O Problema do Corte Bidimensional: uma abordagem utilizando o método de geração de colunas

Alan Augusto Teodoro

Trabalho Final de Mestrado Profissional

O Problema do Corte Bidimensional: uma abordagem utilizando o método de geração de colunas

ALAN AUGUSTO TEODORO

TRABALHO FINAL
DE
MESTRADO PROFISSIONAL APRESENTADO
AO
INSTITUTO DE COMPUTAÇÃO DA UNICAMP

PARA OBTENÇÃO DO GRAU
DE
MESTRE EM COMPUTAÇÃO

ÁREA DE CONCENTRAÇÃO: ENGENHARIA DA COMPUTAÇÃO

ORIENTADOR: FLÁVIO KEIDI MIYAZAWA

Julho de 2003

Instituto de Computação

Universidade Estadual de Campinas

O Problema do Corte Bidimensional : uma abordagem utilizando o método de geração de colunas

Banca Examinadora:

- Prof. Dr. Flávio Keidi Miyazawa (Orientador)
  (Instituto de Computação – UNICAMP)
- Prof. Dr. Horacio Hideki Yanasse
  (Laboratório Assoc. de Comp. e Matemática Aplicada – INPE)
- Prof. Dr. Arnaldo Vieira Moura
  (Instituto de Computação – UNICAMP)
- Prof. Dr. Ricardo Dahab (Suplente)
  (Instituto de Computação – UNICAMP)

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP

Teodoro, Alan Augusto

T264p O problema do corte bidimensional: uma abordagem utilizando o método de geração de colunas / Alan Augusto Teodoro -- Campinas, [S.P. :s.n.], 2003.

Orientador : Flávio Keidi Miyazawa

Trabalho final (mestrado profissional) - Universidade Estadual de Campinas, Instituto de Computação.

1. Otimização combinatória. 2. Programação inteira. 3. Pesquisa operacional. I. Miyazawa, Flávio Keidi. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

O Problema do Corte Bidimensional: uma abordagem utilizando o método de geração de colunas

Este exemplar corresponde à redação final do Trabalho de Conclusão do curso de Mestrado Profissional em Computação devidamente corrigida e defendida por Alan Augusto Teodoro e aprovada pela Banca Examinadora.

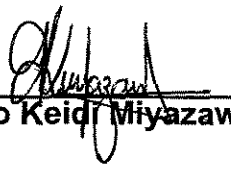
Campinas, 27 de agosto de 2003.

Prof. Dr. Flávio Keidi Miyazawa (Orientador).

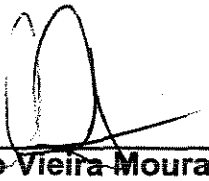
Trabalho Final apresentado ao Instituto de Computação, Unicamp, para obtenção do título de Mestre em Computação na área de Engenharia de Computação.

TERMO DE APROVAÇÃO

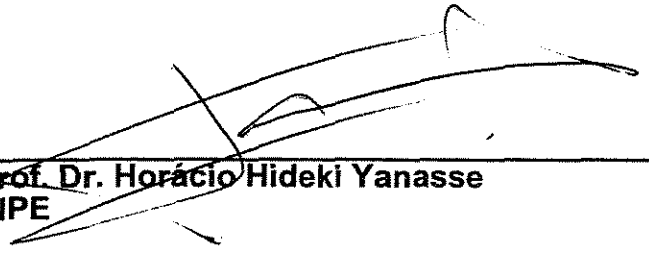
Tese defendida e aprovada em 27 de agosto de 2003, pela Banca Examinadora composta pelos Professores Doutores:



Prof. Dr. Flavio Keidi Miyazawa
IC - UNICAMP



Prof. Dr. Arnaldo Vieira Moura
IC - UNICAMP



Prof. Dr. Horácio Hideki Yanasse
INPE

Ao Oscar e à Neuza, meus queridos pais.

Agradecimentos

Muitas pessoas participaram direta ou indiretamente para o desenvolvimento deste trabalho. Aproveito este espaço para agradecê-las.

Agradeço ao Flávio, pela excelente orientação, paciência e amizade.

Agradeço aos professores e funcionários do IC.

Agradeço aos professores Horácio e Arnaldo por aceitarem o convite para formar a banca examinadora e pelas sugestões que fizeram para melhorar este trabalho.

Agradeço a todos os colegas de mestrado que fizeram do curso um ambiente muito agradável.

Agradeço aos meus professores de graduação, Hermes Senger e Carlos Eduardo pelas cartas de recomendação.

Agradeço ao Fernando Oliveira pela amizade e por entender a importância deste trabalho e não cobrar dedicação exclusiva à nossa empresa.

Agradeço aos meus pais e à minha irmã, a quem devo tudo o que tenho e ao que sou e por me motivarem para alcançar todos os meus objetivos.

Por fim agradeço a Deus por ter colocado todas essas pessoas em minha volta.

Muito Obrigado!

O Problema do Corte Bidimensional: uma abordagem utilizando o método de geração de colunas

ALAN AUGUSTO TEODORO

Resumo do trabalho final apresentado ao IC-UNICAMP como parte dos requisitos necessários para a obtenção do título de Mestre em Computação.

Resumo

Neste trabalho é realizado um estudo experimental de técnicas de otimização para gerar soluções eficientes para o problema do corte bidimensional, que pode ser definido como: dado um número finito n de itens retangulares de largura l_i, comprimento c_i e demanda d_i, a serem obtidos de retângulos maiores de dimensão LxC, encontrar padrões de corte que atendam a uma demanda de itens utilizando o menor número possível de retângulos maiores.

O problema foi formulado através de um modelo de programação linear inteira. Para obter soluções de custo reduzido para o problema, aplicamos o método de geração de colunas, obtendo então soluções viáveis para o problema relaxado do programa linear inteiro. Utilizamos um algoritmo de aproximação para obter uma solução inicial de qualidade e métodos de arredondamento com tratamento de problema residual para transformar a solução fracionária em soluções viáveis para o problema.

Finalmente, diversos estudos são realizados através de testes computacionais.

Palavras-chave: Otimização Combinatória, Programação Linear Inteira, Método de Geração de Colunas.

Orientador: FLÁVIO KEIDI MIYAZAWA

Two-dimensional Cutting Stock Problem: a column genneration method approach

ALAN AUGUSTO TEODORO

Abstract of the work presented to IC-UNICAMP as partial fulfillment for the requirementes for the degree of Master of Computer Science.

Abstract

In this work we describe an experimental study of optimization techniques to generate efficient results for the two-dimensional cutting stock problem which can be defined as follows: given a finite number n of rectangular items of width l_i, length c_i and demand d_i, to be cut from larger rectangles with dimensions LxC, find cutting patterns which attend the demand of the requested items minimizing the number of larger rectangles.

The problem is formulated as an integer programming model. To obtain solutions with reduced cost to the problem, we apply the column generation method, obtaining feasible solutions for the relaxed integer program. We use an approximation algorithm to generate a good initial solution and rounding techniques with treatment of the residual problem to transform the fractional solution into feasible solutions to the problem.

Finally, several studies are realized through computational experiments.

Keywords: Combinatorial Optimization, Integer Programming, Column Generation Method.

Supervisor: FLÁVIO KEIDI MIYAZAWA

ÍNDICE

Lista de Figuras

Lista de Tabelas

CAPÍTULO 1

Introdução

Atualmente o mundo vem passando por uma situação econômica delicada e as empresas vêem-se obrigadas a reduzir os custos para se manterem competitivas. Para isso, algumas empresas tomam diversas medidas como a redução no quadro de funcionários e o corte de benefícios e investimentos. Neste cenário, as que procuram rever seus processos para fabricar seus produtos de maneira otimizada (com o menor custo possível), podem levar vantagem perante as demais.

Entretanto, existem muitos casos nos quais realizar uma tarefa de maneira ótima pode ser impraticável, pois a busca por uma solução ótima é quase sempre uma enumeração para percorrer um conjunto grande de soluções possíveis. Pertencentes a uma classe conhecida como *Otimização Combinatória,* alguns exemplos desta categoria de problemas são: o corte e empacotamento de materiais, alocação de recursos, localização de facilidades, determinação de centros de distribuição, escalonamento de tarefas, seqüenciamento de genes de DNA, entre outros.

O problema considerado neste trabalho é o de corte de placas, conhecido na literatura como o *problema de corte bidimensional restrito (two-dimensional constrained cutting stock problem),* que é comum para empresas, por exemplo, de móveis, que compram peças retangulares de madeireiras e precisam cortá-las em itens menores para compor suas matérias-prima, como cadeiras, mesas e armários.

Como será visto posteriormente, para os problemas estudados neste trabalho não são esperados algoritmos que os resolvam de forma exata e rápida para grandes instâncias e em função da importância destes problemas, diversas técnicas são utilizadas para se obter soluções de compromisso.

1.1 Motivação

Problemas de natureza combinatória são largamente estudados pela comunidade acadêmica devido ao fato de ainda não serem conhecidos algoritmos eficientes para resolvê-los de forma exata e em tempo adequado para qualquer instância. Devido a esta dificuldade, diversas técnicas como *heurísticas* e *algoritmos de aproximação* são desenvolvidas para buscar soluções próximas da ótima e em tempo razoável.

Além da importância acadêmica, estes problemas aparecem constantemente na vida real, principalmente em processos industriais, em que as empresas buscam continuamente aumentar sua produtividade e eliminar custos. Dependendo da escala de produção, pequenas melhorias nos processos das empresas podem eliminar custos desnecessários e gerar grandes economias.

Em alguns setores, fabricar produtos de forma otimizada pode ser uma questão de sobrevivência, pois existe a concorrência de empresas estrangeiras que investem muito em pesquisas.

Neste trabalho é realizado um estudo experimental de técnicas de otimização para encontrar soluções eficientes para o *problema do corte bidimensional*, porém as técnicas estudadas podem ser utilizadas em diversos outros problemas de otimização combinatória. Estes problemas compartilham entre si características comuns, o que torna a leitura de artigos de diversas áreas uma boa prática, pois uma técnica para resolver com eficiência uma determinada necessidade pode ser aplicada em outros problemas, muitas vezes adaptada com poucas alterações.

1.2 Resumo do Trabalho

A abordagem estudada neste trabalho foi proposta por Gilmore e Gomory [4, 5, 6] e consiste em gerar soluções eficientes para o *problema de corte* utilizando *programação linear inteira* juntamente com o *método de geração de colunas*. O objetivo é encontrar *padrões de corte* que atendam uma demanda solicitada e minimizem a quantidade de material utilizado.

Para diminuir o tempo de processamento, é utilizada a estratégia de primeiramente executar um *algoritmo de aproximação* que fornece, com grande eficiência, uma solução de boa qualidade. O resultado será um conjunto inicial de *padrões de corte* que será usado para formular um *problema de programação linear inteira.*

Devido à complexidade de resolver um *problema de programação linear inteira* quando o número de variáveis é muito grande, uma *relaxação linear* é realizada, tornando-o um *problema de programação linear.* Após resolver o *problema de programação linear*, os *valores duais* das variáveis são utilizados para determinar o valor de importância dos itens para a geração de um novo *padrão de corte.*

O processo continua com a geração de *padrões de corte* melhores através de um algoritmo recursivo proposto por Herz [1]. Este algoritmo se destaca pela facilidade de implementação e por posicionar os itens na placa de modo que o desperdício de material seja o mínimo possível. Com o novo *padrão de corte* gerado pelo algoritmo de Herz, é verificado se este irá melhorar nossa solução e, em caso positivo, incluído no *problema de programação linear.* Este processo se repetirá até que não existam mais *padrões de corte* que melhorem a solução. O processo termina com a aplicação de técnicas para a obtenção de soluções inteiras.

O trabalho é concluído através da realização de testes computacionais.

1.3 Organização do Texto

No capítulo 2, os problemas e os principais conceitos encontrados neste trabalho são definidos. O objetivo deste capítulo é familiarizar o leitor com os *problemas de corte e empacotamento* e mostrar quais problemas serão tratados especificamente. Serão abordadas também, de forma resumida, técnicas de otimização muito utilizadas na indústria e aplicadas em diversos outros problemas.

No capítulo 3 o *problema do corte bidimensional,* juntamente com suas restrições e complexidade, é detalhado. Será visto que a indústria pode nos impor algumas limitações no processo de corte e os algoritmos devem ser adaptados para atendê-las.

No capítulo 4 é apresentada a abordagem estudada. É mostrado como modelar o problema no formato de *programação linear inteira*. A *heurística* que gera o conjunto de soluções inicial é detalhada e o leitor verá como é feita a interação do sistema com o algoritmo que gera os *padrões de corte*. O capítulo é concluído com a apresentação das técnicas que foram utilizadas para a obtenção de soluções inteiras.

No capítulo 5 testes computacionais são realizados através do sistema que foi implementado para analisar a qualidade da solução e sentir a dificuldade de resolver um problema de otimização combinatória. Através de instâncias geradas aleatoriamente é verificado como o sistema se comporta com certas modificações que foram impostas, como por exemplo à redução dos *pontos de discretização*.

Finalmente, no capítulo 6 é feita uma conclusão sobre o trabalho realizado e outros que poderão ser estudados no futuro são mencionados.

CAPÍTULO 2

Preliminares

2.1 Introdução

Neste capítulo são apresentados os principais problemas, técnicas e conceitos encontrados neste trabalho. O objetivo é familiarizar o leitor com futuros termos que serão utilizados na abordagem estudada.

2.2 Complexidade Computacional

Mesmo com a atual capacidade dos computadores[1], existem alguns problemas para os quais não são esperados algoritmos que os resolvam de forma exata e rápida para grandes instâncias. Essa afirmação tem por base o estudo da *complexidade computacional dos problemas.*

Provar que não é possível desenvolver um algoritmo eficiente para resolver determinado problema pode ser tão difícil quanto encontrar um algoritmo eficiente para resolvê-lo.

A eficiência citada é medida em relação ao tempo de execução de um algoritmo, normalmente no seu pior caso.

Basicamente analisamos a taxa de crescimento do tempo de execução em relação ao espaço requerido para representar a instância de um problema. Uma das importâncias em se descobrir a complexidade computacional de um algoritmo que será utilizado para solucionar

[1] Na data de escrita deste trabalho, os computadores pessoais possuem, em geral, a seguinte configuração: Processador de 1.7Ghz de velocidade, 256 MB de memória RAM e 40GB de disco rígido.

determinado problema é que podemos nos concentrar em técnicas adequadas para sua resolução, tanto para obter soluções exatas como para obter soluções aproximadas.

Os problemas estudados neste trabalho aparecem freqüentemente ou como *problemas de decisão*, onde a resposta do problema é simplesmente "sim" (verdadeiro) ou "não" (falso) à determinada pergunta ou como *problemas de otimização*, onde existem muitas possibilidades de soluções concorrendo entre si e o objetivo é encontrar a melhor delas satisfazendo critérios pré-definidos.

Ao fazer a análise da complexidade computacional de um problema, queremos saber se podemos desenvolver um algoritmo com *complexidade de tempo polinomial* que possa resolvê-lo. Um algoritmo de complexidade de tempo polinomial é aquele cuja função de complexidade de tempo é $O(p(n))$ para alguma função polinomial p, onde n é o tamanho da entrada. Um função $f(n)$ é $O(g(n))$ se existe uma constante c e n_0 tais que $|f(n)| \leq c.|g(n)|$, para todo $n \geq n_0$. Informalmente, um algoritmo de complexidade de tempo polinomial é computacionalmente tratável, requerendo um tempo de execução limitado por uma função polinomial.

Existem outros problemas para os quais a quantidade de operações cresce muito rapidamente em relação ao tamanho do problema. Para muitos problemas apenas algoritmos com complexidade de tempo exponencial são conhecidos, o que torna o problema inviável de ser resolvido em tempo adequado.

Para formalizar esta teoria, os problemas considerados foram restritos a problemas de decisão que apesar da aparente simplicidade, conservam a dificuldade computacional de sua versão não decisória. As principais classes de complexidade são:

Classe P (*Deterministic Polynomial Time*)

Classe dos problemas de decisão que podem ser decididos por algoritmos de complexidade de tempo polinomial.

Classe NP (*NonDeterministic Polynomial Time*)

Classe dos problemas de decisão para os quais, quando positivos, existem certificados que podem ser verificados em tempo polinomial, em geral tal certificado é dado pela própria

solução do problema. Cabe frisar que apenas a verificação do certificado (solução) é feita em tempo polinomial e não a resolução do problema. Esta classe de problemas é bem grande [3].

Classe NP-Completo (*NonDeterministic Polynomial Time Complete*)

É um subconjunto de NP, onde se localizam os problemas considerados mais difíceis desta classe. Um problema é dito ser NP-Completo se todo problema de NP pode ser reduzido a ele em tempo polinomial. Por exemplo, a resolução de um problema NP-Completo em tempo polinomial implica que todos os problemas de NP podem ser resolvidos em tempo polinomial.

Classe NP-Difícil (*NP-Hard*)

São problemas que não estão necessariamente em NP (onde estão apenas os problemas de decisão) mas são pelo menos tão difíceis quanto os problemas em NP. Nesta classe se encontram muitos problemas de otimização e a resolução de um problema NP-Difícil por um algoritmo de complexidade de tempo polinomial nos leva a obtenção de algoritmos de complexidade de tempo polinomial para todos os problemas em NP.

Existem nesta área algumas questões em aberto, como por exemplo determinar se $P = NP$, isto é, se existem algoritmos de complexidade de tempo polinomial para todo problema da classe NP. Caso essa igualdade se mostre verdadeira, todos os problemas dessa classe poderão ser resolvidos em tempo polinomial. A maioria dos pesquisadores acredita que $P \neq NP$.

Os problemas tratados nesta monografia são todos NP-Difíceis. Nossa estratégia é a de estudar algoritmos que apesar de não garantir soluções ótimas, apresentam soluções de custo reduzido em tempo razoável.

Para maiores detalhes sobre complexidade computacional, uma excelente referência é o livro de Garey e Johnson [3].

2.3 Problemas de Corte e Empacotamento

Quando queremos cortar itens de um determinado material (papel, madeira, vidro, alumínio, isopor, etc) de forma a atingir um objetivo (menor desperdício, menor quantidade de material utilizado, etc), estamos lidando com um problema de corte. Ao definir configurações para posicionar itens dentro de recipientes de maneira otimizada, dizemos que estamos tratando de um problema de empacotamento.

Claramente pode-se perceber que existe uma relação entre os problemas de corte e os problemas de empacotamento e que estes devem ser melhor definidos. Ao analisar as restrições impostas por cada problema, entendemos melhor suas semelhanças e diferenças.

Nesta seção, os problemas de corte e empacotamento encontrados neste trabalho são conceituados de maneira informal através de suas principais características e algumas aplicações são apresentadas.

2.3.1 Classificação dos Problemas de Corte e Empacotamento

Os problemas de corte e empacotamento podem ser classificados, em geral, por certas características que serão determinantes para a escolha da técnica ou algoritmo que irão resolvê-los. Algumas das principais características são:

Dimensionalidade

Ao analisar a dimensionalidade de um problema, verificamos quantas dimensões são relevantes ao problema. Para o caso unidimensional, por exemplo, no corte de bobinas de aço, o comprimento é a única dimensão que será tratada. No caso bidimensional, onde temos o corte de chapas, devemos considerar a largura e o comprimento dos objetos. Já no caso tridimensional, consideramos também a altura dos objetos e este problema é comum, por exemplo, para empresas de transporte que procuram fazer o menor número de viagens empacotando as caixas nos caminhões da melhor maneira possível.

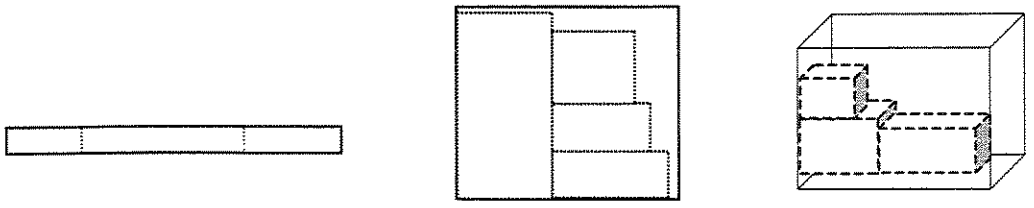


Figura 2.1: Objetos onde respectivamente uma, duas e três dimensões são relevantes.

Temos ainda o caso n-dimensional ou multidimensional, onde podemos ter mais de 3 variáveis relevantes ao processo, neste caso além das dimensões dos objetos, outras características do problema são consideradas, tais como o tempo disponível para realizar os cortes.

Formato dos objetos

Os objetos a serem cortados e/ou empacotados, podem ter sua forma geométrica regular ou irregular. Figuras regulares possuem lados e ângulos iguais entre si. A maioria dos problemas encontrada na literatura trata formas de objetos com formas regulares, principalmente retângulos e quadrados.

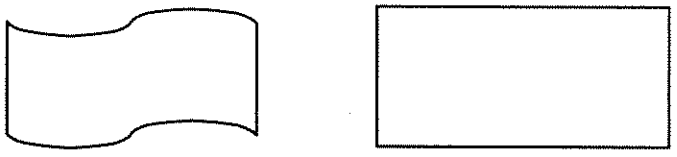


Figura 2.2: Objetos com formato irregular e regular, respectivamente.

Entretanto, pode-se encontrar na indústria diversos problemas que tratam formas irregulares, por exemplo, no problema de corte de tecidos, normalmente existem diversas peças de formas diferentes e deve-se encontrar a melhor maneira de dispô-las de forma a minimizar o desperdício de tecido.

Demanda

Em um problema de corte, a demanda representa a quantidade de itens que devem ser obtidos dos objetos maiores, normalmente a variação dos itens é pequena e a demanda é alta. Nos problemas de empacotamento acontece o contrário.

Diz-se que o problema é restrito quando a demanda imposta deve ser respeitada. Este é o caso que será tratado neste trabalho.

Orientação

O que define se o problema é orientado ou não é se o item pode ou não ser girado, isto é, sofrer rotações quanto a alguns dos eixos.

Ao determinar que os itens podem ser girados, a complexidade do problema aumenta, pois temos que considerar todas as rotações possíveis. É comum encontrar em caixas a indicação de "este lado para cima" (ou "para baixo"). Este tipo de restrição deve ser adicionado ao problema e respeitado pelo algoritmo que irá resolvê-lo.

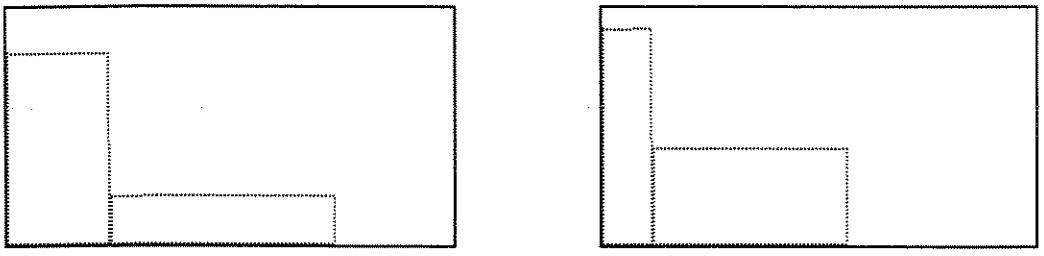


Figura 2.3: Empacotamentos com itens iguais e orientações diferentes.

Neste trabalho será tratado apenas o caso orientado, isto é, rotações dos itens não são permitidas.

Disponibilidade

A disponibilidade considera no problema a quantidade disponível de objetos e a ordem que podem ser utilizados.

Os algoritmos que fazem o empacotamento de uma lista de itens sem precisar da informação dos itens que não foram empacotados são chamados algoritmos *on-line*. Quando o algoritmo conhece todos os itens e pode escolher a ordem em que serão empacotados, ele é denominado um algoritmo *off-line* e claramente é mais eficiente que o *on-line*, pois permite que diversas estratégias sejam utilizadas, por exemplo uma ordenação dos itens antes de fazer o empacotamento.

Objetivo(s)

Podemos encontrar na literatura diversos trabalhos sobre problemas de corte e empacotamento. Um dos pontos principais que os diferem são os objetivos que os autores escolhem para priorizar. Alguns exemplos são:

- Minimizar a quantidade de recipientes utilizados.
- Minimizar o desperdício nos *padrões* (ou maximizar o aproveitamento).
- Determinação da melhor seqüência de corte.
- Determinação de soluções exatas (neste caso a qualidade da solução é o foco).
- Determinação de soluções aproximadas (neste caso o tempo disponível é escasso).

Outros estudos consideram a combinação de dois ou mais objetivos. Este trabalho está concentrado em encontrar *padrões de corte* de modo a minimizar a quantidade de retângulos maiores utilizados respeitando determinadas restrições que serão detalhadas no próximo capítulo.

2.3.2 Problemas de corte em placa(s)

Conforme visto, os problemas de corte aparecem quando temos um material e precisamos cortá-lo da melhor maneira possível para obter itens que provavelmente serão ou irão compor uma determinada matéria-prima.

No caso do corte em placas, dado um número finito n de itens retangulares de largura l_i, comprimento c_i e demanda d_i, a serem obtidos de retângulos maiores com dimensão LxC, o objetivo é encontrar padrões de corte que atendam a demanda de itens solicitados utilizando o menor número possível de retângulos maiores.

Os problemas de corte possuem certas características que aparecem como restrições nos algoritmos criados para resolvê-los. Algumas dessas características são:

Nível de Corte

A indústria pode impor algumas restrições ao procedimento de corte. Uma delas é o nível de corte das máquinas, isto é, quantas vezes a placa será cortada para obter-se os itens desejados.

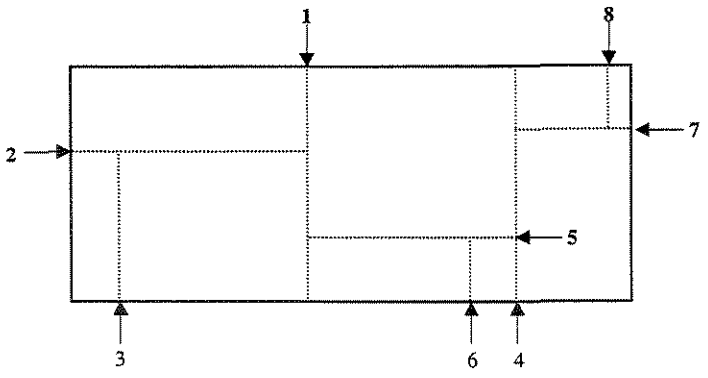


Figura 2.4: Níveis de corte em um padrão.

Os números indicam a quantidade de níveis que o objeto de corte necessitou para cortar o *padrão* indicado pela figura.

Estágio de Corte

Algumas máquinas de corte realizam apenas o corte por estágios, isto é, os retângulos são obtidos fazendo uma seqüência de vários cortes em uma mesma direção, digamos horizontais e depois, para o que sobrou, faz-se vários cortes verticais e assim por diante, alternando as direções. Em cada estágio a direção dos cortes é ortogonal a do estágio anterior.

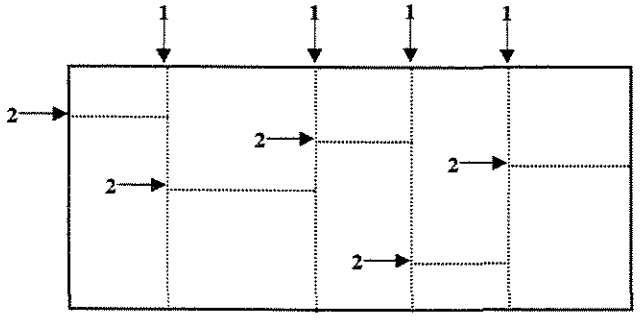


Figura 2.5: Estágios de corte em um padrão.

Uma forma de visualizar o corte por estágios seria através de várias serras cortando em paralelo. Os números indicam a quantidade de estágios que o objeto de corte necessitou para cortar o *padrão* indicado pela figura.

Neste trabalho será feito um estudo comparativo entre o corte por níveis e por estágios, verificando a eficiência das soluções geradas por ambas as formas de cortar.

Corte Guilhotina

O tipo de corte feito por determinado equipamento de corte pode ser do tipo guilhotina, isto é, cada vez que uma placa ou sub-placa é cortada, a lâmina atravessa a placa de ponta a ponta, ou do tipo não guilhotinado, onde a máquina pode mudar de direção durante o corte.

Esta limitação pode ser imposta pelo equipamento utilizado para cortar (existem máquinas de corte que permitem apenas o corte guilhotina) ou na escolha das técnicas de resolução.

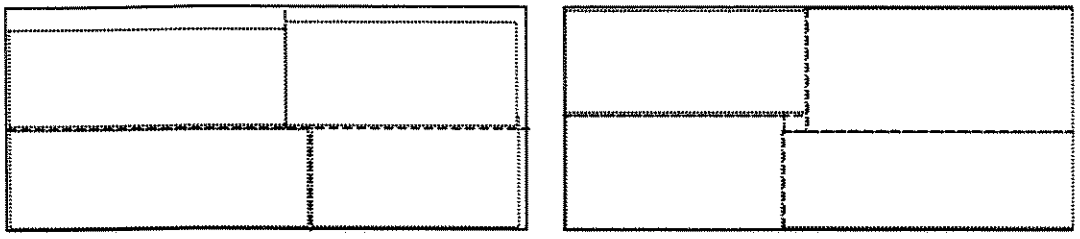


Figura 2.6: Corte guilhotinado e corte não guilhotinado.

Os algoritmos implementados neste trabalho trabalham apenas com cortes de tipo guilhotina.

2.3.3 Problema de empacotamento bidimensional de peso máximo

Problemas de empacotamento bidimensional aparecem quando precisamos determinar o melhor posicionamento dos itens em uma placa ou determinada região da placa. Neste trabalho serão encontradas especificamente duas variações, o empacotamento em placa para obtenção de peso máximo e o empacotamento em faixa.

O problema de empacotamento bidimensional de peso máximo consiste em, dado um número finito n de itens retangulares de dimensões $l_i x c_i$ e valor de importância v_i, encontrar um sub-conjunto destes itens empacotados em um retângulo maior de dimensão LxC de modo que o valor total dos itens empacotados seja o maior possível.

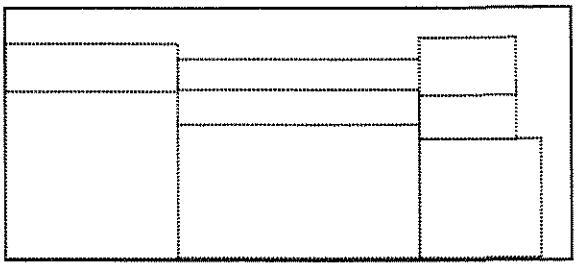


Figura 2.7: Exemplo de empacotamento em placa.

A partir de agora o termo padrão de corte será utilizado para simbolizar um determinado empacotamento em placa. Mais adiante será que visto que o problema de empacotamento em placa será encontrado como um sub-problema neste trabalho.

2.3.4 Empacotamento em Faixa

O problema de empacotamento em faixa consiste em, dado um número finito n de itens retangulares de dimensões $l_i x c_i$, encontrar a melhor maneira de posicioná-los em um retângulo maior de dimensão LxC de modo que a altura do empacotamento seja a menor possível, onde L representa a altura do retângulo.

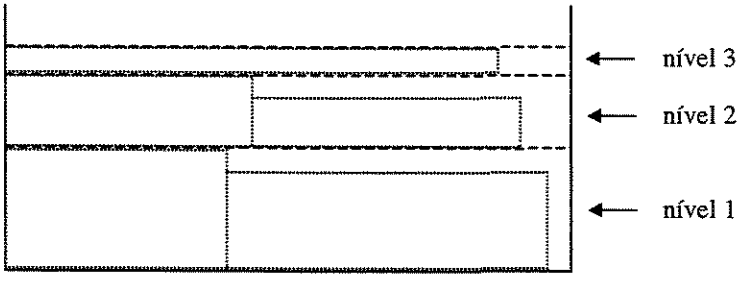


Figura 2.8: Exemplo de empacotamento em faixa.

Será utilizado neste trabalho um algoritmo de aproximação que fornece padrões de corte de grande qualidade e em tempo adequado. Esse algoritmo é uma combinação de dois outros, um utilizado para resolver problemas de empacotamento unidimensional e outro utilizado em empacotamento em faixa. Cabe frisar que existem diversos algoritmos, com as mais variadas técnicas de empacotamento para ambos os casos. Posteriormente veremos como os algoritmos foram implementados.

2.3.5 Algumas aplicações

Serão descritas algumas situações práticas onde os problemas de corte e empacotamento bidimensional aparecem com freqüência:

Empresas de Móveis

Ao fabricar materiais como cadeiras, mesas, armários, portas, entre outros, uma empresa de móveis tem em mãos grandes placas de madeira e necessita encontrar a melhor maneira de cortá-las de modo a economizar na quantidade de placas necessárias. Devido às diferentes matérias-prima que podem ser trabalhadas, a quantidade e a variedade de retângulos são altas.

Vidraçarias

Considere as janelas e divisórias onde devem ser colocados vidros de diferentes tamanhos, por exemplo, em um prédio. A construtora deve fazer um pedido de compra informando suas necessidades para uma vidraçaria que deve cortar pedaços de vidro de modo a satisfazer a demanda dos clientes e usar a menor quantidade possível de material.

Empresa de Tecelagem

Diversas empresas necessitam de tecidos para produzirem suas matérias-prima ou utilizá-los para fazer faixas, bandeiras, entre outros materiais de propaganda. As empresas de tecelagem dispõem de certa quantidade de tecido e como seu custo é proporcional ao tipo de material e ao comprimento, é importante saber qual é a quantidade mínima necessária para posteriormente economizar na compra desse material.

Alocação de Tarefas

Quando desejamos alocar um conjunto de tarefas de modo a minimizar o tempo total necessário para processá-las, podemos representar estas tarefas como retângulos, onde o comprimento é igual ao tempo necessário de processamento e a largura é igual ao número de processadores (citando como exemplo alocação de tarefas em multiprocessadores) ou uma quantidade contínua de memória usada pela tarefa.

Temos diversas outras aplicações de grande importância, mas os exemplos citados podem fornecer uma idéia de como existem oportunidades de aplicação do nosso estudo. No último exemplo, percebe-se que podem existir ainda problemas que não tratam de caixas ou placas, mas podem ser formulados como problemas de corte e empacotamento.

2.4 Programação Linear

Programação Linear é uma área de conhecimento que é utilizada extensivamente em setores industriais, militares, governamentais, entre outros, como uma importante ferramenta de planejamento para a utilização dos recursos disponíveis com a maior eficiência possível. A sua importância é devido à capacidade de modelar e resolver problemas grandes e complexos em tempo razoável (polinomial no caso médio). Seus benefícios são exatamente aqueles procurados pelas empresas: diminuição dos custos e aumento dos lucros.

A sua origem ocorreu durante a segunda guerra mundial quando grupos de assessores da força militar americana foram formados para resolver problemas estratégicos e táticos que não eram até então estudados pelos militares. Foram desenvolvidos métodos, processos e teorias. Como os resultados obtidos foram significativos, estes grupos continuaram a trabalhar mesmo após o final da guerra, redirecionando seus objetivos para abranger também a resolução de problemas civis ou não-militares [9].

Ao se formular um problema sob a forma de um modelo matemático, o objetivo é o de ajudar no processo de decidir quais alternativas e quanto de cada uma devem ser escolhidas, a fim de satisfazer um objetivo, que normalmente é maximizar (minimizar) uma função das atividades, geralmente lucros (perdas), perante atividades (variáveis de decisão) que competem entre si pela utilização de recursos escassos (restrições).

Um problema de programação linear genérico pode ser representado pelo seguinte sistema de inequações lineares:

$$\min(\max)\ z = c_1 x_1 + c_2 x_2 + \cdots + c_n x_n \tag{1}$$

$$\text{sujeito a} \begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \geq b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \leq b_2 \\ \vdots \quad \vdots \quad \vdots \quad \vdots \quad \cdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n = b_m \\ x_1, x_2, \cdots, x_n \geq 0 \end{cases} \tag{2}$$

Podemos ler o problema acima da seguinte maneira:

Encontre uma solução de modo que minimize (maximize) a função objetivo (1) respeitando as restrições impostas (2). A função objetivo, também chamada de função de custo, determina a qualidade da solução. Para resolver um problema utilizando Programação Linear, temos que garantir que as restrições e a função objetivo são lineares.

O conjunto de soluções que satisfazem todas as restrições é chamado de região viável ou conjunto de pontos viáveis. Uma solução ótima é aquela que faz com que a função objetivo retorne o mínimo (máximo) valor possível respeitando todas as restrições. O valor obtido é então denominado valor ótimo.

Existem vários métodos para resolver problemas de programação linear, o mais conhecido é o método Simplex que apesar de ter complexidade de tempo exponencial no seu pior caso, é polinomial no caso médio.

2.4.1 Simplex

O Método Simplex foi criado por George B. Dantzig [9] em 1947 para resolver com eficiência (polinomial no caso médio) problemas de programação linear, fornecendo em um número finito de iterações a solução ótima ou a inexistência de uma solução ótima para o problema.

Conhecido como o pai da programação linear, um fato interessante sobre Dantzig e o método simplex [15] é que durante uma das aulas de estatística que cursava, ele chegou atrasado e copiou um problema (de programação linear) pensando ser uma tarefa para ser feita após o horário, mas na verdade era um problema de estatística que havia sido apresentado como insolúvel por seu professor. Alguns dias depois ele entregou o problema resolvido e essa estória

ficou famosa, pois existe a seguinte curiosidade: teria ele resolvido o problema se soubesse que pouco antes seu professor havia dito que não existia solução ?

O algoritmo simplex pode ser resumido da seguinte maneira:

Algoritmo Simplex

1 Colocar o problema na *forma padrão*
2 Procurar uma *solução básica viável* inicial.
3 Se não encontrar
  3.1 Retorne que o problema é inviável
4 Senão
  4.1 Enquanto conseguir melhorar a função objetivo faça:
    4.1.1 Pesquise a próxima *solução básica viável.*
5 Retorne solução ótima ou indicação de solução ilimitada

fim algoritmo

Como vimos no primeiro passo, o problema deve ser formulado na forma padrão, isto é, acrescentando variáveis de folga e colocando as restrições em forma de igualdade, desta forma temos:

$$\min(\max)\ z = c_1x_1 + c_2x_2 + \cdots + c_nx_n$$

$$\text{sujeito a} \begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n + x_{n+1} = b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n + x_{n+2} = b_2 \\ \vdots \quad \vdots \quad \vdots \quad \vdots \quad \cdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n + x_{n+m} = b_m \\ x_1, x_2, \cdots, x_n, x_{n+1}, \cdots, x_{n+m} \geq 0 \end{cases}$$

Sabendo que as restrições impostas formam um poliedro convexo, o algoritmo passa então a procurar a melhor solução em um dos pontos extremos da região viável que torne a função mínima (máxima).

Podemos ainda representar o problema na forma matricial, formato utilizado pelos algoritmos computacionais que resolvem problemas de programação linear. Em cada iteração do

método simplex, a matriz é dividida em duas partes, uma base A_B e o restante da matriz A_N. As variáveis associadas às colunas de A_B são ditas variáveis básicas e as variáveis associadas às colinas de A_N são chamadas de não-básicas. Assim temos:

$$c = \begin{bmatrix} c_1 & c_2 & \cdots & c_n & \cdots & c_{n+m} \end{bmatrix}$$

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} & 1 & 0 & \cdots & 0 \\ a_{21} & a_{22} & \cdots & a_{2n} & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ a_{m1} & a_{m2} & & a_{mn} & 0 & 0 & \cdots & 1 \end{bmatrix}$$

$$x_B = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad x_N = \begin{bmatrix} x_{n+1} \\ x_{n+2} \\ \vdots \\ x_{n+m} \end{bmatrix} \quad b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

Onde o vetor x_B representa o vetor das variáveis básicas e o vetor x_N representa o vetor das variáveis não básicas ou de folga. O sistema pode ser escrito agora como:

$$\begin{array}{ll} \min & z = cx \\ \text{sujeito a} & A_B x_B + A_N x_N = b \\ & x \geq 0 \end{array}$$

Assim, temos que:

$$x_B = A_B^{-1} b - A_B^{-1} A_N x_N$$

$$z = c_B A_B^{-1} b + \left(c_N - c_B A_B^{-1} A_N\right) x_N$$

Se atribuirmos zero para todas as variáveis não básicas temos uma solução básica inicial, isto é:

$$\bar{x}_N = 0, \bar{x}_B = A_B^{-1} b \text{ e } \bar{z} = c_B A_B^{-1} b$$

Pela notação matricial, podemos resumir o algoritmo simplex da seguinte maneira:

Encontre uma solução básica inicial viável:

$$\bar{x}_N = 0, \bar{x}_B = A_B^{-1} b \geq 0$$

A solução corrente é ótima se

$$\left(c_N - c_B A_B^{-1} A_N\right) \geq 0$$

Se existe uma coluna tal que:

$$\left(c_s - c_B A_B^{-1} A_s\right) < 0,$$

então esta coluna entra na base e alguma coluna $l \in$ B deixa a base para produzir uma nova base (pivotação). Repita o processo enquanto houver colunas que melhorem a função objetivo.

A notação matricial será utilizada a partir de agora por ser mais compacta e por facilitar o entendimento da seção onde será explicado o método de geração de colunas.

2.4.2 Simplex Revisado

Simplex Revisado é a técnica utilizada na maioria dos *solvers* (sistemas especializados em resolver problemas de programação matemática), para resolver problemas como o tratado neste trabalho, onde é inviável gerar todas as colunas para posteriormente aplicar o método simplex.

Este método difere do Simplex padrão por não exigir que todas as colunas sejam geradas *a priori*, evitando desperdício de tempo e memória. Podemos trabalhar com novas colunas conforme a necessidade.

Em algoritmos computacionais, o Simplex Revisado é mais eficiente que o Simplex por utilizar menor quantidade de memória e por realizar menor número de operações.

Todos os conceitos apresentados sobre o Simplex continuam válidos, bem como o reconhecimento da existência de uma única solução, de infinitas soluções, de um problema ilimitado ou de um problema inviável.

Normalmente o número de variáveis envolvidas no problema de otimização é muito maior que a quantidade de restrições. O Simplex Revisado trabalha com uma solução básica

viável inicial e não utiliza mais do que uma matriz quadrada para realizar os cálculos, onde o número de colunas é igual ao número de restrições.

2.5 Programação Linear Inteira

Ao formular um problema de programação linear, a solução ótima pode significar utilizar parte dos recursos disponíveis. Por exemplo, se estivéssemos formulando um problema de alocação de recursos, não poderíamos alocar uma pessoa e meia para uma determinada tarefa, possivelmente teríamos que decidir entre uma ou duas pessoas.

Isso ocorre freqüentemente em problemas reais e um problema de programação linear deve ser modelado de modo a prever situações como esta. Uma alternativa para este problema seria arredondar a solução, mas em alguns casos poderíamos estar nos afastando muito da solução ótima.

Para resolver esta situação, acrescenta-se ao problema novas restrições, forçando a solução a conter apenas variáveis inteiras. Com isso, o problema passa a ser de programação linear inteira.

Apesar da formulação não sofrer grandes alterações, computacionalmente o problema de programação linear inteira é muito mais complexo, pois dependendo do número de variáveis existentes, acredita-se que a solução ótima com todas as variáveis inteiras só pode ser obtida com algoritmos de complexidade de tempo exponencial. É conhecido que um problema de programação linear inteira é NP-Difícil [3].

Um problema de programação linear inteira genérico possui o seguinte formato:

$$\begin{array}{ll} \min & z = cx \\ \text{sujeito a} & Ax = b \\ & x \geq 0 \text{ e inteiro} \end{array}$$

Cabe citar que existem ainda outros problemas de programação linear inteira que utilizam formulações particulares desta, como por exemplo, programação binária. Caso a solução do

problema aceite que algumas variáveis sejam inteiras e outras contínuas (fracionárias), estaremos tratando de um problema de programação linear mista.

2.5.1 Branch and Bound

Devido à dificuldade de resolver um problema de programação linear inteira, algumas técnicas alternativas podem ser utilizadas. Uma delas é a relaxação linear do problema e aplicação do método de Ramificação e Poda (*Branch and Bound*).

Após relaxar o programa de programação linear inteira, a solução poderá conter algumas ou todas variáveis fracionárias. A técnica *Branch and Bound* consiste em pegar a solução fracionária e escolher, iterativamente, uma das variáveis com valor fracionário para dividir o problema em dois subproblemas, um onde a variável escolhida é limitada superiormente pelo valor fracionário arredondado para baixo e outro onde a variável escolhida é limitada inferiormente pelo valor fracionário arredondado para cima.

Para facilitar o entendimento desta técnica, vejamos o seguinte exemplo:

Suponha que a solução encontrada recomenda utilizar 1.8 unidades do recurso do tipo P_1 e 2 unidades do recurso do tipo P_2. Desta maneira o algoritmo divide o problema conforme a figura a seguir:

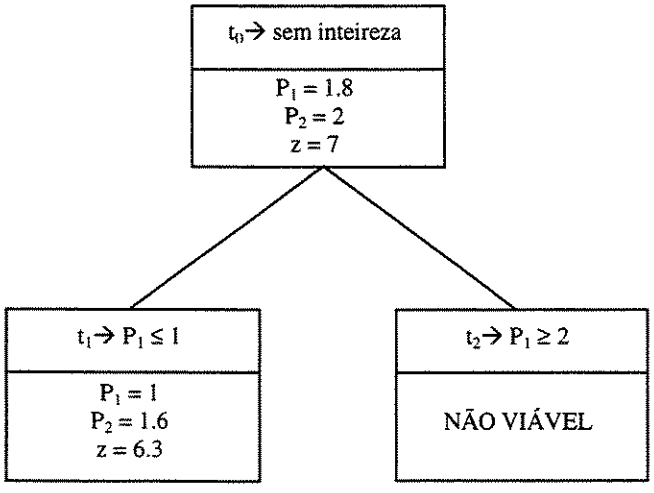


Figura 2.9: Exemplo de execução da técnica Branch and Bound.

Agora, um dos nós que não está ramificado é escolhido arbitrariamente e resolvido. Após isto, pode-se encontrar uma solução inteira ou repete-se o processo de divisão novamente (*branch*). Conforme a figura 2.10, se encontra no nó esquerdo a solução do problema recomendando utilizar uma unidade de cada item.

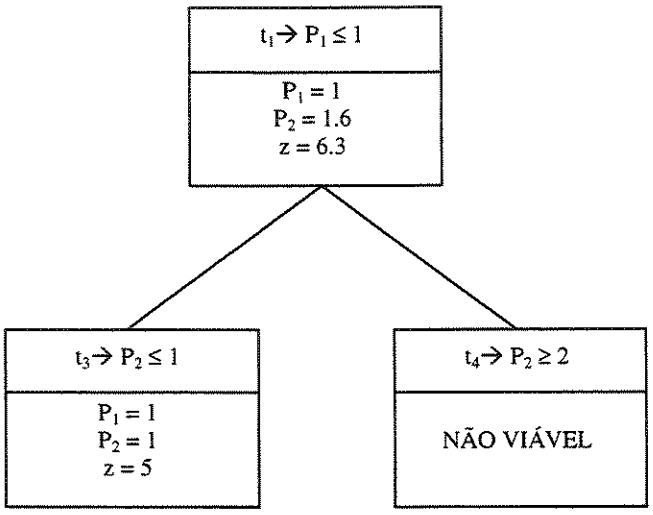


Figura 2.10: Continuação do exemplo de execução da técnica Branch and Bound.

Durante o processo, trabalha-se com todos os nós e, os que indicarem inexistência, inviabilidade ou simplesmente não levam a soluções melhores que a melhor encontrada até o momento são podados (bound) desta árvore e não são mais utilizados. No final tem-se uma solução factível inteira ou a inexistência de uma solução. Sempre que um novo ramo resulta numa solução não viável, pode-se abandonar todas as possíveis descendências.

2.6 Heurísticas e Algoritmos Aproximados

Devido a intratabilidade de problemas NP-Completos, os esforços para resolver problemas desta classe de complexidade podem ser direcionados ao estudo de outros métodos, como heurísticas e algoritmos aproximados.

Quando o tempo disponível para encontrar uma solução é escasso, métodos heurísticos são amplamente utilizados. Apesar de fornecerem soluções muitas vezes de boa qualidade, esta qualidade não pode ser garantida.

Existem ainda as meta-heurísticas, que são heurísticas de busca no domínio de soluções adaptadas para realizarem buscas em vizinhanças ou populações de soluções, com estratégias para escapar de armadilhas de ótimos locais. Pode-se encontrar na literatura diversas abordagens sobre problemas de otimização utilizando meta-heurísticas, dentre as quais destacam-se: Simulated Annealing, Busca Tabu e os Algoritmos Genéticos.

Algoritmos exatos produzem soluções ótimas, porém os conhecidos para problemas NP-difíceis possuem complexidade de tempo pelo menos exponencial, sendo então aplicáveis apenas para problemas de pequeno e médio porte.

Com a impossibilidade de se utilizar algoritmos exatos, algoritmos aproximados combinados com outras técnicas de refinamento são normalmente usados. Diferentes das heurísticas, os algoritmos aproximados possuem garantia de desempenho e podem ser adaptados conforme a necessidade de cada problema.

Essa garantia é uma medida (constante) da proximidade da solução encontrada pelo algoritmo em relação à solução ótima do problema, isto é, dado um algoritmo A para uma classe de problemas de P e seja $A(I)$ o valor da solução encontrada pelo algoritmo A para uma instância $I \in P$ e $OPT(I)$ o valor da solução ótima de I. Considerando o seguinte resultado:

$$A(I) \leq \alpha \cdot OPT(I) + \beta \text{ para todo } I \in P$$

Dizemos que α é um limite de *desempenho assintótico* para o algoritmo A. Se $\beta = 0$, dizemos que α é um limite de *desempenho absoluto*.

CAPÍTULO 3

O Problema do Corte Bidimensional

3.1 Introdução

Neste capítulo o problema tratado neste trabalho será detalhado. Serão listadas as restrições que foram consideradas e uma análise da sua complexidade computacional será feita.

O problema de corte em placas ou corte bidimensional consiste em: dado um número finito n de itens retangulares de largura l_i, comprimento c_i e demanda d_i, a serem obtidos de retângulos maiores de dimensão LxC, encontrar padrões de corte que atendam a demanda de itens solicitados utilizando o menor número possível de retângulos maiores.

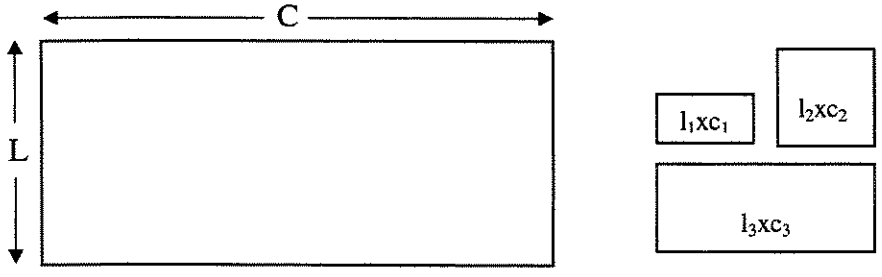


Figura 3.1: Placa principal e itens a serem obtidos.

O objetivo de nosso estudo é encontrar padrões de corte que minimizem a quantidade de placas (retângulos maiores) utilizadas através de técnicas de programação matemática, algoritmos de aproximação e o método de geração de colunas. Na literatura o problema é conhecido como *"two-dimensional constrained cutting stock problem"*, onde o número de itens diferentes a serem obtidos é pequeno e a demanda por eles é alta. Outra característica do objeto em estudo é que

estamos tratando de um problema restrito, isto é, o número de retângulos menores possui uma demanda que deve ser respeitada.

Neste estudo a quantidade de estágios/níveis de corte não será restringida, essa informação ficará em aberto para que o usuário do sistema possa trabalhar conforme sua necessidade e essa restrição será respeitada durante a determinação dos padrões de corte. No final do trabalho, testes computacionais serão realizados para a verificação da influência do estágio/nível de corte na solução e o usuário do sistema poderá determinar qual opção se adequa melhor às suas necessidades.

Outras restrições que foram impostas ao problema devido às técnicas que serão utilizadas para obtenção de padrões e por aparecerem constantemente no processo de corte das fábricas são:

i) Os itens a serem obtidos devem respeitar as medidas da placa principal, portanto $l_i \leq L$ e $c_i \leq C$.

ii) Os itens possuem uma orientação fixa e não podem ser alteradas durante o processo de corte (problema orientado).

iii) As medidas das placas devem ser inteiras.

iv) O corte feito será do tipo guilhotina.

v) Os retângulos maiores são todos do mesmo tamanho.

Não pode ser esquecido que a lâmina possui uma espessura e existe uma pequena sobra entre os cortes. Esse "problema" é contornado pelo algoritmo adicionando o valor da espessura em todos os itens e no retângulo maior.

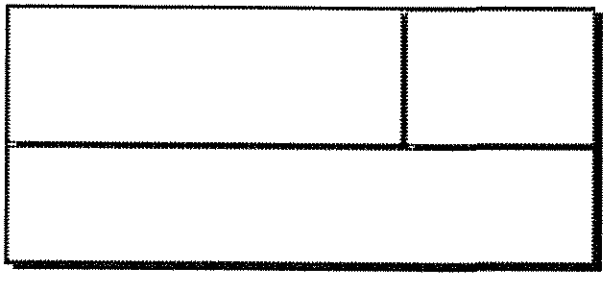


Figura 3.2: Solução para o problema da espessura da lâmina

3.2 Complexidade do Problema

Para resolver este problema, poderíamos utilizar nossa intuição definindo alguns padrões de corte e escolhendo o melhor por tentativa e erro. Mas como garantir que não existem padrões de corte melhores? Precisaríamos conhecer todos os padrões para decidir, o que na prática só é possível quando a quantidade de padrões que podem ser gerados é pequena.

Outra estratégia poderia ser enumerar todos os padrões possíveis e aplicar técnicas de programação matemática para determinar os melhores.

Na prática poderemos ter uma grande quantidade de padrões, onde até mesmo o computador, com sua incrível capacidade de armazenamento e processamento, poderá não conseguir armazenar e processar todos em um tempo viável.

Ao analisar as técnicas empregadas para a resolução do problema do corte bidimensional, faz-se necessário verificar qual a complexidade computacional do problema.

Se reduzirmos o problema de empacotamento unidimensional, que é um dos problemas clássicos encontrados na literatura e já foi provado ser NP-Difícil [3], ao problema de empacotamento bidimensional, será provado que o problema do corte bidimensional possui, no mínimo, a mesma complexidade.

A redução pode ser feita da seguinte maneira:

Considere que temos uma barra de comprimento C e queremos arranjar n itens da melhor maneira possível, utilizando o menor número possível de barras, isto é:

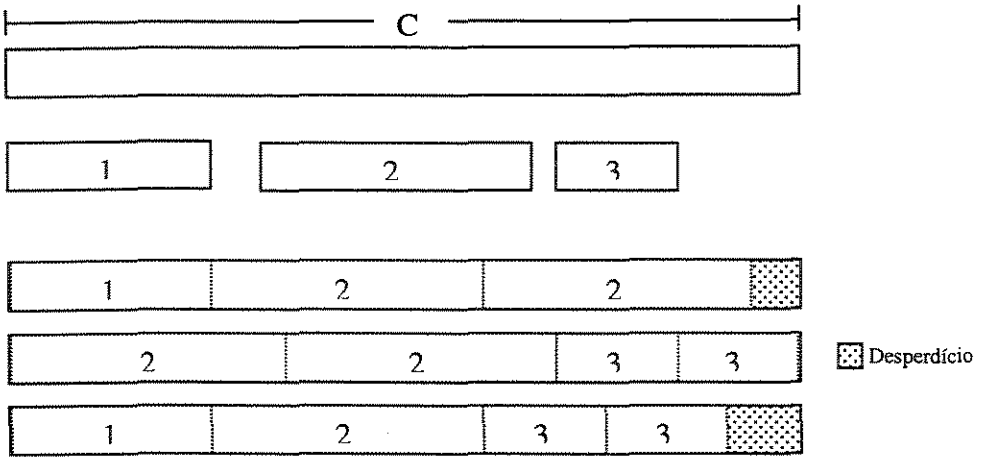


Figura 3.3: Barra principal, itens a serem cortados e 3 possíveis padrões.

Se fixarmos a largura de todos os itens e barras para um mesmo valor, podemos encarar todos os objetos como placas, isto é:

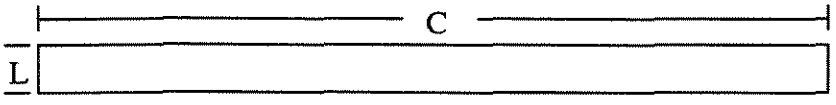


Figura 3.4: Problema unidimensional reduzido ao problema bidimensional.

Claramente podemos concluir que ao resolver o problema do empacotamento bidimensional, estaremos resolvendo também o problema do empacotamento unidimensional. Portanto o problema do empacotamento bidimensional é NP-Difícil.

Assim, ainda não foram desenvolvidos algoritmos que resolvam o problema de forma exata para qualquer instância, e muitas vezes é necessário concentrar os esforços no estudo de técnicas que gerem soluções aproximadas e em tempo adequado.

Abordagem Estudada

4.1 Introdução

Conforme mencionado anteriormente, existem na literatura diversas abordagens que tratam o problema do corte bidimensional, onde são consideradas diferentes restrições e utilizadas diferentes técnicas de resolução.

Optou-se por estudar neste trabalho uma abordagem clássica, que utiliza programação linear inteira e o método de geração de colunas, apresentada em 1961 por Gilmore e Gomory. A primeira aplicação do método de geração de colunas foi proposta para resolver o problema do corte e empacotamento [4], mas esta técnica pode ser aplicada para encontrar soluções para outros problemas.

Para as próximas seções, é recomendável que o leitor possua conhecimentos básicos de orientação a objetos, pois o sistema foi desenvolvido utilizando-se esta metodologia e serão citadas algumas implementações que foram feitas.

4.2 Visão Geral do Processo

Antes de entrar em detalhes nos algoritmos que foram estudados, será apresentada uma visão geral do processo para que futuramente o leitor possa voltar a esta seção para ver como cada parte que estiver lendo se encaixa no processo como um todo e facilitar o entendimento.

O seguinte quadro resume o processo:

Programa Principal::executa()

Entrada: Dados de todas as placas e restrições
Saída: Padrões de corte que minimizam a quantidade de placas necessárias

1 Calcular pontos de discretização
2 Gerar através de um algoritmo aproximado um conjunto inicial de padrões.
3 Formular um problema de programação linear inteira com o conjunto inicial de padrões gerado.
4 Enquanto existirem padrões melhores faça:
  4.1 Resolver o problema de programação linear inteira relaxado.
  4.2 Atribuir valores duais para os itens.
  4.3 Encontrar o melhor padrão com os pesos associados.
  4.4 Se o valor do padrão for maior que 1 continuar processo, senão fim enquanto.
  4.5 Acrescente o padrão no formato de coluna no programa linear.
5 Aplicar uma técnica para obter solução inteira.
6 Retorne padrões encontrados

Fim programa

Note que no passo 4-3, verificamos se o valor de importância ou peso do padrão é maior que 1 (um), esse número é devido ao coeficiente da função objetivo.

A seguir detalhamos, por ordem de execução, cada etapa existente na abordagem estudada.

4.3 Entrada de Dados e Inicializações

Inicialmente, é necessário conhecer o tamanho da placa (retângulo maior) e dos itens, que devem ser informadas pelo usuário do sistema. Para guardar essas informações foi criada uma classe chamada placa_principal, que tem o objetivo de guardar as características da placa a ser cortada e um vetor de classes do tipo placa_menor, onde serão guardadas as características dos itens a serem obtidos. Ambas as classes herdam as características da classe placa, que contém algumas operações e informações básicas como altura, comprimento e cor.

Os itens são armazenados de forma que fiquem ordenados de forma decrescente pela sua largura. O motivo é que alguns algoritmos tiram proveito desta ordenação.

Além destas informações, o usuário poderá configurar o sistema com as seguintes opções:

- Tempo de execução do sistema (o sistema irá devolver a melhor solução encontrada no tempo indicado).
- Espessura da guilhotina.
- Tipo dos cortes (estágio ou nível) e quantidade de vezes que a guilhotina irá atravessar a placa para gerar os padrões de corte.
- Percentual de redução dos pontos de discretização.

O segundo passo é determinar as melhores posições para realizar os cortes. Para isso, determina-se as combinações das medidas de todas as placas nas posições horizontal e vertical. Esse trabalho será feito por uma classe chamada gerador_posicao e a qualquer momento pode-se consultá-la para conhecer as posições de corte.

Gerador_Posicao::executa()

Entrada
Comprimento C e largura L do retângulo maior
Comprimento c_i, largura l_i e demanda d_i dos itens a serem obtidos.
Saída:
Pontos de discretização

1 Verificar todas as combinações horizontais dos itens e atualizar vetor de pontos de discretização P.
2 Verificar todas as combinações verticais dos itens e atualizar vetor de pontos de discretização Q.

Fim Gerador_Posicao

Um detalhe que está encapsulado no algoritmo é que, a cada passo, é verificado se o tamanho do item excede a metade do tamanho da placa principal. Isso foi feito para evitar o empacotamento em áreas iguais geradas por dois cortes diferentes. Esta estratégia é conhecida como empacotamento espelho.

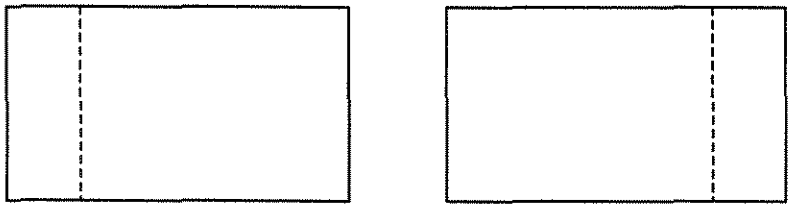


Figura 4.1: Empacotamento espelho.

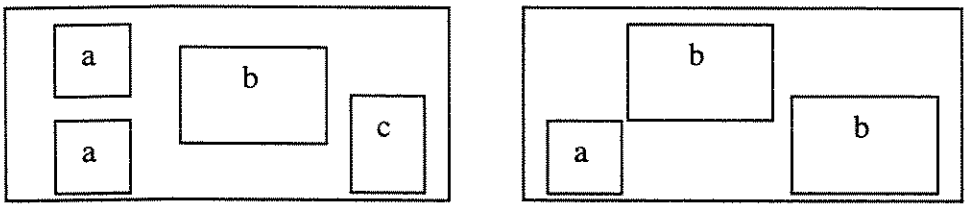
As informações obtidas até o momento são suficientes para instanciar uma classe do tipo gerador_padrao, responsável pela geração e armazenamento de todos os padrões gerados. Essa classe possui um método denominado gera_padrao que encapsula o algoritmo de Herz [1] utilizado para encontrar padrões de corte com o menor desperdício possível para o problema de empacotamento bidimensional. Este algoritmo também será visto em detalhes ainda neste capítulo.

4.4 Formulação do Problema

Para adequar o problema do corte bidimensional no formato de um problema de programação linear inteira, tem-se:

- Objetivo: usar o menor número possível de placas.
- Restrições: o número de itens obtidos com os padrões de corte usados deve ser maior ou igual à demanda.
- Informações conhecidas: demanda de cada item, padrões de corte (será visto como gerar os padrões na próxima seção).
- Informações desconhecidas: quantas vezes um determinado padrão de corte será utilizado.

Considere os padrões p e q que satisfazem a demanda d_i pelos itens a, b e c, representados pela figura abaixo:



(P_1) (P_2)

Figura 4.2: Padrões de corte.

Pode-se colocar essas informações em formato matricial, onde cada coluna representa um padrão e cada linha representa a quantidade de itens utilizados pelo padrão, isto é:

	P_1	P_2
a	2	1
b	1	2
c	1	0

Se chamarmos essa matriz de A, podemos formular o problema da seguinte maneira:

$$\begin{aligned} \min \quad & z = cx \\ \text{sujeito a} \quad & Ax = b \\ & x \geq 0 \text{ e inteiro} \end{aligned}$$

A coluna denotada pela letra b, representa a demanda de cada item. Assim temos o problema formulado como programação linear inteira, devido à inviabilidade da utilização de soluções fracionárias para este problema.

Dessa forma, o problema consiste em minimizar o número de placas, atendendo a uma demanda b_i de itens de dimensões $l_j x c_j$, sendo $i,j \geq 1$ e x inteiro.

Por se tratar de um problema NP-Dificil, pode não ser viável encontrar a solução ótima para um problema de programação linear inteira com muitas variáveis, portanto uma relaxação será feita no problema, eliminando as restrições que obrigam a solução encontrada ser inteira. Posteriormente serão aplicadas técnicas de arredondamento.

Uma alternativa para formular o problema, seria iniciar com uma matriz identidade uma vez que não temos nenhum padrão gerado, mas essa opção é muito ineficiente. Uma boa alternativa é obter um conjunto inicial de padrões através de alguma heurística ou algoritmo aproximado que gere padrões de qualidade em tempo polinomial.

4.5 Gerando uma solução inicial

Algoritmos de aproximação são técnicas bastante utilizadas para fornecer soluções boas para problemas complexos em um rápido período de tempo, porém sem garantia de que a solução ótima será encontrada. Devido a essa eficiência, essa foi a estratégia utilizada para gerar o conjunto inicial de padrões.

O algoritmo utilizado foi o *Hybrid First Fit*, que respeita todas as restrições impostas pelo nosso escopo e fornece soluções de alta qualidade (em termos assintóticos). Este algoritmo é uma combinação de outros dois, que serão apresentados a seguir.

4.5.1 Algoritmo First Fit Decreasing

O algoritmo FFD é um algoritmo off-line de empacotamento unidimensional que consiste em primeiramente ordenar os itens de maneira decrescente e posicionar o primeiro item na parte de baixo de uma placa principal (retângulo maior). Fazemos o mesmo para o segundo item. Se este item não couber na primeira placa principal (logo acima do 1° item), uma nova placa é requerida. Os itens subseqüentes continuam o processo tentando empacotar nas placas na ordem em que foram requisitadas e usando novas caso não haja espaço nas existentes.

Algoritmo FFD

Entrada: Informações da placa e dos itens, onde as placas e os itens têm o mesmo comprimento.
Saída: Conjunto de padrões

1 Faça uma ordenação não crescente dos itens pela largura
2 Enquanto existirem itens não empacotados faça
  2.1 Tente empacotar o item nas placas anteriores na ordem em que foram criadas
  2.2 Se não conseguir, empacote o item em uma nova placa
3 Retorne padrões encontrados.

Fim algoritmo

Vejamos um exemplo prático:

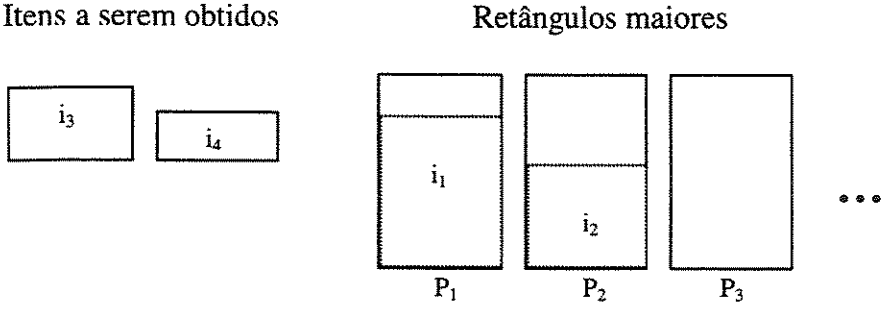


Figura 4.3: Representação gráfica do FFD através de exemplo.

Neste exemplo, o próximo item a ser empacotado é o i_3. Pela lógica do FFD, este item seria colocado na placa P_2 e o último item i_4, na placa P_1, totalizando dois padrões de corte necessários:

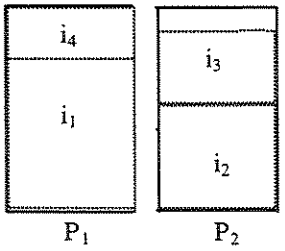


Figura 4.4: Resultado final do exemplo prático do FFD.

O algoritmo FFD garante que a solução encontrada é:

$$FFD(L) \leq \frac{11}{9}OPT(L)+1$$

sendo L a lista de itens e $OPT(L)$ o número de placas usadas em um empacotamento ótimo destes itens.

4.5.2 Algoritmo First Fit Decreasing Height

O algoritmo FFDH é um algoritmo de empacotamento em faixa e funciona da seguinte maneira: primeiramente ordena os itens por ordem não crescente de largura[2] e coloca o primeiro item na parte de baixo, justificado a esquerda. Pega então o segundo item e o coloca, se possível, ao lado do primeiro item. Se não for possível, traça uma faixa na altura do primeiro item e coloca o segundo item em cima, alinhado a esquerda. Repete-se o procedimento para os itens subseqüentes, tentando posicioná-los nas faixas na ordem em que foram criadas ou criando novas faixas caso não seja possível empacotar os itens.

[2] No artigo original, os autores consideram o termo altura no lugar de largura. Optamos pelo termo largura quando estamos tratando de duas dimensões e altura para uma terceira dimensão.

Algoritmo FFDH

Entrada: Informações da placa e itens.
Saída: Conjunto de faixas

1 Faça uma ordenação não crescente dos itens pela largura.
2 Enquanto houver itens não empacotados faça:
  2.1 Tente empacotar o próximo item nas faixas existentes, na ordem em que foram geradas, dispondo os retângulos lado a lado dentro da mesma faixa.
  2.2 Se não conseguir, empacote o item em uma nova faixa com o comprimento da placa e largura do item sendo empacotado.
3 Retorne faixas geradas.

Fim algoritmo

Vejamos um exemplo prático:

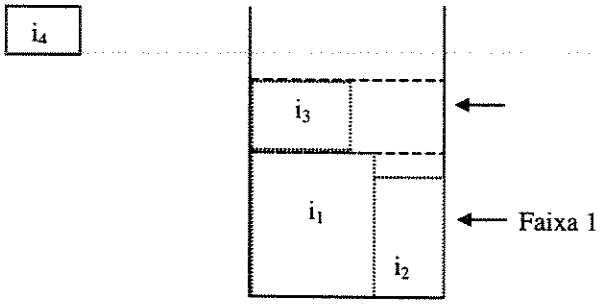


Figura 4.5: Representação Gráfica do FFDH através de exemplo.

Neste exemplo, o próximo item a ser empacotado é o i_4. Pela lógica do algoritmo FFDH, este item seria colocado na segunda faixa:

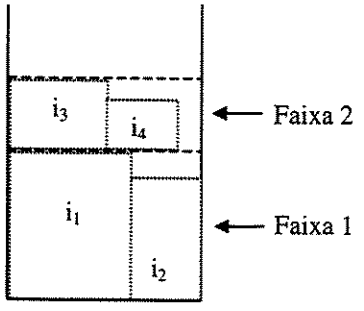


Figura 4.6: Resultado final do exemplo prático do FFDH.

O algoritmo FFDH garante que a solução encontrada é:

$$FFDH(L) \leq \frac{17}{10} OPT(L) + 1$$

sendo L a lista de itens e $OPT(L)$ o número de placas usadas em um empacotamento ótimo destes itens.

4.5.3 Hybrid First Fit

O Algoritmo Hybrid First Fit (HFF) é um algoritmo híbrido, isto é, ele é baseado em dois outros algoritmos de empacotamento simples e combina a potencialidade de cada um, gerando um resultado de grande qualidade. Escolhemos utilizar este algoritmo devido a sua eficiência, por utilizar o padrão de corte guilhotina e trabalhar com a orientação das placas fixa.

Esta técnica foi proposta por Chung, Garey e Johnson [2] em 1982 para o problema de empacotamento em placas e garante que a solução encontrada é

$$HFF(L) \leq \frac{17}{8} OPT(L) + 5,$$

sendo L a lista de itens e $OPT(L)$ o número de placas usadas em um empacotamento ótimo destes itens.

O HFF trabalha da seguinte maneira: primeiramente devemos criar as faixas através do FFDH e posteriormente utilizamos o FFD para empacota-las nas placas principais.

Algoritmo HFF

Entrada: Informações da placa e itens
Saída: Conjunto de padrões

1 Aplique o algoritmo FFDH para criar faixas
2 Empacote as faixas em placas usando o algoritmo FFD
3 Retorne padrões encontrados

Fim algoritmo.

Na prática temos faixas como as representadas pela figura abaixo:

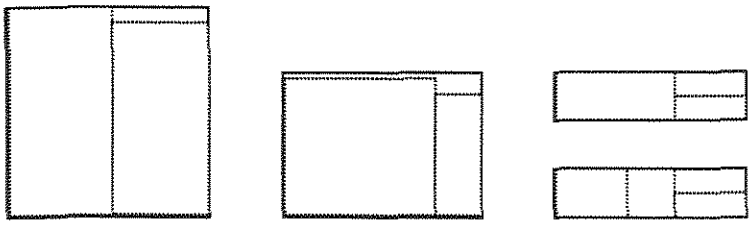


Figura 4.7: Faixas geradas pelo FFDH

Assim, o algoritmo HFF aplica o algoritmo FFD para empacotá-las em placas:

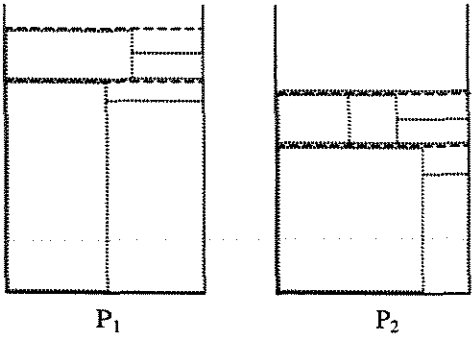


Figura 4.8: Resultado final do exemplo prático do HFF.

Após a geração do conjunto inicial de soluções, iniciamos o problema de programação linear, o resolvemos e pegamos os valores duais das variáveis que utilizamos para determinar o valor de importância de cada item, de modo que um padrão que otimize a função objetivo seja encontrado pelo método de geração de colunas.

4.6 O Método de Geração de Colunas

Baseada no trabalho de decomposição de Dantzig e Wolfe (*Dantzig-Wolfe decomposition*), a primeira aplicação prática desta técnica foi a determinação de padrões de corte unidimensionais, no trabalho escrito por Gilmore e Gomory [4]. A técnica de geração de colunas é normalmente aplicada em problemas de grandes dimensões, no caso de não se dispor de todas as colunas *a priori,* mas essa não é uma regra para a utilização desta técnica. A partir de um conjunto inicial de colunas, que geramos através do algoritmo HFF, resolve-se o problema de programação linear (problema mestre), obtendo-se as variáveis duais que serão utilizadas por um sub-problema para determinar novas colunas a serem consideradas no problema mestre.

Até o momento temos um problema de programação linear a ser otimizado e que foi inicializado com um conjunto inicial de colunas representadas pelos padrões gerados pelo algoritmo HFF. Com essas informações já podemos iniciar a resolução do problema pelo método do simplex revisado.

O objetivo agora é encontrar uma nova coluna com custo reduzido para entrar na base de modo a melhorar a solução, mas não sabemos quantas colunas podem existir. Para determinar o custo reduzido de uma variável x_j associada a uma coluna A_j, temos:

$$\bar{c}_j = c_j - c'_B B^{-1} A_j$$

Como o coeficiente de custo é igual a 1 (um) e $c'_B B^{-1}$ é o vetor das variáveis duais, denotado agora por y', temos:

$$\bar{c}_j = 1 - y' A_j$$

Portanto a coluna A_j tem custo reduzido negativo e conseqüentemente deve entrar na base se, e somente se, $1 - y' A_j < 0$, o que é equivalente a $y' A_j > 1$.

Mas como encontrar uma coluna que satisfaça tal condição? Podemos encontrar a resposta resolvendo um sub-problema de maximização, onde optamos por utilizar o algoritmo de Herz.

4.6.1 Algoritmo de Herz

Como vimos anteriormente, cada coluna do problema de programação linear corresponde a um padrão de corte e pode ser inviável armazenar todos os padrões de corte. Assim, a geração de padrões de corte é feita conforme a necessidade.

O algoritmo de Herz trabalha com o corte por níveis. Algumas máquinas de corte trabalham apenas com o corte por estágios. Devido a essa restrição, foi feita uma modificação no algoritmo original, permitindo que o usuário opte pelo tipo de corte que lhe convier. Em estudos computacionais será visto qual a eficiência de ambos os tipos de corte. Um tipo de padrão utilizado pelo algoritmo de Herz é o homogêneo, que consiste em colocar em uma placa vários itens do mesmo tipo.

Algoritmo de Herz

Entrada:
Informações da placa, itens e posições de corte.
Posição do último corte realizado.
Coordenadas da área gerada após o corte.
Coordenadas onde os itens devem começar a ser empacotados.
Tipo de Corte, número do nível ou estágio de corte, última direção no caso de estágio.
Saída:
Padrão de Corte

1. Criar e guardar *padrão homogêneo* com item que proporcione maior peso total (o peso é o valor de importância de cada item obtido através das variáveis duais do problema mestre).
2. Verificar se número do estágio ou nível não excede o informado pelo usuário.
    2.1 Se exceder, retorne *padrão homogêneo.*
    2.2 Se não, continuar processo.
3. Para todas as posições de corte horizontais faça:
    3.1 Se for corte por níveis
        3.1.1 Chamar a função recursivamente com as coordenadas do corte para cada lado com um nível a mais.
        3.1.3 Verificar se padrão retornado de cada lado somado tem peso maior que o melhor encontrado até agora.
        3.1.4 Guardar melhor padrão e deletar da memória o anterior.
    3.2 Se for corte por estágios
        3.2.1 Se direção for horizontal
            3.2.1.2 Chamar a função recursivamente com as coordenadas do corte para cada lado com mesmo nível.
        3.2.2 Se direção for vertical
            3.2.2.1 Chamar a função recursivamente com as coordenadas do corte cada lado com um nível a mais.
        3.2.3 Verificar se padrão retornado de cada lado somado tem peso maior que o encontrado até agora.
        3.2.4 Guardar melhor padrão e deletar da memória o anterior.
4. Repita o procedimento para todas as posições verticais, passando um nível a mais quando direção anterior for horizontal em cortes por estágio.
5. Retorna melhor padrão

Fim do algoritmo

A cada corte são gerados dois sub-problemas a serem atacados recursivamente pelo algoritmo e a divisão continuará até que não seja possível empacotar o menor item ou atinja o nível de divisões passado como parâmetro.

No início do processo é realizado um empacotamento homogêneo dos itens, isto é, é verificado qual item proporciona peso máximo em um empacotamento na área passada.

O empacotamento será feito por um objeto denominado empacotador, que considera a área disponível para o empacotamento, o valor de importância de cada placa menor e sua respectiva demanda. Ele fornecerá para o padrão o melhor empacotamento homogêneo possível.

Iremos representar graficamente os primeiros passos do algoritmo:

Primeiramente, verificamos através do objeto empacotador qual é o melhor empacotamento para a placa principal antes de realizar o corte:

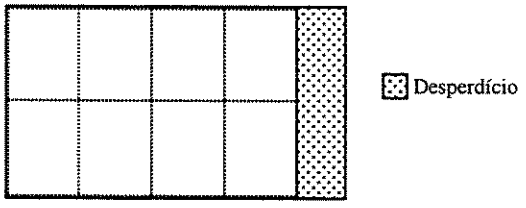


Figura 4.9: Processo de determinação de padrão – Passo 1.

Agora um corte é realizado na primeira posição fornecida pelo objeto gerador_posicao e o objeto empacotador se encarregará de fornecer o melhor empacotamento para cada lado, que chamamos de ladoA e ladoB.

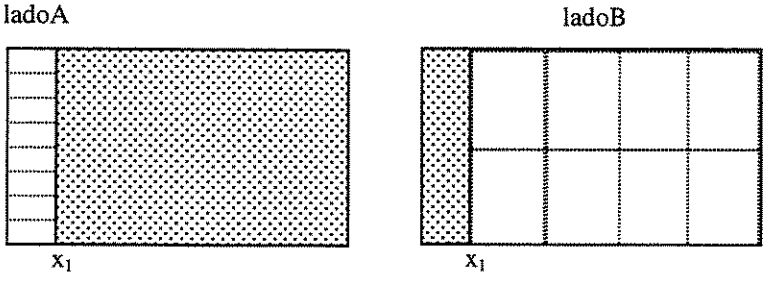


Figura 4.10: Processo de determinação de padrão – Passo 2.

A figura anterior nos mostra o melhor empacotamento de cada lado resultante do corte x_1. Ao voltar da recursão, os dois lados são combinados gerando o padrão abaixo.

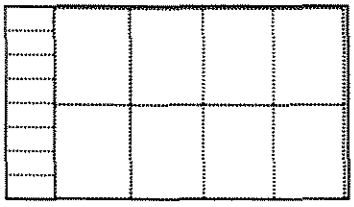


Figura 4.11: Processo de determinação de padrão – Passo 3.

Neste ponto o algoritmo compara este padrão com o outro gerado antes do corte(fig. 4.9) e guarda o melhor. Esse processo continuará até que o empacotamento seja testado em todas as áreas geradas através das posições de corte (horizontais e verticais) fornecidas pelo objeto gerador_posicao. A cada retorno da recursão, será testado se o peso do padrão gerado antes do corte é maior que o peso do padrão gerado após o corte e no final uma árvore que representa o padrão com o menor desperdício possível é retornada.

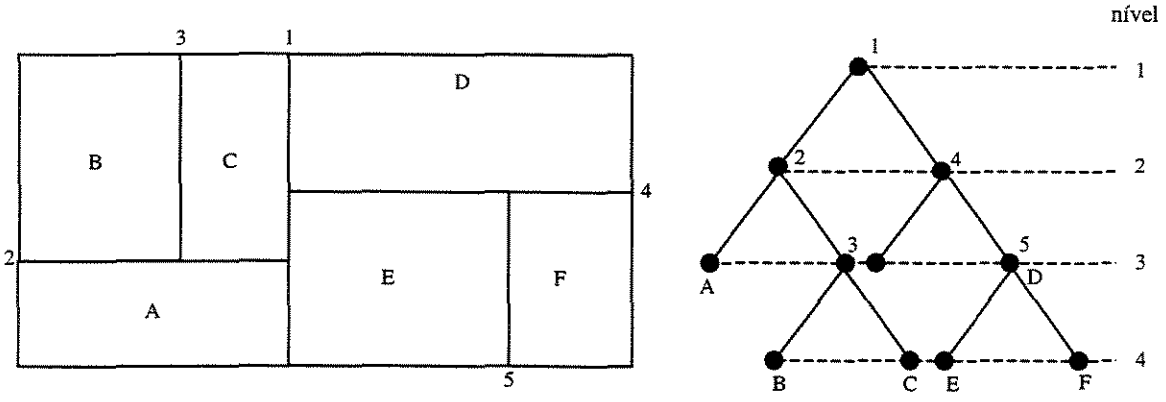


Figura 4.12: Padrão genérico por níveis e sua representação em árvore.

No início a árvore possui apenas um nó (raiz) que representa o melhor empacotamento na placa inteira, conforme os cortes são feitos, os nós são gerados dinamicamente e guardados caso sejam melhores que os anteriores.

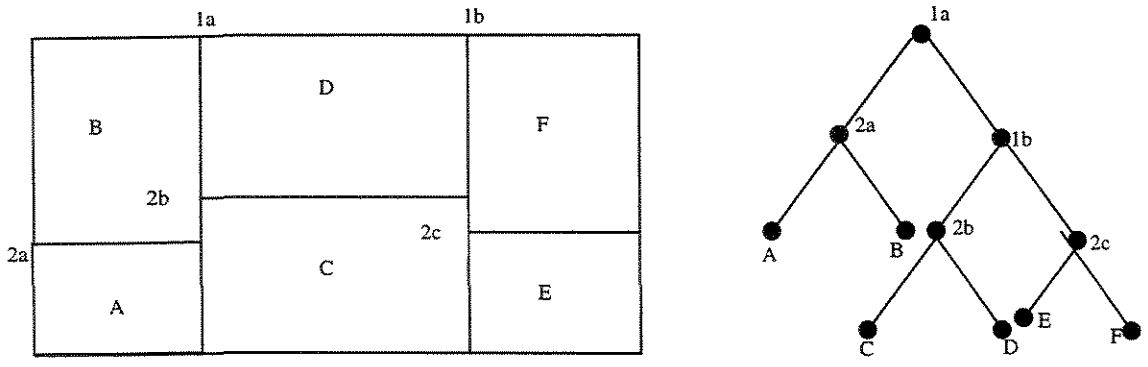


Figura 4.12: Padrão genérico por estágios e sua representação em árvore

Portanto a classe padrão representa um padrão de corte, com todas as informações necessárias para geração de colunas do problema de programação linear e para cálculos de qualidade de resultados gerados.

4.7 Obtendo soluções inteiras

Vamos supor que a loja de móveis citada como exemplo no início do trabalho esteja atrasada para entregar uma grande encomenda de um importante cliente. Nesse caso podemos não dispor de tempo para encontrar a melhor solução e uma solução aproximada está de bom tamanho.

Em situações como esta, onde o tempo de processamento é mais importante, podemos deixar de aplicar técnicas mais demoradas como o *branch and bound* e apenas arredondar uma solução fracionária.

Descreveremos a seguir como podemos arredondar as soluções fracionárias e mostraremos no próximo capítulo uma comparação da qualidade das soluções obtidas através dos arredondamentos e outras combinações.

4.7.1 Arredondamento para cima

Em sua primeira abordagem sobre o problema de corte, Gilmore e Gomory [4] propuseram fazer uma relaxação do problema, isto é, eliminar as restrições que obrigavam a solução ser inteira e arredondar a solução para cima. Eles utilizaram o arredondamento para cima por estarem mais

preocupados em resolver o problema de forma eficiente (em tempo viável) e menos com a solução exata do problema.

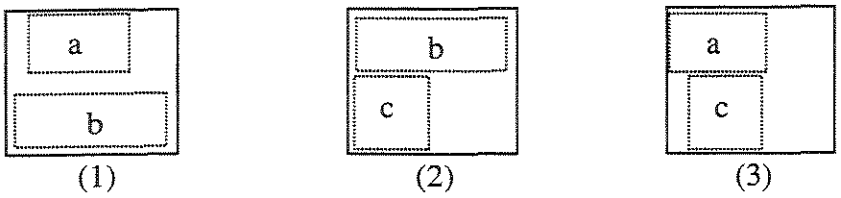


Figura 4.13: Exemplo de solução para o problema de corte.

Vamos considerar a solução representada pela figura 4.13. Supondo que para atender a demanda de 3 itens de cada tipo, a solução seja utilizar 1.5 padrões (1), 1.5 padrões (2) e 1.5 padrões (3). Claramente, se arredondarmos para cima, iremos utilizar 4 itens de cada tipo.

Arredondar para cima garante que estamos cumprindo o objetivo de atender a demanda com padrões de corte inteiros.

Veremos em resultados computacionais qual a diferença das soluções que sofreram arredondamento para cima em relação às outras soluções.

4.7.2 Arredondamento para baixo

Seguindo o mesmo exemplo representado pela figura 4.13, isto é, para atender a demanda de 3 itens de cada tipo, a solução seja utilizar 1.5 padrões (1), 1.5 padrões (2) e 1.5 padrões (3). Se arredondarmos para baixo implica em utilizar dois itens de cada tipo.

Portanto, se optarmos por fazer o arredondamento para baixo, não estaremos atendendo a demanda solicitada. Neste caso dizemos que temos um problema residual a ser tratado.

Neste exemplo temos que empacotar mais um item de cada tipo. Uma boa estratégia é aplicar uma heurística no problema residual.

Caso o algoritmo encontre um padrão que utilize os três itens restantes, estaríamos melhorando a solução em relação ao arredondamento para cima (1 placa principal de economia), porém pagamos pelo tempo necessário para encontrar essa solução.

No capítulo de resultados computacionais poderemos ver a diferença entre as duas opções.

4.7.3 Branch and Bound

Relaxando o problema para o formato de programação linear e através do método de geração de colunas, encontramos uma solução ótima fracionária para o problema do corte bidimensional. Como não podemos utilizar padrões quebrados, precisamos tornar nossa solução inteira.

Devido às suas características, o método *Branch and Bound* se mostra uma eficiente opção para obtermos uma solução inteira.

A combinação da técnica *Branch and Bound* com o método de geração de colunas dá-se o nome de *Branch and Price*. Esta técnica não foi implementada em nosso trabalho e ficará como um melhoramento a ser desenvolvido em trabalhos futuros.

CAPÍTULO 5

Resultados Computacionais

5.1 Introdução

Neste capítulo apresentamos um estudo prático da abordagem estudada. São apresentadas algumas variações do problema e alterações de algumas restrições para vermos como o sistema desenvolvido se comporta.

Para resolver os problemas de programação matemática, utilizamos um sistema de programação linear desenvolvido por terceiros. Existem diversos destes programas disponíveis no mercado, dentre os quais podemos destacar o CPLEX e o XPRESS, conhecidos por sua eficiência, porém ambos são comerciais. Outros solvers, apesar de não serem comerciais são muito bons, dentre os quais escolhemos utilizar o LP_SOLVE, desenvolvido em linguagem C ANSI por Michel Berkelaar e está na sua versão 3.2. Segundo dados do autor [14], este solver possui capacidade de resolver problemas de programação linear inteira mista com 30.000 variáveis e 50.000 restrições.

Este sistema recebe como dado de entrada o sistema que representa o problema de programação matemática e aplica, conforme nossa escolha, um ou mais métodos de otimização (simplex, simplex revisado, *branch and bound*, etc) retornando os valores ótimos das variáveis. Cabe lembrar que a qualidade da solução também é afetada pela qualidade do solver utilizado, pois temos que respeitar suas limitações.

Para fornecer a matriz inicial e as colunas do problema, desenvolvemos um sistema orientado a objetos utilizando a linguagem C++. Através deste sistema podemos preparar os dados para a formulação dos modelos, analisar a qualidade dos resultados e mostrar o resultado graficamente.

Os testes foram realizados com diversas instâncias geradas de forma pseudo-aleatória e o equipamento utilizado foi um Pentium 4 Intel com CPU de 1.6 GHz e 256 MB de memória.

As instâncias foram divididas em classes denominadas D (Diversificado), P (Pequenos), G (Grandes) e M (Médios), contendo itens (x_i, y_i, d_i) geradas pseudo-aleatóriamente da seguinte forma:

D: x_i, y_i pertencentes a [10% , 100%] do correspondente tamanho da placa
d_i pertencente ao intervalo [5...100]

P: x_i, y_i pertencentes a [5% , 25%] do correspondente tamanho da placa
d_i pertencente ao intervalo [5...100]

G: x_i, y_i pertencentes a [25% , 100%] do correspondente tamanho da placa
d_i pertencente ao intervalo [5...100]

M: x_i, y_i pertencentes a [20% , 60%] do correspondente tamanho da placa
d_i pertencente ao intervalo [5...100]

Para todas as instâncias, as dimensões das placas são 100x200.

As colunas que serão vistas nas tabelas contendo os resultados dos testes são :

n = número de itens.

C = Classe do problema (P,M,G ou D).

NP_HFF = Número de placas usadas pelo algoritmo HFF.

AC = Número de placas usadas pelo algoritmo que arredonda para cima.

AB = Número de placas usadas pelo algoritmo que arredonda para baixo e resolve o residual pelo HFF.

LB = Quantas placas é o mínimo que devemos usar calculado pela área. (soma das áreas de todos os retângulos / área da placa). Este limitante inferior, que não é o valor ótimo, será utilizado apenas como uma referência em relação à solução encontrada.

T = Tempo computacional necessário para a execução do processo (em minutos).

NP # E = Número de placas com # estágios.

NP # N = Número de placas com # níveis;

T # E = Tempo necessário para executar com # estágios (em minutos).

T # N = Tempo necessário para executar com # níveis (em minutos).

Para o tempo foi estipulada uma faixa de processamento, isto é, o programa deve retornar a melhor solução que encontrar no tempo informado. Em nossos testes fixamos que o programa deve retornar a solução a partir de 15 minutos e não passe de 25 minutos. Essa faixa foi escolhida devido ao acompanhamento dos testes e percepção de que até 20 minutos de execução, a geração de padrões converge com mais eficiência. Após esse tempo a geração de padrões pode se tornar muito lenta (em alguns casos o padrão seguinte demorou mais de 1 hora para ser gerado).

Por exemplo, Vamos supor que para uma determinada execução, ele tenha gerado os padrões com o seguinte tempo:

1° padrão – 0,5 minuto.
2° padrão – 2 minutos.
3° padrão – 10 minutos.

Na soma, a execução demorou 12,5 minutos. Supondo que o próximo padrão demore mais que vinte e cinco minutos (ou mais 12,5 minutos – tempo estipulado), a execução é abortada e a solução retornada é aquela que foi encontrada em 10 minutos (3° padrão), senão a retornada vai ser a próxima que passar de 15 minutos.

Vejamos agora alguns comparativos para que o leitor possa ter uma perspectiva da qualidade dos resultados obtidos e possa escolher qual solução atende suas necessidades em relação à qualidade de solução e o tempo de processamento (o tempo de input/output não foi considerado).

5.2 Análise da eficiência do sistema

Antes de iniciar os testes, verificamos se o sistema irá gerar os resultados conforme o esperado. Para garantir isso, criamos uma instância onde já sabemos qual será a solução ótima, de modo que é esperado que o algoritmo chegue nessa solução para que os testes sejam realizados de forma correta.

Primeiramente, definimos as dimensões da placa e realizando cortes do tipo guilhotina, determinamos as dimensões dos itens.

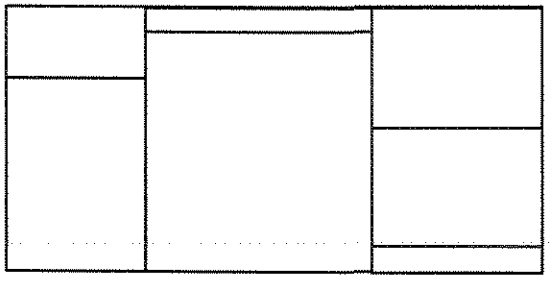


Figura 5.1: Teste da eficiência do sistema.

As dimensões estão organizadas na tabela abaixo:

Placa: 100 x 200			
Id	Largura	Comprimento	Quantidade
1	20	30	1
2	80	30	1
3	10	90	1
4	90	90	1
5	45	80	2
6	10	80	1

Tabela 5.1: Dimensões dos itens.

Dessa forma temos um padrão gerado. Se multiplicarmos a quantidade por uma constante, digamos 5 (cinco), temos que o resultado final deverá ser a utilização de 5 (cinco) placas. Após executar o algoritmo, obtivemos o seguinte resultado:

Quantidade de estágios	Quantidade de Placas
2	5

Tabela 5.2: Resultado do teste.

O padrão ótimo encontrado pelo sistema está representado pela figura a seguir.

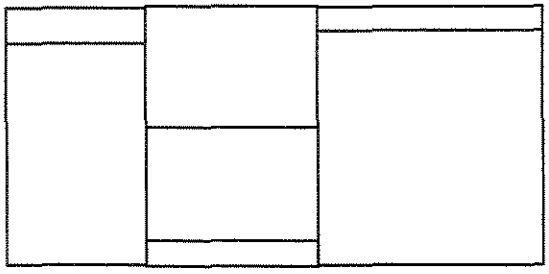


Figura 5.2: Padrão ótimo retornado.

Veja que o padrão encontrado não é exatamente igual ao que cortamos manualmente, mas demonstra ser algo bem próximo.

Confirmamos a validade do programa obtendo a solução ótima desta instância e de várias outras geradas da mesma forma. Assim, podemos continuar os testes computacionais com maior segurança[3].

5.3 Corte por Estágios x Corte por Níveis

É comum encontrar limitações na indústria quanto ao tipo de corte. Devido a este fato, o algoritmo de Herz foi modificado de modo a atender, além de corte por níveis, corte por estágios.

O algoritmo HFF trabalha com dois estágios e é comum encontrar na literatura abordagens tratando essa quantidade de estágios de corte. No sistema implementado, o estágio ou nível não fica limitado a um número, mas claramente quanto maior for o número de estágios/níveis, maior será a complexidade e conseqüentemente o tempo necessário para determinar o melhor padrão de corte também aumentará.

[3] Este é um teste dentre vários, não poderíamos garantir que o programa está correto com apenas uma instância.

Nesse estudo, foram testados para cada instância o corte por 2 estágios e por 2 e 3 níveis. A partir de 3 estágios e 4 níveis, o sistema se mostrou excessivamente lento, não retornando no tempo estipulado nenhum padrão de qualidade. Com o aumento de estágios ou níveis, o número de recursões aumenta exponencialmente, tornando a geração de um padrão muito mais lenta.

Ins	Tam	Dem	n	LB	HFF	NP 2 E	T 2 E	NP 2 N	T 2 N	NP 3 N	T 3 N
1	P	D	10	14	15	15	00:17	15	00:01	15	02:31
2	P	D	10	16	17	17	13:19	17	00:01	16	03:45
3	P	D	10	13	14	14	12:44	14	00:02	14	08:27
4	M	D	10	85	107	102	04:42	102	00:01	102	07:33
5	M	D	10	71	84	82	00:01	80	00:00	79	00:13
6	M	D	10	77	143	140	00:01	125	00:00	125	00:46
7	G	D	10	219	295	288	00:01	288	00:00	288	00:01
8	G	D	10	209	294	291	00:00	291	00:00	291	00:00
9	G	D	10	272	426	426	00:01	426	00:01	426	00:01
10	D	D	10	154	217	199	00:45	199	00:01	199	04:57
11	D	D	10	156	246	246	00:00	246	00:00	246	00:01
12	D	D	10	121	182	165	00:00	165	00:04	165	15:09

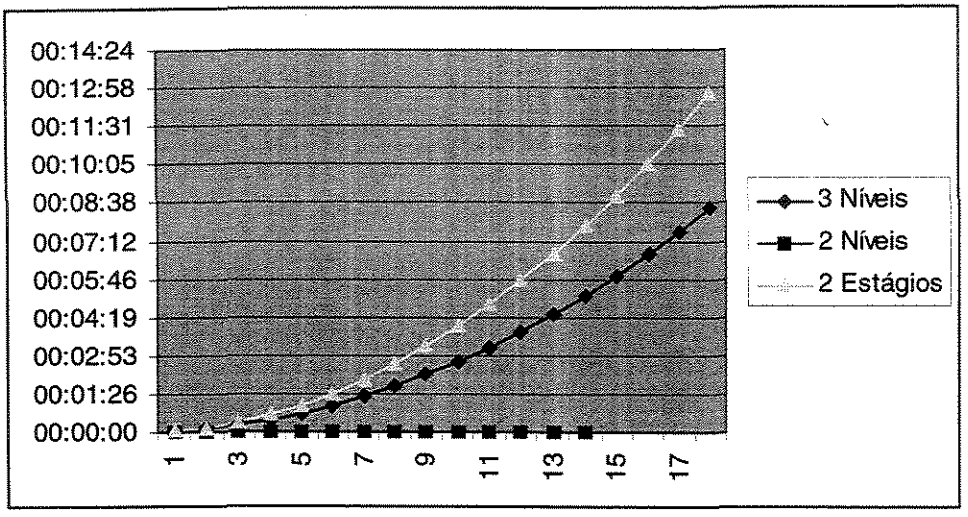
Tabela 5.3: Corte por estágios e por níveis.

Através dos resultados coletados, percebe-se que o tempo necessário para executar em 3 níveis é maior, contudo o resultado costuma ser melhor para itens pequenos. Esta opção será utilizada nos próximos estudos e uma forma de diminuir o tempo de execução será apresentada.

Na tabela podemos perceber uma variação muito grande de tempo que acontece para instâncias do mesmo tipo. Devido aos testes terem sido gerados pseudo-aleatoriamente, não há garantias que em alguns casos aconteça de gerar muitos itens pequenos em relação à porcentagem de tamanho estipulada em relação ao tamanho da placa.

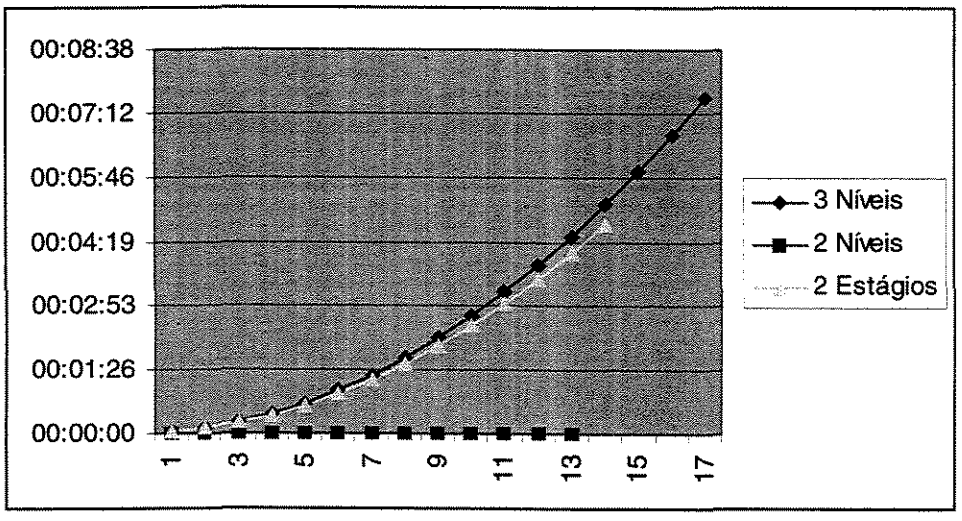
Para analisar a taxa de crescimento da complexidade em gerar padrões, geramos um gráfico das instâncias 3, 4 e 10 para todos os tipos de corte, escolhidas por possuírem maior tempo de execução nos testes executados e conseqüentemente para podermos analisar quanto foi o tempo necessário para gerar cada padrão e a respectiva quantidade.

Taxa de crescimento da Instância 3



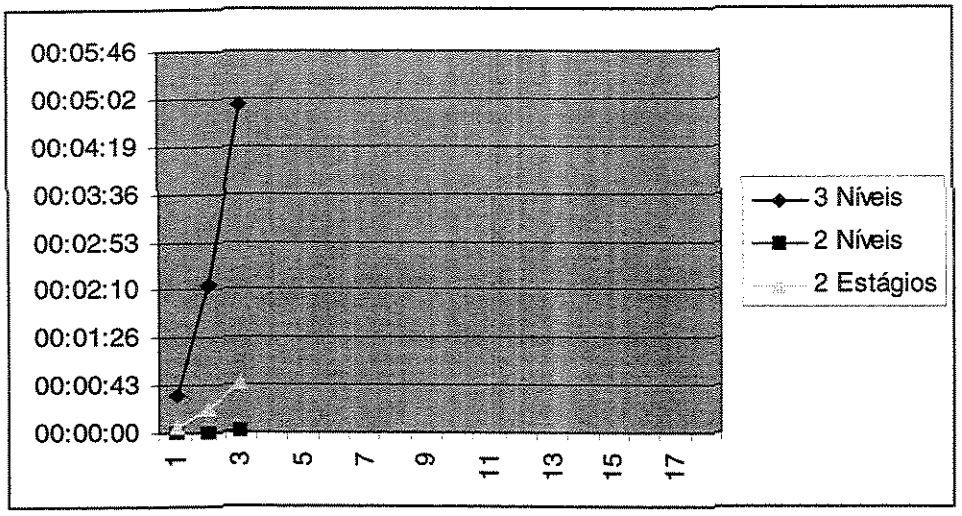
No eixo vertical temos o tempo que foi necessário para gerar os padrões e no eixo horizontal, a quantidade de padrões gerados. Podemos perceber que para 3 níveis e 2 estágios o programa gerou um número maior de padrões e conseqüentemente o tempo de execução foi maior.

Taxa de crescimento da Instância 4



Nesta instância a diferença entre o número de padrões gerados para 2 níveis e as outras opções de corte foi mantida, porém a taxa de crescimento da execução de 3 níveis foi maior.

Taxa de crescimento da Instância 10



Nesta instância, de itens diversificados, o programa gerou apenas 3 padrões, porém a dificuldade em gerar esses padrões cresceu muito rapidamente, praticamente dobrando para cada padrão.

5.4 Redução dos Pontos de Discretização

Conforme visto no estudo anterior, quanto maior o nível de corte, melhor poderá ser a solução. Infelizmente o tempo de execução cresce exponencialmente, mas para baixar esse tempo podemos reduzir o número de pontos de discretização, isto porque para cada ponto é realizada uma chamada recursiva no algoritmo de Herz e esses pontos podem estar muito juntos.

A redução aplicada nestes testes foi de fazer um espaçamento entre pontos de corte de 5% do tamanho da placa, isto é, para uma largura de 100 centímetros, os pontos devem ter pelo menos um espaçamento de 5 centímetros.

Ins	Tam	Dem	n	LB	HFF	NP Com Redução	T Com Redução	NP Sem Redução	T Sem Redução
1	D	B	5	12	13	13	00:09	13	20:20
2	D	B	5	9	12	11	00:03	11	00:15
3	D	B	5	10	13	12	00:00	12	00:05
4	D	M	5	70	113	113	00:00	113	00:01
5	D	M	5	65	95	95	00:00	95	00:00
6	D	M	5	60	69	69	00:00	69	00:02
7	D	G	5	177	268	247	00:00	247	00:00
8	D	G	5	80	130	103	00:00	97	00:00
9	D	G	5	115	175	160	00:00	160	00:00
10	D	B	15	38	48	46	00:07	46	08:25
11	D	B	15	50	64	60	02:25	60	10:03
12	D	B	15	72	107	105	00:51	103	06:29
13	D	M	15	134	206	206	00:08	198	03:56
14	D	M	15	217	271	254	03:52	254	16:38
15	D	M	15	145	203	192	00:37	192	19:35
16	D	G	15	317	432	409	03:54	403	04:35
17	D	G	15	459	676	635	15:41	638	15:31
18	D	G	15	275	354	332	16:50	341	15:18

Tabela 5.4: Estudo da redução dos pontos de discretização.

O resultado melhorou consideravelmente, apesar de que em alguns casos a solução se mostrou de pior qualidade.

Um fato interessante aconteceu neste teste. Analisando as duas últimas instâncias, notamos que apesar da redução dos pontos de discretização, a solução encontrada foi melhor que com todos os pontos. Acreditamos que nestes casos, a redução do número de pontos possibilitou percorrer por padrões com formas diferenciadas. No caso da geração de padrões sem redução dos pontos de corte gerou padrões muito próximos uns dos outros não percorrendo padrões melhores pela limitação no tempo.

Estaremos priorizando o tempo de execução nos próximos testes e por este motivo a redução continuará sendo aplicada.

5.5 Influência do tamanho dos itens

O objetivo deste estudo é verificar qual é a influência do tamanho dos itens em relação à solução gerada. Foram realizados testes para itens pequenos, médios, grandes e diversificados utilizando 3 níveis de corte, demanda diversificada e redução de apenas 5% nos pontos de discretização.

Itens Pequenos

A primeira instância, cujos testes computacionais são apresentados na tabela seguinte, conterá apenas itens pequenos:

Ins	n	HFF	AC	T AC	AB	T AB	LB	HFF/LB	AC/LB	AB/LB	T
1	5	16	20	01:43	15	01:43	13	1,23	1,54	1,15	01:43
2	5	5	8	20:51	5	20:51	5	1,00	1,60	1,00	20:51
3	5	11	16	04:47	11	04:47	10	1,10	1,60	1,10	04:47
4	10	12	17	21:04	12	21:23	11	1,09	1,55	1,09	21:23
5	10	17	22	19:23	17	19:25	16	1,06	1,38	1,06	19:25
6	10	9	10	19:20	9	19:24	8	1,13	1,25	1,13	19:24
7	15	22	29	15:42	22	15:44	21	1,05	1,38	1,05	15:44
8	15	28	38	15:19	28	15:19	26	1,08	1,46	1,08	15:19
9	15	19	26	18:12	19	18:15	18	1,06	1,44	1,06	18:15

Tabela 5.5: Estudo de itens pequenos.

Para os itens pequenos a solução obtida é muito próxima do limite inferior, isto ocorre porque os itens pequenos se encaixam facilmente, formando padrões melhores. Em contrapartida, a quantidade de padrões possíveis é grande e o tempo computacional necessário para gerar todos é alto.

Itens Médios

A segunda classe de problemas é para itens médios:

Ins	n	HFF	AC	T AC	AB	T AB	LB	HFF/LB	AC/LB	AB/LB	T
1	5	48	51	00:02	47	00:02	42	1,14	1,21	1,12	00:02
2	5	131	104	00:00	104	00:00	63	2,08	1,65	1,65	00:00
3	5	52	52	00:00	48	00:00	33	1,58	1,58	1,45	00:00
4	10	56	63	10:09	54	10:09	47	1,19	1,34	1,15	10:09
5	10	110	104	03:59	97	03:59	78	1,41	1,33	1,24	03:59
6	10	116	115	17:38	105	17:38	95	1,22	1,21	1,11	17:38
7	15	168	164	02:11	154	02:11	134	1,25	1,22	1,15	02:11
8	15	173	151	01:41	144	01:41	123	1,41	1,23	1,17	01:41
9	15	148	151	01:53	144	01:53	125	1,18	1,21	1,15	01:53

Tabela 5.6: Estudo de itens médios

Conforme esperado, o tempo necessário para geração dos padrões diminuiu e a distância da solução em relação ao limite inferior aumentou. Isso ocorre devido à impossibilidade de colocar dois itens que tenham mais que 50% em alguma das coordenadas em relação ao tamanho da placa.

Itens Grandes

Itens grandes limitam a quantidade de padrões possíveis e conseqüentemente diminuem o tempo computacional. Vejamos os resultados obtidos:

Ins	n	HFF	AC	T	AB	T	LB	HFF/LB	AC/LB	AB/LB	T
1	5	118	116	00:00	115	00:00	73	1,62	1,59	1,58	00:00
2	5	101	96	00:00	94	00:00	52	1,94	1,85	1,81	00:00
3	5	83	81	00:01	78	00:01	67	1,24	1,21	1,16	00:01
4	10	340	334	00:02	333	00:02	223	1,52	1,50	1,49	00:02
5	10	253	249	00:01	244	00:01	204	1,24	1,22	1,20	00:01
6	10	254	254	00:00	254	00:00	152	1,67	1,67	1,67	00:00
7	15	618	589	00:04	587	00:05	375	1,65	1,57	1,57	00:05
8	15	424	412	00:11	407	00:11	315	1,35	1,31	1,29	00:11
9	15	605	573	00:03	570	00:04	432	1,40	1,33	1,32	00:04

Tabela 5.7: Estudo de itens grandes.

Dificilmente mais de dois itens são empacotados em uma placa, o que torna a geração de padrões mais rápida e de menor qualidade.

Itens Diversificados

Itens diversificados devem aparecer mais freqüentemente na indústria e comportam todos os tamanhos estudados anteriormente.

Ins	n	HFF	AC	T	AB	T	LB	HFF/LB	AC/LB	AB/LB	T
1	5	61	57	00:01	52	00:01	42	1,45	1,36	1,24	00:01
2	5	149	150	00:00	149	00:00	115	1,30	1,30	1,30	00:00
3	5	113	117	00:00	113	00:00	76	1,49	1,54	1,49	00:00
4	10	138	129	01:38	123	01:38	98	1,41	1,32	1,26	01:38
5	10	227	226	02:24	224	02:24	156	1,46	1,45	1,44	02:24
6	10	139	121	06:43	114	06:43	90	1,54	1,34	1,27	06:43
7	15	410	377	01:16	369	01:16	273	1,50	1,38	1,35	01:16
8	15	286	287	05:59	277	05:59	214	1,34	1,34	1,29	05:59
9	15	394	360	14:00	348	14:00	306	1,29	1,18	1,14	14:00

Tabela 5.8: Estudo de itens diversificados

Os itens diversificados podem conter grande quantidade de itens grandes e assim comprometer a solução. Os que têm execução em tempo maior provavelmente contêm itens menores e conseqüentemente geram mais possibilidades de soluções. Em todos os testes a execução do algoritmo HFF não passou de 1 (um) segundo, tanto para a geração da solução inicial quanto para o tratamento do problema residual.

5.6 Técnicas de Arredondamento

Como vimos, é computacionalmente difícil resolver problemas de programação linear inteira com muitas colunas. Devido a essa dificuldade, relaxamos a condição de integralidade das variáveis e utilizar posteriormente outras técnicas para obtenção de soluções inteiras.

Utilizamos em nosso trabalho duas técnicas para gerar soluções inteiras e veremos agora computacionalmente a qualidade das soluções utilizando cada técnica de arredondamento.

A primeira técnica é fazer o arredondamento para cima e a segunda técnica é fazer o arredondamento para baixo e resolver o problema residual.

Conforme as tabelas 5.5 a 5.8 pode-se perceber que o tempo necessário para gerar uma solução inteira através de arredondamento é praticamente igual para ambas as técnicas. O

resultado, ao contrário, mostrou que o arredondamento para baixo e tratamento do problema residual fornece soluções de melhor qualidade.

5.7 A importância da geração do conjunto de soluções iniciais

Ao iniciar o problema de programação linear, geramos um conjunto inicial de padrões que seriam melhorados posteriormente pela técnica de geração de colunas. Veremos nesta seção a importância da utilização desta estratégia.

Ins	Tam	Dem	Nível	n	LB	HFF	NP Com HFF	T Com HFF	NP Sem HFF	T sem HFF
1	D	D	3	5	56	76	76	00:02	76	00:47
2	D	D	3	5	50	94	94	00:01	94	00:11
3	D	D	3	5	50	74	65	00:05	65	00:13
4	D	D	3	5	132	215	206	00:01	206	00:01
5	D	D	3	10	137	221	184	00:45	184	02:23
6	D	D	3	10	192	281	273	00:16	273	01:08
7	D	D	3	10	97	129	107	18:54	122	18:48
8	D	D	3	10	130	189	189	00:05	189	05:05
9	D	D	3	15	176	232	227	00:05	240	04:17
10	D	D	3	15	275	405	391	01:43	391	12:57
11	D	D	3	15	234	349	349	00:03	349	12:14
12	D	D	3	15	254	373	364	00:15	368	03:03

Tabela 5.10: Estudo da importância do conjunto inicial.

Os resultados comprovam que com a geração de um conjunto inicial de padrões, a solução converge mais rapidamente e, quando limitamos o tempo, a probabilidade de termos uma solução melhor é maior.

Conclusão e Trabalhos Futuros

Neste trabalho realizamos um estudo experimental utilizando técnicas de otimização para resolver com eficiência o problema do corte bidimensional. O estudo consistiu na leitura da abordagem proposta por Gilmore e Gomory para resolver problemas de corte e empacotamento através de programação linear inteira e o método de geração de colunas. Livros e artigos relacionados ao problema foram consultados com o objetivo de otimizar as diversas fases do processo de resolução, como a geração dos padrões de corte.

Percebemos através do estudo realizado que problemas de aparente simplicidade de formulação podem esconder uma alta complexidade na resolução, vimos na prática a explosão combinatória gerada pelas possíveis soluções para o problema e como pode ser impossível gerar todas *a priori*.

As técnicas de otimização que foram estudadas e aplicadas são utilizadas na resolução não apenas do problema do corte bidimensional, mas em diversos outros. Novas descobertas em determinados problemas podem contribuir muito para o estudo de problemas NP-Difíceis de diferentes áreas de conhecimento. É um campo da teoria da computação que ainda pode ser muito explorado de modo que novos algoritmos sejam desenvolvidos e os problemas sejam cada vez melhor resolvidos.

Com base na experiência adquirida através do estudo do problema e na implementação do sistema para resolvê-lo com eficiência, pretendemos futuramente realizar um estudo com outras etapas integradas ao processo de corte e empacotamento de placas, visto que o processo ainda pode envolver outras dificuldades como determinar a melhor seqüência de corte. Pode ser interessante ver o processo como um todo, onde as fases da resolução iriam compartilhar informações entre si, por exemplo, no momento da definição dos padrões de corte, procurar obter informações para depois utilizar no processo de determinação da melhor seqüência de corte.

Devido às diversas restrições que são impostas pela indústria, outra oportunidade de aprimoramento desse estudo é deixá-lo mais flexível para que novas restrições que venham a aparecer sejam facilmente integradas ao processo.

O sistema foi desenvolvido utilizando a metodologia orientada a objetos, o que futuramente pode nos permitir desenvolver componentes de otimização que combinem diversas técnicas, sendo facilmente adaptado para diversos problemas relacionados. Estes componentes seriam de domínio público e poderiam ser cada vez mais aprimorados, da mesma maneira que é feita atualmente com o sistema operacional linux.

Referências Bibliográficas

[1] J. C. Herz. *A recursive computational procedure for two-dimensional stock cutting*. IBM J. Res. Develop., páginas 462-469, 1972.

[2] F. R. K. Chung, M. R. Garey, e D. S. Johnson. *On packing two-dimensional bins*. SIAM J. ALG. DISC. METH., Vol 3, No. 1, 1982.

[3] M. R. Garey e D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, San Francisco, 1979.

[4] P. C. Gilmore e R. E. Gomory. 1961. *A Linear Programming Approach to the Cutting Stock Problem*. Opns Res. 9,849-859.

[5] P. C. Gilmore e R. E. Gomory. 1963. *A Linear Programming Approach to the Cutting Stock Problem – Part II*. Opns Res. 11,863-888.

[6] P. C. Gilmore e R. E. Gomory. 1965. *Multistage cutting Stock Problems of Two and More Dimensions*. Opns. Res. 13,94-120.

[7] H. Dyckhoff. 1981. *A New Linear Programming Approach to the Cutting Stock Problem*

[8] R. W. Haessler. 1979. *A Note on Computational Modifications to the Gilmore-Gomory Cutting Stock Algorithm.*

[9] G. B. Dantzig. 1963. *Linear Programming and Extensions*. Princeton, New Jersey: Princeton University Press.

[10] V. Chvátal. 1983. *Linear Programming*. W. H. Freeman and Company, New York.

[11] F. K. Miyazawa, *Algoritmos de Aproximação para Problemas de Empacotamento*, tese de doutorado, Universidade de São Paulo IME-USP, São Paulo-SP, Brasil, novembro 1997.

[12] K. C. Poldi, *Algumas extensões do problema de corte de estoque*, tese de mestrado, Universidade de São Paulo, São Carlos-SP, março 2003.

[13] G. F. Cintra, *Algoritmos Híbridos para Problemas de Corte Unidimensional*, dissertação de mestrado, Instituto de Matemática e Estatística da Universidade de São Paulo, 1998.

[14] Informações sobre o algoritmo LP_SOLVE. Disponível em: 29/04/1996. http://www.cs.sunysb.edu/~algorith/implement/lpsolve/implement.shtml. Acesso em: 20/10/2002.

[15] Informações sobre George Dantzig. Disponível em: 01/04/2003. http://www-gap.dcs.st-and.ac.uk/~history/Mathematicians/Dantzig_George.html . Acesso em: 10/06/2003.