Curso de Ciência da Computação
Universidade Estadual de Mato Grosso do Sul

Comparação entre duas metodologias de otimização para maximização do aproveitamento de área de chapas de vidro

Davi Pereira Rocha

Prof. Dr. Rubens Barbosa Filho (Orientador)

Curso de Ciência da Computação
Universidade Estadual de Mato Grosso do Sul

Dourados - MS
2014

Comparação entre duas metodologias de otimização para maximização do aproveitamento de área de chapas de vidro

Davi Pereira Rocha

Este exemplar corresponde à redação final da monografia da disciplina Projeto Final de Curso devidamente corrigida e defendida por Davi Pereira Rocha e aprovada pela Banca Examinadora, como parte dos requisitos para a obtenção do título de Bacharel em Ciência da Computação.

Dourados, 30 de novembro de 2014.

Prof. Dr. Rubens Barbosa Filho (Orientador)

A Deus que para mim é Pai, Filho e Espírito Santo, que me sustentou em todos os passos desta jornada, a Virgem Maria, Mãe de Deus que não cessou de olhar por mim, meus queridos pais, minha irmã e minha amada namorada que me apoiaram no cumprimento desta etapa.

“Foi para liberdade que Cristo nos libertou (Gl 5,1)”

Agradecimentos

A Deus e Nossa Senhora, ao meus queridos pais, Domingos Lira Rocha e Benedita Pereira Lúcio Rocha, pelo apoio e incentivo que deram durante a minha graduação e a minha irmã Janaína.

A minha amada namorada Liara que esteve comigo nos momentos tranquilos e nos momentos difíceis me apoiando durante todos estes anos de graduação.

Ao professor Rubens Barbosa Filho, pela disponibilidade e orientação neste trabalho e por deixar o exemplo de um grande profissional para minha vida.

Davi Pereira Rocha

Resumo

Este trabalho apresenta uma comparação entre duas metodologias para a maximização do aproveitamento de área de chapas de vidros. Existem uma ou várias peças para serem cortadas sobre uma chapa de vidro de um tamanho pré-determinado qualquer, qual é a melhor metodologia para se aplicar afim de obter o aproveitamento máximo da área da chapa de vidro? ou equivalentemente, maximizar o seu uso?

Uma das metodologias que foi comparada é definida em três regras básicas e três estratégias de otimização e é denominada de "Heurística *O(mn)* para o Corte Guilhotinado Bidimensional".

A outra metodologia, é baseada em Programação Dinâmica, uma técnica de programação que se aplica a problemas que possuam estrutura recursiva. Quando a recursão é aplicada para a resolução de um problema, geralmente cálculos são feitos repetidamente.

A Programação Dinâmica quando faz um determinado cálculo na recursão, armazena este resultado para um eventual uso futuro, economizando assim o tempo utilizado para calculá-lo novamente.

Para a comparação entre as duas metodologias foram elaboradas dez instâncias de teste. Cada instância possui um determinado número de peças, sendo que não há repetições de peças dentro da mesma instância.

As peças de cada instância foram geradas aleatoriamente e são sempre menores que a chapa em que foi aplicada as metodologias. As instâncias de teste foram aplicadas em quatro chapas de áreas diferentes.

O trabalho apresenta tabelas e gráficos com o resultado da aplicação de cada metodologia, sendo possível observar o desempenho que as duas metodologias alcançaram em relação a cada instância de teste.

Para a coleta dos resultados da aplicação da metodologia "Heurística O(mn) para o Corte Guilhotinado Bidimensional foi usado um programa desenvolvido por [1] e seu executável está disponível em [5].

Para a coleta dos dados para o Corte Bidimensional com Programação Dinâmica foi implementado um programa com a metodologia.

Palavra-Chave: Corte Bidimensional; Heurística O(mn) para o Corte Guilhotinado Bidimensional; Programação Dinâmica; Chapa de Vidro; Otimização.

Sumário

Lista de figuras

Lista de tabelas

Capítulo 1

1.1 Introdução

Um problema clássico que as empresas que trabalham com o corte de vidro enfrentam é o desperdício. O tipo de corte aplicado neste material, é conhecido por corte bidimensional, que é uma generalização do problema do corte unidimensional[1].

Supõe-se que uma loja qualquer venda o m² de um certo tipo de vidro por 360,00 R$. Supõe-se também que por dia uma empresa qualquer desperdiça 10% desse material, que é comprado da loja citada acima.

Levando em conta o preço que a empresa compra o material, em 30 dias a empresa teria um desperdício em dinheiro de 1080,00 R$. Em 365 dias teria um desperdício de 13140,00 R$ e em 10 anos, um prejuízo enorme de 131400,00 R$.

Uma solução que resolveria este problema seria um planejamento antecipado mostrando as várias combinações de corte possível, de forma a permitir que o administrador faça a melhor escolha possível no momento do corte desse vidro.

Assim, minimizando a quantidade de material utilizado, ou equivalentemente, maximizando o seu aproveitamento, uma economia seja feita, tanto em vidro quanto em dinheiro.

Foram duas as metodologias adotadas para solução do problema. A primeira, é uma técnica de programação denominada Programação Dinâmica que o resultado final de sua aplicação é a melhor forma de como escolher quais peças de vidro obtém a maximização de uso de determinada chapa para corte.

A Programação Dinâmica, se aplica a problemas que tenham estrutura recursiva e a solução da instância total do problema depende das soluções de instâncias menores do mesmo[2]. Ou seja, a instância total contém subproblemas que solucionados constroem a solução do problema.

Para o problema do Corte Bidimensional a instância total do problema é a resposta de quais peças serão cortadas sobre uma determinada chapa de vidro que maximize o seu uso e as instâncias menores são as possibilidades de inserir ou não inserir cada peça.

A grande vantagem da Programação Dinâmica é que ela armazena as soluções das várias instâncias menores no primeiro instante de seu cálculo, aproveitando assim o resultado armazenado nas operações futuras, ao invés de recalculá-las como no caso que geralmente a recursão faz. Assim sendo, há uma economia grande de operações e tempo.

A segunda metodologia para a solução do problema é composta por três regras, acompanhadas de três estratégias de otimização, como afirmam seus autores em [1]. Esta tecnologia tem nome de: "Heurística *O(mn)* Para o Corte Guilhotinado Bidimensional".

Para a comparação entre as metodologias foram criadas dez instâncias de teste. Estas bases são compostas por peças a serem cortadas em uma chapa de corte. Cada peça é composta por largura e altura no formato *largura x altura*.

Em cada instância de teste não existem peças repetidas e as mesmas são sempre menores que a chapa em que a metodologia vai ser aplicada. As peças foram geradas de forma aleatória para cada instância de teste.

As instâncias de teste elaboradas para os testes entre as duas metodologias estão localizadas nos CDs entregues juntamente com o trabalho.

O trabalho apresenta os resultados da aplicação de cada metodologia com tabelas e gráficos, podendo visualizar o desempenho que cada metodologia alcançou para cada instância de teste.

O objetivo deste trabalho é realizar uma comparação de desempenho entre duas metodologias de corte bidimensional de vidro. A primeira metodologia usa Programação Dinâmica e o segunda usa uma heurística baseada em três regras gerais e acompanhadas por três estratégias de otimização. Além de contribuir com uma informação válida sobre qual metodologia obtém um melhor desempenho neste tipo de problema.

O trabalho está organizado por capítulos. No capítulo dois é abordado a técnica da Programação Dinâmica e como ela se aplica ao Corte Bidimensional. No capítulo três, é abordado a metodologia "Heurística *O(mn)* para o Corte Guilhotinado Bidimensional".

O capítulo quatro apresentará a metodologia empregada e os resultados da aplicação dos métodos em conjunto com a base de dados e o capítulo cinco as conclusões em relação ao desempenho das duas metodologias e sugestões para trabalhos futuros e um levantamento do trabalho ao todo.

Capítulo 2

2.1 Programação Dinâmica e suas características

A Programação Dinâmica se aplica a problemas de otimização em que escolhas devem ser feitas, a fim de alcançar uma solução ótima[2]. Para conseguir imaginar a economia da aplicação da Programação Dinâmica, ela pode em transformar facilmente algoritmos de tempo exponencial em algoritmos de tempo polinomial[2].

Assim sendo, a Programação Dinâmica se encaixa muito bem como uma solução para o problema do Corte Bidimensional, que precisa decidir que peças serão cortadas sobre uma determinada chapa de vidro para maximizar o uso da mesma.

A Programação Dinâmica resolve os problemas combinando as soluções de subproblemas. A mesma se aplica quando estes subproblemas não são independentes, ou seja, os subproblemas compartilham de subproblemas menores ainda[2].

Na Programação Dinâmica, a solução ótima para o problema é construída partir de soluções ótimas para os seus subproblemas. Ou seja, a instância total do problema consiste em instâncias menores que primeiramente solucionadas e depois combinadas é construída a solução para instância maior do problema.

O ponto destaque que permite economizar espaço da Programação Dinâmica é a economia que se obtém ao aplica-la. No momento em que ela resolve um subproblema, ela armazena essa solução para que em um eventual uso futuro, ela não recalcule este subproblema, mas somente consulte a sua solução no lugar em que estão armazenados os resultados dos subproblemas.

O termo “programação”, da Programação Dinâmica não tem relação a programação de computadores. A palavra programação significa um certo planejamento e refere-se à construção da tabela que armazena as soluções dos subproblemas[4].

“O desenvolvimento de um algoritmo de Programação Dinâmica pode ser desmembrado em quatro etapas[2]”. A primeira etapa para a construção de um algoritmo de Programação Dinâmica consiste em caracterizar uma estrutura de uma solução ótima para o problema.

Essa estrutura de solução ótima é o raciocínio que deve ser seguido para a resolução de cada subproblema. Ou seja, para a resolução dos subproblemas deve-se seguir um raciocínio que conduz a solução ótima.

A segunda é definir recursivamente o valor de uma solução ótima. A terceira é calcular o valor de uma solução ótima em um processo de baixo para cima e por fim construir uma solução ótima a partir das informações calculadas.

Definir o valor recursivamente significa estabelecer a fórmula matemática que será usada na recursão. Calcular o valor de uma solução ótima de “baixo para cima” significa a resolução do menor subproblema até chegar a resolução do problema total através de combinação dos subproblemas.

Construir uma solução ótima a partir das informações calculadas significa consultar as informações calculadas afim de construir a solução para um determinado problema.

A base da solução de um problema usando Programação Dinâmica é composta pelas etapas um e três e a etapa quatro pode ser omitida se apenas o valor de uma solução ótima é exigido[2].

Isso significa que em determinando problemas que são resolvidos com Programação Dinâmica, no fim de sua aplicação não é preciso construir a solução a partir dos resultados armazenados mas somente uma consulta na última solução armazenada por exemplo.

A fim de facilitar a construção de uma solução ótima, no momento da execução da etapa quatro, são mantidas informações adicionais obtidas durante a computação na etapa três[2].

Isso ocorre quando a solução de um subproblema é usada na solução do problema “acima” dele.

Um exemplo clássico onde pode-se visualizar um problema onde a técnica de Programação Dinâmica se encaixa é um problema conhecido como sequência de Fibonacci[6].

A sequência de Fibonacci é definida por uma sequência numérica observada na reprodução de coelhos[6]. A figura com a sequência é apresentada abaixo:

Figura 1 – Sequência de Fibonacci[7].

A variável *i* corresponde ao índice da sequência. Por exemplo *i* = 2 resultará no segundo número 1, pois a contagem é iniciada no índice 0.

Para a resolução da sequência de Fibonacci, sabe-se que o *i-esimo* item da sequência (para i maior que 1) é resultante da soma dos seus dois antecessores da sequência.

Para a construção da sequência podemos utilizar um algoritmo recursivo como é descrito na figura abaixo:

```
fib(n):
    Se n=0 ou n=1 então
        retornar n
    Senão
        retornar fib(n-1) + fib(n-2)
```

Figura 2 – Algoritmo recursivo de Fibonacci[7]

Na figura 1 o algoritmo recursivo pode ser visto na chamada fib(n-1) + fib(n-2). Os pontos negativos desta implementação é que no modo recursivo de resolver o problema, existem cálculos que são feitos repetidamente, como podemos observar na árvore de cálculos:

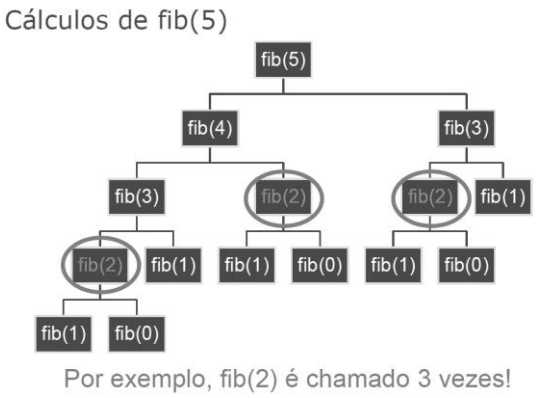

Figura 3 – Demonstração dos cálculos na recursão de Fibonacci[7]

Na Figura 3, existem os cálculos sendo feitos repetidamente. Fib(0) é chamado 3 vezes, fib(1) é chamado 5 vezes, fib(3) é chamado 2 vezes e fib(2) é chamado 3 vezes.

Nestes tipos de problema a Programação Dinâmica tem boa aplicabilidade, pois existem cálculos repetitivos onde a tecnologia armazena o resultado destes cálculos na primeira vez em que são calculados para um eventual uso futuro.

Sabe-se que o *i-esimo* (i como índice da sequência) item da sequência (para i maior que 1) é resultante da soma dos seus dois antecessores da sequência. Assim sendo, podemos extrair o raciocínio que vai caracterizar a estrutura de solução ótima para o problema dos cálculos repetitivos.

A estrutura de solução ótima para Fibonacci é o armazenamento dos dois últimos elementos de cada *i-ésimo* número da sequência. Assim, não são feitos cálculos repetitivos como no caso da recursão. Somente é feita uma consulta das duas soluções anteriores.

Pode-se imaginar que cada número da sequência é um subproblema e a sua solução é a soma da resolução dos dois subproblemas anteriores. Então indiretamente para chegar no *i-ésimo* número desejado é preciso o cálculo de todos os subproblemas anteriores para *i* maior que 1.

Pode-se armazenar estes valores em uma estrutura de dados simples como um vetor. A segunda etapa informa que devemos definir recursivamente o valor da solução ótima, ou seja, definir matematicamente como vai ser feita a recursão.

No caso de Fibonacci não é preciso uma recursão em termos de “uma função chamar ela mesma”, mas recursão no sentido que cada número da sequência irá usar o mesmo recurso, o de consultar a solução dos dois subproblemas anteriores.

Como exemplo um vetor *v* e o seu índice *i* a ser calculado. Matematicamente o recurso utilizado para cada número será $v[i] = v[i-1] + v[i-2]$. Ou seja, o *i*-ésimo item é sempre calculado através dos números contidos nos dois índices anteriores para *i* maior que 1.

A etapa três, define que é preciso calcular os valores de baixo para cima. Ela é executada desde a primeira vez que o recurso é utilizado por um subproblema. No caso de Fibonacci, o menor subproblema é solucionado quando o *i-ésimo* item da sequência é o 2. Pois é o primeiro utilizar o recurso. Quando o *i-ésimo* número a se saber da sequência é zero ou um, ele simplesmente é retornado propriamente, então não usa o recurso.

Assim sendo, cada *i-ésimo* item após o 2 é resolvido através da solução dos dois subproblemas anteriores. Por isso o termo de “baixo para cima”. Ou seja do menor subproblema para o maior.

A etapa quatro define que deve-se construir a solução ótima através dos resultados obtidos. Nesse caso ela é omitida para o problema geral, pois ela não usa diretamente de todos os resultados obtidos da sequência mas usa somente os dois “anteriores”. Ela é executada na construção da solução de cada subproblema. Ou seja, combinando as soluções dos dois subproblemas anteriores eu construo a solução para o *i-ésimo* item.

É importante ressaltar o objetivo que você quer alcançar com a programação dinâmica, esta técnica é geralmente utilizada para economizar. No caso de Fibonacci ela foi utilizada para economizar cálculos e consequentemente tempo.

Nem todos os autores seguem as quatro etapas sugeridas por [2]. Como exemplo o autor [8], para determinados problemas nem cita as etapas. Para determinados problemas [8] usa outro tipo de estratégia para a aplicação da Programação Dinâmica, usa hipótese de indução.

Mas as características da Programação Dinâmica de guardar as soluções dos subproblemas para um eventual uso futuro e os subproblemas compartilharem do uso do mesmo recurso para serem solucionados são algumas das características que devem ser mantidas.

A Programação Dinâmica é um nome fantasia para a recursão apoiada com uma tabela para consulta. Ao invés de resolver os problemas de forma recursiva, onde geralmente são feitos cálculos repetitivos ou que a recursão vai acarretar em algum desperdício de operações, resolve-se os subproblemas de forma sequencial e armazena suas soluções em alguma estrutura de dados que é chamada de tabela[9].

O truque é resolvê-los na ordem correta para que sempre que é necessária a solução para um subproblema, a mesma já esteja disponível na tabela[9]. Assim, os cálculos são realizados sem repetições e somente consultas são necessárias se o cálculo já tiver sido feito.

2.2 Programação Dinâmica para inserção de peças na área de uma chapa de vidro

O problema do Corte Bidimensional solucionado com Programação Dinâmica que é apresentado neste trabalho, consiste em decidir quais peças com prioridade de corte de um determinado conjunto das mesmas, devem ser inseridas para corte em uma chapa de vidro afim que se obtenha a maximização do uso de área da chapa.

O problema pode ser formulado da seguinte forma: Dada uma área *K* de uma chapa de vidro e *n* peças de diferentes áreas com prioridades para corte. De tal forma que cada peça tenha uma área *Ki* e uma prioridade *Vi*. Encontrar um subconjunto de peças cujas áreas maximizem o uso da chapa e as peças de maior prioridade sejam cortadas primeiro.

A prioridade de corte é inserida em cada peça, visando o problema real que uma vidraçaria ou instituição que trabalha com o corte de vidro enfrentam no dia a dia.

A entrada do problema é o conjunto de peças desejadas para corte (*n*), suas respectivas prioridades e a área da chapa em que as peças serão inseridas (*K*). A saída será o subconjunto de peças que está contido no conjunto inicial de peças da entrada que maximiza o uso da chapa de área *K*.

Para resolve-lo com a Programação Dinâmica foi utilizado hipótese de indução. Vamos designar o problema do Corte Bidimensional por P*(n, K)* de modo que *n* indica o número de peças e *K* indica a área da chapa em que as peças serão inseridas.

Cada peça do conjunto de entrada pode ser considerada como um subproblema. E a solução ótima para o subproblema consiste na maximização alcançada por duas hipóteses. A primeira hipótese é determinada como a peça dentro do conjunto solução. E a segunda, se ela estiver fora do conjunto solução.

Deve-se levar em conta o tamanho *Ki* (área da *i-ésima* peça) de cada peça. Pois se *Ki* for maior que *K* (área da chapa) ela não pode ser considerada no conjunto solução. Pois não existe forma de inserir a área de *Ki* em *K*. Assim, P*(i, K)* indica o problema com as *i* primeiras peças (*i* sendo como índice para as peças).

Se a área da *i-ésima* peça (*Ki*) for maior que a área da chapa (*K*), a função do problema deve ser considerada sem a determinada peça. A função do problema é estabelecida como P(*i, K*) = P(*i-1, K*). Ou seja, o *K* (área da chapa) vai ser mantida pelo fato que a peça não vai ser inserida na chapa e a *i-ésima* peça vai ser desconsiderada quando (*i-1*) for calculado.

Se a área da *i-ésima* peça (*Ki*) for menor que a área da chapa (*K*), ela pode estar ou não no conjunto de solução ótima. E é definido por hipótese que a solução ótima é o maior valor obtido com a peça dentro do conjunto solução ou fora do conjunto solução.

A função do problema é estabelecida dessa forma se área da *i-ésima* peça for menor que a área da chapa:

$$P(i, K) = \text{MAX } \{P(i - 1, K - Ki) + Vi \text{ ou } P(i - 1, K)\}$$

Ou seja, a função consultará e somará a prioridade da *i-ésima* peça e armazenará o maior valor obtido através das soluções das hipóteses. Na primeira hipótese, a *i-ésima* peça é inserida na chapa, P(*i – 1, K – Ki*) + *Vi,* note a diminuição do total de área disponível (*K – Ki*). E na segunda hipótese a área será mantida e serão consideradas as *i – 1* peças. Ou seja, sem a *i-ésima* peça no conjunto solução.

Cada resultado da consulta vai ser armazenada em uma tabela (matriz), onde estão as soluções ótimas anteriores armazenadas. Como a programação dinâmica divide o problema em subproblemas menores, a instância menor do problema é caracterizada pela capacidade que uma chapa de área 0 com nenhuma peça a ser inserida. Como sabe-se uma chapa de área 0 é nula então todos os seus resultados também serão nulos.

Aplicando a hipótese que é definida para alcançar a solução ótima para cada subproblema sucessivamente (área 0 com peça 1, área 1 com peça 1, área 2 com peça 1), são geradas e armazenadas as soluções ótimas, para que os subproblemas mas a "frente" como (área 2 com peça 2) possa consultar a solução ótima do problema anterior, ou seja, qual é a solução ótima com a área 2 e peça 2 é com a peça 1 no conjunto ou não?

Os subproblemas são solucionados utilizando sempre o mesmo recurso. Até se chegar na solução desejada que é capacidade da mochila (coluna da tabela) e a quantidade de itens (linhas da tabela).

2.3 Exemplo da aplicação da Programação Dinâmica para inserção de peças na área de uma chapa

Tem-se quatro peças para serem cortadas e com determinadas prioridades e áreas. As peças são denominadas em x_1, x_2, x_3 e x_4; x_1 tem prioridade 10 de corte e área de 2m²; x_2 tem prioridade 7 e área de 1m²; x_3 tem prioridade tem prioridade 25 e área de 6m² e x_4 com prioridade 24 e área de 5m². A chapa de corte possui capacidade de 7 m². Qual é o valor de prioridade alcançada e as peças que a constituem este valor?

Os valores armazenados na tabela são as prioridades de corte de cada peça ou a soma de prioridades de várias peças que determinam a maximização de área da chapa. Como temos a função para o problema como P(*i*, *K*) ela é um índice também para a tabela (Figura 4). Por exemplo qual é o a solução para o subproblema P(2, 4)? Ou seja, duas peças com uma chapa de área 4? São duas peças que a soma de suas prioridades é 17.

A soma das áreas dessas peças será o valor de área alcançado. Para se saber quais peças geraram determinada soma de prioridades, basta aplicar novamente as hipóteses na linha da peça. O resultado pode vir do índice P(i – 1, K) ou de MAX {P(*i – 1, K – Ki*) + *Vi ou* P(*i – 1*, *K*). Os dois valores referentes a estes índices são 10 implicando que peça 1 (prioridade 10) está inserida no conjunto juntamente com a peça 2 (prioridade 7).

Assim sendo, o exemplo pede o valor de prioridade da chapa com uma área de 7 m² e 4 peças denominadas em x_1, x_2, x_3 e x_4. A resposta está no índice da Figura 4 abaixo no índice P(*i*, *K*) = P(4, 7) = 34. Ou seja, a soma das prioridades de peças que maximizam o uso da chapa resulta em 34.

Para saber quais peças resultaram este valor vamos aplicar as funções das hipóteses. O índice P(*i – 1, K*) informa uma solução ótima com 32 de prioridade. E o índice P(*i – 1, K – Ki*) informa 10. Somando a prioridade (*Vi*) da peça 4 a 10 resulta o valor da solução ótima 34. Se somarmos 24 que é o valor da prioridade da peça 4 a 32 não resulta 34 então sabe-se que o resultado não vem da primeira hipótese.

Se o resultado não vem da primeira hipótese ele então vem da segunda. De onde vem o valor 10? P(*i – 1, K – Ki*) = P(3, 2). Das 3 primeiras peças a única de prioridade 10 que consegue maximizar o uso da mochila é a peça x_1. Então conclui-se que as peças que resultaram o número de prioridade 34 são as peças x_1 e x_4; x_1 possui área de 2cm² e x_4 possui área de 5cm², somando as duas área resulta em 7cm².

	0	1	2	3	4	5	6	7	K
0	0	0	0	0	0	0	0	0	
1	0	0	10	10	10	10	10	10	
2	0	7	10	17	17	17	17	17	
3	0	7	10	17	17	17	25	32	
4	0	7	10	17	17	24	31	34	
i									

Figura 4 – Tabela de resultados Programação Dinâmica aplicada a área da chapa do Corte Bidimensional

A figura 4 destaca a solução do exemplo, índice P(4, 7). Com as prioridades acumuladas das peças que devem ser cortadas para maximização do uso da chapa. No capítulo 3 o assunto abordado é a outra metodologia a ser comparada com a Programação Dinâmica.

Capítulo 3

3.1 Heurística O(*mn*) para o Corte Bidimensional Guilhotinado

3.1.1 A heurística

"As heurísticas ou algoritmos heurísticos foram desenvolvidos com a finalidade de se resolver problemas de elevado nível de complexidade em tempo computacional razoável" [3]. A partir disso, pode-se dizer que uma heurística nada mais é que um método ou processo que resolve problemas e sua finalidade é alcançar uma economia. Descreve-se uma heurística para uma otimização para o Corte Guilhotinado Bidimensional.

A Heurística O(*mn*) aplicada ao Corte Guilhotinado Bidimensional é executada através de três regras: a primeira determina que deve-se cortar sempre a maior *peça* cuja a dimensão caiba no menor *pedaço* disponível; a segunda determina que deve-se cortar uma *peça* sempre no canto inferior esquerdo do menor *pedaço* disponível[1].

Um *pedaço* é qualquer chapa disponível para corte que a heurística será aplicada e *peça* são as peças que deseja-se cortar sobre a chapa (*pedaço*). Caso o menor pedaço (*chapa de corte*) não suportar nenhuma *peça*, ele é considerado uma sobra e é descartado.

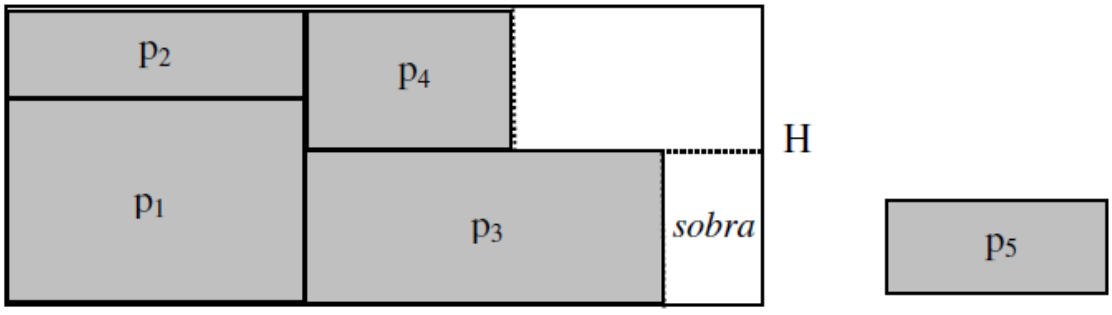

Figura 5 – Esquema mostrando um pedaço que se tornou sobra e irá ser descartado[1]

Existe ainda um pedido *P5* para ser atendido. O menor pedaço denominado "sobra" na imagem, não o suporta então o pedaço "sobra" é descartado.

Dois tipos de cortes são possíveis: principal vertical ou principal horizontal. Após o corte há a possibilidade de surgirem novos *pedaços*. A última regra consiste em escolher o tipo de corte (vertical ou horizontal) a ser aplicado conforme três estratégias, são elas: maximizar o menor pedaço restante; maximizar o maior pedaço restante; e maximizar o menor pedaço quanto a largura e a altura[1].

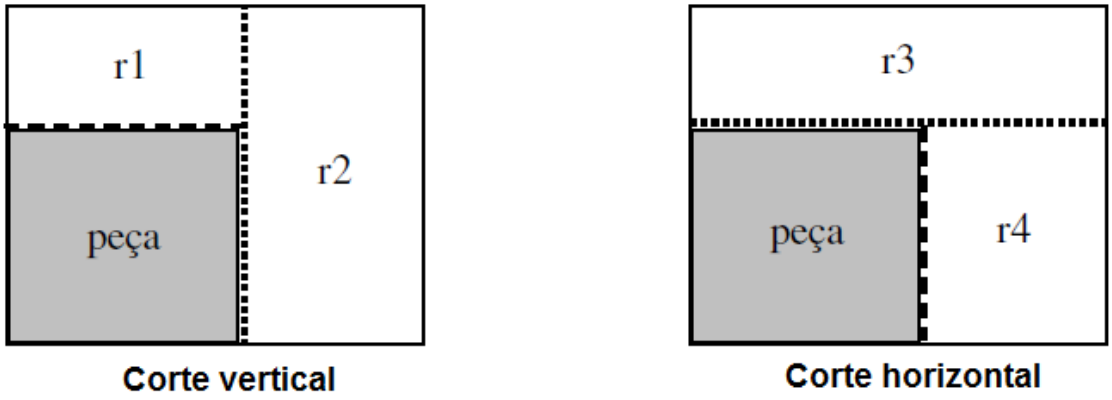

Figura 6 – Exemplo de tipos de corte [1]

O corte vertical é visto mais à esquerda na imagem. Depois de aplicado os cortes, dependendo podem gerar novos pedaços como *r1* e *r2*.

Maximizar o maior pedaço significa escolher o corte que resultará no maior pedaço. Ou seja, aplica-se uma simulação dos cortes vertical e horizontal com peça normal e a peça rotacionada.

Faz-se uma verificação de qual corte resultou o maior pedaço. A partir disso é escolhido o tipo de corte que vai ser aplicado para determinada *peça*. Maximizar o menor pedaço é escolher o corte que maximiza o menor pedaço resultante dos cortes.

Verifica-se os menores pedaços resultantes da aplicação dos cortes. Verifica qual é o menor pedaço maior e aplica o corte que o resultará. A terceira estratégia de otimização que maximiza o menor pedaço quanto a altura e a largura não foi abordada pelo fato que não foi usada nos testes de comparação.

Essas três estratégias de otimização tem o objetivo de que não sobrem pedaços com áreas muito pequenas[1]. Também devemos considerar que a orientação das peças também pode ser considerada no momento da escolha do corte, caso seja possível a rotação das peças.

O resultado da aplicação destas regras é a definição de um *layout* contendo as posições de como cortar peças retangulares sobre uma chapa também retangular[1].

Para a geração de um layout através da heurística precisa-se de uma lista de pedaços (*chapas de corte*) para atender os pedidos (*peças para corte*) que são peças menores. Inicialmente a lista de *pedaços* contém um único pedaço de dimensão igual à da chapa de vidro que o usuário da heurística quer aplicar a otimização.

A lista de pedaços é mantida ordenada do menor para o maior pedaço até o fim da heurística. Se o pedaço do topo da lista, ou seja o menor pedaço, não suportar nenhum pedido, ele é considerado como uma sobra e é descartado.

A repetição das três regras da heurística é repetida até que não haja mais pedidos para serem atendidos. Caso existam muitos pedidos e os pedaços acabarem, a lista de pedaços é reiniciada com um pedaço igual ao da entrada (chapa inicial) neste momento, um layout está completo e outro layout é iniciado.

Quando a lista de pedaços é reiniciada há a possibilidade de ocorrer um problema, pois se existir algum pedido de dimensão maior que a chapa inicial, este nunca poderá ser atendido, pois nem o maior pedaço disponível o suportará.

A lista de pedaços é uma lista que contém as coordenadas do tipo (x_1, y_1, x_2, y_2) que determinam a diagonal inferior (x_1, y_1) e a diagonal superior (x_2, y_2) de qualquer pedaço da lista. Essas coordenadas são usadas no momento de determinar a posição de uma peça para o corte na chapa e estabelecer as coordenadas dos pedaços restantes[1].

3.1.2 Exemplo da aplicação da Heurística O(mn) para o Corte Guilhotinado Bidimensional

Como exemplo, existe uma lista constituída de 7 peças para corte e uma lista de pedaços disponíveis constituída inicialmente de um único *pedaço* em que será aplicada a metodologia da heurística O(*mn*) abordada neste capítulo.

A maior peça da lista é a peça de número 7. Como é a maior peça da lista ela é retirada da mesma para ser cortada se possível no único pedaço disponível na lista de pedaços.

Se for possível o corte da peça sobre o pedaço é simulada uma aplicação de cada tipo de corte (vertical ou horizontal) com a peça normal e com a peça rotacionada. Após isso, dependo da estratégia escolhida (maximizar o maior pedaço ou maximizar o menor pedaço) é escolhido o tipo de corte a ser aplicado.

Depois que o corte escolhido for aplicado podem ser gerados novos *pedaços* para corte. Esses, vão para a lista de pedaços. E as regras da heurísticas se repetem até que não existam mais peças para corte.

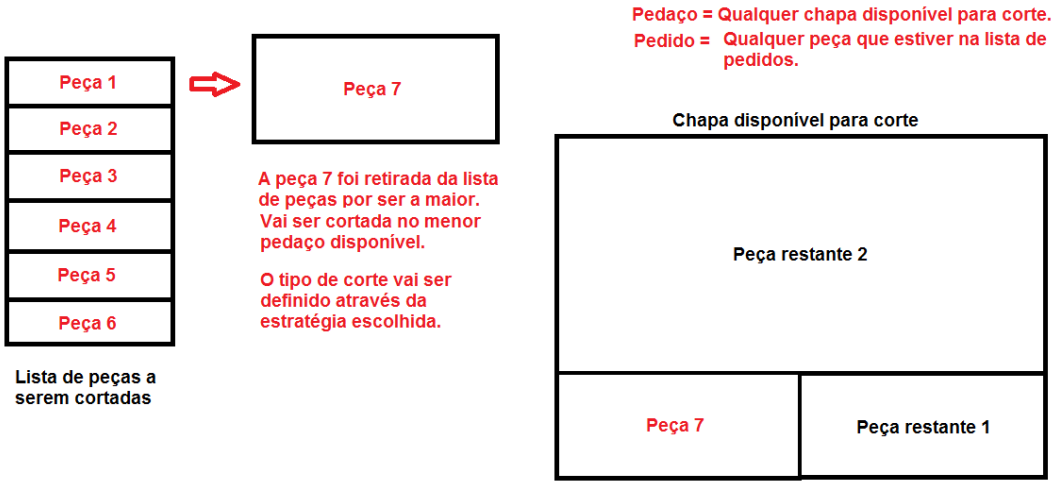

Figura 7 – Esquema do exemplo da Heurística O(mn) para o Corte Guilhotinado Bidimensional

Note que a partir do corte escolhido foram gerados dois novos pedaços (Figura 7). Estes são enviados para a lista de *pedaços* disponíveis para corte. Nesse caso, o corte determinado foi o horizontal com a peça sem rotação.

3.1.3 Análise da complexidade

A complexidade de espaço que deve ser levada em conta é no momento que devemos encontrar a maior peça que caiba no menor pedaço que é O(*m*), onde *m* é a quantidade de pedidos. Mas a complexidade total depende da quantidade de elementos da lista de pedaços[1].

Para cada pedaço da lista são duas operações possíveis: a primeira é o descarte do pedaço se ele for considerado sobra; ou o pedaço é retirado da lista, a fim de permitir o posicionamento de uma peça, e a inserção de duas novas áreas de pedaços (a_2 e a_3) na lista, o que faz com que a lista de áreas de pedaços cresça linearmente[1].

Além disso, a cada operação com um pedaço, ele é descartado ou um pedido é removido, assim a lista de pedaços cresce no máximo, $n+1$ elementos, onde n é o número total de peças inicialmente[1].

No pior caso, são feitas n iterações para posicionar todas as peças (pedidos), mais $n+1$ iterações para analisar algumas áreas de pedaços restantes. O que implica em uma complexidade $O(n)$[1].

Deve existir um laço mais externo que garanta caso não seja mais possível posicionar todas as peças em uma única chapa, o processo seja reiniciado para uma nova chapa[1].

A complexidade total é então definida pela complexidade $O(n)$ multiplicado pela complexidade da procura pelo maior pedaço, resultando $O(mn)$. No caso de cada pedido ser composto de uma única peça ($q_i = 1$, *para* $i = 1,2, ..., m$), temos $n=m$, o que implica em uma complexidade $O(n^2)$[1].

A heurística proposta em [1] possui uma estrutura recursiva. A recursividade decorre do fato de cada pedaço obtido a partir do corte uma peça sobre uma chapa inicial pode ser analisado com uma nova chapa de tamanho menor. Podendo assim implementar também uma versão recursiva da heurística.

A complexidade da heurística é O(mn), onde n é a quantidade de peças especificadas em uma lista de m pedidos, sendo pois bastante eficiente para os problemas práticos de cortes[1].

Capítulo 4

4.1 Metodologia e análise de resultados

As duas metodologias de otimização estudadas neste trabalho visam comparar o espaço ocupado de uma determinada chapa de vidro. Para as duas metodologias de otimização foram construídas dez instâncias de teste onde foram coletados os resultados e comparados.

As instâncias de testes foram construídas de forma que cada instância possui um determinado número de peças geradas aleatoriamente e sem repetições. Ou seja, não existem repetições de peças dentro de uma mesma instância. Com a condição que nenhuma peça dentro da instância possui área maior que chapa inicial disponibilizada para aplicação das metodologias.

Escolheu-se dez instâncias para os testes porque essas, apresentam todas as características necessárias para testar os métodos. Por características foram estabelecidas o tamanho da chapa disponível para corte e as peças desejadas para corte. As peças consistem de dimensões *largura x altura*.

Para a técnica da Programação Dinâmica implementou-se um programa escrito em linguagem C. A entrada para o programa é a lista de peças desejadas para corte com suas respectivas prioridades de corte (Capítulo 2) e a chapa inicial para corte.

Em todas as peças de cada instância de teste foi determinado que possuem prioridade 1. Ou seja, a mesma prioridade para que as peças sejam neutras em relação as outras na aplicação da metodologia.

Se fossem determinadas prioridades diferentes para as peças a comparação entre as metodologias seria diferente pelo fato que a metodologia “Heurística O(*mn*) para o Corte Guilhotinado Bidimensional” não determina uma prioridade de corte para as peças em que será aplicada. A única prioridade desta metodologia é a imposta em sua heurística (Capítulo 3) e não a prioridade de corte das peças, que são prioridades diferente

As instâncias de teste foram aplicadas para as duas metodologias. A partir disso comparou-se quatro itens: O aproveitamento de área da chapa para corte (cm²), a porcentagem de uso da chapa (%) que foi aplicada a metodologia, tempo de execução que cada programa obteve aplicando a metodologia (em segundos) e a quantidade de peças que cada metodologia conseguiu atender do conjunto de peças para corte.

A plataforma computacional utilizada foi um processador Intel Core i3 CPU M380 @ 2.53GHz, 2Gb de memória, com sistema operacional Linux Ubuntu 12.04 LTS. Para coletar os resultados da tecnologia da heurística O(*mn*) para o Corte Guilhotinado Bidimensional, foi utilizado um programa desenvolvido por [1], disponível em [5] para ambiente Windows.

Criou-se tabelas e gráficos com o resultados obtidos da aplicação de cada metodologia. Nas tabelas os campos estão organizados com a indicação da instância de teste, a indicação da chapa que as metodologias foram aplicadas, a quantidade de peças da instância e os itens para comparação relacionados com as metodologias.

A metodologia 1 é a aplicação da Programação Dinâmica (Capítulo 2), a metodologia 2 com estratégia 1 é a aplicação da "Heurística O(*mn*) para o Corte Guilhotinado Bidimensional com a estratégia de maximização do menor pedaço e a metodologia 2 com estratégia 3 é a aplicação da "Heurística O(*mn*) para o Corte Guilhotinado Bidimensional com a estratégia de maximização do maior pedaço.

Todas as instâncias de teste onde está a descrição das peças usadas para comparação foram entregues juntamente com este trabalho em mídias ópticas (CDs).

Instância de teste 1 – Chapa de 100x100 (10000 cm²) com 21 peças a serem atendidas			
Itens para comparação	***Metodologia 1***	***Metodologia 2 Estratégia 1***	***Metodologia 2 Estratégia 3***
Aproveitamento de área (cm²):	9890	7819	9054
Porcentagem de uso (%):	98,9	78,19	90,54
Tempo de execução (segundos):	3.835	0,0	0,0
Quantidade de peças atendidas:	21	14	20

Tabela 1 – Instancia de testes 1

Segundo a Tabela 1 pode-se observar que a execução com os parâmetros relacionados apresenta as dimensões da chapa (100x100) e o tamanho de sua respectiva área (10000 cm²)

que as metodologias foram aplicadas, a quantidade de peças contidas na instância 1 (21) e os quatro itens determinados para comparação.

A Programação Dinâmica (Metodologia 1) obteve um aproveitamento de área de 9890 cm² com 21 peças atendidas da instância de teste 1. A porcentagem de uso alcançada de 98,9% da chapa e o tempo de sua execução foi de 3.835 segundos.

A "Heurística O(*mn*) para o Corte Guilhotinado Bidimensional" com a estratégia da maximização do menor pedaço (Metodologia 2 com estratégia 1) obteve um aproveitamento de área de 7819 cm² com 14 peças atendidas da instância de teste 1. A porcentagem de uso alcançada é de 78,19% da chapa e o tempo de sua execução foi zero.

A "Heurística O(*mn*) para o Corte Guilhotinado Bidimensional" com a estratégia da maximização do maior pedaço (Metodologia 2 com estratégia 3) obteve um aproveitamento de área de 9054 cm² com 20 peças atendidas da instância de teste 1. A porcentagem de uso alcançada é de 90,54% da chapa e o tempo de sua execução foi zero.

O item tempo de execução com a metodologia 2 (estratégia 1 e 3) é a mesma mostrada no programa disponível em [5]. O programa não converte o resultado para segundos.

Instância de teste 2 – Chapa de 100x100 (10000 cm²) com 23 peças a serem atendidas			
Itens para comparação	***Tecnologia 1***	***Tecnologia 2 Estratégia 1***	***Tecnologia 2 Estratégia 3***
Aproveitamento de área (cm²):	9998	7964	8853
Porcentagem de uso (%):	99,98	79,64	88,53
Tempo de execução (segundos):	14.1	0,0	0,0
Quantidade de peças atendidas:	23	15	20

Tabela 2 – Instancia de testes 2

Segundo a Tabela 2 pode-se observar que a execução com os parâmetros relacionados apresenta as dimensões da chapa (100x100) e o tamanho de sua respectiva área (10000 cm²) que as metodologias foram aplicadas, a quantidade de peças contidas na instância (23) e os quatro itens determinados para comparação.

A Programação Dinâmica (Metodologia 1) obteve um aproveitamento de área de 9998 cm² com 23 peças atendidas da instância de teste 2. A porcentagem de uso alcançada de 99,98% da chapa e o tempo de sua execução foi de 14.1 segundos.

A “Heurística O(*mn*) para o Corte Guilhotinado Bidimensional” com a estratégia da maximização do menor pedaço (Metodologia 2 com estratégia 1) obteve um aproveitamento de área de 7964 cm² com 15 peças atendidas da instância de teste 2. A porcentagem de uso alcançada é de 79,64% da chapa e o tempo de sua execução foi zero.

A “Heurística O(*mn*) para o Corte Guilhotinado Bidimensional” com a estratégia da maximização do maior pedaço (Metodologia 2 com estratégia 3) obteve um aproveitamento de área de 8853 cm² com 20 peças atendidas da instância de teste 2. A porcentagem de uso alcançada é de 88,53% da chapa e o tempo de sua execução foi zero.

O item tempo de execução com a metodologia 2 (estratégia 1 e 3) é a mesma mostrada no programa disponível em [5]. O programa não converte o resultado para segundos.

Instância de teste 3 – Chapa de 100x100 (10000 cm²) com 27 peças a serem atendidas			
Itens para comparação	***Tecnologia 1***	***Tecnologia 2 Estratégia 1***	***Tecnologia 2 Estratégia 3***
Aproveitamento de área (cm²):	9939	8414	8495
Porcentagem de uso (%):	99,39	84,14	84,95
Tempo de execução (segundos):	19.804	0,0	0,0
Quantidade de peças atendidas:	27	19	21

Tabela 3 – Instancia de testes 3

Segundo a Tabela 3 pode-se observar que a execução com os parâmetros relacionados apresenta as dimensões da chapa (100x100) e o tamanho de sua respectiva área (10000 cm²) que as metodologias foram aplicadas, a quantidade de peças contidas na instância 3 (27) e os quatro itens determinados para comparação.

A Programação Dinâmica (Metodologia 1) obteve um aproveitamento de área de 9939 cm² com 27 peças atendidas da instância de teste 3. A porcentagem de uso alcançada de 99,39% da chapa e o tempo de sua execução foi de 19,804 segundos.

A “Heurística O(*mn*) para o Corte Guilhotinado Bidimensional” com a estratégia da maximização do menor pedaço (Metodologia 2 com estratégia 1) obteve um aproveitamento de área de 8414 cm² com 19 peças atendidas da instância de teste 3. A porcentagem de uso alcançada é de 84,14% da chapa e o tempo de sua execução foi zero.

A “Heurística O(*mn*) para o Corte Guilhotinado Bidimensional” com a estratégia da maximização do maior pedaço (Metodologia 2 com estratégia 3) obteve um aproveitamento de área de 8495 cm² com 21 peças atendidas da instância de teste 3. A porcentagem de uso alcançada é de 84,95% da chapa e o tempo de sua execução foi zero.

O item tempo de execução com a metodologia 2 (estratégia 1 e 3) é a mesma mostrada no programa disponível em [5]. O programa não converte o resultado para segundos.

Instância de teste 4 – Chapa de 100x200 (20000 cm²) com 18 peças a serem atendidas			
Itens para comparação	***Tecnologia 1***	***Tecnologia 2 Estratégia 1***	***Tecnologia 2 Estratégia 3***
Aproveitamento de área (cm²):	19742	14197	15956
Porcentagem de uso (%):	98,71	70,98	79,76
Tempo de execução (segundos):	0,756	0,0	0,0
Quantidade de peças atendidas:	18	15	16

Tabela 4 – Instancia de testes 4

Segundo a Tabela 4 pode-se observar que a execução com os parâmetros relacionados apresenta as dimensões da chapa (100x200) e o tamanho de sua respectiva área (20000 cm²) que as metodologias foram aplicadas, a quantidade de peças contidas na instância 4 (18) e os quatro itens determinados para comparação.

A Programação Dinâmica (Metodologia 1) obteve um aproveitamento de área de 19742 cm² com 18 peças atendidas da instância de teste 4. A porcentagem de uso alcançada de 98,71% da chapa e o tempo de sua execução foi de 0,756 segundos.

A “Heurística O(*mn*) para o Corte Guilhotinado Bidimensional” com a estratégia da maximização do menor pedaço (Metodologia 2 com estratégia 1) obteve um aproveitamento de área de 14197 cm² com 15 peças atendidas da instância de teste 4. A porcentagem de uso alcançada é de 70,78% da chapa e o tempo de sua execução foi zero.

A “Heurística O(*mn*) para o Corte Guilhotinado Bidimensional” com a estratégia da maximização do maior pedaço (Metodologia 2 com estratégia 3) obteve um aproveitamento de área de 15952 cm² com 16 peças atendidas da instância de teste 4. A porcentagem de uso alcançada é de 79,76% da chapa e o tempo de sua execução foi zero.

O item tempo de execução com a metodologia 2 (estratégia 1 e 3) é a mesma mostrada no programa disponível em [5]. O programa não converte o resultado para segundos.

Instancia de teste 5 – Chapa de 100x200 (20000 cm²) com 22 peças a serem atendidas			
Itens para comparação	***Tecnologia 1***	***Tecnologia 2 Estratégia 1***	***Tecnologia 2 Estratégia 3***
Aproveitamento de área (cm²):	19972	17792	17792
Porcentagem de uso (%):	99,86%	88,96%	88,96%
Tempo de execução (segundos):	10,668	0,0	0,0
Quantidade de peças atendidas:	22	19	19

Tabela 5 – Instancia de testes 5

Segundo a Tabela 5 pode-se observar que a execução com os parâmetros relacionados apresenta as dimensões da chapa (100x200) e o tamanho de sua respectiva área (20000 cm²) que as metodologias foram aplicadas, a quantidade de peças contidas na instância 5 (22) e os quatro itens determinados para comparação.

A Programação Dinâmica (Metodologia 1) obteve um aproveitamento de área de 19972 cm² com 22 peças atendidas da instância de teste 5. A porcentagem de uso alcançada de 99,86% da chapa e o tempo de sua execução foi de 10,668 segundos.

A "Heurística O(*mn*) para o Corte Guilhotinado Bidimensional" com a estratégia da maximização do menor pedaço (Metodologia 2 com estratégia 1) obteve um aproveitamento de área de 17792 cm² com 19 peças atendidas da instância de teste 5. A porcentagem de uso alcançada é de 88,96% da chapa e o tempo de sua execução foi zero.

A "Heurística O(*mn*) para o Corte Guilhotinado Bidimensional" com a estratégia da maximização do maior pedaço (Metodologia 2 com estratégia 3) obteve um aproveitamento de área de 17792 cm² com 19 peças atendidas da instância de teste 5. A porcentagem de uso alcançada é de 88,96% da chapa e o tempo de sua execução foi zero.

O item tempo de execução com a metodologia 2 (estratégia 1 e 3) é a mesma mostrada no programa disponível em [5]. O programa não converte o resultado para segundos.

Instância de teste 6 – Chapa de 100x200 (20000 cm²) com 23 peças a serem atendidas			
Itens para comparação	***Tecnologia 1***	***Tecnologia 2 Estratégia 1***	***Tecnologia 2 Estratégia 3***
Aproveitamento de área:	19993	17494	17730
Porcentagem de uso:	99,965%	87,47%	88,65%
Tempo de execução:	15.550	0,0	0,0
Quantidade de pedidos atendidos:	23	19	20

Tabela 6 – Instancia de testes 6

Segundo a Tabela 6 pode-se observar que a execução com os parâmetros relacionados apresenta as dimensões da chapa (100x200) e o tamanho de sua respectiva área (20000 cm²) que as metodologias foram aplicadas, a quantidade de peças contidas na instância (23) e os quatro itens determinados para comparação.

A Programação Dinâmica (Metodologia 1) obteve um aproveitamento de área de 19993 cm² com 23 peças atendidas da instância de teste 6. A porcentagem de uso alcançada de 99,965% da chapa e o tempo de sua execução foi de 15,55 segundos.

A "Heurística O(*mn*) para o Corte Guilhotinado Bidimensional" com a estratégia da maximização do menor pedaço (Metodologia 2 com estratégia 1) obteve um aproveitamento de área de 17494 cm² com 19 peças atendidas da instância de teste 6. A porcentagem de uso alcançada é de 87,47% da chapa e o tempo de sua execução foi zero.

A "Heurística O(*mn*) para o Corte Guilhotinado Bidimensional" com a estratégia da maximização do maior pedaço (Metodologia 2 com estratégia 3) obteve um aproveitamento de área de 17730 cm² com 20 peças atendidas da instância de teste 6. A porcentagem de uso alcançada é de 88,65% da chapa e o tempo de sua execução foi zero.

O item tempo de execução com a metodologia 2 (estratégia 1 e 3) é a mesma mostrada no programa disponível em [5]. O programa não converte o resultado para segundos.

Instancia de teste 7 – Chapa de 100x300 (30000 cm²) com 19 peças a serem atendidas			
Itens para comparação	***Tecnologia 1***	***Tecnologia 2 Estratégia 1***	***Tecnologia 2 Estratégia 3***
Aproveitamento de área (cm²):	29865	23804	21894
Porcentagem de uso (%):	99,55%	79,34	72,98%
Tempo de execução (segundos):	4,506	0,0	0,0
Quantidade de peças atendidas:	19	14	14

Tabela 7 - Instancia de teste 7

Segundo a Tabela 7 pode-se observar que a execução com os parâmetros relacionados apresenta as dimensões da chapa (100x300) e o tamanho de sua respectiva área (30000 cm²) que as metodologias foram aplicadas, a quantidade de peças contidas na instância 7 (19) e os quatro itens determinados para comparação.

A Programação Dinâmica (Metodologia 1) obteve um aproveitamento de área de 19993 cm² com 19 peças atendidas da instância de teste 7. A porcentagem de uso alcançada de 99,55% da chapa e o tempo de sua execução foi de 4,506 segundos.

A "Heurística O(*mn*) para o Corte Guilhotinado Bidimensional" com a estratégia da maximização do menor pedaço (Metodologia 2 com estratégia 1) obteve um aproveitamento de área de 23804 cm² com 14 peças atendidas da instância de teste 7. A porcentagem de uso alcançada é de 79,34% da chapa e o tempo de sua execução foi zero.

A "Heurística O(*mn*) para o Corte Guilhotinado Bidimensional" com a estratégia da maximização do maior pedaço (Metodologia 2 com estratégia 3) obteve um aproveitamento de área de 21894 cm² com 14 peças atendidas da instância de teste 7. A porcentagem de uso alcançada é de 72,98% da chapa e o tempo de sua execução foi zero.

O item tempo de execução com a metodologia 2 (estratégia 1 e 3) é a mesma mostrada no programa disponível em [5]. O programa não converte o resultado para segundos.

Instancia de teste 8 – Chapa de 100x300 (30000 cm²) com 23 peças a serem atendidas			
Itens para comparação	***Tecnologia 1***	***Tecnologia 2 Estratégia 1***	***Tecnologia 2 Estratégia 3***
Aproveitamento de área (cm²):	29968	25703	26550
Porcentagem de uso (%):	99,893%	85,67	88,5
Tempo de execução (segundos):	7,907	0,0	0,0
Quantidade de peças atendidas:	23	17	18

Tabela 8 - Instancia de teste 8

Segundo a Tabela 8 pode-se observar que a execução com os parâmetros relacionados apresenta as dimensões da chapa (100x300) e o tamanho de sua respectiva área (30000 cm²) que as metodologias foram aplicadas, a quantidade de peças contidas na instância 8 (23) e os quatro itens determinados para comparação.

A Programação Dinâmica (Metodologia 1) obteve um aproveitamento de área de 29968 cm² com 23 peças atendidas da instância de teste 8. A porcentagem de uso alcançada de 99,893% da chapa e o tempo de sua execução foi de 7,907 segundos.

A "Heurística O(*mn*) para o Corte Guilhotinado Bidimensional" com a estratégia da maximização do menor pedaço (Metodologia 2 com estratégia 1) obteve um aproveitamento de área de 25703 cm² com 17 peças atendidas da instância de teste 8. A porcentagem de uso alcançada é de 85,67% da chapa e o tempo de sua execução foi zero.

A "Heurística O(*mn*) para o Corte Guilhotinado Bidimensional" com a estratégia da maximização do maior pedaço (Metodologia 2 com estratégia 3) obteve um aproveitamento de área de 26550 cm² com 18 peças atendidas da instância de teste 8. A porcentagem de uso alcançada é de 88, 5% da chapa e o tempo de sua execução foi zero.

O item tempo de execução com a metodologia 2 (estratégia 1 e 3) é a mesma mostrada no programa disponível em [5]. O programa não converte o resultado para segundos.

Instancia de teste 9 – Chapa de 100x300 com 25 peças a serem atendidas			
Itens para comparação	***Tecnologia 1***	***Tecnologia 2 Estratégia 1***	***Tecnologia 2 Estratégia 3***
Aproveitamento de área (cm²):	29982	25659	26616
Porcentagem de uso (%):	99,94%	85,53%	88,72%
Tempo de execução (segundos):	16.244	0,0	0,0
Quantidade de peças atendidas:	25	20	20

Tabela 9 - Instancia de teste 9

Segundo a Tabela 9 pode-se observar que a execução com os parâmetros relacionados apresenta as dimensões da chapa (100x300) e o tamanho de sua respectiva área (30000 cm²) que as metodologias foram aplicadas, a quantidade de peças contidas na instância (25) e os quatro itens determinados para comparação.

A Programação Dinâmica (Metodologia 1) obteve um aproveitamento de área de 29982 cm² com 25 peças atendidas da instância de teste 9. A porcentagem de uso alcançada de 99,94% da chapa e o tempo de sua execução foi de 16,244 segundos.

A "Heurística O(*mn*) para o Corte Guilhotinado Bidimensional" com a estratégia da maximização do menor pedaço (Metodologia 2 com estratégia 1) obteve um aproveitamento de área de 25659 cm² com 20 peças atendidas da instância de teste 9. A porcentagem de uso alcançada foi de 85,53% da chapa e o tempo de sua execução foi zero.

A "Heurística O(*mn*) para o Corte Guilhotinado Bidimensional" com a estratégia da maximização do maior pedaço (Metodologia 2 com estratégia 3) obteve um aproveitamento de área de 26616 cm² com 20 peças atendidas da instância de teste 9. A porcentagem de uso alcançada é de 88,72% da chapa e o tempo de sua execução foi zero.

O item tempo de execução com a metodologia 2 (estratégia 1 e 3) é a mesma mostrada no programa disponível em [5]. O programa não converte o resultado para segundos.

Instancia de teste 10 – Chapa de 1000x1000 (1000000 cm²) com 25 peças a serem atendidas			
Itens para comparação	***Tecnologia 1***	***Tecnologia 2 Estratégia 1***	***Tecnologia 2 Estratégia 3***
Aproveitamento de área (cm²):	996976	850667	866650
Porcentagem de uso (%):	99,6976%	85,07%	86,67%
Tempo de execução (segundos):	17.109	0,0	0,0
Quantidade de peças atendidas:	25	17	19

Tabela 10 - Instancia de teste 10

Segundo a Tabela 10 pode-se observar que a execução com os parâmetros relacionados apresenta as dimensões da chapa (1000x1000) e o tamanho de sua respectiva área (1000000 cm²) que as metodologias foram aplicadas, a quantidade de peças contidas na instância 10 (25) e os quatro itens determinados para comparação.

A Programação Dinâmica (Metodologia 1) obteve um aproveitamento de área de 996976 cm² com 25 peças atendidas da instância de teste 10. A porcentagem de uso alcançada de 99,6976% da chapa e o tempo de sua execução foi de 17,109 segundos.

A "Heurística O(*mn*) para o Corte Guilhotinado Bidimensional" com a estratégia da maximização do menor pedaço (Metodologia 2 com estratégia 1) obteve um aproveitamento de área de 850667 cm² com 17 peças atendidas da instância de teste 10. A porcentagem de uso alcançada foi de 85,07% da chapa e o tempo de sua execução foi zero.

A "Heurística O(*mn*) para o Corte Guilhotinado Bidimensional" com a estratégia da maximização do maior pedaço (Metodologia 2 com estratégia 3) obteve um aproveitamento de área de 866650 cm² com 19 peças atendidas da instância de teste 10. A porcentagem de uso alcançada é de 86,67% da chapa e o tempo de sua execução foi zero.

O item tempo de execução com a metodologia 2 (estratégia 1 e 3) é a mesma mostrada no programa disponível em [5]. O programa não converte o resultado para segundos.

Pode-se verificar que para as instâncias de teste elaboradas para este trabalho a técnica da Programação Dinâmica para ocupação de área da chapa de vidro possui um desempenho maior que a "Heurística O(*mn*) para Corte Guilhotinado Bidimensional" com duas de suas estratégias.

Outra característica que pode-se observar é tempo gasto para aplicação de cada metodologia. A Programação Dinâmica de acordo com as instâncias de testes elaboradas para este trabalho sempre teve um tempo de execução maior que a outra metodologia.

Capítulo 5

5.1 Conclusão

Foram estudadas as duas metodologias afim de fazer uma comparação de desempenho em relação ao uso de uma determinada chapa de vidro. As duas metodologias foram comparadas com dez instâncias de teste contendo cada uma, peças para corte sem serem repetidas dentro da mesma instância e geradas de forma aleatória.

Para a metodologia da Programação Dinâmica implementou-se um programa escrito em linguagem C, onde as entradas são feitas via código-fonte sendo a lista de peças para corte com suas respectivas prioridades e o tamanho da área da chapa em que a metodologia será aplicada.

Para os estudos das metodologias foram estudados artigos e livros que contribuíram para o desenvolvimento deste trabalho.

Para a metodologia "Heurística O(*mn*) para o Corte Guilhotinado Bidimensional", para a coleta dos resultados foi utilizado um programa disponível em [5] desenvolvido por [1].

A partir da aplicação de cada metodologia com as instâncias de teste, conclui-se que para a maximização de uso da área de uma de vidro, a metodologia da Programação Dinâmica obteve um melhor desempenho do que a metodologia da "Heurística O(*mn*) para o Corte Guilhotinado Bidimensional" e duas de suas estratégias de otimização que a acompanham (Capítulo 3).

A metodologia da Programação Dinâmica possui um melhor desempenho que a outra metodologia em relação a ocupação de área da chapa de vidro. Porém, o tempo de sua execução foi maior em relação a outra metodologia para as instâncias de teste elaboradas para este trabalho.

A metodologia da "Heurística O(*mn*) para o Corte Guilhotinado Bidimensional" perde em relação a ocupação de área da metodologia que envolve Programação Dinâmica. Porém, o tempo de execução para sua aplicação é menor que a metodologia da Programação Dinâmica

5.2 Trabalhos futuros

A metodologia da Programação Dinâmica determina quais peças de um determinado conjunto são as que maximizam o uso da área de uma determinada chapa de vidro. Para um trabalho futuro pode-se desenvolver uma metodologia de determinar a posição de corte dessas peças selecionadas para serem cortadas sobre a chapa de vidro de forma correta.

Uma comparação entre a metodologia da Programação Dinâmica com outras metodologias de otimização como por exemplo a Programação Linear[10], poderia resultar em outro trabalho futuro.

O último trabalho futuro sugere-se que poderia ser a implementação de um programa que utilize a Programação Dinâmica em conjunto com outra técnica de otimização.

Apêndice A

Código da metodologia da Programação Dinâmica escrito em linguagem C.

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
/* Estrutura de uma chapa de vidro com o seu “nome”, tamanho de área e prioridade. */
typedef struct {
	const char * nome;
	int tamanho, prioridade;
} chapa_t;
 /* Inicialização com a entrada da lista de peças. */
chapa_t chapa[] = {
	{"Janela",		9,	150},
	{"Porta",		13,	35},
	{"Mesa",		153,	200},
	{"Chapa 1",		50,	160},
	{"Mesa 2",		15,	60},
	{"Janela 2",		68,	45},
	{"Porta 2",		27,	60},
	{"Chapa 2",		39,	40},
	{"Chapa Chico",		23,	30},
	{"Chapa Mario",		52,	10},
	{"Fachada",		11,	70},
	{"Cinzeiro",		32,	30},
	{"Chapa Maria",		24,	15},
	{"Empresa X",		48,	10},
	{"Empresa Y",		73,	40},
	{"Empresa Z",		42,	70},
	{"banheiro",		43,	75},
	{"Box",		22,	80},
	{"Cozinha",		7,	20},
	{"Sala",		18,	12},
	{"Home Theater",	4,	50},
	{"Banheiro 2",		30,	10}
};

#define n_chapas (sizeof(chapa)/sizeof(chapa_t))
typedef struct {
	uint32_t bits; /* 32 bits, consigo calcular até 32 chapas */
	int prioridade;
} solucao;
 /* Função Programação Dinâmica, parâmetro tamanho corresponde a área da chapa. */
void otimo(int tamanho, int idx, solucao *s)
{
	solucao v1, v2;
```

```
        if(idx < 0) {
                s->bits = s->prioridade = 0;
                return;
        }
        if (tamanho < chapa[idx].tamanho) {
                otimo(tamanho, idx - 1, s);
                return;
         }
        otimo(tamanho, idx - 1, &v1);
        otimo(tamanho - chapa[idx].tamanho, idx - 1, &v2);
        v2.prioridade += chapa[idx].prioridade;
        v2.bits |= (1 << idx);
        *s = (v1.prioridade >= v2.prioridade) ? v1 : v2;
}
int main(void)
{
        int i = 0, w = 0;
        solucao s = {0, 0};
        otimo(600, n_chapas - 1, &s);
        for (i = 0; i < n_chapas; i++) {
                if (s.bits & (1 << i)) {
                        printf("%s\n", chapa[i].nome);
                        w += chapa[i].tamanho;
                }
        }
        printf("Tamanho total: %d \n", w);
        return 0;
}
```

Referências bibliográficas

[1] Nascimento, Hugo A. D. do, Longo, Humberto José, Aloise, Dario José. *Uma Heurística O(mn) para o Corte Bidimensional Guilhotinado*. Nos anais do XXXI Congresso da Sociedade Brasileira de Pesquisa Operacional, Juiz de Fora, Brasil, Outubro de 1999.

[2] Cormen, T.H., Leiserson, C.E., Rivest, R.L. & Stein, C. *Algoritmos Teoria e Prática*. 2º edição. 2002.

[3] Charles, A., A. *Modelagens Exata e Heurística para Resolução do Problema do* C*aixeiro Viajante com Coleta de Prêmios.* Universidade Federal de Ouro Preto. Instituto de Ciência exatas e biológicas. Departamento de computação, 2003.

[4] FEOFILOFF, Paulo. *Programação Dinâmica*. Em: http://www.ime.usp.br/~pf/analise_de_algoritmos/aulas/dynamic- programming.html. Acesso em 30/11/2014.

[5] NASCIMENTO, Hugo A. D. do, LONGO, Humberto José, ALOISE, Dario José. Em: http://www.portal.inf.ufg.br/~hadn/old/sydney/research/corte/. Aceso em 30/11/2014.

[6] Oliveira, Fernanda A., Caldas, Mayara D. de A. *Sequência de Fibonacci*. Campinas. Abril de 2013. Em: http://www.ime.unicamp.br/~ftorres/ENSINO/MONOGRAFIAS/F_M1_FM_2013.pdf. Acesso em: 30/11/2014.

[7] Ribeiro, Pedro. *Programação Dinâmica. Uma metodologia de resolução de problemas.* Em: http://www.dcc.fc.up.pt/~pribeiro/pd-pribeiro.pdf. Acesso em: 30/11/2014.

[8] Manber, Udi. *Introduction to Algorithms a Creative Approach*. 1º edição. 1989.

[9] Parberry, Ian., Gasarch, William. *Problems on Algorithms*. 2º edição. Julho, 2002.

[10] Frossard, Afonso C. P. *Programação Linear: Maximização de Lucro e Minimização de Custos*. Em: http://www.flf.edu.br/revista-flf.edu/volume06/V6_02.pdf. Faculdade Lourenço Filho.