definido (GOMES, 2011) como uma área retangular no qual possível encaixar os demais itens. Esse é sempre definido após o ponto de referência ser atualizado ao longo do processo de corte e/ou empacotamento. A altura e largura do *gap* é definida como *gap* vertical e *gap* horizontal, respectivamente. Sendo a largura definida pelo comprimento entre o PR ao ponto direito do objeto (pedaço atual disponível para encaixe). Tal definição é importante no processo visto que, permite saber a cada instante, qual o espaço disponível para encaixe, em relação ao PR para alocar itens. Se para o *gap* calculado não couber nenhum item o PR é novamente atualizado e um novo *gap* é calculado e todo o processo é repetido.

Figura 6 – Ilustração de um *gap* em algoritmos orientados (a) e não orientados a níveis (b)



FONTE: GOMES (2011)

Em Lodi *et al.* (2002) foi realizada uma revisão geral de métodos heurísticos e exatos, bem como limites inferiores para o problema de corte e empacotamento bidimensional.

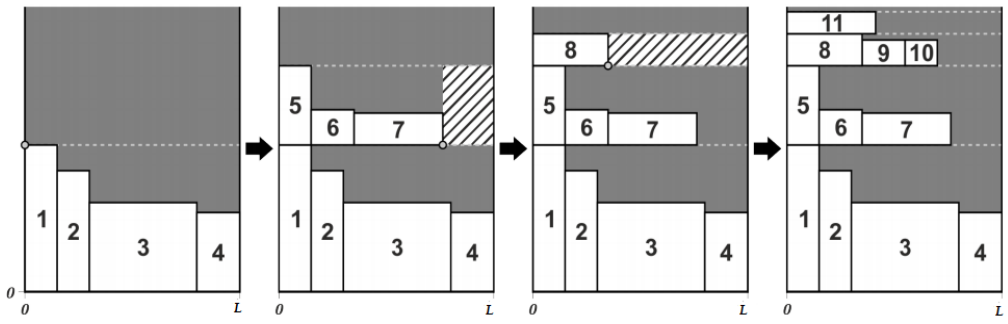
Dentre algumas heurísticas orientadas a níveis que se destacam podemos citar: *Next Fit Decreasing Height* (NFDH) e *First Fit Decreasing Height* (FFDH) propostas por Coffman *et al*. (1980). Na heurística NFDH os itens são justificados à esquerda num nível (faixa) até não haver espaço à direita para acomodar o próximo item. Nesse momento, um próximo nível é criado e o nível atual é encerrado, os novos itens serão adicionados no canto esquerdo do novo nível. A escolha dos itens é sequencial e segue o critério de ordenação por altura. O item *i+1* só pode ser escolhido após o seu antecessor *i* ter sido alocado.

Na heurística FFDH o próximo item a ser empacotado sempre é colocado no primeiro nível, a contar do primeiro nível do objeto, em que exista espaço suficiente para alocar tal item, sendo sempre justificado à esquerda. Caso nenhum dos níveis atuais, já criados, consiga acomodar este item então um novo nível é criado. O que difere os algoritmos NFDH e FFDH é que neste segundo é possível retroceder a um nível anterior para alocar o novo item escolhido, diferentemente da heurística NFDH que só aloca os itens subsequentes no nível atual ou posterior. As Figuras 8 e 9 ilustram exemplos de empacotamento utilizando as heurísticas NFDH e FFDH para uma instância de itens da Figura 7.

Figura 7 – Instância de itens para explicação das heurísticas NFDH, FFDH e MFDH

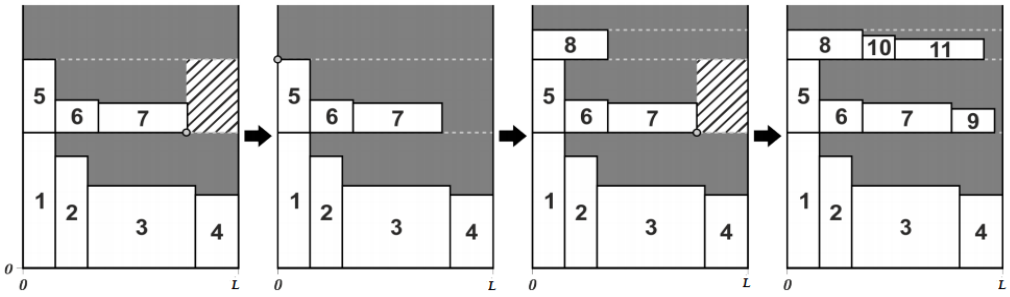
FONTE: GOMES (2011)

Figura 8 – Empacotamento aplicando a heurística NFDH



FONTE: GOMES (2011)

Figura 9 – Empacotamento aplicando a heurística FFDH



FONTE: GOMES (2011)

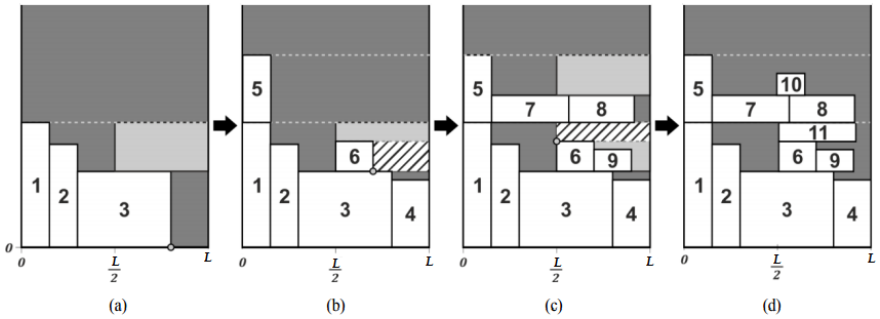
Coffman e Shor (1990) introduziram e apresentaram detalhes para a heurística *Best Fit Decreasing Height* (BFDH). Na heurística *Best Fit Decreasing Height*, insere-se o item no nível (ou faixa) em que a área do item melhor se ajustar a área disponível. Para isso, o método calcula a relação (divisão) entre a área do item e a área disponível na faixa escolhendo a faixa que apresentar essa relação. Se nenhuma faixa existente comportar o item atual, uma nova será criada, repetindo o processo até toda a demanda ser atendida.

Essas estratégias foram originalmente desenvolvidas para o problema de característica unidimensional, mas também têm sido adaptadas para o problema em estudo. Todas as três heurísticas requerem que os itens sejam ordenados de forma decrescente por altura.

A heurística *Modified First Fit Decreasing Height* (MFFDH) (RODE e ROSENBERG, 1987), similar às clássicas, apresenta a estratégia de criar sub-objetos retangulares com o objetivo de aproveitar espaços que sobram entre os diferentes níveis. A heurística MFFDH (RODE e ROSENBERG, 1987) é semelhante à FFDH, entretanto quando um dos itens ultrapassa metade da largura do objeto *(L/2),* podendo ser um único item de largura *(L/2)* ou a soma dos itens já alocados*,* como o item 3 da Figura 10 (a), um sub-objeto retangular é criado, se houver espaço. O sub-objeto criado possui largura *L/2* e sua altura é definida pela diferença entre o topo do item colocado e a altura do nível, como na Figura 10 (a). Após esse processo, as peças podem ser alocadas neste sub-objeto utilizando a estratégia FFDH. O processo de alocação dos demais itens restantes, segue a estratégia FFDH, até o momento em que um outro item em outro nível ultrapassar metade da largura do objeto *(L/2)*, sendo criado um novo objeto. Quando um item ultrapassa metade da largura do objeto e um sub-objeto é criado, a sequência de itens a ser alocado é prosseguida: primeiro tenta encaixar o item no primeiro nível, não havendo espaço,

tenta-se encaixar no primeiro sub-objeto aplicando a estratégia FFDH, passando para o próximo nível e sub-objeto, caso não haja espaço nos anteriores até o item atual conseguir ser alocado. O processo de alocação é encerrado quando todos os itens são atendidos.

Figura 10 – Empacotamento aplicando a heurística MFDH



FONTE: GOMES (2011)

Com o objetivo de também utilizar espaços não aproveitados entre os níveis, em Silva (2003) foi apresentada uma heurística chamada *Adaptative First Fit Decreasing Height* (AFFDH) e a *Split First Fit Decreasing Height* (SFFDH). Na heurística AFFDH quando o item escolhido ultrapassa metade da largura do objeto, o sub-objeto é criado por cima desse item similar à MFFDH. Quando em um nível não é possível mais encaixar itens, nesse momento o processo de alocação é guiado para o sub-objeto, até não ser mais possível alocar itens. A lógica se mantém nos outros níveis até toda a demanda de itens ser atendida. A diferença em relação à heurística MFFDH está na criação do sub-objeto, visto que este, no seu limite esquerdo, pode ser expandido para esquerda até a fronteira dos itens que estão posicionados no mesmo nível podendo sua largura ser superior a *L/2*. Seu limite direito continua a ser definido pela largura *L* do objeto. As heurísticas FFDH, BFDH, NFDH, MFDH, AFFDH e SFFDH têm em comum: a garantia do corte guilhotinado, a rotação dos itens não é permitida e a ordenação correspondente dos itens de entrada é decrescente por altura, em caso de empate na altura, de modo que o desempate é feito por ordem decrescente de largura.

Burke *et al*. (2004) também apresentou uma heurística inspirada na *Best Fit* para o problema de corte bidimensional guilhotinado e não guilhotinado com rotação dos itens permitida.

Frenk e Galambos (1987) apresentaram uma heurística nomeada *Hybrid Next Fit* (HNF) baseada na heurística NFDH. Outra heurística chamada *Hybrid First Fit* foi proposta por Chung *et al*. (1982). Berkey e Wang (1987) propuseram duas heurísticas

chamadas *Finite Next Fit* (FNF) e *Finite First Fit* (FFF) que são adaptações das clássicas para resolução de problemas *bin packing,* cujo o objeto tem suas dimensões finitas. Na heurística FFF os itens são empacotados em níveis criados diretamente dentro dos objetos. Cada item é empacotado dentro do primeiro objeto em que couber, ou, se não há objetos existentes que comporte o item, este é alocado no canto inferior esquerdo de um novo objeto inicializado. Dentro de um objeto, o item é empacotado dentro do nível mais baixo existente. Caso não haja níveis que comporte o item, um novo nível é inicializado.

Os autores Berkey e Wang (1987) também apresentaram duas novas heurísticas. A primeira heurística foi nomeada *Finite Best Strip* (FBS). Nela os itens são alocados como um problema de s*trip packing* (colocado em níveis) e posteriormente os níveis são alocados dentro de objetos com dimensões finitas. A segunda heurística é não orientada a nível e nomeada *Finite Bottom Left* (FBL).

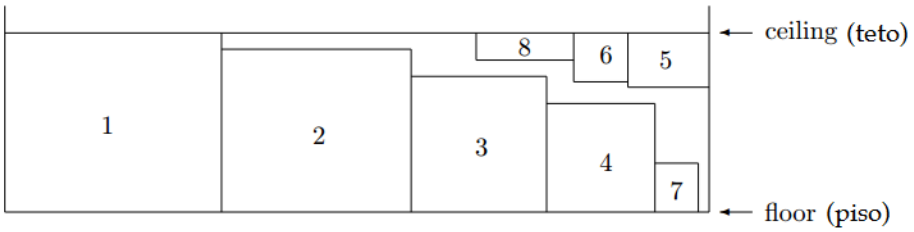
Na heurística FBS, os itens são ordenados de forma decrescente por altura, desempatados por ordem decrescente de largura e alocados um a cada vez. Cada item é adicionado ao nível em que ele couber e que tem o menor espaço horizontal reduzido. Se nenhum nível já inicializado comportar o item, um novo nível é inicializado até todos os itens serem alocados. Por fim, os níveis são alocados dentro de objetos usando a estratégia heurística de melhor encaixe (*Best Fit Decreasing Heuristic* - BFDH) para o problema de corte e empacotamento unidimensional.

Lodi *et al*. (1999) estendeu ambas as heurísticas FBS e FFF de Berkey e Wang (1987) para o caso com rotação dos itens permitida. As heurísticas utilizadas por Lodi et al. são a *Floor-Ceiling* (FC), que é uma abordagem que estende a maneira como os itens são acondicionados em níveis e a *Knapsack Problem* (KP) que é baseada no problema da mochila. A heurística *Floor Ceiling* (FC) proposta por Lodi *et al*. (1999) pode ser vista como uma melhoria da heurística FBS. Ambas as heurísticas FC e KP foram aplicadas ao PCBG. A heurística FC adiciona os itens, da esquerda para a direita, com a sua extremidade inferior no piso do nível corrente. Em seguida, adiciona os itens restantes, aqueles que não puderam ser adicionados na parte inferior, da direita para a esquerda, com a sua borda superior no teto do nível corrente. A heurística KP, primeiro encaixa os itens como no problema da mochila, em seguida separa os níveis conforme o tamanho dos objetos e aplica rotações nos itens de forma a otimizar a solução inicial.

Lodi *et al.* (1999) também apresentou uma outra heurística importante e conhecida, não orientada a níveis e, de uma fase, chamada de *Alternate Direction* (AD). AD ordena os itens de forma decrescente por altura e inicializa L placas, onde L é o limite

inferior do número necessário de caixas para empacotar. Depois disso, a parte inferior das placas são preenchidas da esquerda para direita, uma a uma, aplicando a estratégia heurística *Best-Fit Decreasing*. Nesse contexto, os itens são encaixados em níveis (faixas), da esquerda para direita, e, posteriormente da direita para esquerda utilizando os itens restantes não encaixados, até que não exista mais itens. Caso ainda haja itens e não é possível encaixar nas placas existentes, novas placas são criadas.

Figura 11 – Exemplo do algoritmo *Floor-Ceiling* (FC)



FONTE: LODI ET AL. (1999)

Ayadi *et al.* (2004) propôs e desenvolveu uma heurística híbrida para o problema de corte bidimensional guilhotinado baseada, na combinação dos algoritmos *Bottom Left* e uma heurística orientada a nível. Para este algoritmo, os objetos (bins) disponíveis para encaixe são colecionados em uma lista, assim como uma lista de peças a serem cortadas. Ao projetar padrões, para cada retângulo disponível i, um teste compara as dimensões da peça j (h_j, w_j) com as dimensões do retângulo correspondente (h_{Ri}, w_{Ri}). Retângulos disponíveis e as peças são testadas na ordem em que aparecem nas listas. Se $h_{Ri} \geq h_j$ e $w_{Ri} \geq w_j$, a peça j é selecionada para o retângulo i, se, $w_{Ri} \geq h_j$ e $h_{Ri} \geq w_j$, a peça j também é selecionada para o retângulo i. Caso nenhuma condição anterior seja satisfeita, será testada a próxima peça $j+1$ para o retângulo i, realizando os mesmos testes para as próximas peças disponíveis até que alguma satisfaça a condição; caso nenhuma peça satisfaça a condição, então será aplicado o mesmo processo para o próximo retângulo $i+1$. Os retângulos disponíveis são atualizados (removidos e outros novos inseridos) à medida que as peças vão sendo inseridas e os cortes realizados.

Jylänki (2010) implementou, comparou e apresentou uma revisão de várias técnicas heurísticas (desde as diversas formas de ordenação dos pedidos, os tipos de corte, com ou sem rotação) que podem ser usados na resolução de corte e empacotamento bidimensional.

Em Gomes (2011) foi proposto uma heurística nomeada *Guillotinable Bottom-Left First Fit Decreasing Height* (BLFFDH). Para esta heurística os itens não podem

sofrer rotação. O algoritmo tem em sua base a combinação de uma heurística *Bottom-Left* e a FFDH, citada anteriormente.

Msabah e Ali (2011) propuseram uma nova heurística de encaixe inspirada em uma combinação entre a estratégia *Bottom-Left* e algoritmo genético. A heurística tenta empacotar os itens em níveis e utiliza estratégia de explorar os resíduos entre níveis nas duas direções vertical e horizontal, garantindo restrições de guilhotina. Em seguida, a metaheurística Algoritmo Genético é aplicada, em que operações de *crossover* e mutação são executadas sobre a população no qual a representação do cromossomo é definida pela ordem dos itens encaixados utilizando a estratégia *Bottom-Left*.

Blum *et al.* (2012) abordou o problema de corte e empacotamento bidimensional com restrição de cortes guilhotinados. O primeiro algoritmo proposto é uma versão multipartida de uma heurística conhecida da literatura. Adicionalmente apresentaram um outro método de resolução evolucionário, com a heurística multipartida anteriormente proposta incorporada a este. Ambos os métodos de resolução expostos por Blum *et al.* (2012) são fortemente baseados na heurística LGFi, desenvolvida por Wong e Lee (2009). Wong e Lee (2009), apresentaram primeiramente uma heurística nomeada *Lowest Gap Fill* (LGFi), onde a rotação não é permita, e uma heurística denominada *Improved Lowest Gap Fill* ($LGFi_{OF}$), para resolver casos em que a rotação não é permitida. Ambas com duas etapas: etapa de pré-processamento e etapa de empacotamento.

A heurística LGFi (WONG e LEE, 2009) é uma adaptação da heurística LFG proposta por Lee (2008) e se caracteriza por ser uma heurística de dois estágios. Nessa heurística, os itens são ordenados em uma lista decrescente por área e desempatados pela diferença absoluta entre a largura e altura dos itens, em uma fase chamada de pré-processamento, enquanto os itens são empacotados e/ou recortados na fase de empacotamento. A fase de empacotamento é um processo iterativo, de modo que primeiramente a posição mais inferior e à esquerda (estratégia *Bottom-Left*) para a alocação do item é identificada e passa a ser a posição corrente. Em seguida, dois *gaps* são calculados o *gap* horizontal e vertical. Como já mencionado anteriormente, o *gap* horizontal é definido pela distância entre a posição corrente e a borda direita da placa. A distância entre a posição corrente e a borda superior da placa define o *gap* vertical. O valor do menor *gap* é chamado de *gap* corrente. O *gap* corrente é comparado com a largura dos itens a ser empacotados se o gap é horizontal e com a altura dos itens se o *gap* é vertical. O primeiro item que encaixar completamente no *gap* é colocado no canto inferior esquerdo indicado. Nesse *gap* se nenhum item encaixar completamente, o

primeiro item que couber sem sobrepor os limites é colocado na posição do canto inferior esquerdo atual. Se, ainda, não houver itens que possam ser encaixados, então a área é declarada como área de desperdício. Finalmente, se nenhuma posição corrente puder ser encontrada na placa atual e ainda existir itens para serem alocados então uma nova placa é criada (LEE, 2008).

A abordagem de Blum *et al.* (2012) utiliza a estratégia da heurística LGFi, citada anteriormente, de uma maneira probabilística dentro de um estágio de pré-processamento. Como já mencionado, a heurística LGFi, em sua fase de pré-processamento ordena seus itens de maneira decrescente por área, gerando uma sequência *s* de entrada de todos os itens. Na heurística multipartida *Multistart* LGFi (MS-LGFi), a cada iteração, uma nova sequência *s* de entrada é probabilisticamente criada com base na sequência original. Então esta nova entrada é passada para a heurística LGFi sendo executada a fase de empacotamento. No fim da execução do algoritmo a melhor solução encontrada, seguindo um critério explanado pelo autor, é selecionada.

Um segundo método de resolução proposto por Blum *et al.* (2012) é a aplicação de um algoritmo evolucionário (EA-LGFi) com a justificativa que as boas sequências de entradas encontradas são esquecidas a cada iteração. Um método de resolução escolhido foi o algoritmo genético formulado para o problema em questão, de modo que operações de *crossover* e mutação são aplicadas e a heurística LGFi é utilizada no processo de encaixe e avaliação da solução. Assim, boas sequências são preservadas e sequências de encaixe não promissoras são removidas.

Por fim, outro trabalho recente, para o problema de *bin packing* (corte bidimensional) com restrições de guilhotina foi proposto por (FLESZAR, 2013), onde três novas heurísticas construtivas, nomeadas, *First-Fit Insertion Heuristic* (FFIH), *Best-Fit Insertion Heuristic* (BFIH), *Critical-Fit Insertion Heuristic* (CFIH) e uma heurística de justificação ou melhoria, chamada *Justification* (J) foram apresentadas. O autor utiliza e propõe uma representação de padrões de corte (soluções) na forma de árvores, de modo que cada árvore representa um padrão de corte em uma placa. Nessa representação, todos os nós folhas da árvore representam itens, e os nós intermediários representam a disposição dos itens (lado-a-lado ou acima) em relação aos outros. Para todas as novas heurísticas apresentadas pelo autor, é utilizado uma estrutura de dados baseada em árvore para representar os padrões de corte guilhotinado, além de dispor de um novo critério para enumerar as possibilidades de inserção nas árvores e um novo critério de fitness para a escolha da melhor inserção. Com isso, os itens são ordenados seguindo um critério

específico e inseridos iterativamente um a um com base no valor do critério fitness.

Todas as três heurísticas de inserção de Fleszar (2013) são inspiradas na *First-Fit Decreasing* (FFD) e *Best-Fit Decreasing* (BFD) para o problema de empacotamento unidimensional. Tais heurísticas consideram, como visto anteriormente, os itens ordenados de forma decrescente por área. A solução é construída iterativamente onde os itens vão sendo inseridos nas árvores um de cada vez, ou seja, empacotados até não restar mais itens. Cada item é inserido em uma das *bins*(árvores) já iniciadas, caso não seja possível, uma nova bin é criada e esse item é inserido na mesma. A melhor inserção é escolhida entre as inserções possíveis pelo novo critério de fitness. A heurística FFIH tenta inserir um item de cada vez nas bins já iniciadas e, se possível, adota a melhor inserção na primeira bin (placa) que couber. A heurística BFIH adota a melhor inserção entre todas as inserções possíveis entre todas as bins. É importante ressaltar que a posições dos itens nas bins (árvores) são relativas, e só tornam-se absolutas ao término do algoritmo. A heurística de melhoria *Justification* assume que as bins estão ordenadas e busca reduzir o número de bins usadas desmontando o último padrão e tentando realocar os itens nas bins anteriores (FLESZAR, 2013). A representação de árvores em tal trabalho, segue um padrão não binário de cortes, poderia ser utilizada futuramente no presente trabalho, como forma de melhoria na altura das árvores de corte influenciando principalmente na otimização do processo de corte com a guilhotina.

Com relação a metaheurísticas a primeira metaheurística desenvolvida para o problema de corte e empacotamento bidimensional é a metaheurística Busca Tabu (do inglês *Tabu Search* - TS) LODI *et al.* (1999; 2002). As soluções iniciais são criadas utilizando heurísticas construtivas como FBS, AD ou KP. O processo de melhoria, ou movimentos de melhora, são baseados na tentativa de esvaziar certas placas (ou *bins*) para tentar reempacotar seus itens em outras placas.

Outra metaheurística, apresentada por Faroe *et al.* (2003), é baseada no método de busca local guiada (do inglês *Guided Local Search* - GLS). Esta metaheurística utiliza memória para guiar o processo de busca longe de regiões já exploradas do espaço de busca. Para isso, nesse processo é adicionado um valor de penalidade para a função objetivo penalizando soluções de características ruins visitadas previamente. A metaheurística é baseada em uma estratégia de satisfação de restrição.

Boschetti e Mingozzi (2003) apresentaram uma metaheurística muito simples, nomeada HBP, fundamentada em uma heurística gulosa. Tal metaheurística utiliza a estratégia de atribuir um *score* a cada item. Em seguida, em uma etapa de construção da

solução, os itens são considerados de acordo com o valor decrescente dos *scores*. Ao término da etapa de construção, os *scores* são atualizados seguindo um critério específico. Este procedimento é iterativo e termina quando um critério de parada é satisfeito.

Andrade, Temponi e Souza (2007) trataram o problema de corte bidimensional utilizando metaheurísticas populacionais.

Velasco (2005) e Velasco *et al.* (2008) também estudaram o problema do corte bidimensional guilhotinado e restrito utilizando uma formulação heurística baseada na metaheurística GRASP utilizando a estratégia de geração de faixas.

Temponi (2007) apresentou um estudo para os problemas de corte e empacotamento bidimensional guilhotinado e de dois estágios, quanto na forma não guilhotinada, não considerando o movimento de rotação dos itens. Para este fim, foi proposto a sua resolução utilizando três abordagens metaheurísticas, sendo elas: *Iterated Local Search* (ILS), *Greedy Randomized Adaptative Search Procedure* (GRASP) e uma metaheurística híbrida GRASP-ILS, que utiliza a metaheurística GRASP como geradora de soluções iniciais para a metaheurística ILS. Temponi (2007), na fase de construção, utiliza uma heurística parcialmente gulosa que funciona da seguinte forma: o primeiro item a ser alocado, em cada faixa, é escolhido aleatoriamente. Em seguida, são inseridos os itens que minimizam a diferença entre altura do primeiro item da faixa e o novo item a ser inserido. Pode-se perceber, que o primeiro item define a altura da faixa e os demais itens possuem altura igual ou menor a altura desta, mantendo assim, uma ordem decrescente na altura dos itens inseridos. Na etapa de busca local o método da descida randômico é aplicado. A abordagem proposta, para o caso guilhotinado, foi analisada com instâncias da literatura de Lodi *et al.* (2004).

Loh *et al.* (2009) propuseram uma abordagem nomeada por *Weight Annealing* (WA) para resolver o problema de corte e empacotamento bidimensional guilhotinado e com orientação fixa dos itens. A técnica também é uma extensão de uma heurística gulosa. Nessa técnica, pesos são atribuídos a diferentes partes do espaço de solução e têm uma influência na decisão da heurística gulosa no processo de construção de uma solução. Tais pesos são alterados durante a execução do algoritmo com base nas soluções geradas. O método WA divide-se em três fases. Na primeira fase, os itens são ordenados de forma decrescente por altura, e em seguida é construída uma solução inicial inserindo os itens já ordenados, da esquerda para direita, em sequência, dentro de níveis. Após esse processo de encaixe, há uma troca de itens entre níveis com o objetivo de minimizar o número de níveis. Na segunda fase, a solução é construída aplicando FFDH utilizando a altura do

nível a_i como o tamanho do item e a altura da placa A. Nessa fase, é realizada a troca de níveis entre placas (*bins*) com o objetivo de minimizar o número de placas *(bins)*. A terceira fase emprega estratégias para preencher espaços não utilizados em cada nível, verificando os níveis com menos itens encaixados e, deslocando esses itens para espaços não utilizados nos outros níveis não desmontados.

Teixeira *et al.* (2010) abordou o problema de corte em estudo direcionado à indústria de tecido, para o qual desenvolveram duas estratégias heurísticas: a primeira, baseada em busca local *Drop-Add* (Retirar-Adicionar) e a segunda, uma metaheurística *Simulated Annealing* com o objetivo de determinar a quantidade de cada padrão que deverá ser cortado de modo a atender a demanda de itens minimizando perdas. Para resolução do problema, foram consideradas as seguintes características: os itens alocados ortogonalmente no objeto padrão, não permissão para rotacionar itens e corte guilhotinado e restrito.

Outro método de resolução atual foi proposto em Parreño *et al.* (2010) com boa performance e resultados. Ele é uma metaheurística híbrida entre a *Greedy Randomized Adaptative Search Procedure* (GRASP) e *Variable Neighborhood Descente* (VND).

Conforme apresentado, é possível perceber que, ao longo dos anos, diversos métodos têm sido aplicados com o objetivo de encontrar boas soluções para o problema de corte e empacotamento bidimensional e guilhotinado. Dentre eles, destacam-se algumas das heurísticas clássicas apresentadas anteriormente, bem como a combinação dessas estratégias associadas a métodos metaheurísticos, como Busca Tabu, GRASP, *Iterated Local Search* (ILS) e até algoritmos evolucionários.

Por fim, para complementar o entendimento do nosso trabalho é importante explanar algumas particularidades do problema de corte. Uma delas é a forma de visualização e a outra é a representação dos padrões de corte, importantes na concepção dos algoritmos heurísticos apresentados aqui.

2.2.3 Visualização de padrões de corte

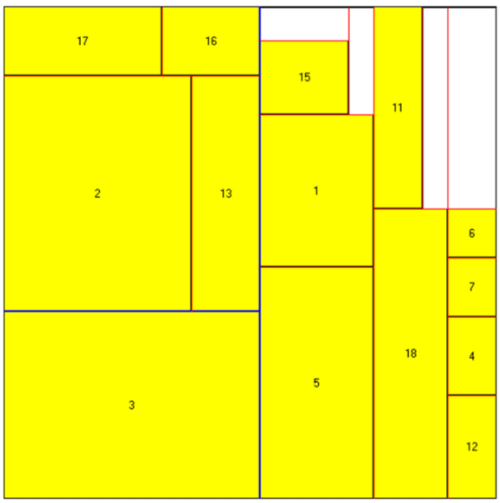
Dois tipos de visualizações de padrões de corte foram pesquisados: o layout de corte clássico, utilizado em diversos software de corte; e árvore binária de corte, apresentada nos trabalhos de (MOREIRA, 2004) e (CARDOSO, 2004).

2.2.3.1 Layout de Corte

O layout de corte é a apresentação clássica utilizada por diversos softwares de corte, onde peças são ilustradas conforme cortadas na placa. Algumas formas de interação podem ser imaginadas para manipular peças nessa visualização. Por exemplo, Moreira (2004) e Cardoso (2004) propõem três operações de interação envolvendo um *layout* de corte, o usuário e um algoritmo de corte guilhotinado. São elas:

- *Re-execução* – dada uma seleção de peças, a re-execução do algoritmo considera como conjunto de entrada o conjunto de peças selecionadas.
- *Alteração manual* – o usuário pode movimentar peças de um lugar para outro por meio de operações *drag-and-drop*, bem como rotacioná-las.
- *Restrições* – o usuário pode forçar que determinadas peças possuam posições relativas (acima, abaixo, esquerda e direita) em relação a uma outra; fixar as peças em determinadas posições e na re-execução de algoritmos tais restrições são consideradas.

Figura 12 – Layout clássico exibindo um padrão de corte



FONTE: (NASCIMENTO, ALOISE e LONGO, 1999)

2.2.3.2 Árvore Binária de Corte

Nos trabalhos de (MOREIRA, 2004) e (CARDOSO, 2004), outra forma de visualização apresentada é uma árvore binária, cujas folhas são ou peças ou sobras. É importante evidenciar que no tipo de corte guilhotinado, cada corte realizado gera outros dois pedaços, sendo cada um deles representado perfeitamente por uma sub-árvore binária.

A árvore binária de corte representa o processo hierárquico dos cortes por meio de um conjunto de nós, de modo que os nós intermediários representam um pedaço de chapa cortado, sendo rotulados como “V” ou “H”, para nomear o corte vertical ou horizontal, e as folhas podem ser peças ou sobras. Um exemplo de árvore representando o mesmo padrão da Figura 12 pode ser visualizado na Figura 13.

A estrutura de dados de cada placa (padrão de corte) utiliza essa estrutura hierárquica em árvore, a qual mantém todo o processo de corte. Segundo os autores, se o *layout* do corte for representado fisicamente usando tal estrutura, muitas funcionalidades interativas ficam mais fáceis de serem implementadas e utilizadas.

A vantagem mais clara da visualização em árvore binária é enfatizar a hierarquia dos cortes de uma solução. O usuário tem ideia de como todo o processo de corte foi realizado, aumentando seu entendimento sobre o que pode melhorar na solução. A árvore de corte permite também descobrir rapidamente a quantidade de cortes necessários para retirar uma determinada peça. Ainda, informações adicionais podem ser colocadas na representação em árvore (como a porcentagem de desperdício ou aproveitamento) sendo útil, por exemplo, na avaliação da qualidade da solução.

A desvantagem da utilização de árvore (como citada por Moreira (2004)) é que ela não reflete visualmente o posicionamento das peças no objeto final, além da impossibilidade de visualizar de forma intuitiva o tamanho das peças. A Figura 13 ilustra uma visualização baseada em árvore.

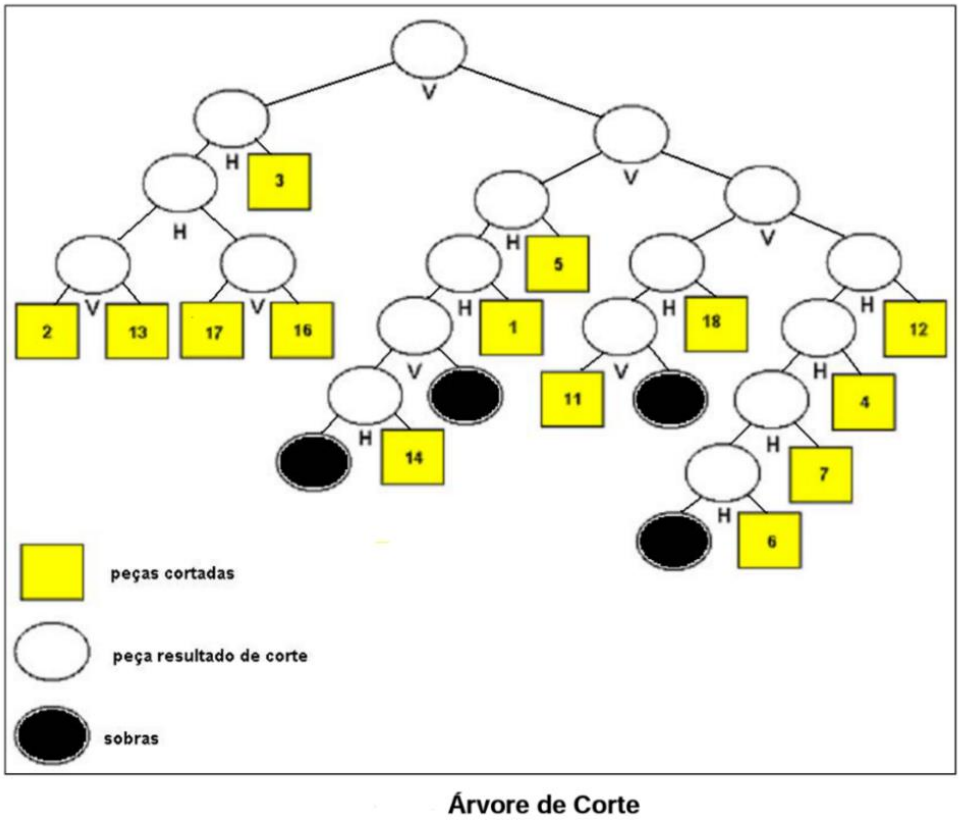
Em consequência das vantagens apresentadas pela árvore binária de corte, Cardoso (2004) e Moreira (2004) afirmaram que ela permite novas abordagens de interação com o usuário, complementando funcionalidades oferecidas pela visualização clássica (layout de corte). Três principais tipos de interação do usuário com sistemas de corte baseados em árvore foram assim propostas:

- *Re-execução* – como a árvore de corte permite a identificação rápida de trechos problemáticos (com baixa taxa de aproveitamento e/ou alta profundidade), torna-se simples focar o algoritmo em sub-regiões mal aproveitadas. Focando a exibição em determinadas sub-árvores, o usuário pode concentrar-se na interação com as partes problemáticas das mesmas.
- *Restrições* – o usuário, por exemplo, pode determinar que um conjunto de peças seja colocado em níveis superiores (necessário quando deseja-se destacar rapidamente peças de pedidos de um certo cliente); uma segunda

restrição é marcar uma sub-árvore como um *agrupamento* sendo então este considerado pelo algoritmo como uma única peça.

- *Alterações Manuais* – alterações simples como retirar ou inserir um nó, ou realizar a troca de nós. A principal vantagem das operações manuais na árvore de corte é que a restrição de guilhotina é explícita (como nós pais) e pode ser mais facilmente garantida.

Figura 13 – Layout na forma de árvore binária representando o mesmo padrão



Árvore de Corte

FONTE: (CARDOSO, 2004)

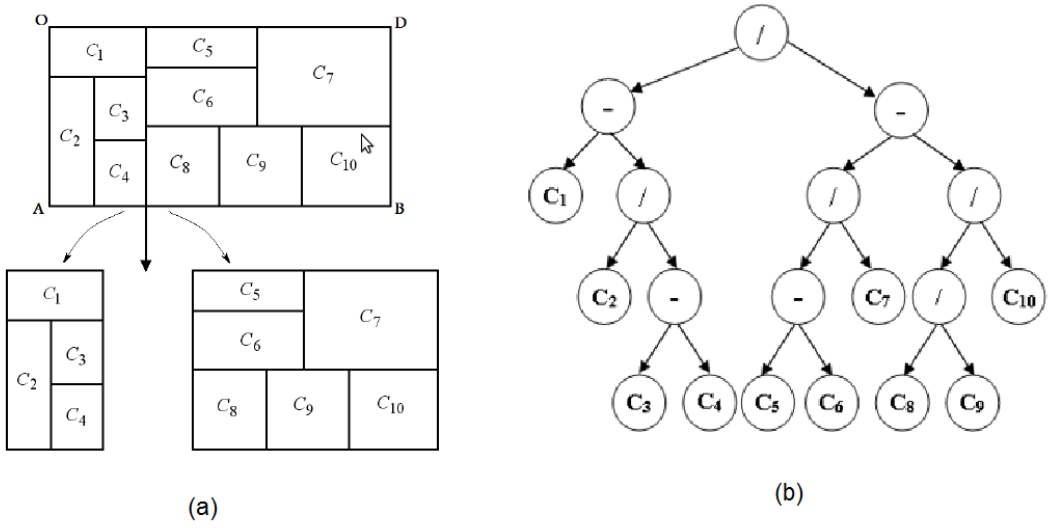
Similarmente à Cardoso (2004) e Moreira (2004), outros autores exploraram uma representação de árvore tais como, MARINESCU, BAICOIANU, *et al.*, (2009), por exemplo, apresentaram um algoritmo para determinar restrições de guilhotina considerando uma representação da árvore de corte. Nessa abordagem, *gaps*/sobras não foram tratados.

Na Figura 14 é apresentado um plano de corte e sua representação e notação na forma de árvore utilizada por (MARINESCU, BAICOIANU, *et al.*, 2009).

Conforme a Figura 14 (b e c), o símbolo “ / ” indica um corte vertical, e o símbolo

“ - ” representa um corte horizontal. Como o problema de corte é guilhotinado a cada corte realizado produz dois retângulos menores, isso pode ser visualizado na forma de uma árvore binária.

Figura 14 – Representação do padrão de corte (a) e sua notação em árvore (b e c)



(a) (b)

$$\oslash \ominus C_1 \oslash C_2 \ominus C_3 C_4 \ominus \oslash \ominus C_5 C_6 C_7 \oslash \oslash C_8 C_9 C_{10}$$

(c)

FONTE**:** (MARINESCU, BAICOIANU, *et al.*, 2009)

A representação em árvore se mostrou interessante e adequada ao nosso problema, dado que sua estrutura garante restrição de guilhotina. Assim, ela foi utilizada neste trabalho, porém em uma abordagem totalmente algorítmica, sem interação com o usuário.

Nesse âmbito, a representação do corte na forma de árvore serviu como base para concepção de um dos algoritmos de melhoria que utilizou essa estrutura no processo de busca local (refinamento) de soluções já contidas na memória. A ideia de re-execução em sub-regiões foi adotada em agentes de melhoria que, ao selecionar uma solução, utiliza as informações de aproveitamento e desperdício das sub-árvores de corte que a compõem. Os detalhes são apresentados na seção que descreve os agentes de melhoria.

2.3 MÉTODOS DE RESOLUÇÃO

Como o foco do trabalho é a resolução do Problema do Corte Bidimensional e Guilhotinado (PCBG) por meio de um *A-Team* composto por agentes heurísticos de características construtivas e de melhoria, as próximas seções têm o objetivo de

contextualizar a utilização dos métodos heurísticos utilizados na resolução do problema.

2.3.1 Heurísticas Construtivas

As heurísticas construtivas são responsáveis por construir soluções viáveis para o problema em questão. A construção da solução é geralmente feita de forma incremental, onde a cada iteração um novo elemento da lista de candidatos é escolhido para compor a solução, de acordo com uma função de avaliação. Algumas heurísticas construtivas foram descritas na Seção 2.2.1 deste capítulo. Na Figura 15, um pseudocódigo geral para heurísticas construtivas é apresentado (SOUZA, 2007).

Figura 15 – Pseudocódigo de uma Heurística Construtiva.

```
procedimento HeuristicaConstrutiva(f(·), s)
1   s ← ∅;
2   Inicialize o conjunto C de elementos candidatos
3   enquanto (C ≠ ∅) faça
4       e ← escolha um elemento do conjunto C
5       s ← s ∪ e;
6       Atualize o conjunto C de elementos candidatos
7   fim-enquanto;
8   Retorne s;
fim HeuristicaConstrutiva();
```

FONTE: SOUZA (2007)

Uma heurística construtiva pode escolher os itens para compor uma solução de forma aleatória ou usar um critério específico, como, por exemplo, a escolha de itens que maximiza ou mínima a função objetivo do problema a resolver. A vantagem das heurísticas construtivas aleatórias é que são de fácil implementação, porém a qualidade das soluções geradas é inferior as das heurísticas construtivas guiadas por um critério. No entanto, existem métodos de refinamento que, a partir de uma solução inicial, procura obter melhorias por meio de operações diversas (HOPPER e TURTON, 2000).

2.3.1.1 HHDHeuristic

A heurística de NASCIMENTO, ALOISE e LONGO (1999) foi escolhida como uma das heurísticas construtivas para a resolução do problema em estudo. Dada uma lista

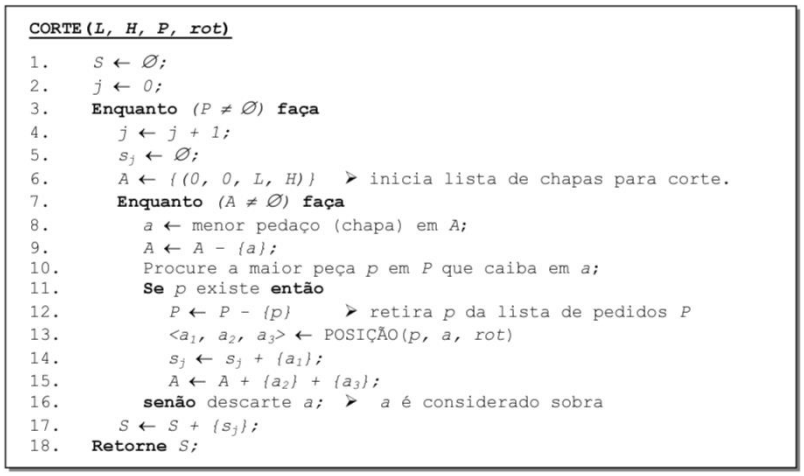
de pedidos, a serem atendidos, o algoritmo funciona da seguinte maneira:

- A lista de pedidos sempre é mantida ordenada de forma decrescente e a lista de sobras de forma crescente, ambas pela área do pedaço.
- Procura-se cortar a maior peça (pedido) cujas dimensões caibam no menor pedaço disponível (sobra);
- A peça será cortada sempre no canto inferior esquerdo da chapa ou pedaço, de modo que possa ser imediatamente destacada, isto é: pode ser realizado um corte horizontal e em seguida um corte vertical, ou vice-versa. Caso o tamanho da peça seja menor que o pedaço sendo cortado, restarão um ou dois pedaços após a operação.
- Deve ser escolhido o tipo de corte a ser realizado (horizontal ou vertical) com base em algumas estratégias:
    - Escolher o corte que maximiza o aproveitamento da área, não nula, do menor pedaço restante;
    - Escolher o corte que maximiza o aproveitamento da menor das dimensões, quanto à largura e a altura, de todas as possíveis áreas, não nulas dos pedaços restantes;

Uma descrição mais detalhada é apresentada nos pseudocódigos das 15 e 16. Eles consistem das duas rotinas fundamentais na concepção do algoritmo. A primeira rotina é chamada CORTE e a segunda é a rotina POSIÇÃO (NASCIMENTO, ALOISE e LONGO, 1999).

A rotina CORTE recebe como entrada as dimensões L (largura) e H (altura) de uma chapa (objeto) padrão, uma lista P contendo n peças com suas respectivas largura, altura e demanda, representada, na forma (l_i, h_i, q_i) e um último parâmetro *rot* referindo-se a permissão de rotação das peças. A saída gerada pela rotina é uma lista $S = \{s_1, s_2, s_3, ..., s_r\}$ contendo r padrões (layouts) de corte, onde $s_j = \{(x_1^1, y_1^1, x_2^1, y_2^1), (x_1^2, y_1^2, x_2^2, y_2^2), ..., (x_1^k, y_1^k, x_2^k, y_2^k)\}$ é uma lista de coordenadas de como cortar k peças de P para o layout de índice j, $1 \leq j \leq k$.

Figura 16 – Rotina principal CORTE.

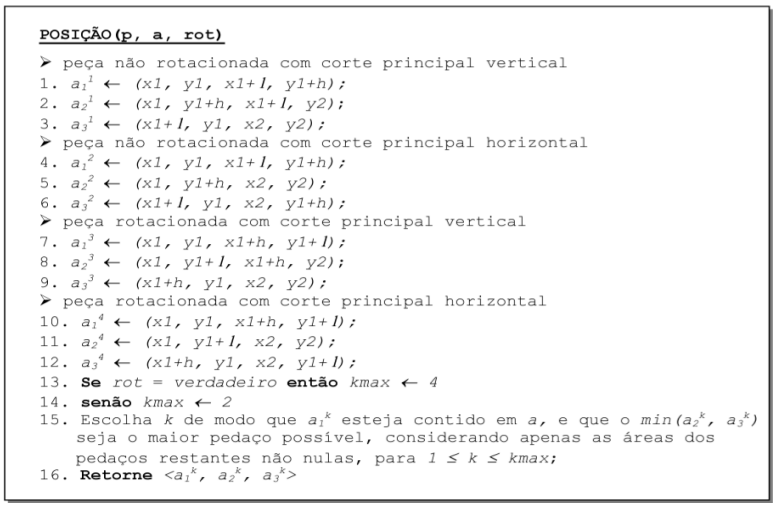


```
CORTE(L, H, P, rot)
1.    S ← Ø;
2.    j ← 0;
3.    Enquanto (P ≠ Ø) faça
4.       j ← j + 1;
5.       s_j ← Ø;
6.       A ← {(0, 0, L, H)}   ➢ inicia lista de chapas para corte.
7.       Enquanto (A ≠ Ø) faça
8.          a ← menor pedaço (chapa) em A;
9.          A ← A - {a};
10.         Procure a maior peça p em P que caiba em a;
11.         Se p existe então
12.            P ← P - {p}      ➢ retira p da lista de pedidos P
13.            <a1, a2, a3> ← POSIÇÃO(p, a, rot)
14.            s_j ← s_j + {a1};
15.            A ← A + {a2} + {a3};
16.         senão descarte a;  ➢  a é considerado sobra
17.      S ← S + {s_j};
18.   Retorne S;
```

FONTE**:** (NASCIMENTO, ALOISE e LONGO, 1999)

As variáveis A e s_j são listas de coordenadas da forma (x_1, y_1, x_2, y_2) que determinam o canto inferior esquerdo (x_1, y_1) e superior direito (x_2, y_2) de uma região retangular sobre a chapa. A lista A armazena as coordenadas dos pedaços de chapa disponíveis para o corte, enquanto s_j contém as coordenadas das peças já cortadas para o layout j. Os símbolos + e - representam as operações de inserção e remoção de um elemento nas listas, respectivamente. A rotina POSIÇÃO é responsável por determinar a posição onde uma dada peça deve ser cortada em uma chapa, assim como os pedaços restantes do corte realizado. A Figura 17, a seguir, apresenta a rotina POSIÇÃO.

Figura 17 – Sub-rotina POSIÇÃO



```
POSIÇÃO(p, a, rot)
➢ peça não rotacionada com corte principal vertical
1. a1^1 ← (x1, y1, x1+l, y1+h);
2. a2^1 ← (x1, y1+h, x1+l, y2);
3. a3^1 ← (x1+l, y1, x2, y2);
➢ peça não rotacionada com corte principal horizontal
4. a1^2 ← (x1, y1, x1+l, y1+h);
5. a2^2 ← (x1, y1+h, x2, y2);
6. a3^2 ← (x1+l, y1, x2, y1+h);
➢ peça rotacionada com corte principal vertical
7. a1^3 ← (x1, y1, x1+h, y1+l);
8. a2^3 ← (x1, y1+l, x1+h, y2);
9. a3^3 ← (x1+h, y1, x2, y2);
➢ peça rotacionada com corte principal horizontal
10. a1^4 ← (x1, y1, x1+h, y1+l);
11. a2^4 ← (x1, y1+l, x2, y2);
12. a3^4 ← (x1+h, y1, x2, y1+l);
13. Se rot = verdadeiro então kmax ← 4
14. senão kmax ← 2
15. Escolha k de modo que a1^k esteja contido em a, e que o min(a2^k, a3^k)
    seja o maior pedaço possível, considerando apenas as áreas dos
    pedaços restantes não nulas, para 1 ≤ k ≤ kmax;
16. Retorne <a1^k, a2^k, a3^k>
```

FONTE: (NASCIMENTO, ALOISE e LONGO, 1999)

A rotina POSIÇÃO recebe como entrada a descrição da peça p; a descrição do pedaço disponível para o corte a e um parâmetro lógico *rot* que indica se a rotação será

permitida ou não no processo de corte. A saída resultante da rotina POSIÇÃO é uma sequência, na forma $<a_1, a_2, a_3>$, de modo que a_1 indica as coordenadas da peça *p* no pedaço *a*, e a_2 e a_3 representa as coordenadas dos pedaços restantes resultante do corte de *a* realizado (NASCIMENTO, ALOISE e LONGO, 1999).

2.3.2 Heurísticas de Refinamento (Melhoria)

As heurísticas de melhoria ou refinamento, também conhecidas como técnicas de busca local tem como principal desígnio melhorar uma solução inicial de entrada qualquer, gerada de forma aleatória ou por uma heurística construtiva. Para esse fim, tais heurísticas utilizam o conceito de vizinhança para explorar o espaço de soluções do problema de otimização em questão (PARREÑO, ALVAREZ VALDÉS, *et al.*, 2010).

Uma estrutura de vizinhança determina para toda solução $s \in S$, um conjunto de vizinhos $V(s) \subseteq S$. Uma solução *s'* faz parte da vizinhança da solução *s* se, e somente se resultar de uma modificação em *s* de tal modo que continue a fazer parte do conjunto de soluções possíveis (TEMPONI, 2007). Dentre os métodos de busca local mais conhecidos e utilizados na resolução de problemas de otimização combinatória encontram-se: o Método de Descida e o Método de Descida Randômico.

Para o problema em estudo, as heurísticas de melhoria propostas consistem em adaptações da heurística construtiva HHDHeuristic e da metaheurística GRASP-2D trabalhando sobre um conjunto reduzido de itens. Para isso, optou-se por remover total ou parcialmente placas de uma determinada solução, seguindo um critério específico (no caso, aproveitamento) e reexecutar os métodos de resolução em sua concepção original.

2.3.3 Metaheurísticas

Metaheurísticas são definidas por Dorigo e Stützle (2004) como um conjunto de conceitos algorítmicos e de estruturas de dados genéricas para desenvolvimento e aplicação de algoritmos heurísticos voltados à resolução satisfatória de problemas NP-Difíceis. Em geral, tais algoritmos não garantem encontrar a solução ótima, porém implementam estratégias para fugir de ótimos locais o que os diferenciam de heurísticas simples. Avaliações estatísticas dessas estratégias têm mostrado o alcance com regularidade de soluções de boa qualidade com tempo computacional aceitável (MULATI, CONSTANTINO e SILVA, 2013).

As metaheurísticas são heurísticas que possuem caráter geral, podendo ser aplicadas na resolução de diferentes problemas de otimização. De acordo com Osman e Laporte (1996), Voß *et al.* (1999) *apud* Blum e Roli (2003), uma metaheurística é definida formalmente como um processo de geração iterativo que guia e modifica as operações das heurísticas subordinadas, combinando inteligentemente diferentes conceitos para explorar o espaço de busca no intuito de produzir eficientemente soluções de ótima qualidade. As heurísticas subordinadas podem ser procedimentos de alto ou baixo nível, como: uma busca local ou métodos construtivos.

Existem várias metaheurísticas e estas são classificados segundo características e pontos de vista específicos. Segundo Blum e Roli (2003), a principal forma de classificação das metaheurísticas pode ser: inspiradas ou não inspiradas na natureza e populacionais ou não populacionais.

As metaheurísticas inspiradas na natureza ensaiam e simulam situações naturais existentes na natureza. Outra característica relevante utilizada na classificação de metaheurísticas é o número de soluções usadas ao mesmo tempo para a análise do espaço de busca (uma única solução ou população de soluções). Se o algoritmo trabalha com uma única solução é considerado não-populacional. As metaheurísticas consideradas populacional executam etapas de busca que produzem a evolução de um conjunto de soluções no espaço de busca. Tais algoritmos populacionais mantém as boas soluções e tentam combiná-las, com o propósito de produzir soluções cada vez melhores (TEMPONI, 2007).

Exemplos dessas metaheurísticas são: Recozimento Simulado (*do inglês Simulated Annealing* – AS), Algoritmos Genéticos (*do inglês Genetic Algorithm* – GA) (HOLLAND, 1975), GRASP (FEO e RESENDE, 1995), Colônia de Formigas (*do inglês Ant Colony System – ACS*) (DORIGO e CARO, 1999), Enxames de Partículas (*do inglês Particle Swarm Optimization*), Programação Genética (*do inglês Genetic Programming* – GP) (KOZA, 1994), Local Iterativa (*do inglês Iterated Local Search – ILS*) (Lourenço *et al.*, 2003), Busca Tabu (GLOVER, 1986) e (GLOVER e LAGUNA, 1997), V*ariable Neighborhood Search*, Times Assíncronos (*do inglês Assynchronous Team – A-Team*) (TALUKDAR e DE SOUZA, 1990) entre outros. Essas metaheurísticas vêm sendo empregados com sucesso na solução dos mais diversos problemas de otimização, incluindo os problemas de corte e empacotamento e suas variações.

Entretanto, não é nosso propósito discorrer longamente sobre dezenas de algoritmos para o problema de corte, mas sim nos concentrar naqueles algoritmos e

trabalhos relacionados fortemente com a nossa proposta, sendo mencionados nas subseções a seguir. Assim uma breve revisão literária sobre algumas das abordagens metaheurísticas aplicada ao problema em estudo foi apresentada na Seção 2.2.1 e aqui será apresentada somente a metaheurística GRASP fundamental na concepção desse trabalho.

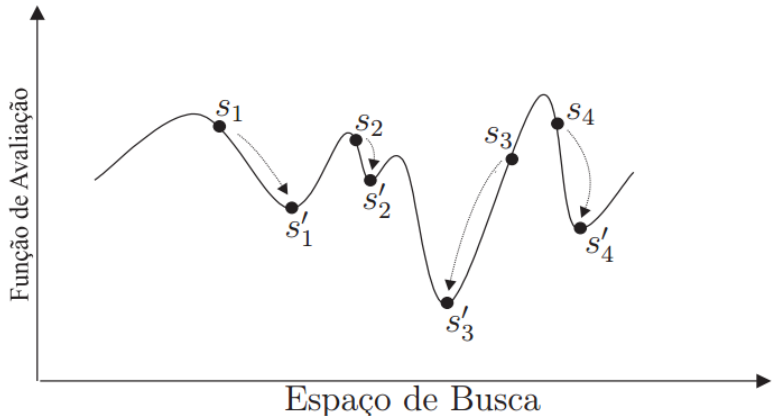
2.3.3.1 Greedy Randomized Adaptive Search Procedure (GRASP)

A GRASP é uma metaheurística multipartida proposta por (FEO e RESENDE, 1995) para otimização combinatória que aplica o método de busca local repetidamente a partir de soluções construídas por um algoritmo guloso aleatório. Desse modo, a metaheurística se caracteriza por ser iterativa e constitui-se de duas etapas: uma construtiva, na qual uma solução é gerada, elemento a elemento e uma de melhoria. Uma característica especialmente atraente dessa metaheurística é a sua fácil implementação, dado que poucos parâmetros precisam ser ajustados (RESENDE e SILVA, 2013). O primeiro parâmetro diz respeito ao número de iterações dos métodos construtivos e de busca local a serem aplicados. O segundo parâmetro, alfa, controla o enfoque entre aleatório e guloso do método construtivo.

As características gulosa-aleatória e multipartida tornam a metaheurística GRASP eficaz e muito interessante, dado que esta pode varrer um grande espaço de soluções diferenciadas. Tal característica foi fundamental para a escolha deste algoritmo a ser abordado nesse trabalho.

O método de construção guloso aleatório da metaheurística permite construir um conjunto diversificado de soluções iniciais de boa qualidade que sofrem melhorias na fase posterior de busca local. Assim a ideia básica da metaheurística consiste em utilizar diferentes soluções como ponto de partida. Com isso, o algoritmo executa, a cada iteração a heurística de construção gulosa-aleatória, gerando uma solução *s*, e em seguida, aplica a etapa de busca local sob esta solução *s,* encontrando uma solução ótima local s'. Esse processo é seguido pela avaliação da função objetivo $f(s')$ da solução ótima local s'. Caso a função $f(s')$ é melhor do que o valor da melhor solução corrente s^* essa nova solução é atribuída a solução s^* (TEMPONI, 2007). A Figura 18 ilustra o comportamento de partidas múltiplas da metaheurística GRASP, seguido das etapas de melhorias sob o espaço de soluções possíveis do problema em questão.

Figura 18 – Exemplo do comportamento da metaheurística GRASP.



FONTE: TEMPONI (2007)

Neste trabalho foi selecionado uma metaheurística GRASP (VELASCO, 2005), nomeada GRASP-2D, aplicada ao Problema de Corte Bidimensional Guilhotinado aliada a estratégia de busca local iterativa para implementação de um agente de melhoria GRASP que aplica o processo de busca local sobre um conjunto reduzido do espaço de soluções.

2.3.3.1.1 GRASP-2D aplicada ao Problema do Corte Guilhotinado Bidimensional

Como já foi citado anteriormente é uma metaheurística gulosa aleatória, caracterizada por partidas múltiplas, onde a cada iteração realiza duas fases: a primeira chamada de fase construtiva e a segunda de fase de melhoria. É caracterizada por existir um componente alfa, que determina a escolha aleatória de um dos melhores candidatos da lista de todos os candidatos, a essa lista dá-se o nome de Lista Restrita de Candidatos, conhecida também como LRC (RESENDE e SILVA, 2013).

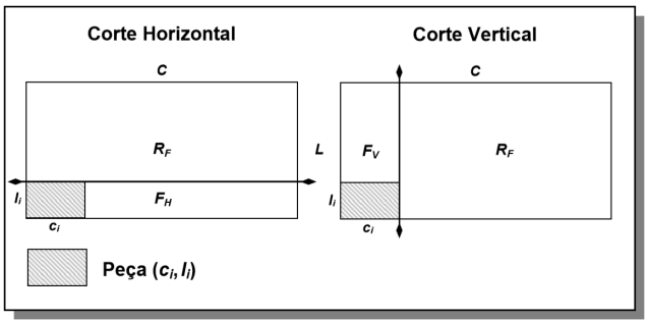
Na fase construtiva uma solução viável é produzida iterativamente, inserindo na solução parcial um elemento de cada vez, esses elementos são chamados de candidatos, que podem ser adicionados a solução respeitando as restrições de viabilidade. Na fase de melhoria são aplicados procedimento que buscam melhorar a solução construída na etapa anterior (RESENDE e SILVA, 2013).

Ao iniciar o algoritmo GRASP devem ser passados como parâmetros a lista de pedidos a ser atendidos, o parâmetro de probabilidade alfa e o número máximo de iterações. É dado como entrada para, o problema de corte, um conjunto P = {(c_1, l_2), (c_2,

l_2), ..., (c_n, l_n)} de uma lista de pedidos a ser atendidos ordenados de modo decrescente por área e as dimensões de uma placa retangular R = (C, L) de comprimento C e largura L, em quantidades suficientes. A lista de pedidos compõe a lista de candidatos e o componente alfa determina a composição da lista restrita de candidatos para o qual será selecionado um candidato (pedido) a compor a solução. Para o problema em estudo deseja-se atender toda a demanda de pedidos, minimizando a quantidades de chapas utilizadas.

A metaheurística GRASP escolhida utiliza a estratégia de lista de faixas para gerar a solução com os padrões de corte final até todo os elementos da lista de candidatos acabar. Para o problema, todas as peças são alocadas sempre no canto inferior esquerdo do retângulo R = (C, L) escolhido para ser trabalhado. Ao escolher um pedido para trabalhar, dois tipos de corte guilhotinado hipotéticos são executados sobre a placa R, corte horizontal e corte vertical, conforme ilustrado a seguir na Figura 19.

Figura 19 – Cortes Guilhotinados e faixas guilhotina



FONTE: VELASCO (2005)

Executando o corte horizontal como no exemplo da Figura 19 têm-se dois novos retângulos um chamado R_f = (C, L - l_i) que será utilizado na origem de outras faixas e um pedaço chamado F_h = (C, l_i). De modo análogo ao executar um corte vertical são produzidos os retângulos R_f = (C - c_i, L) e F_v = (c_i, L). A composição das faixas se constituirá pelo agrupamento de peças na faixa horizontal (F_h) e vertical (F_v) a cada etapa de construção e melhoria do algoritmo. A composição das faixas pelas peças deve respeitar as dimensões da faixa e o pedaço restante após a inserção de cada pedido selecionado. Após todo o processo iterativo pedaços que não são aproveitados na geração de novas faixas configurarão como perda externa no padrão de corte. Do mesmo modo as faixas geradas também podem apresentar pedaços improdutivos chamados de perda interna.

A seguir busca-se apresentar detalhadamente a metaheurística GRASP escolhida para compor o time assíncrono proposto nesse trabalho, bem como têm-se o intuito descrever as duas fases (construtiva e melhoria) características da metodologia GRASP-2D aplicada ao problema de corte em questão.

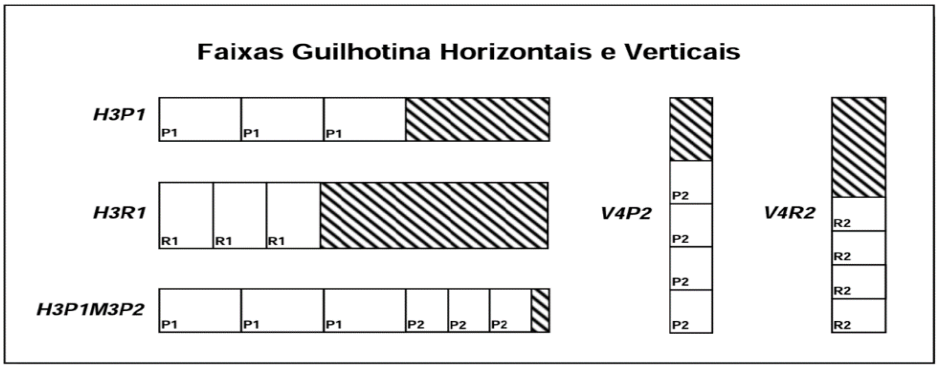
Fase Construtiva

Com base no valor atribuído a probabilidade, constrói-se uma Lista Restrita com esses candidatos (LRC), contendo os melhores elementos pertencentes a C. O valor de alpha $\alpha \in [0,1]$ irá determinar o nível de gulosidade e aleatoriedade na escolha de peças candidatas. Dessa forma, quanto mais próximo de zero for o valor do alpha, o algoritmo utilizará inicialmente peças maiores na definição das faixas (será mais guloso na escolha das peças). Em contrapartida, quanto mais próximo do valor um, maior será a chance de escolher qualquer peça aleatória para compor inicialmente as duas faixas.

Em seguida, um candidato (uma peça p_k) é selecionado de forma aleatória na LRC. Se a peça p_k resultante for maior a área da faixa a ser preenchida, excluir p_k da lista de candidatos e a escolha de uma outra peça é feita. Caso contrário, a peça resultante escolhida vai construir inicialmente e temporariamente duas faixas, uma vertical gerada por um corte vertical F_v e uma faixa horizontal gerada pelo corte guilhotinado horizontal F_h, de acordo com as dimensões da peça, como mostrado na Figura 19.

Após a inserção da peça escolhida nas faixas (vertical e horizontal) o algoritmo verifica se existe outras peças de mesmas dimensões (demanda) e insere tais peças na faixa atual até que as peças com as dimensões escolhidas extrapolem o limite restante da faixa ou toda sua demanda seja atingida, em seguida a demanda da peça é atualizada.

Figura 20 – Alocação de peças com dimensões iguais.



FONTE: VELASCO (2005)

Fase de Melhoria

- Após a etapa de construção a etapa de melhoria é iniciada, onde para cada faixa temporária (F_v e F_h) seleciona outros itens na lista de candidatos que caibam nos pedaços restantes das faixas vertical e horizontal (nessa etapa cria-se duas listas de itens, chamadas BV e BH, respectivamente). Vale ressaltar que as listas BV e BH pode conter itens diferentes;
- A lista BV compõe-se de itens ordenados por largura e refere-se a itens que podem ser encaixados no pedaço restante na faixa vertical.
- A lista BH compõe-se de itens ordenados por altura e refere-se a itens que podem ser encaixados no pedaço restante na faixa horizontal.
- Após a construção das duas listas BV e BH tenta encaixar os itens contidos na lista *BV* na Faixa Vertical e os itens da lista *BH* na Faixa Horizontal até ambas as faixas não ter mais espaço suficiente;
- Após esse processo de encaixe dos itens nas duas faixas temporárias, calcula-se o aproveitamento das duas faixas;
- A melhor faixa é selecionada (faixa com menor perda interna) e incluída definitivamente no plano de corte, em seguida:
    - Atualiza a lista de candidatos, de modo que somente os pedidos pertencentes a faixa escolhida, sejam removidos da lista de candidatos;
    - Repete o processo, voltando a fase construtiva com a nova lista de pedidos (candidatos). Onde uma nova faixa será criada a partir do pedaço restante, caso ainda exista; se ainda existir pedidos na lista e as faixas não comportar, uma nova placa será criada.

A escolha da metaheurística GRASP para composição dos agentes de inicialização e melhoria, ocorreu pelo fato da metaheurística possuir característica gulosa-aleatória determinada por um alpha, que permite a construção de soluções desde mais aleatórias a soluções mais gulosas, além de permitir a combinação de vários agentes com configurações e parâmetros diferenciados permitindo uma diversificação maior das soluções geradas. Aliada a essa metaheurística também se utilizou a ideia de busca local na implementação de agentes de melhorias, onde a metaheurística é aplicada sobre um conjunto reduzidos de itens permitindo intensificar em sub-regiões das soluções produzidas.

3. TIMES ASSÍNCRONOS

Neste Capítulo 3, será apresentado um método multi-algoritmo para a resolução de problemas combinatórios, denominado A-Team (do inglês *Asynchronous Team* ou time assíncrono) que se baseia na combinação de diversos algoritmos interagindo entre si de maneira assíncrona, a fim de obter soluções de qualidade superior às geradas pelos algoritmos de forma isolada. Será apresentado uma visão geral e introdutória aos times assíncronos: conceitos, características e topologias, bem como a descrição de memórias e agentes.

3.1 VISÃO GERAL

Algoritmos que se propõem a encontrar soluções aproximadas para diversos problemas de natureza complexa, geralmente são adequados e eficientes somente em circunstâncias particulares e específicas, determinando a diferença de eficiência entre estes. O que levou a indagações e questionamentos sobre quais heurísticas são mais adequadas a resolver um determinado problema de otimização para uma dada instância e como utilizar mais de uma heurística existente tirando proveito das características particulares de cada uma na resolução destes problemas. Tais indagações influenciaram a ideia do desenvolvimento de times assíncronos (RODRIGUES, 1996).

Segundo Talukdar e De Souza (1992) a concepção de times assíncronos surgiu de sistemas naturais e sintéticos que combinam caraterísticas particulares e individuais dos agentes como: pássaros, insetos, peixes, humanos para formar superagentes, tais como: bando de pássaros, colônia de insetos, cardumes de peixes e corporações. Definem ainda, times assíncronos como quaisquer superagentes cujos agentes são autônomos, o fluxo de dados é cíclico e as comunicações entre esses agentes são assíncronas.

Entre muitas características e vantagens de sistemas multi-agentes, as principais podem ser destacadas como: autonomia dos agentes e possibilidade de paralelização. A mais relevante é que sistemas multi-agentes oferecem um caminho natural para execução de paradigmas de cooperação, de modo que cada elemento autônomo (agente) pode ser responsável por tarefas específicas e pode implementar métodos e estratégias de soluções particulares no intuito de resolver um problema em comum.

A capacidade de inserir e retirar agentes heurísticos autônomos com características

diferentes e particulares sem a necessidade de um agente de supervisão é uma das grandes vantagens na utilização de times assíncronos, além da capacidade de encontrar soluções de boa qualidade influenciadas pela capacidade e características individuais de cada agente, aliado a utilização de recursos distribuídos.

De acordo com Peixoto e De Souza (1994) um time assíncrono consiste em uma rede cíclica de agentes que trabalham cooperativamente entre si, sem sincronismo e de forma paralela, compartilhando soluções entre os membros do time mediado por uma memória, de maneira que soluções geradas em um momento por determinado agente heurístico possam ser trabalhadas e reutilizadas por outro agente heurístico de características internas diferentes.

3.2 CARACTERÍSTICAS

A fim de compreender melhor o funcionamento de um Time Assíncrono entender suas características e componentes é extremamente relevante e faz-se necessário. Deste modo, as principais características de times assíncronos conforme (SOUZA, 1993), (TALUKDAR e DE SOUZA, 1992) e (NASCIMENTO, 1997):

- **Autonomia –** não existe agentes supervisores, os agentes trabalham de forma autônoma; não existindo relacionamento hierárquico entre agentes.
- **Comunicação assíncrona –** não deve haver sincronismo na execução dos agentes, ou seja, nenhum agente deve esperar pela execução ou processamento de soluções por outro agente. Além disso, agentes não trocam informações diretamente entre e sim estas são mediadas pelo uso de memória compartilhada.
- **Fluxo cíclico dos dados –** permite soluções (soluções contidas na memória) geradas por um determinado agente possam ser trabalhadas por outros, que acessam memórias compartilhadas. Assim os dados de saída de um agente podem ser os dados de entrada de outro. Desta maneira cada agente pode contribuir com suas características específicas para a melhorias das soluções na memória como um todo;
- **Sinergia –** acontece quando a combinação dos algoritmos se dá de maneira cooperativa produzindo resultados melhores que a execução dos agentes individualmente.
- **Consenso gradual –** ao iniciar a execução do time um grande conjunto de

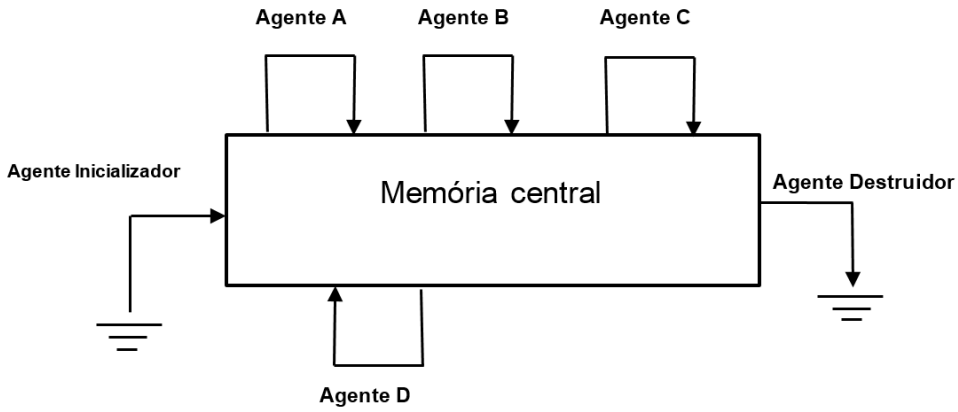
soluções variadas é considerado, mas passado algum tempo de processamento o time irá ser conduzido a um conjunto de boas soluções, restando algumas alternativas, onde estas são disponibilizadas a todos os agentes.

3.3 TOPOLOGIA E REPRESENTAÇÃO GRÁFICA

A representação gráfica para os elementos de um time assíncrono se dá da seguinte maneira: utiliza-se retângulos na representação de memórias e setas para os fluxos de entrada e saída dos agentes. As direções das setas indicam os fluxos de entrada e saída de soluções manipuladas pelos agentes (NASCIMENTO, 1997).

Na Figura 21 um exemplo de representação gráfica de um time assíncrono é exposto, onde a memória, os agentes (A, B, C e D) que o compõe são apresentados, bem como o fluxo de entrada e saída de soluções trabalhadas pelos agentes. Tal representação busca evidenciar a disposição de agentes e memórias, assim como o fluxo de dados entre esses elementos.

Figura 21 – Exemplo de representação gráfica de um Time Assíncrono.



FONTE: AUTORIA PRÓPRIA

Como visto na Figura 21, temos no exemplo um agente inicializador, responsável pelo preenchimento inicial de soluções na memória, um agente destruidor, responsável por manter a memória sempre atualizada eliminando soluções não promissoras. Há também a existência de outros quatro agentes (A, B, C, D) que podem simular construção e melhorias de novas soluções para memória central.

Ainda no exemplo da figura acima, uma solução ou conjunto de soluções da memória central, pode ser resultante de uma sequência de transformações pelos agentes

dispostos: uma ou mais transformações pelo agente A, uma ou mais transformações pelos demais agentes B, C, D.

A comunicação entre os algoritmos é realizada pela estrutura de dados usada nas memórias. Não importa o procedimento interno adotado pelo algoritmo para obter a solução, mas as soluções geradas devem ser formatadas (padronizadas) e armazenadas de forma apropriada na memória compartilhada (RODRIGUES, 1996). De modo complementar Alves e Longo (2013) apontam que o atendimento a vários requisitos, como restrições de acesso a soluções pelos agentes ou a obrigatoriedade de realizar uma sequência de transformações nas soluções pode ser necessário ao definir a topologia do time.

3.4 PROJETO DE UM TIME ASSÍNCRONO

Para a elaboração de um A-Team, devem ser observados alguns aspectos básicos, de forma a permitir diferentes formas de implementação, configuração e execução. Alves e Longo (2013) destacam alguns aspectos relevantes na elaboração de um time: análise e decomposição do problema a ser resolvido, definir parâmetros de configuração, definição da topologia dos agentes e memórias, garantir o fluxo cíclico de dados entre agentes e sequência de execução de execução de agentes. Os aspectos a seguir são descritos conforme (ALVES e LONGO, 2013).

3.4.1 Análise e Decomposição do Problema

Etapa onde o problema a ser tratado deve ser definido corretamente, identificando e compreendendo suas características, variáveis, domínio e restrições. Em seguida, é necessário a identificação dos resultados esperados na execução do time.

A definição e identificação do padrão de qualidade desejado busca diminuir a complexidade do problema e muitas vezes é guiada pela decomposição do problema em subproblemas. Cada decomposição pode ser associada a uma ou mais memórias e depende muito das características do problema, sendo uma tarefa complexa.

Os projetistas devem buscar e utilizar a melhor combinação entre os diversos métodos e algoritmos da literatura para resolver cada decomposição, com o principal intuito de manipular propriedades que possam influenciar na eficácia da proposta.

3.4.2 Representação e Qualidade das Soluções

Conforme Rodrigues (1996) uma vez que as soluções devem ser compartilhadas por todos os componentes do time, é necessário a definição de uma representação compreensível a todos os agentes e que estes devem entendê-la. Tal representação deve contemplar todas as informações relevantes a todos os algoritmos envolvidos e que esta é influenciada e dependente do problema a ser resolvido.

Apesar da exigência de uma padronização na representação de soluções, esta não é uma tarefa trivial. O cuidado para não despender muito tempo em detrimento de outras tarefas, deve ser tomado.

Em relação a qualidade das soluções, a busca é sempre pela *solução ótima*. Um tomador de decisão é uma escolha importante, já que o time fornece como resultado final um conjunto de soluções, e não apenas uma única solução final. A escolha se dá por meio de uma função de avaliação, que deve ser definida a priori, e mantém as soluções sempre ordenadas por tal critério.

3.4.3 Topologia do Time Assíncrono (A-Teams)

É na etapa de definição da topologia do *A-Team* que é estabelecida a disposição dos agentes e memórias, o fluxo de entrada e saída entre os elementos. Etapa que influencia diretamente à execução e desempenho do time.

O conjunto de agentes escolhidos, pode ser aplicado de diferentes maneiras:

- Um time formado pela execução de cópias do mesmo algoritmo, com diferentes configurações de parâmetros;
- Um time formado por diferentes tipos de algoritmo, com funções distintas dentro do time;
- A combinação das duas opções anteriores, ou seja, várias cópias do mesmo algoritmo e algoritmos distintos.

A topologia básica mostrada na Figura 20 apresenta apenas uma única memória e um conjunto de seis agentes, um agente inicializador, quatro agentes trabalhando em um fluxo cíclico de trocas de informações sobre as soluções desta memória e um agente destruidor, atualizando e removendo soluções não promissoras. Vale ressaltar que na topologia da Figura 20 apesar de simples, o conjunto de soluções nesta memória deve

possuir a mesma representação e estrutura interna, para que ocorra uma troca de informações coerente entre os agentes.

3.4.4 Fluxo de Dados Cíclicos

O fluxo de dados (soluções) cíclicos em um time deve ser garantido, permitindo que todos os agentes troquem informações e cooperem entre si, ou seja, a sequência de transformações e alterações realizadas por um agente sejam armazenadas na memória central compartilhada e novamente disponibilizadas aos demais componentes do *A-Team* (NASCIMENTO, 1997).

O fluxo de dados cíclicos na memória é exigido, de modo que as execuções dos agentes formem laços e todos agentes possam ler e escrever soluções da memória.

3.4.5 Inicialização das Memórias

As memórias são responsáveis por armazenar soluções ou informações necessárias na resolução do problema. É por meio das memórias que os resultados produzidos por um agente podem ser partilhados e manipulados por outros.

Uma vez que os resultados produzidos por um determinado agente estarão disponíveis aos demais; outros agentes podem escrever e ler sempre que desejarem, já que são autônomos, sempre respeitando políticas de acesso a esta memória. A leitura de soluções deve respeitar políticas de seleção, assim como a destruição de soluções não promissoras, executadas pelo agente destruidor, também deve respeitar políticas de destruição (RODRIGUES, 1996).

Com base nas informações acima apresentadas a diversidade de soluções na memória, é de fundamental importância e influencia diretamente o desempenho de um A-Teams. Tal problemática reforça a importância de uma boa inicialização do conjunto de soluções, ponderando a diversidade e quantidade de soluções na memória. Um objetivo básico no projeto de um A-Teams é estabelecer um percentual sempre mantendo a alta diversidade, mas sem tornar lento processo de convergência para soluções ótimas (ALVES e LONGO, 2013).

O processo de inicialização das memórias deve ser a primeira atividade a ser realizada durante a execução do time como abordado anteriormente e conforme

Nascimento (1997) podem ser iniciadas de três formas diferentes:

- **Aleatória –** a memória é preenchida com soluções geradas de forma aleatória;
- **Algoritmos de construção –** a memória é preenchida com algoritmos heurísticos construtivos;
- **Mista –** parte das soluções são geradas de forma aleatória e parte são obtidas por algoritmos de construção.

3.4.6 Agentes

Em Alves e Longo (2013) agentes são definidos como elementos responsáveis por criar, manipular e remover soluções nas memórias de um *A-Teams*. Um agente consiste em um algoritmo heurístico que se propõe a resolver determinado problema (ou parte do problema). Devido à complexidade dos problemas tratados pelo time, os agentes passam muito mais tempo processando dados para resolução do problema do que efetuando comunicações com a memória. A autonomia dos agentes permite adicionar agentes ou retirar sem qualquer aviso e sem alterar e comprometer no trabalho dos demais agentes na construção de novas soluções, podendo ser classificados, como especificado nas seções seguintes.

3.4.6.1 Agentes Construção e Inicialização

Nos times assíncronos toda comunicação acontece por meio de memórias compartilhadas. Não existindo um agente supervisor, cabe ao agente decidir sua seleção de entradas, sendo livre para escolher quando e que soluções processar.

Um agente inicializador é responsável pela construção e preenchimentos de novas soluções na memória a partir de uma definição do problema. Já um agente construtor consiste em um procedimento heurístico que se propõe resolver o problema ou parte dele e gerar novas soluções durante a execução.

3.4.6.2 Agentes de Melhoria

Agentes com função específica com a intenção de melhorar a qualidade de soluções extraídas da memória. Geralmente uma solução processada por esses agentes são depositadas novamente na memória substituindo soluções antigas. Ou respeitando

critérios específicos.

3.4.6.3 Agentes de destruição

Agentes com a função de monitorar, julgar, selecionar e eliminar soluções de uma memória, abrindo espaços para inserção de novas soluções evitando que memórias recebam quantidades de soluções indesejadas. Para isso utilizam políticas de destruição. Nos times assíncronos, estes agentes têm a importância de eliminar soluções não promissoras, geralmente sob um critério ou função de avaliação de qualidade das soluções (NASCIMENTO, 1997).

3.4.7 Políticas de Seleção e Destruição

Características como a diversidade de soluções devem ser consideradas, uma vez que uma baixa diversidade pode acarretar em uma rápida convergência do processo, prejudicando a busca por soluções de boa qualidade.

Além da diversidade outra característica extremamente relevante, é a forma como os agentes se comportam ao selecionar novas soluções. Uma escolha correta de políticas de seleção e devida associação aos agentes deve ser feita (ALVES e LONGO, 2013). Os autores ainda apresentam as três principais políticas de seleção:

- **Seleção gulosa –** sempre escolhendo a melhor ou pior solução candidata na memória;
- **Seleção com distribuição uniforme de probabilidade –** todas as soluções contidas na memória têm a mesma probabilidade de serem selecionadas;
- **Seleção com distribuição linear de probabilidade –** há uma possibilidade crescente de serem selecionadas, podendo ser da melhor para pior ou da pior para melhor.

O projetista pode escolher diferentes formas de implementação de políticas de seleção de solução. Pode escolher entre implementar políticas de seleção para cada memória, ou políticas diferentes para cada agente. Na primeira opção de implementação todos os agentes utilizam a mesma política, já na segunda cada agente pode implementar políticas de seleção particular.

3.4.8 Implementação de um Time Assíncrono (A-Teams)

Um time assíncrono pode ser implementado segundo uma arquitetura cliente-servidor, onde agentes funcionam como clientes e memórias funcionam como servidores. Os agentes (clientes) requisitam informações (soluções) à memória (servidor). Tal modelo de implementação permite que agentes e memória sejam executados concorrentemente em um computador, com um ou mais processadores, assim como, distribuídos em uma rede, com um computador alocado para cada algoritmo. Vale salientar que aplicações como times assíncronos são complexas e exigem elevado grau de processamento, sendo ideal que o código a ser desenvolvido suporte aplicações distribuídas (ALVES e LONGO, 2013).

Alves e Longo (2013) afirmam que para um bom desempenho do time as informações mantidas nas memórias compartilhadas necessitam de um controle preciso que se estende desde a inicialização, quando soluções iniciais preenchem a memória, até o momento de decisão de quais soluções não fazem parte da solução.

Há muitas formas de implementação de um Time Assíncrono. Nascimento (1997) enumera três modelos principais relacionados a forma de codificação de um time, são eles:

- **Procedimentos de um programa –** nesse modelo os agentes são implementados como procedimentos de um programa simples. Esse modelo tem a vantagem de uma fácil codificação;
- **Threads –** agentes são implementados na forma de linhas de execução independentes em um programa multi-thread. É importante ressaltar que a integridade das informações contidas na memória deve ser garantida por mecanismos de exclusão mútua. Nesse modelo é possível explorar paralelismo em sistema operacional multi-thread e cada agente pode ser alocado aos processadores disponíveis na máquina;
- **Programas independentes –** nesse modelo a memória possui uma região compartilhada pelos agentes que compõe o time e cada agente é implementado na forma de programas independentes. A abordagem cliente-servidor adota o modelo de programas independentes. Nessa abordagem os agentes são programas independentes e representam clientes e as memórias representam os servidores, fornecendo serviços aos clientes.

No modelo de programas independentes a implementação é mais complexa exigindo uma forma de comunicação entre os agentes e memórias mais sofisticadas, além de um esforço maior demandado pelo sistema operacional. Tal modelo tem a grande vantagem de permitir um alto grau de paralelismo e executar um Time Assíncrono em uma rede de computadores, onde os agentes podem ser distribuídos em máquinas paralelas e/ou convencionais.

3.5 APLICAÇÕES DE TIMES ASSÍNCRONOS

A estratégia de Times Assíncronos desde sua concepção tem sido aplicada nos mais diversos problemas de otimização combinatória. Um dos estudos pioneiros foi apresentado por Sousa (1993) aplicado ao problema do Caixeiro Viajante (*Traveling Salesman Problem - TSP*). Rodrigues (1996) aplicou a estratégia times assíncronos na busca por soluções ótimas para o Problema do Caixeiro Viajante com Várias Matrizes de Distância (*Multi Distance Traveling Salesman Problem - MDTSP*). Peixoto (1995) utilizou times para o Problema de Escalonamento de Tarefas (*Flow Shop Problem - FSP*). Em Longo (1995) uma abordagem de times assíncronos para o problema de cobertura de conjunto (*Set Covering Problem - SCP*) foi abordada e aplicada.

Nascimento (1997) aplicou a ideia de times assíncronos sugerindo uma abordagem que pudesse ser aplicada a desenho de grafos para atender diversos critérios estéticos e para manipular várias classes de grafos.

Bernardi (2001) propôs a utilização de técnicas de times assíncronos para o Problema de Corte Unidimensional. Sanchez (2008) estudou e implementou times assíncronos baseados no modelo híbrido linear aplicado a um Problema de Planejamento da Expansão da Transmissão de Energia Elétrica.

Visto que as aplicações de times assíncronos tem se mostrado eficaz na resolução dos mais variados problemas para o quais foi aplicada, e a forte característica de combinar estratégias particulares de heurísticas diferenciadas para um mesmo problema foram fatores fundamentais e motivacionais para escolha da proposta.

Em resumo, ao abordar a estratégia de A-Teams, alguns elementos são considerados essenciais na definição e construção de um time assíncrono:

- **Tamanho da população –** é muitas vezes definido experimentalmente como um compromisso entre a eficácia da abordagem sugerida e computacional do tempo

necessário para o sistema durante o processo de execução com o objetivo de encontrar resultados satisfatórios.

- **Representação da solução –** deve refletir as características típicas do problema, assegurando um tratamento eficaz da solução pelos agentes de otimização que compõe o time.
- **Métodos de criação da população inicial –** tipicamente podem utilizar mecanismos aleatórios ou heurísticas construtivas dedicadas.
- **Critérios de seleção –** usado para determinar como os agentes escolhem uma solução a partir da memória compartilhada.
- **Critério de aceitação e atualização –** como sabemos é usado para decidir se a nova solução devolvida por um agente de otimização (melhoria) é aceita para ser adicionado à na memória. A condição de aceitação/atualização utilizado nesse trabalho é a solução nova obtida ser melhor do que a pior solução produzida por todos os agentes de otimização, com a ressalva de que a nova solução deve ser diferente das soluções já contidas na memória, uma vez que a diversidade da população deve ser garantida.
- **Critério de destruição –** estabelece o critério de destruição das soluções contidas na memória.
- **Critério de parada –** utilizado para determinar quando parar o processo de execução do time. Para o critério de parada adotado neste trabalho foi a definição de um período de tempo permitido para execução.

4. TIME ASSÍNCRONO APLICADO AO PROBLEMA DE CORTE BIDIMENSIONAL GUILHOTINADO

Existe uma grande variedade de problemas de corte e empacotamento, para os quais há uma gama de algoritmos heurísticos capazes de resolvê-los e produzir soluções satisfatórias, conforme mencionado no Capítulo 2.

Segundo Nascimento (1997), na prática, para solucionar problemas de características altamente combinatória, três alternativas podem ser consideradas:

- A primeira seria selecionar e aplicar apenas um dos algoritmos. Entretanto, tem-se a desvantagem de não saber previamente qual algoritmo é mais adequado; além disso, a qualidade dos resultados pode variar conforme o tamanho das instâncias.
- A segunda alternativa se caracteriza por aplicar vários algoritmos de estratégias distintas isoladamente e identificar qual a melhor solução.
- A terceira consiste em selecionar vários algoritmos e combiná-los de tal modo que eles cooperem entre si a fim de produzir soluções satisfatórias.

A terceira alternativa se torna interessante por construir um programa que combine várias estratégias heurísticas de algoritmos distintos na resolução de um problema em comum. Esta ideia motivou a produção do presente trabalho, uma vez que ela é menos comum na resolução de problemas de corte e empacotamento.

Por conseguinte, neste Capítulo 4 apresentamos a metodologia A-Teams proposta na resolução do problema de corte estudado, bem como: a estrutura de dados utilizada, a representação das soluções e memórias e os critérios de avaliação.

4.1 VISÃO GERAL SOBRE A ABORDAGEM PROPOSTA

Este trabalho consiste em aplicar a metaheurística A-Teams para o problema de corte bidimensional guilhotinado e mostrar que é possível combinar algoritmos de problemas de corte em um time assíncrono.

4.2 ESTRUTURA DO TIME

O Time assíncrono proposto é constituído por nove agentes distintos e uma única

memória. Dois agentes construtores, seis agentes de melhoria e um agente destruidor. Os agentes construtores e de melhorias são baseados nas versões originais dos algoritmos de Velasco (2005) e Nascimento *et al.* (1999) e adaptações para obter agentes com características de melhoria.

4.2.1 Memória

A memória é formada por uma lista de soluções. O tamanho da memória é definido pelo número máximo da lista de soluções que ela possui. O tamanho da memória deve ser grande o suficiente para permitir uma alta diversidade de soluções e deve ser definido inicialmente (NASCIMENTO, 1997).

Cada solução armazena informações úteis a todos os agentes que compõe o time, como por exemplo, informações de qualidade representadas por uma função de avaliação (FAV) que avalia o aproveitamento geral da solução, quantidade total de linhas de corte, somatório total da área das sobras, quantidade de sobras, média das sobras e a lista de placas com os padrões de corte (plano de corte).

O processo de preenchimento da memória é a primeira etapa a ser realizado durante a execução do time. As soluções produzidas nessa etapa são chamadas de soluções iniciais.

A política de acesso à memória pelos agentes é delineada por políticas de seleção, de modo semelhante aos apresentados no capítulo anterior. Entre as políticas disponíveis, em nossa abordagem foram adotadas as políticas de seleção da melhor ou pior solução, e soluções aleatórias contidas na memória. As definições das políticas de seleção são parâmetros configuradas dentro de cada agente que irá compor o time.

4.2.1.1 Representação das soluções

Conforme foi citado a memória é composta por uma lista de soluções, com tamanho definido no início da execução. E como sabemos, a memória é compartilhada e é a única forma de comunicação entre os agentes que compõe o time se fazendo necessário uma padronização.

Em vista disso, para cada placa (plano de corte), a estrutura que representa o plano de corte da solução é composta por três listas:

- Um conjunto da lista de peças que foram cortadas. Cada peça possui uma posição associada indicando a região da chapa em que foi cortada.
- Um conjunto da lista de cortes efetuados para obtenção de todas as peças, consistindo da orientação do corte e sua posição no eixo em que foi realizada;
- Um conjunto da lista de sobras não aproveitadas durante o processo de corte e a posição destas na chapa.

Com base nessa representação é possível manter a padronização na representação das soluções, uma vez que cada algoritmo, especialmente, o GRASP e o HHDHeuristic, abordados na seção sobre agentes, trabalham com estruturas internas diferentes.

Deste modo, os agentes possuem informações e referências suficientemente relevantes para as demais representações e procedimentos internos utilizadas por cada um, como por exemplo, a conversão para a árvore binária de corte que será utilizado por um agente específico, os cálculos de desperdício, o aproveitamento, a quantidade de linhas de corte, a quantidade e área das sobras durante a sua execução. Não importa o tratamento e estrutura de dados interna de cada algoritmo, o agente é encarregado de codificar e decodificar as soluções obtidas da memória.

4.2.1.2 Avaliando a qualidade das soluções

Com o objetivo de avaliar e ordenar as soluções contidas na memória, a cada solução são associadas informações com base nos critérios desejados, respeitando a ordem de prioridade, são eles:

1. Função de Qualidade (FAV)
2. Quantidade total de chapas utilizadas (QCU)
3. Quantidade total de linhas de corte (QLC)
4. Quantidade total de sobras (somadas de todas as chapas) (QTS)

Antes de explicar cada critério alguns pontos são relevantes. O somatório da área dos itens, a área de desperdício e o desperdício em porcentagem de uma placa é dado pelas seguintes fórmulas, mostradas a seguir, respectivamente:

Equação 1 – Somatório da área dos itens em uma placa

$$Somatório\ área\ itens\ da\ placa = \sum_{i=1}^{n} l_i * c_i$$

FONTE: AUTORIA PRÓPRIA

Equação 2 – Área de desperdício de uma placa

$$Área\ de\ Desperdício\ da\ placa = C * L - \sum_{i=1}^{n} l_i * c_i$$

FONTE: AUTORIA PRÓPRIA

Equação 3 – Desperdício de uma placa em porcentagem

$$Desperdício\ da\ placa = 100 - \left(\left(\frac{Somatório\ área\ itens\ da\ placa}{(C * L)}\right) * 100\right)$$

FONTE: AUTORIA PRÓPRIA

- C – refere-se ao comprimento da chapa;
- L – refere-se a largura da chapa;
- c_i – refere-se ao comprimento do item *i* especificado;
- l_i – refere-se a largura do item *i* especificado;
- n – refere-se a quantidade de itens contidos na placa;

Por meio do desperdício é possível calcular o seu inverso (aproveitamento) de cada placa que constitui a solução. A função de qualidade (FAV) utilizada como critério de avaliação das soluções pode ser obtida pela seguinte equação:

Equação 4 – Função de avaliação de uma solução

$$FAV = \frac{\left(\sum_{i=1}^{j} (Somatório\ área\ itens\ da\ placa)_i\right) + d_k}{j * C * L}$$

FONTE: AUTORIA PRÓPRIA

- j – refere-se a quantidade de placas total da solução;
- d_k – está relacionado a maior área de desperdício entre todas as placas da solução. Vale salientar que antes do cálculo é necessário que todas as placas estejam ordenadas por ordem decrescente de aproveitamento;
- C * L – faz referência a área de uma placa.

Seguindo a ordem dos critérios, quando as soluções possuem a mesma função de qualidade (FAV), o critério de desempate será a quantidade de placas (menor número de placas), seguido, da quantidade total de linhas de cortes da solução (menor número), quantidade total da área de sobras e a quantidade de sobras. Tais critérios também são utilizados para diferenciar as soluções contidas na memória.

Para inserção de uma nova solução na memória duas atividades são necessárias: a primeira é ordenar as soluções já contidas na memória; a segunda é verificar se a quantidade limite de soluções já foi atingida, se não a nova solução é inserida imediatamente; caso a memória já esteja com a capacidade total atingida, então é necessário a comparação da nova solução gerada com a pior solução contida na memória. Na condição da nova solução ser melhor que a pior solução da memória e diferente das demais pertencentes a memória, então um agente destruidor é chamado e encarregado de destruir a pior solução. Ao terminar o processo de destruição a nova solução é adicionada a memória.

Os funcionamentos de cada agente são explicados detalhadamente nas subseções de agentes de inicialização/construção e melhoria, a seguir. A utilização dessas estratégias e especificamente da metaheurística GRASP, foi influenciada por sua característica gulosa-aleatória e por ser uma metaheurísticas de partidas múltiplas o que permite agentes com uma variação maior no grau de diversidade das soluções geradas, caraterística extremamente relevante para o time.

4.2.2 Agentes

Como já citado anteriormente em um time existe duas categorias de agentes produtores de soluções são os agentes inicializadores/construtores e os agentes de melhoria (refinamento).

4.2.2.1 Agentes de Construção e Inicialização

Em nossa abordagem os agentes de construção e inicializadores implementados para o problema de corte em estudo têm como base a metaheurística GRASP-2D e a heurística construtiva HHDHeuristic em sua versão original, apresentadas no Capítulo 2. Cada agente utiliza como representação inicial das soluções a representação especificada anteriormente.

Sempre ao final do processo de construção de uma nova solução, a saída é codificada no padrão de lista de placas, contendo três listas (peças, cortes e sobras), e o agente se encarrega de calcular os demais atributos que armazenam informações sobre a solução e que serão armazenados na memória.

4.2.2.2 Agentes de Melhoria/Refinamento/Modificação

Os agentes de melhoria são adaptações feitas dos algoritmos GRASP-2D e HHDHeuristic, a fim de atender características de busca local.

4.2.2.2.1 Agentes de Melhoria – Grasp2D

É um agente adaptado da GRASP construtiva original. O agente recebe como parâmetros: um alfa, um número de iterações e os parâmetros adicionais de política de seleção que determina qual solução será escolhida pelo agente, como também um valor referente ao aproveitamento das placas que influenciará quais placas serão desmontadas. O funcionamento do agente é determinado pelos seguintes passos:

- Seleciona uma solução da memória conforme a política de seleção definida;
- As placas da solução escolhida são ordenadas por aproveitamento de forma decrescente de aproveitamento;
- O agente desmonta parte da solução escolhida. Para isso, escolhe placas com valor de aproveitamento abaixo do valor definido previamente e estas são desmontadas totalmente;
- Os itens pertencentes as placas desmontadas tornam-se novamente candidatos no processo heurístico;

- Aplica o algoritmo GRASP-2D construtivo original, com o alfa, número de iterações, sobre parte da solução (novos itens candidatos);
- Ao final do processo, a solução parcial gerada (sobre o conjunto reduzido de itens) é unida à que foi mantida (placas que não foram desmontadas);
- A nova solução gerada é avaliada e os atributos de qualidade são calculados.

4.2.2.2.2 Agentes de Melhoria – HHDHeuristic

É um agente adaptado da heurística HHDHeuristic construtiva original. O agente recebe como parâmetros adicional: a política de seleção determinante para solução que será escolhida pelo agente, e um valor referente ao aproveitamento das placas que influenciará quais placas serão desmontadas por completo, além de um valor referente ao aproveitamento do nível na árvore binária de corte, que influenciará o desmonte de sub-regiões (níveis) das placas com aproveitamento inferior ao informado, de maneira que o pedaço do nível desmontado se torna novamente uma sobra e inserido na lista de sobras; e as peças pertencentes ao pedaço desmontado são removidas da lista de peças para que sejam utilizadas novamente no processo heurístico do agente de melhoria.

A representação dos planos de corte na forma de um conjunto de listas (peças, sobras e cortes), como exposto anteriormente é convertida para uma representação interna diferenciada, uma estrutura hierárquica de corte (árvore binária). Nessa representação a solução final é composta por uma "floresta de árvores" (lista de árvores).

A representação foi apresentada por Cardoso (2004) e Moreira (2004), como forma de visualização e representação do processo de corte para ser utilizada no processo interativo com o usuário. Os autores apresentam como principal vantagem da forma de visualização em árvore permitir ao usuário visualizar e desmontar trechos problemáticos (regiões mal aproveitadas) da árvore de corte e reexecutar um algoritmo sobre sub-regiões mal aproveitadas de forma interativa.

As vantagens apresentadas pelos autores inspiram a implementação de um agente capaz de converter uma solução para uma estrutura de informação baseada em árvore binária. O algoritmo utiliza as informações das listas de peças, sobras e cortes de cada placa da solução e converte para uma árvore, sem perder referências para as três listas, de maneira que manipular a árvore de corte interfere também na atualização das três listas. O agente de melhoria se encarrega de realizar cálculos adicionais de aproveitamento e

desperdício em cada nível (nó) da árvore permitindo trabalhar posteriormente em sub-regiões problemáticas.

Nessa representação os nós intermediários da árvore representam corte vertical "V" ou horizontal "H". E os nós folhas representam sobra ou peça. Por meio dessa nova estrutura de dados e representação é possível percorrer e conhecer todo o processo de corte. Para cada nó da árvore informações das dimensões de largura e altura são inseridos.

Deste modo, o agente implementado agora utiliza as vantagens da árvore, antes utilizada de forma interativa (com dicas e ajuda do usuário) agora em um processo totalmente algorítmico e automático. O funcionamento do agente de melhoria acontece seguindo as seguintes etapas:

- Seleciona uma solução da memória conforme a política de seleção definida;
- Realiza conversão para uma lista de árvores de corte. Realiza cálculos de dimensões e aproveitamento em cada nó da árvore.
- As placas da solução escolhida (árvores de corte) são ordenadas por aproveitamento de forma decrescente;
- Procura por placas com valor de aproveitamento abaixo do valor definido previamente (utiliza informações do nó raiz da árvore), se existir alguma placa com valor igual ou abaixo do especificado essa é desmontada por completo; caso a placa possua um valor superior o agente realiza uma busca em largura na árvore de corte e desmonta subníveis da árvore (sub-regiões) da placa com um valor de aproveitamento igual ou abaixo do valor de aproveitamento de nível especificado;
- Assim, o agente faz uma varredura para todas as placas (representadas agora como árvore) da solução escolhida, desmontando-as por completo ou parte da placa (nível da árvore). Para isso, utiliza informações do nó raiz e nós intermediários.
- Ao final do processo as peças antes pertencentes as placas que foram desmontadas voltam a ser pedidos não atendidos. Em caso de desmonte de um nível uma referência para as dimensões do nó desmontado é inserido na lista de sobras e as peças pertencentes a essa região da placa é inserida na lista de pedidos.
- As listas de pedidos e sobras são atualizadas depois do processo de varredura para todas as árvores da solução.
- Por conseguinte, o algoritmo HHDHeuristic é aplicado, sobre parte da solução (sobre o conjunto reduzido de itens que foram desmontados) que tenta

primeiramente verificar se existe alguma peça que caiba nas sobras das placas que não foram desmontadas.

- Se nenhum pedido couber nas sobras das placas já existentes, o algoritmo se encarrega de criar novas placas até atender toda a demanda de itens restantes;
- Por fim, a qualidade e atributos da nova solução são novamente calculados.

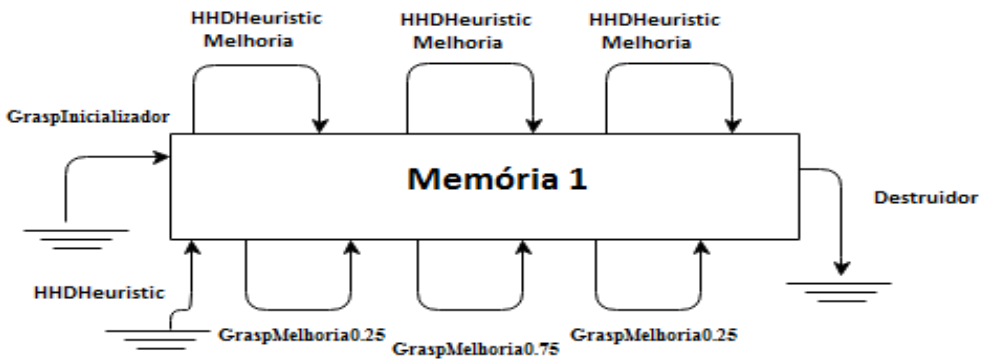
Ao término do processo é feita a decodificação da árvore novamente para a representação de lista de soluções, e cada solução é representada pelas três listas (peças, cortes e sobras).

A configuração do time para etapa de testes foi constituída da seguinte forma:

- Dois agentes chamados “GRASP_Construtivo” inicializadores. Para cada agente “GRASP_Construtivo” foram atribuídas duas configurações de alfa distintas. Os parâmetros de alpha atribuídos a cada agente foram respectivamente (0.5; 0.75) e 20 soluções geradas pelos agentes objetivando preencher a memória com soluções iniciais.
- Um terceiro agente inicializador HHDHeuristic também foi incorporado ao time gerando uma única solução, condicionado pelo fato da heurística ser determinística.
- Três agentes “HHDHeuristicMelhoria” configurados com parâmetros e políticas de seleção diferentes;
- Três agentes “GraspMelhoria” também configurados de forma distinta;
- Um agente destruidor das piores soluções;

O diagrama da Figura 22 ilustra a estrutura de composição do time para esta configuração. Seguido pela Tabela 1 que apresenta os parâmetros para o qual os agentes foram configurados.

Figura 22 – Diagrama do time assíncrono proposto



FONTE: AUTORIA PRÓPRIA

Tabela 1 – Tabela com os parâmetros de configuração dos agentes.

Agentes	Políticas de Seleção	Aproveitamento Placa	Aproveitamento Nível	Alfa
HHDHeuristicMelhoria	Melhor Solução	96%	80%	-
HHDHeuristicMelhoria	Solução Aleatória	96%	80%	-
HHDHeuristicMelhoria	Pior Solução	96%	80%	-
Grasp2d_Melhoria	Melhor Solução	96%	-	0.25
Grasp2d_Melhoria	Solução Aleatória	97%	-	0.75
Grasp2d_Melhoria	Pior Solução	97%	-	0.25

- **GraspInicializador** – constitui os agentes inicializadores que preenchem inicialmente a memória, executando a metaheurística GRASP em sua versão original.
- **HHDHeuristic –** agente inicializador que implementa o algoritmo HHDHeuristic.
- **HHDHeuristicMelhoria –** agente que lê uma solução na memória, procura por placas com aproveitamento inferior ao valor especificado, caso encontre, estas são desmontadas e os itens inseridos em uma lista. Se não encontrar placas abaixo de um aproveitamento, o mesmo percorre por níveis mais internos da placa desmontando um nível com aproveitamento igual ou abaixo do valor especificado como parâmetro. Ao desmontar um nível, o pedaço desmontado se torna sobra e é inserido na lista de sobras. Ao termino do processo de desmonte de placas, o agente procura encaixar os itens da lista de pedidos primeiramente em sobras das placas anteriores não desmontadas por completo. Posteriormente, novas placas são criadas até todos os itens da lista ser atendidos. As chapas são cortadas seguindo a heurística HHDHeuristic já comentada na seção anterior. Dessa maneira sempre é possível trabalhar sob um conjunto reduzindo de itens e encontrar arranjos melhores sob este conjunto. E com isso, convergir para boas soluções já que mantém as placas com bom aproveitamento, e tem a possibilidade de aproveitar sobras contidas em placas não desmontadas.
- **GraspMelhoria –** semelhante a metaheurística GRASP, entretanto, a cada vez que lê uma solução da memória primeiramente desmonta placas com aproveitamento abaixo do valor de aproveitamento especificado. Portanto, sempre trabalha sob um conjunto reduzindo de itens podendo convergir para boas soluções, uma vez que, mantém as placas com bom aproveitamento e tem a oportunidade de encontrar arranjos melhores sob este conjunto.

- **Destruidor –** remove as piores soluções quando a quantidade de soluções limite na memória é atingida e uma nova solução melhor foi encontrada.

4.2.2.3 Agente de Destruição

Conforme explanado acima, o agente de destruição só entra em execução quando uma solução deve ser alocada na memória e esta já está em sua capacidade total. Para isso, a política de destruição adotada e escolhida para a memória nos testes realizados, foi de característica gulosa eliminando sempre as soluções não promissoras (piores soluções).

4.3 POLÍTICAS DE SELEÇÃO

Dentre as políticas de seleção adotadas pelos agentes consiste em escolher a melhor, a pior solução ou uma solução aleatória. De acordo com essas políticas cada agente incorporado ao time pode escolher uma das três políticas de seleção na memória.

4.4 EXECUÇÃO DO TIME

Antes dos agentes de melhoria darem início ao processo de execução, o (s) agente (s) de inicialização preenchem a memória com as soluções iniciais a ser otimizadas. A partir desse momento, todos os demais agentes de melhoria iniciam seus fluxos de execução, só encerrando quando o critério de parada (tempo limite) é atingido.

Os experimentos computacionais foram realizados sobre uma classe entre dez conhecidas na literatura. Cada classe tem cinquenta entradas, dividida em dez instâncias com quantidades *n* de itens diferentes, com *n* representando a quantidade de itens {20, 40, 60, 80 e 100}. As classes se diferem pelo tamanho da placa variando de 10 x 10 à 300 x 300 e quanto a heterogeneidade (diversidade) dos itens. A classe 10 foi a classe escolhida entre as dez classes existentes disponível por Lodi *et al.* (1999) na OR Library.

Para a configuração de time escolhida foram realizados testes extras a fim de encontrar parâmetros que melhor se ajustassem aos agentes contidos no time e a classe testada.

4.4.1 Detalhes da implementação e tempo de resposta

O A-Team para o problema de corte, em estudo, abordado nesse trabalho foi implementado na linguagem Java utilizando a abordagem cliente-servidor, onde os agentes funcionam como clientes (em programas independentes) e as memórias funcionam como servidores, fornecendo serviços (acesso e escrita) aos agentes a ela conectados.

Para obter os resultados, foram testadas 50 entradas com 20, 40, 60, 80 e 100 itens, sendo 10 instâncias de cada uma. Para cada uma das 50 entradas foi realizada uma rodada de dez execuções, com tempo médio de dez minutos cada.

Os testes foram executados em um computador Intel Core 2 Duo 2.1GHz, com 6 GB de memória RAM, utilizando o sistema operacional Linux Ubuntu 12.04. Os resultados obtidos para os testes computacionais são apresentados no próximo capítulo.

4.4.2 Tamanho da Memória

Em particular estabelecemos o tamanho da memória de 50 soluções para os testes com todas as instâncias. Essa relação foi obtida de modo empírico, por meio de testes extras a priori e se mostrou suficiente na maioria dos casos. O aumento no tamanho da memória não apresentou melhorias significativas no time, além do aumento considerável no tempo de execução, já que um controle maior sobre as soluções é exigido.

Quando a capacidade total da memória é atingida e uma nova solução precisar ser armazenada na mesma, o agente destruidor é convocado e elimina a pior solução, assim abrindo espaço para novos dados.

4.4.3 Diferenciação das soluções e critérios de avaliação das soluções

Um aspecto relevante no desenvolvimento de A-Team aplicado ao problema de corte guilhotinado bidimensional diz respeito à diferenciação das soluções, ou seja como as soluções são diferenciadas uma das outras. Para este fim, a memória é sempre mantida ordenada e suas soluções são diferenciadas por uma função de qualidade e desempatadas de acordo com a seguinte ordem: menor número de placas da solução, menor quantidades de linhas de corte total da solução obtida e menor somatório da área total de sobras.

5. RESULTADOS

Este capítulo apresenta os resultados alcançados com a aplicação da abordagem proposta para cada uma das dez instâncias testadas. A fase de testes se dividiu em cinco etapas: primeiro foram realizados os testes para as dez instâncias de 20 itens, seguidos pelas dez instâncias de 40 itens e sucessivamente para 60, 80 e 100 itens.

Em cada tabela os resultados apresentados são divididos em informações referentes aos construtores no início da execução (1º bloco da tabela – parte superior) e pelo time completo no final da execução (2º bloco – parte inferior). Nas tabelas são exibidos para cada instância, os valores referentes a função de avaliação (FAV) para: a melhor solução, a pior solução, a média e o desvio padrão das soluções encontradas entre as dez execuções testadas. Para o valor de qualidade (FAV) quanto mais próximo de 1, melhor o aproveitamento geral da solução encontrada, sendo que o valor 1 representa padrões de cortes perfeitos em todas as placas.

Vale salientar que os valores apresentados na tabela de forma decimal podem ser representados também na forma de porcentagem.

5.1 INSTÂNCIAS DE 20 ITENS

A Tabela 2 apresenta os resultados obtidos para os testes com as dez instâncias de 20 itens. Os valores que constituem a tabela fazem referência a função de avaliação (FAV) que representa o aproveitamento geral da solução. Para cada instância foram selecionadas a melhor e a pior solução encontrada entre as 10 rodadas de execução, a média das funções de avaliação e o desvio padrão do valor das funções de avaliação obtidas entre as dez execuções realizadas.

Na Tabela 2 é possível perceber que para as instâncias (I1, I5 e I8) o time não conseguiu melhorar o valor da função de avaliação obtidas pelos agentes construtores para a melhor solução (observa-se que a primeira linha na melhor solução encontrada pelos agentes construtores é igual a melhor solução encontrada pelos agentes no time completo), mas obteve a média da qualidade superior à média dos construtores. Para essas três instâncias as qualidades das melhores soluções não foram diferentes, sendo 87,81% a FAV para melhor solução encontrada na instância 1. Na instância 5, o aproveitamento foi de 93,31% e para a instância 8 foi de 93,24%.

Para as demais instâncias o time conseguiu melhorar a qualidade das soluções geradas pelos agentes inicializadores no início da execução, bem como manteve a média geral superior à média dos agentes construtores.

Tabela 2 – Resultados obtidos para as dez instâncias de 20 itens.

Instância	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10
Melhor	0.8781	0.9503	0.9301	0.9218	0.9331	0.9383	0.9138	0.9324	0.9294	0.9336
Pior	0.7678	0.7433	0.7814	0.7620	0.7287	0.7494	0.7248	0.6859	0.8005	0.7916
Média	0.8209	0.8564	0.8486	0.8231	0.8351	0.8539	0.8337	0.8388	0.8491	0.8708
Desv. Padrão	0.0410	0.0536	0.0323	0.0439	0.0399	0.0346	0.0465	0.0424	0.0331	0.0275
Melhor	0.8781	0.9680	0.9584	0.9349	0.9331	0.9652	0.9380	0.9324	0.9513	0.9719
Pior	0.8781	0.9625	0.9584	0.9228	0.9331	0.9533	0.9347	0.9145	0.9405	0.9540
Média	0.8781	0.9649	0.9584	0.9266	0.9331	0.9640	0.9374	0.9270	0.9476	0.9651
Desv. Padrão	0.00	0.0028	0.00	0.0033	0.00	0.0038	0.0014	0.0086	0.0047	0.0059

5.2 INSTÂNCIAS DE 40 ITENS

Para todas as instâncias de 40 itens o time conseguiu melhorar as soluções geradas pelos construtores no início da execução e convergir sempre para boas soluções. Analisando a Tabela 3, a melhor solução encontrada pelo Time completo, bem como a média das funções de avaliação das dez rodadas foram superiores às soluções geradas pelos agentes construtores inicializadores. O desvio padrão para todas as instâncias foi inferior ao dos agentes construtores.

Por exemplo, para a instância 1 a melhor solução encontrada pelos construtores entre as dez execuções foi de 92,12% com uma média de aproveitamento geral de 86,13%. Para a mesma instância o time conseguiu encontrar a melhor solução com aproveitamento de 96,30% e uma média de 95,57%. Para a instância 2 a melhor solução entre as dez execuções obtidas pelo construtor foi de 89,69% com média de 81,86%, a melhor solução obtida pelo time após o tempo total de execução foi de 93,03% e com média de 92,99%. Também é possível visualizar que as piores soluções obtidas pelo time ao final da execução foram superiores em qualidade às obtidas pelos agentes construtores geradas no início.

Tabela 3 – Resultados obtidos para as dez instâncias de 40 itens.

Instância	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10
Melhor	0.9212	0.8969	0.9215	0.9363	0.9237	0.9286	0.9334	0.9135	0.9212	0.8564
Pior	0.7905	0.7202	0.7754	0.7501	0.7182	0.7067	0.7385	0.7473	0.7569	0.7007
Média	0.8613	0.8186	0.8572	0.8432	0.8330	0.8369	0.8400	0.8432	0.8485	0.7935
Desv. Padrão	0.0247	0.0380	0.0392	0.0403	0.0442	0.0350	0.0250	0.0247	0.0348	0.0376
Melhor	0.9630	0.9303	0.9490	0.9725	0.9607	0.9659	0.9739	0.9569	0.9623	0.9334
Pior	0.9527	0.9283	0.9395	0.9475	0.9511	0.9518	0.9630	0.9429	0.9494	0.9237
Média	0.9557	0.9299	0.9453	0.9579	0.9547	0.9623	0.9661	0.9433	0.9570	0.9284
Desv. Padrão	0.0028	0.0009	0.0034	0.0068	0.0032	0.0040	0.0031	0.0043	0.0040	0.0036

5.3 INSTÂNCIAS DE 60 ITENS

No intuito de verificar e avaliar o comportamento de sinergia do time proposto, constituído pelos seis agentes, também foram realizados 2 testes extras com duas configurações de time diferenciadas. A primeira configuração se constituiu somente por agentes GRASP de melhoria (três agentes) e a segunda configuração apenas por agentes de melhoria HHDHeuristic (três agentes). Ambos os agentes com os mesmos parâmetros de configuração do time completo composto pelos seis agentes de melhoria. A ideia foi avaliar o comportamento de sinergia do time completo em relação as outras duas configurações.

Para os testes foram escolhidas somente as instâncias com quantidades de 60, 80 e 100 itens. Como já foi abordado, a sinergia acontece quando a combinação dos agentes se dá de maneira cooperativa produzindo resultados melhores que a execução dos agentes individualmente. Para isso, nas demais tabelas (60, 80 e 100 itens), que referenciam o time completo, foi incluída a informação de sinergia que reflete o número de vezes (entre as dez execuções) em que o time foi melhor do que às outras duas configurações. De forma complementar mais adiante foi inserida tabelas adicionais, que apresentam os resultados encontrado pelos times compostos somente por agente GRASP e por agentes HHDHeuristic. Onde será interessante observar as linhas correspondentes entre as três tabelas (o 2º bloco, na parte inferior, da tabela referente ao time completo e as demais tabelas).

Analisando a Tabela 4, na linha que referencia sinergia, para as instâncias I3 e I6

o time completo teve desempenho inferior à configuração composta só por agentes GRASP, onde só conseguiu uma função de qualidade superior apenas em 3 das 10 execuções realizadas para a instância 3 e somente 4 das 10 execuções para a instância 6. A melhor solução encontrada pelo time completo com os seis agentes para a instância 3 obteve uma FAV de 95,13%, todavia a configuração composta exclusivamente por agentes GRASP encontrou sua melhor função de avaliação 95,95%. Para a instância 6 apesar de ter perdido em 6 das 10 execuções realizadas a melhor solução encontrada pela configuração de agentes GRASP obteve 95,93% enquanto a melhor solução encontrada pelo time completo foi de 96,27%.

Para as instâncias I2, I4, I5, I7, I8, I9 e I10, o time completo obteve resultados melhores do que as outras duas configurações na maioria das execuções realizadas se mantendo superior em 7, 8, 6, 8, 6, 6 e 10, respectivamente para cada instância, entre as dez execuções realizadas. É importante observar que para a instância 10 de 60 itens a configuração do time completa foi superior em todas as dez execuções realizadas em comparação às outras duas configurações composta só por agentes GRASP e agentes HHDHeuristic, para tal instância a função de qualidade da melhor solução encontrada pelo time entre todas as execuções foi de 96,91% e uma média de qualidade de 95,84%.

Em relação as soluções geradas pelos agentes inicializadores/construtores, o time se manteve superior em todas as execuções e instâncias, sempre conseguindo evoluir para soluções melhores.

Tabela 4 – Resultados obtidos para as dez instâncias de 60 itens (Time completo).

Instância	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10
Melhor	0.9011	0.9034	0.9109	0.9059	0.9349	0.8969	0.9277	0.8803	0.9165	0.9078
Pior	0.7311	0.7771	0.7772	0.7586	0.7933	0.7342	0.7992	0.7335	0.7494	0.7494
Média	0.8331	0.8560	0.8498	0.8352	0.8697	0.8278	0.8736	0.8173	0.8406	0.8463
Desv. Padrão	0.0309	0.0247	0.0273	0.0319	0.0251	0.0271	0.0221	0.0328	0.0368	0.0322
Melhor	0.9391	0.9518	0.9513	0.9752	0.9751	0.9627	0.9701	0.9509	0.9784	0.9691
Pior	0.9269	0.9440	0.9303	0.9648	0.9644	0.9471	0.9590	0.9353	0.9601	0.9508
Média	0.9319	0.9487	0.9375	0.9688	0.9710	0.9529	0.9648	0.9421	0.9686	0.9584
Desv. Padrão	0.0042	0.0023	0.0073	0.0040	0.0033	0.0056	0.0031	0.0060	0.0056	0.0063
Sinergia	5	7	3	8	6	4	8	6	6	10

Tabela 5 – Resultados obtidos para as dez instâncias de 60 itens (Time GRASP).

	Instância	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10
60 itens	Melhor	0,93840	0,95260	0,95950	0,97300	0,97680	0,95930	0,96870	0,94460	0,97150	0,96130
	Pior	0,92660	0,93980	0,93420	0,95900	0,96340	0,94640	0,95430	0,93450	0,96580	0,94250
	Média	0,93252	0,94533	0,94819	0,96528	0,97010	0,95259	0,96190	0,93881	0,96812	0,94990
	Desv.Padrão	0,00397	0,00430	0,00874	0,00463	0,00411	0,00401	0,00416	0,00417	0,00223	0,00572

Tabela 6 – Resultados obtidos para as dez instâncias de 60 itens (Time HHDHeuristic).

	Instância	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10
60 itens	Melhor	0,91703	0,92619	0,91328	0,96641	0,95624	0,91879	0,96038	0,92561	0,96070	0,91830
	Pior	0,90130	0,90862	0,90651	0,95710	0,94344	0,91572	0,95323	0,89925	0,93057	0,90920
	Média	0,90545	0,91483	0,91095	0,96157	0,95096	0,91664	0,95636	0,91277	0,94383	0,91447
	Desv.Padrão	0,00544	0,00574	0,00247	0,00314	0,00422	0,00148	0,00216	0,01184	0,01013	0,00304

5.4 INSTÂNCIA DE 80 ITENS

Analisando a Tabela 7, para as instâncias de 80 itens, o time completo teve um desempenho inferior para as instâncias I8 e I10 em comparação ao resultado obtido pela configuração composta só por agentes GRASP, no qual só conseguiu valores de aproveitamento (FAV) superior em 4 das 10 execuções para a instância 8 e em apenas 3 das 10 execuções realizadas para a instância 10. Para a instância 8 a melhor solução encontrada pelo time completo foi de 97,58%, já para a configuração de agentes GRASP a melhor solução encontrada foi de 97,45%.

Para a instância I3, o time obteve valores de aproveitamento superior em 9 entre as dez execuções realizadas, só perdendo em uma execução, onde a configuração composta somente por agentes HHDHeuristic obteve um aproveitamento de 97,07% e o time completo 96,78%. Entretanto, é possível observar que a melhor solução encontrada pelo time entre as dez execuções ainda é superior à 97,07% com o valor de 97,67%.

Nas demais instâncias manteve qualidade igual e superior às outras duas configurações. Observa-se também que para todas as instâncias o time conseguiu manter a média de aproveitamento superior às geradas pelos agentes construtores.

Tabela 7 – Resultados obtidos para as dez instâncias de 80 itens (Time completo).

Instância	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10
Melhor	0.9252	0.9100	0.9094	0.9378	0.8916	0.9100	0.9117	0.9233	0.9179	0.9100
Pior	0.7282	0.7373	0.7679	0.7961	0.7494	0.7373	0.8121	0.8104	0.8071	0.7373
Média	0.8438	0.8291	0.8508	0.8635	0.8441	0.8276	0.8607	0.8862	0.8653	0.8354
Desv. Padrão	0.0338	0.0312	0.0295	0.0262	0.0299	0.0325	0.0229	0.0220	0.0222	0.0320
Melhor	0.9611	0.9706	0.9767	0.9709	0.9643	0.9506	0.9651	0.9758	0.9736	0.9481
Pior	0.9423	0.9527	0.9658	0.9609	0.9373	0.9296	0.9426	0.9557	0.9588	0.9352
Média	0.9515	0.9590	0.9695	0.9658	0.9471	0.9429	0.9496	0.9651	0.9641	0.9402
Desv. Padrão	0.0064	0.0059	0.0034	0.0037	0.0078	0.0083	0.0067	0.0074	0.0052	0.0043
Sinergia	5	5	9	7	5	7	6	4	7	3

Tabela 8 – Resultados obtidos para as dez instâncias de 80 itens (Time GRASP).

	Instância	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10
80 itens	Melhor	0,96810	0,96610	0,96520	0,97130	0,95920	0,94130	0,95310	0,97450	0,96482	0,94530
	Pior	0,94210	0,95340	0,95660	0,95650	0,93370	0,93410	0,94430	0,96030	0,95185	0,92730
	Média	0,95195	0,95978	0,96100	0,96388	0,94479	0,93778	0,94848	0,96769	0,95807	0,94168
	Desv.Padrão	0,00870	0,00448	0,00246	0,00487	0,00766	0,00261	0,00291	0,00460	0,00397	0,00577

Tabela 9 – Resultados obtidos para as dez instâncias de 80 itens (Time HHDHeuristic).

	Instância	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10
80 itens	Melhor	0,93920	0,95240	0,97070	0,95280	0,94390	0,92750	0,94200	0,95662	0,95600	0,93770
	Pior	0,92070	0,94500	0,96010	0,92600	0,93370	0,92070	0,92930	0,94050	0,94140	0,93150
	Média	0,92810	0,94847	0,96480	0,94117	0,93641	0,92389	0,93456	0,95059	0,94798	0,93530
	Desv.Padrão	0,00514	0,00199	0,00312	0,00904	0,00385	0,00207	0,00356	0,00712	0,00463	0,00210

5.5 INSTÂNCIA DE 100 ITENS

Para as instâncias de 100 itens o time ganhou em 7 das 10 execuções das outras duas configurações para as instâncias I1, I2 e I3. Na instância 1 a melhor solução encontrada pelo time de agentes GRASP foi de 96,51%, já o time completo foi de 96,68%. Para a instância 2 a melhor solução encontrada pelo time completo foi de 95,95% perdendo para a melhor solução encontrada pelo time de agentes GRASP com 96,05%. Já para a instância 3 a melhor solução do time completo foi 97,37% sendo superior à configuração de agentes GRASP com melhor aproveitamento de 96,76%.

Para as instâncias I4, I5 e I8 ganhou em 6 execuções. O time obteve desempenho inferior às outras duas configurações para as instâncias I6 e I9 ganhando somente em três execuções na instância 6 e somente em uma 1 execução para a instância 9. Em relação às soluções geradas pelos agentes inicializadores no início das execuções o time conseguiu melhorar todas as soluções, sempre encontrando soluções de qualidade superior. Como pode ser visto na Tabela 7 comparando a qualidades das melhores, piores e a média para todas as instâncias o time completo conseguiu encontrar soluções melhores e com média bem superior.

Tabela 10 – Resultados obtidos para as dez instâncias de 100 itens (Time completo).

Instância	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10
Melhor	0.9095	0.9241	0.9099	0.8984	0.9106	0.9096	0.9376	0.8846	0.9144	0.9068
Pior	0.8240	0.8014	0.7741	0.7664	0.7804	0.7951	0.8311	0.7371	0.7831	0.8160
Média	0.8682	0.8668	0.8580	0.8374	0.8620	0.8585	0.8782	0.8218	0.8636	0.8715
Desv.Padrão	0.0233	0.0236	0.0257	0.0297	0.0242	0.0243	0.0226	0.0298	0.0237	0.0169
Melhor	0.9668	0.9595	0.9737	0.9607	0.9636	0.9643	0.9737	0.9457	0.9613	0.9597
Pior	0.9554	0.9502	0.9602	0.9378	0.9441	0.9369	0.9654	0.9372	0.9311	0.9321
Média	0.9620	0.9552	0.9651	0.9455	0.9553	0.9513	0.9711	0.9417	0.9434	0.9436
Desv. Padrão	0.0039	0.0032	0.0039	0.0063	0.0054	0.0074	0.0027	0.0022	0.0100	0.0091
Sinergia	7	7	7	6	6	3	5	6	1	5

Tabela 11 – Resultados obtidos para as dez instâncias de 100 itens (Time GRASP).

	Instância	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10
100 itens	Melhor	0,96510	0,96050	0,96760	0,96320	0,95960	0,95950	0,97730	0,94710	0,96564	0,95610
	Pior	0,95310	0,94510	0,95470	0,92860	0,94710	0,94500	0,96390	0,93160	0,94081	0,93150
	Média	0,95894	0,95269	0,96322	0,94292	0,95387	0,95439	0,96982	0,93981	0,95437	0,94444
	Desv.Padrão	0,00398	0,00446	0,00362	0,00966	0,00434	0,00532	0,00509	0,00519	0,00780	0,00978

Tabela 12 – Resultados obtidos para as dez instâncias de 100 itens (Time HHDHeuristic).

	Instância	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10
100 itens	Melhor	0,95100	0,94890	0,95850	0,94280	0,94360	0,93700	0,96510	0,94510	0,93910	0,93530
	Pior	0,94180	0,93920	0,93520	0,92600	0,93960	0,91980	0,94970	0,93820	0,92450	0,92125
	Média	0,94561	0,94236	0,94627	0,93338	0,94132	0,92891	0,95593	0,94134	0,92865	0,92792
	Desv.Padrão	0,00302	0,00279	0,01033	0,00623	0,00147	0,00442	0,00469	0,00227	0,00414	0,00478

Para as execuções e instâncias em que o time apresentou qualidade inferior às outras duas configurações a diferença foi mínima. Uma justificativa deve-se ao fato da

característica gulosa-aleatória da metaheurística GRASP influenciada pelo alfa pode explorar diferentes espaços de soluções. Para tais instâncias seria interessante uma nova calibração e testes com alfas diferenciados, uma vez que para determinadas instâncias uma combinação diferente de parâmetros pudesse se mostrar superior.

5.6 AVALIAÇÃO DA ABORDAGEM

Com o objetivo de avaliar o comportamento geral do time, o resultado para as cinquenta entradas da classe 10 foram comparadas com resultados encontrados por outros algoritmos disponíveis na literatura para a mesma classe de instâncias. Dentre os métodos de resolução selecionados podemos destacar a abordagem GRASP híbrida de (PARREÑO, ALVAREZ VALDÉS, *et al.*, 2010), a abordagem HBP de (BOSCHETTI e MINGOZZI, 2003), os algoritmos Busca Tabu (TS) de (LODI, 1999) e (LODI, MARTELLO e MONACI, 2002) , a abordagem busca local guiada (GLS) de (FAROE, PISINGER e ZACHARIASEN, 2003), a metaheurística *Weight Annealing* (WA) de (LOH, GOLDEN e WASIL, 2009), a heurística multipartida MS-LGFi e abordagem evolucionária EA-LGFi, ambas de (BLUM, SCHMID e BAUMGARTNER, 2012), as heurísticas de inserção *First-Fit Insertion Heuristic* (FFIH), *Best-Fit Insertion Heuristic* (BFIH) e *Critical-Fit Insertion Heuristic* (CFIH), e a combinação destas com uma heurística de melhoria *Justification* (J) de (FLESZAR, 2013), bem como as heurísticas *Finite Best-Strip* (FBS2) e *Floor-Ceiling (*FC2), adaptadas de (BERKEY e WANG, 1987) por (LODI, MARTELLO e VIGO, 1999) também encontradas em (FLESZAR, 2013) e apresentados na Tabela 7.

Tabela 13 – Comparação dos resultados obtidos.

Método/Nº itens	20	40	60	80	100	Ano
FBS2	42	75	103	131	163	1999
FC2	41	74	101	130	163	1999
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
GRASP	42	74	100	129	159	2010
HBP	42	74	102	130	160	2003
TS	43	75	104	130	166	2002
GLS	42	74	102	130	162	2003
WA	43	74	102	129	159	2009
MS-LGFi	42	74	101	129	160	2012
EA-LGFi	42	74	101	128	160	2012
Time Assíncrono	41	73	100	127	159	2013/2014

A Tabela 13 acima refere-se a quantidade total de placas encontradas (somadas) para cada instância (1 à 10) para as quantidades de itens (20, 40, 60, 80 e 100). As últimas três colunas se refere a três configurações de time testadas. A configuração "GRASP-HHDHeuristic" faz referência a configuração apresentada anteriormente nesse trabalho.

A Tabela 14 apresenta o número de placas encontradas para a metaheurística GRASP configurada com diferentes alfas e a heurística construtiva e determinística HHDHeuristic. É possível perceber que as duas estratégias utilizadas isoladamente apresentaram resultados bem inferiores aos apresentados por algumas heurísticas da Tabela 13, entretanto ao ser incorporadas dentro de um time e aliadas a estratégias de melhoria apresentaram resultados de qualidade superior em termos de placas encontradas e no arranjo dos itens.

Tabela 14 – Resultados obtidos para testes com a metaheurística GRASP e heurística HHDHeuristic.

GRASP/nº itens	Alfa	20	40	60	80	100
	0.1	41	74	102	130	163
	0.2	41	73	101	130	164
	0.25	41	73	102	130	164
	0.5	41	74	102	133	165
	0.75	41	75	105	135	169
	0.8	41	75	105	137	170
	0.9	41	75	105	138	170
	1	41	75	106	140	175
HHDHeuristic	—	46	76	105	134	166

5.6.1 Evolução das soluções obtidas

Nas figuras a seguir é apresentada a evolução da qualidade das soluções geradas durante a melhor entre as dez execuções realizadas para as instâncias de 1 e 2. O propósito aqui é apresentar alguns exemplos dos gráficos de dispersão das soluções geradas durante a execução do time. Para mostrar o quadro de evolução foram selecionadas as entradas de 60, 80 e 100 itens.

O eixo x representa o tempo, em segundos, que a solução foi gerada. O eixo y representa a função de qualidade (FAV), em porcentagem, da solução gerada. O gráfico de dispersão evidencia as variações das qualidades das soluções geradas durante o tempo de execução do time.

Os agentes inicializadores levam em média de 5 a 20 segundos para preencher inicialmente a memória, sendo encerrado em seguida para o início dos agentes de

melhoria, que só finalizam o processo de otimização após o tempo limite ser atingido.

Figura 23 – Variação da qualidade de soluções para Instância 1 - 60 itens.

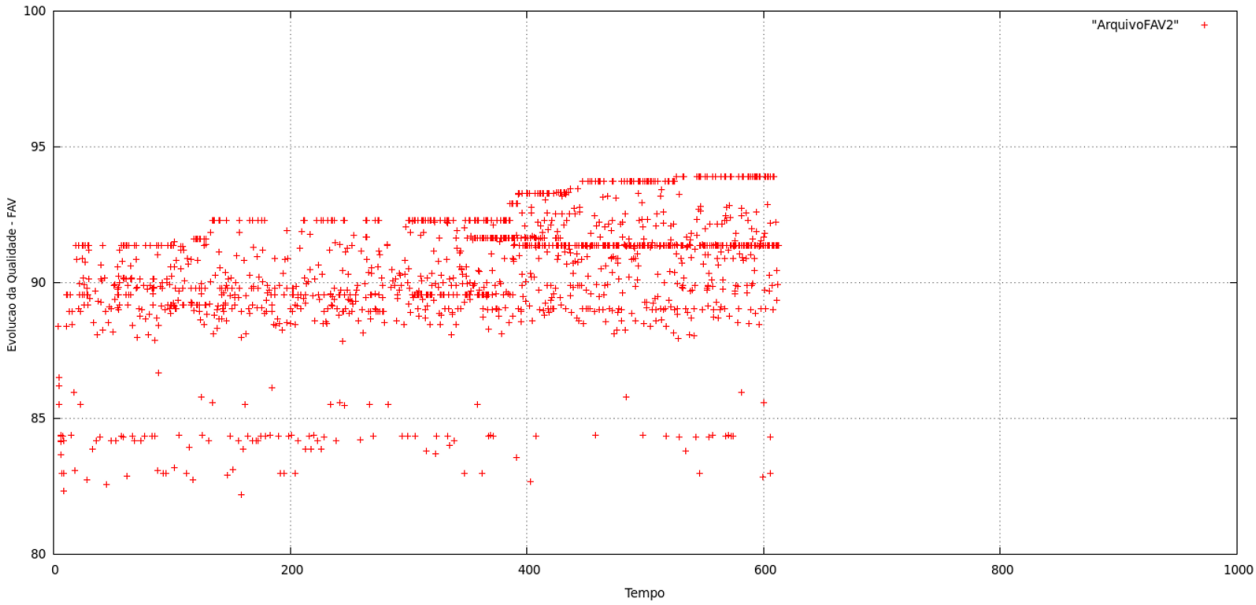


Figura 24 – Variação da qualidade de soluções para Instância 1 - 80 itens.

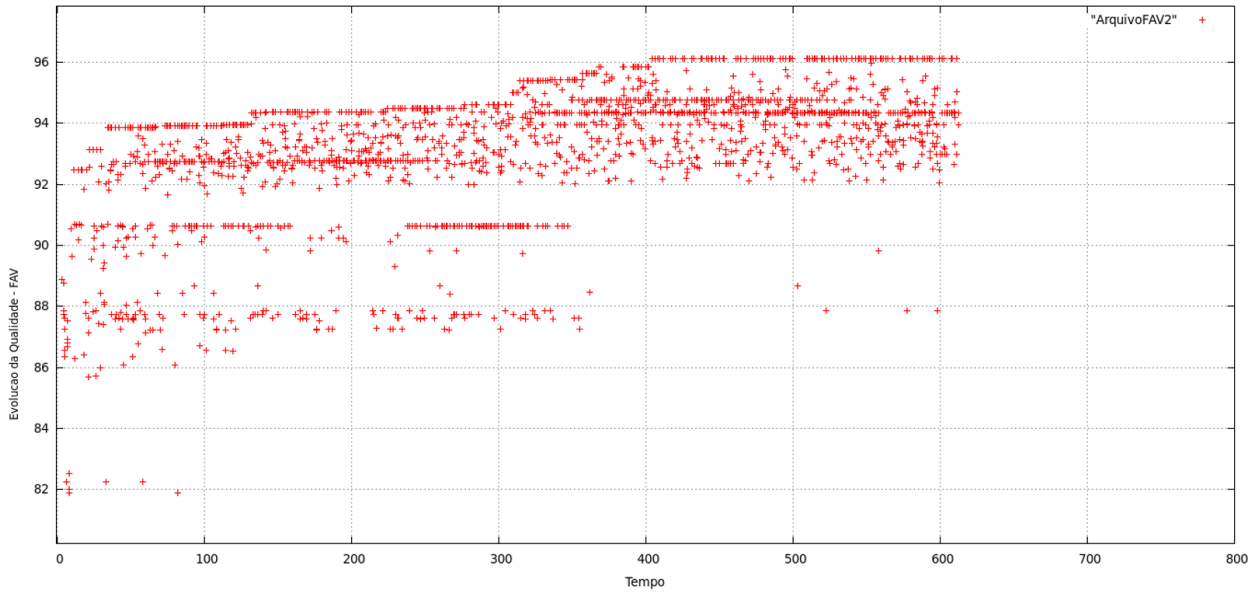
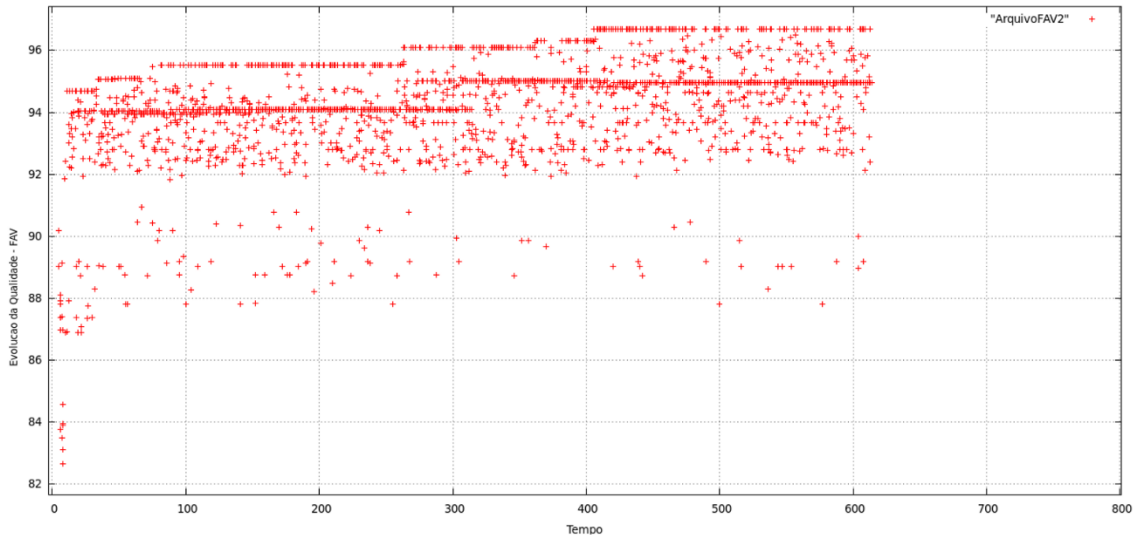


Figura 25 – Variação da qualidade de soluções para Instância 1 - 100 itens.



Analisando as Figuras 23, 24 e 25 é possível verificar que gradativamente as soluções geradas pelos agentes inicializadores nos primeiros 20 segundos são melhoradas ao longo da execução do time, quando os agentes de melhoria entram em ação. É possível observar que em determinados momentos soluções com qualidade inferior também são geradas pelos agentes de melhoria, mas que só são incluídas na memória se atender os critérios que se diferenciam das demais soluções já contidas na memória e possuir uma função de qualidade melhor do que a pior solução.

Figura 26 – Variação da qualidade de soluções para Instância 2 - 60 itens.

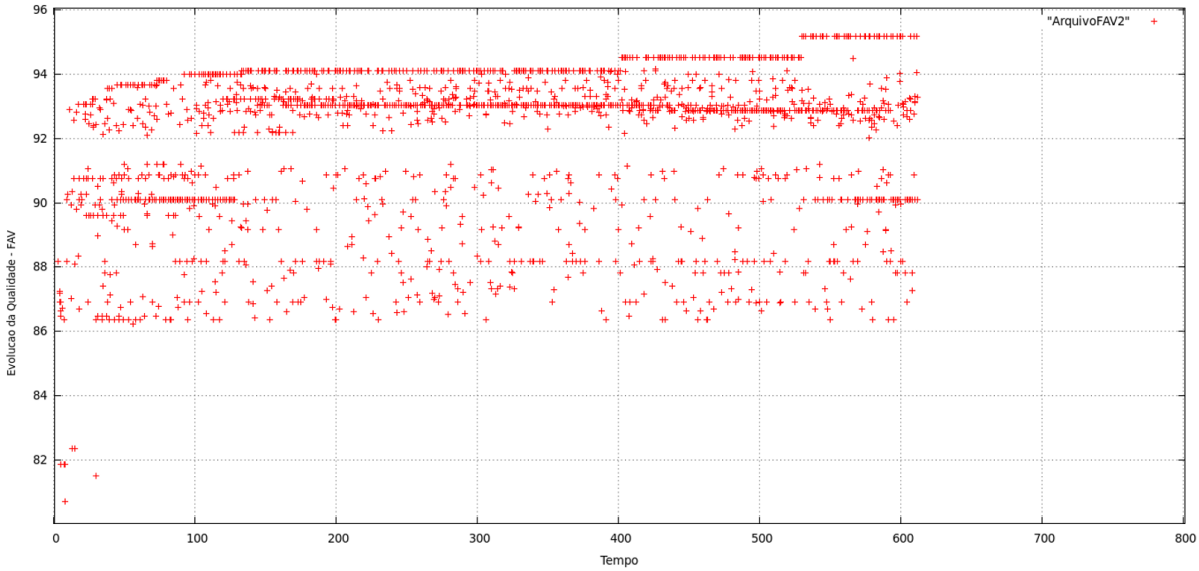


Figura 27 – Variação da qualidade de soluções para Instância 2 - 80 itens

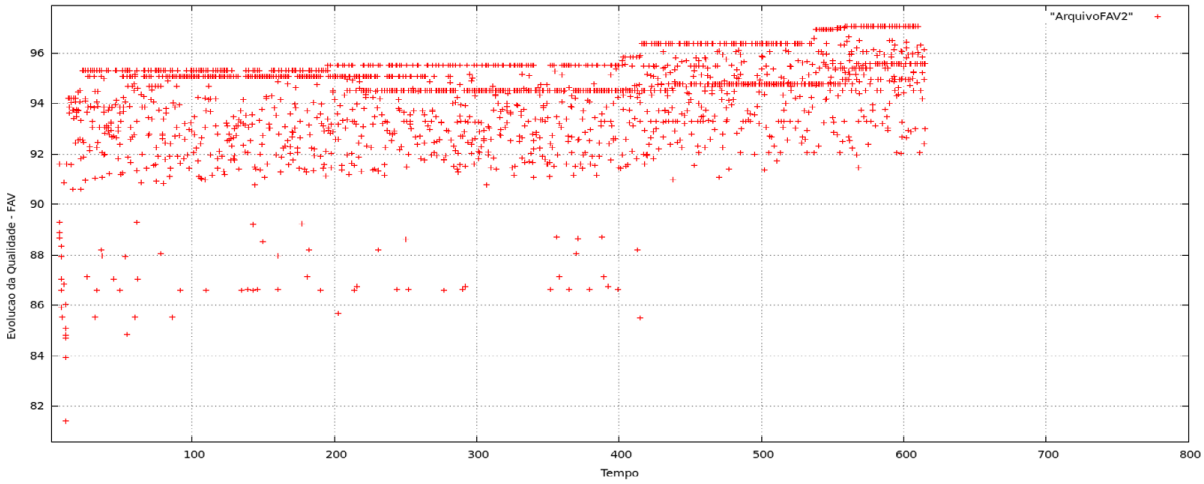
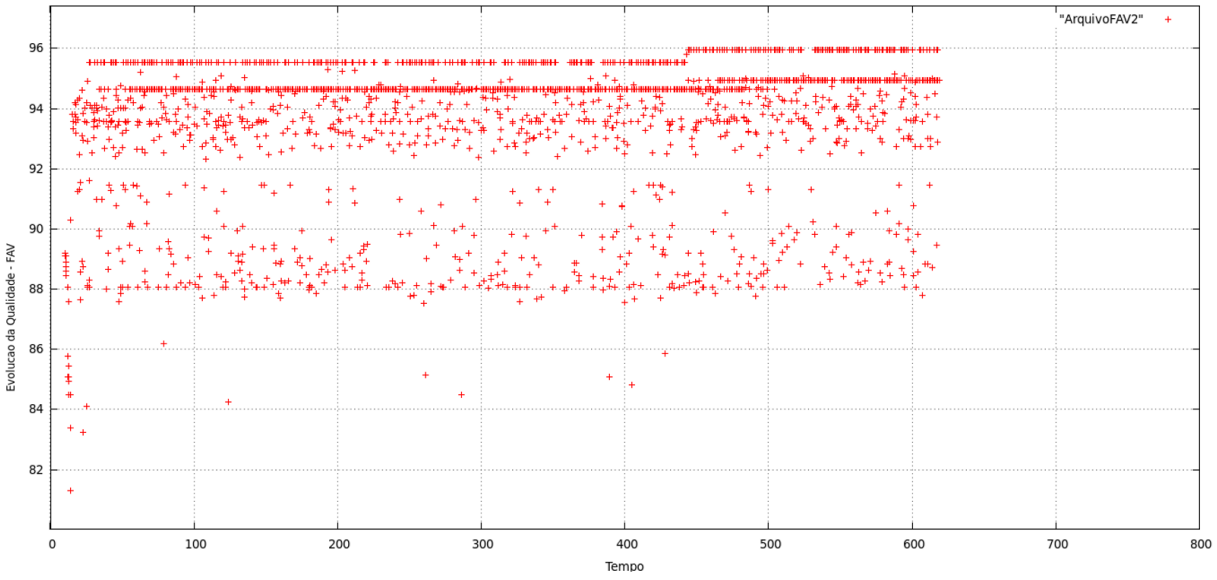


Figura 28 – Variação da qualidade de soluções para Instância 2 - 100 itens.



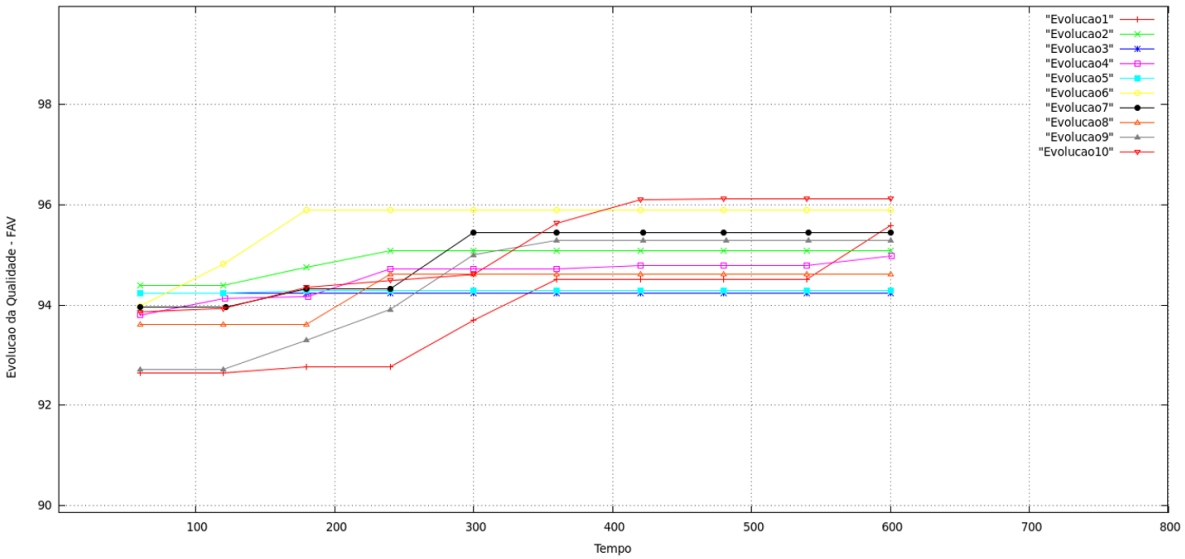
Como é possível visualizar na sequência das Figuras 26, 27 e 28, o time consegue encontrar soluções de boa qualidade depois de 400 segundos de execução e outras acima de 500 segundos.

Percebe-se também que devido a característica gulosa-aleatória da metaheurística GRASP, os agentes que a implementam geram soluções bastante diversificadas ao longo da execução do time, varrendo um espaço de soluções bem diferenciado.

Ainda com o propósito de melhor observar o comportamento e a evolução de algumas das melhores soluções contidas na memória foram selecionadas as mesmas instâncias 1 e 2 e para a quantidade de 80 itens. As Figuras 29 e 30 mostram as evoluções

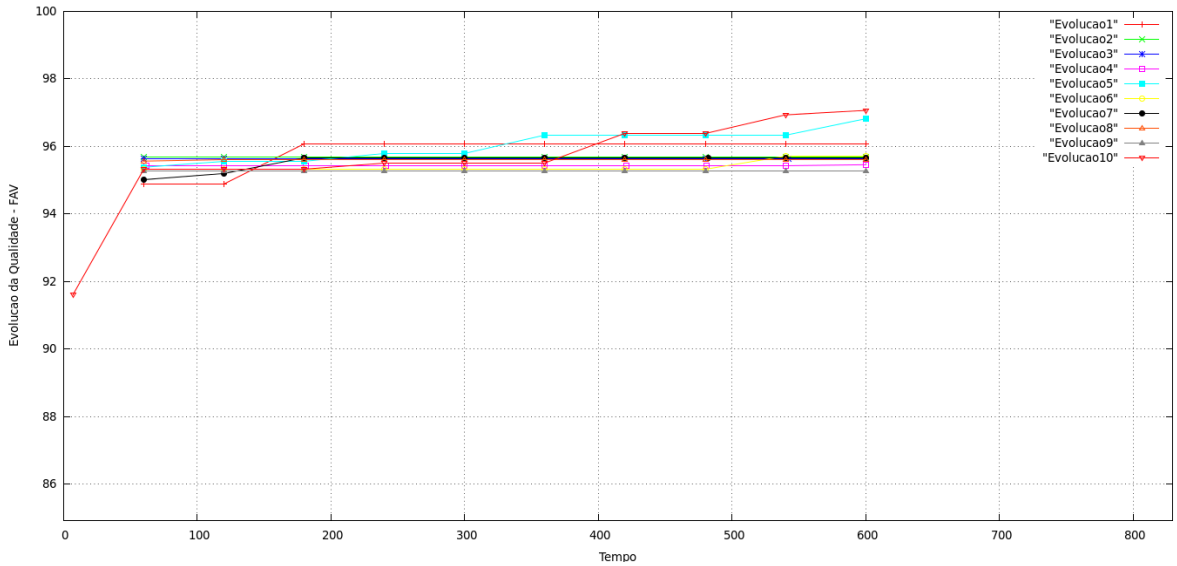
da qualidade das melhores soluções, a cada minuto de execução do time, para cada uma das dez execuções realizadas para tal instância e quantidade de itens.

Figura 29 – Evolução da qualidade das melhores soluções para Instância 1 - 80 itens.



Como exposto no gráfico de evolução da Figura 29 em cada execução realizada a qualidade das soluções se diferenciam bastante a cada minuto, entretanto para a instância 2 a evolução e diferença entre as soluções é bem menor.

Figura 30 – Evolução da qualidade das melhores soluções para Instância 2 - 80 itens.



5.6.2 Agentes que compõe o Time assíncrono e Aperfeiçoamento do time

Analisando as figuras é possível perceber que o time no decorrer do tempo, consegue convergir para soluções de qualidade superior àquelas geradas pelos agentes inicializadores no início da execução, mas que, em determinados momentos, gera soluções de qualidade bem inferior ou fica preso em ótimos locais.

6. CONCLUSÕES E TRABALHOS FUTUROS

Esta dissertação demonstrou que A-Teams apresentam-se como uma ferramenta de grande valor quando aplicados na solução de problemas de otimização combinatória como o Problema de Corte Bidimensional Guilhotinado. Também foi demonstrado que A-Teams permitem, com sucesso, a combinação de algoritmos de características construtivas e melhoria trabalhando, interagindo assíncrona e iterativamente para aplicações em problemas considerados complexos.

Um ponto de grande relevância é o que diz respeito aos resultados iniciais obtidos e apresentados no capítulo 5. Analisando estes resultados verifica-se que mesmo com uma implementação reduzida de agentes, entre as muitas possibilidades de combinação e configuração de agentes dentro de um time assíncrono os resultados produzidos, na maioria das vezes se mostrou melhores que os resultados até então conhecidos. Um grande desafio no que concerne aos times assíncronos diz respeito a quantidade de diferentes configurações de agentes e parâmetros que pode ser realizado. Além das diferentes estruturas de dados que os algoritmos utilizam internamente.

Com variações de dois algoritmos construtivos gulosas-aleatórias e estratégias de melhoria (busca local), e a combinação destes, os resultados obtidos para a classe 10 testada se mostrou interessante em relação ao número de placas encontradas e no aproveitamento geral das placas utilizadas. Sendo assim, testes para as demais classes poderão ser realizados, bem como testes com variação no tamanho das memórias e tempo de execução diferenciados para o time em função do tamanho das instâncias.

A estratégia de trabalhar com soluções parciais se mostrou bastante eficaz, uma vez que os bons arranjos são mantidos e novas combinações de características melhores podem ser encontradas no espaço de soluções. A metaheurística GRASP apresentou bom comportamento ao encontrar soluções prévias de boa qualidade, incluídas no processo de melhoria. Tal metaheurística se mostrou ideal, uma vez que podemos equilibrar agentes mais gulosos e aleatórios em espaço de busca reduzidos, apresentando grande diversidade nas soluções obtidas.

Com esta análise espera-se que com a implementação de agentes mais poderosos (com adaptações mais sofisticadas) e inclusão de algoritmos mais complexos como Path Relink, teríamos uma grande melhora no desempenho de qualidade das soluções obtidas pelo time.

Como trabalhos futuros pode se investigar a possibilidade de trabalhar com o problema de corte bidimensional guilhotinado bi-objetivo levando em consideração o número total de linhas de cortes a ser realizado. Para isso, pode-se estudar e investigar a possibilidade de representação de uma função objetivo mais robusta capaz de contemplar todas as características odo problema. Para este fim, está sendo investigada uma função que atribua pesos diferenciados aos atributos de uma solução e testes adicionais serão realizados futuramente. A ideia de atribuição de pesos pode ser interessante uma vez que os pesos atribuídos configuram o grau de prioridade no problema, podendo ser priorizado por exemplo minimizar o número de cortes a ser realizado, ou maximizar a função de aproveitamento.

Outro aspecto relevante complementar a função objetivo seria investigar uma modelagem de estrutura de dados mais genérica de forma a contemplar todas as características do problema e combinar diferentes algoritmos com estratégias internas diferentes, uma vez que esta apresentou-se com um dos grandes desafios para trabalhar com times assíncronos aplicados ao problema de corte em estudo.

REFERÊNCIAS BIBLIOGRÁFICAS

ALVES, J. D. S.; LONGO, H. J. Times Assíncronos. In: LOPES, H. S.; RODRIGUES, L. C. D. A.; STEINER, M. T. A. **Meta-Heurísticas em Pesquisa Operacional**. Omnipax, 2013. Cap. 9, p. 129-143.

ARYANEZHAD, M.-B. *et al.* A simple approach to the two-dimensional guillotine cutting stock problem. **Journal of Industrial Engineering International**, v. 8, n. 1, 2012. p. 1-10.

AYADI, O. ; Cheikhrouhou, N. ; MELLOULI, A.; MASMOUDI, F. , A hybrid heuristic to solve the two dimensional cutting stock problem with consideration of forecasts. **In: Computers & Industrial Engineering, 2009. CIE 2009. International Conference on**, IEEE, 2009. p. 221-226.

BERKEY, J. O.; WANG, P. Y. Two dimensional finite bin packing algorithms. Journal of the Operational Research Society. 1987. p. 423-429

BERNARDI, R. D. Aplicando a técnica de times assíncronos na otimização de problemas de empacotamento unidimensional, São Paulo, 2001. Dissertação de Mestrado.

BLUM, C.; ROLI, A. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. **ACM Computing Surveys**, v. 35, n.3, 2003. p. 268–308.

BLUM, C.; SCHMID, V.; BAUMGARTNER, L. Solving the Two-Dimensional Bin Packing Problem with a probabilistic multi-start heuristic. **Proceedings of the 5th international conference on Learning and Intelligent Optimization.** Rome, Italy 2011. p. 76-90.

BOSCHETTI, M. A.; MINGOZZI, A. The two-dimensional finite bin packing problem. Part II: New lower and upper bounds. **Quarterly Journal of the Belgian, French and Italian Operations Research Societies**, v. 1, n. 2, 2003. p. 135-147.

CARDOSO, M. C. **Desenvolvimento de uma interface gráfica para otimização interativa em um programa de corte de vidro**. Monografia de conclusão de curso em Ciências da Compautação. Universidade Federal de Goiás. Goiânia. 2004.

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

COFFMAN JR, E. G.; SHOR, P. W. Average-case analysis of cutting and packing in two dimensions. **European Journal of Operational Research**, v. 44, n. 2, 1990. p. 134-144.
COFFMAN, E. G. *et al.* Performance bounds for level-oriented two-dimensional packing algorithms. **SIAM Journal on Computing**, v. 9, n. 4, 1980. p. 808-826.

CORMEN, T. H. *et al.* **Introduction to Algorithms**. 3ª. ed. Cambridge, USA: MIT Press, 2009.

DA BIANCO, Cliceres Mack; DA SILVA, Alexandre Dias. Problema de Corte Bidimensional Guilhotinado: Uuma abordagem através de ferramentas CAD e heurísticas de posicionamento. XXX Encontro Nacional de Engenharia de Produção - ENEGEP. São Carlos - SP, 2010.

DORIGO, M.; CARO, G. D. Ant colony optimization: A new meta-heuristic. **Proceedings of the Congress on Evolutionary Computation**, Piscataway, USA: IEEE Press, 1999. p. 1470-1477.

DORIGO, M.; STÜTZLE, T. Ant Colony Optimization, Cambridge, USA: MIT Press, 2004.

DYCKHOFF, H. A typology of cutting and packing problems. **European Journal of Operational Research**, v. 44, n. 3, 1990. p. 145-159.

FAROE, O.; PISINGER, D.; ZACHARIASEN, M. Guided local search for the three-dimensional bin-packing problem. **INFORMS Journal on Computing**, v.15, n. 3, 2003. p. 267-283.

FEO, T. A.; RESENDE, M. G. C. Greedy randomized adaptive search procedures. **Journal of Global Optimization**, 1995. p. 109-133.

FLESZAR, K. Three insertion heuristics and a justification improvement heuristic for two-dimensional bin packing with guillotine cuts. **Computers & Operations Research**, v. 40, n. 1, 2013. p. 463-474.

FRENK, J. B. G.; GALAMBOS, G. Hybrid next-fit algorithm for the two-dimensional rectangle bin-packing problem. **Computing**, v.39, n. 3, 1987. p. 201-217.

GLOVER, F. Future paths for integer programming and links to artificial intelligence. **Computers an Operations Research**, 1986. p. 533-549.

GLOVER, F.; LAGUNA, M. Tabu Search., Norwich, USA: Kluwer Academic Publishers, 1997.

GOMES, J. A. R. Problema de corte bidimensional: Aplicação a um caso real, Dissertação de Mestrado. Novembro, 2011.

HOLLAND, J. H. Adaptation in Natural and Artificial Systems, East Lansing, USA: University of Michigan Press, 1975.

HOPPER, E.; TURTON, B. A genetic algorithm for a 2D industrial packing problem. **Computers & Industrial Engineering**, v. 37, n. 1, 1999. p. 375-378.

HOPPER, E.; TURTON, B. C. H. An Empirical Investigation of Meta-heuristic and Heuristic Algorithms for a 2D Packing Problem. **European Journal of Operational Research**, 128, n. 1, 2000. p. 34-57.

JYLÄNKI, J. A thousand ways to pack the bin - A practical approach o two-dimensional rectangle bin packing. **Retirado de http://clb. demon. fi/files/RectangleBinPack. pdf.** Fevereiro, 2010. Acessado em

KOZA, J. R. **Genetic Programming II Videotape:** The Next Generation. Cambridge, MA: MIT Press, v. 55, 1994.

LEE, L. S. A genetic algorithm for two-dimensional bin packing problem. **Math Digest: Research Bulletin Institute for Mathematical Research**, v. 2, n. 1, 2008. p. 34-39.

LODI, A. Algorithms for two-dimensional Bin Packing and assignment problems. Tese de Doutorado, **Università Degli Studi di Bologna**, 1999.

LODI, A.; MARTELLO, S.; MONACI, M. Two-dimensional packing problems: A survey. **European Journal of Operational Research**, v. 141, n. 2, 2002. p. 241-252.

LODI, A.; MARTELLO, S.; VIGO, D. Approximation algorithms for the oriented two-dimensional bin packing problem. **European Journal of Operational Research**, v. 112, n. 1, 1999. p. 158-166.

LODI, A.; MARTELLO, S.; VIGO, D. Heuristics and metaheuristic approaches for a class of two-dimensional bin packing problems. **INFORMS Journal on Computing**, v. 11, n. 4, 1999. p. 345-357.

LODI, A.; MARTELLO, S.; VIGO, D. Recent advances on two-dimensional bin packing problems. **Discrete Applied Mathematics**, v. 123, n. 1, 2002. p. 379-396.

LODI, A.; MARTELLO, S.; VIGO, D. Models and bounds for two-dimensional level packing problems. **Journal of Combinatorial Optimization**, v. 8, n. 3, 2004. p. 363-379.

LOH, K.; GOLDEN, B.; WASIL, E. A Weight Annealing Algorithm for Solving Two-dimensional Bin Packing Problems. **In: Operations Research and Cyber-Infrastructure**, Springer US, 2009. p. 121-146.

LONGO, H. J. Aplicação de A-Teams ao Problema de Recobrimento de um Conjunto, Dissertação de mestrado. São Paulo, 1995.

LOURENÇO, H. R.; MARTIN, O. C.; STÜTZLE, T. Iterated local search. In: GLOVER,

F.; KOCHENBERGER, G. A. **Handbook of Metaheuristics**. Norwell, USA: Kluwer Academic Publishers, 2003. p. 321-353.

MARINESCU, D.; BAICOIANU, A. . **The determination of the guillotine restrictions for a rectangular covering model**. WSEAS International Conference. Proceedings. Recent Advances in Computer Engineering. 2009.

MOREIRA, R. A. M. **Otimização interativa de planos de corte - estrutura de dados e algoritmos**. Monografia de conclusão de curso em Ciências da Compautação. Universidade Federal de Goiás. Goiânia. 2004.

MSABAH, S. A.; BABA-ALI, A. R. A new guillotine placement heuristic combined with an improved genetic algorithm for the orthogonal cutting-stock problem. **In: Industrial Engineering and Engineering Management (IEEM), 2011 IEEE International Conference on**, 2011. p. 482-486.

MULATI, M. H.; CONSTANTINO, A. A.; SILVA, A. F. D. Otimização por Colônia de Formigas. In: LOPES, H. S.; RODRIGUES, L. C. D. A.; STEINER, M. T. A. **Meta-Heuristicas em Pesquisa Operacional**. Ominipax, 2013. Cap. 4, p. 53-68.

NASCIMENTO, H. A. D. Uma Abordagem para Desenho de Grafos Baseada na Utilização de Times Assíncronos, Dissertação de Mestrado. Maio, 1997.

NASCIMENTO, H. A. D. ; ALOISE, D. J.; LONGO, H. **Uma heurística O(mn) para o corte bidimensional guilhotinado**. XXXI SBPO-Simpósio Brasileiro de Pesquisa Operacional. 1999. p. 1197-1206.

OSMAN, I. H.; LAPORTE, G. Metaheuristics: A bibliography. **Operational Research**, v. 63, 1996. p. 513–623.

PARKER, R. G. **Discrete optimization**. Universidade de Michigan: Academic Press, 1988.

PARKER, R. G.; RARDIN, R. L. Discrete Optimization. **Computer Science and**

Scientific Computing Academic Press, 1988.

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

PEIXOTO, H. P. Metodologia de especificação de times assíncronos para problemas de otimização combinatória, Dissertação de mestrado. Campinas: IMECC - UNICAMP - SP, 1995.

PEIXOTO, H. P.; DE SOUZA, P. S. Times Assíncronos: Uma nova técnica para o flow shop problem., Seminário Integrado de Software e Hardware. UFMG - Belo Horizonte, 1994. p.597-610

POLYAKOVSKY, S.; HALLAH, R. M. An agent-based approach to the two-dimensional guillotine bin packing problem. **European Journal of Operational Research**, v. 192, 2009. p. 767-781.

RESENDE, M. G. D. C.; SILVA, R. M. D. A. GRASP: Procedimentos de Busca Gulosos, Aleatórios e Adaptativos. In: LOPES, H. S.; RODRIGUES, L. C. D. A.; STEINER, M. T. A. **Meta-Heurísticas em Pesquisa Operacional**. Omnipax, 2013. p. 1-20.

RODE, M.; ROSENBERG, O. An analysis of heuristic trim-loss algorithms. **Engineering Costs and Production Economics**, 1987. p. 71-78.

RODRIGUES, R. D. F. Times assíncronos para resolução de problemas de otimização combinatória com múltiplas funções objetivo. Dissertação de mestrado, Campinas: IC - UNICAMP - SP, 1996.

SANCHEZ, F. R. L. Times Assíncronos Inicializadores para o Planejamento da Expansão da Transmissão de Energia Elétrica Baseados no Modelo Híbrido Linear. Tese de Doutorado. UNIVERSIDADE PAULISTA. 2008.

SILVA, H. J. Heurísticas orientadas à camada para problemas de empacotamento rectangular. Dissertação de mestrado. Escola de Gestão do Porto - Universidade do Porto,

2003.

SOUZA, M. J. F. Inteligência Computacional para Otimização. Notas de aula. Departamento de Computação, Instituto de Ciências Exatas e Biológicas, Universidade Federal de Ouro Preto. Ouro Preto, MG, 2007.

SOUZA, P. S. **Asynchronous Organizations for Multi-Algorithm Problems.** PhD Thesis. Department of Electrical and Computer Engineering, Carnegie Mellon University. Pittsburgh, USA. 1993.

SUBRAMANIAN, A.; PENNA P. H. V.; OCHI, L. S.; SOUZA, M. J. F. . Um Algoritmo Heurístico Baseado em Iterated Local Search para os Problemas de Roteamento de Veículos. In: LOPES, H. S.; RODRIGUES, L. C. D. A.; STEINER, M. T. A. **Meta-Heurísticas em Pesquisa Operacional**. Omnipax, 2013. p. 165-180.

TALUKDAR, S. N.; DE SOUZA, P. S. Scale efficient organizations. **IEEE International Conference on Systems, Man and Cybernetics, 1992, IEEE**, New York, 1992. p. 1458-1463.

TALUKDAR, S. N.; DE SOUZA, P. S. Asynchronous teams. **SIAM SYMPOSIUM ON NUMERICAL SOLUTION OF NONLINEAR PROBLEMS, 1990**, São Francisco: SIAM Publications, 1990.

TEMPONI, E. C. C.; SOUZA S. R. D.; ANDRADE, M. S. F; SANTOS, F. A. . Open Dimensional Cutting Problem: uma abordagem Híbrida via GRASP e ILS. XXVII Encontro Nacional de Engenharia de Produção - ENEGEP. Foz do Iguaçu - PR, 2007.

TEMPONI, E. C. C. Uma Proposta de Resolução do Problema de Corte Bidimensional via Abordagem Metaheurística, Belo Horizonte - MG. Dissertação de mestrado. Dezembro 2007.

VELASCO, A. S. GRASP para o Problema de Corte Bidimensional e Restrito. Dissertação de Mestrado. Campos dos Goytacazes - RJ, Agosto 2005.

VELASCO, A. S.; PAULA JUNIOR, G. G. D.; NETO, E. V. Um algoritmo heurístico baseado na GRASP para o problema de corte bidimensional guilhotinado e restrito. **Gestão da Produção, Operações e Sistemas.** 2008. p. 129.

WÄSCHER, G.; HAUßNER, H.; SCHUMANN, H. An improved typology of cutting and packing problems. **European Journal of Operational Research**, v. 4, 2006. p. 44–454.

WÄSCHER, G.; HAUßNER, H.; SCHUMANN, H. An improved typology of cutting and packing problems. **European Journal of Operational Research**, v. 183, n. 3, 2007. p. 1109-1130.

WONG, L.; LEE, L. S. Heuristic placement routines for two-dimensional bin packing problem. **Journal of Mathematics and Statistics**, v. 5, n. 4, 2009. p. 334.