Trabalho Final - Projeto e Análise de Algoritmos

1st July Werneck
Instituto de Ciências Exatas e Informática
Pontifícia Universidade Católica de Minas Gerais
Belo Horizonte, Minas Gerais
werneckjuly@gmail.com

2st Sofia Bhering
Instituto de Ciências Exatas e Informática
Pontifícia Universidade Católica de Minas Gerais
Belo Horizonte, Minas Gerais
sofiabhering28@gmail.com

3st Thiago Amado Costa
Instituto de Ciências Exatas e Informática
Pontifícia Universidade Católica de Minas Gerais
Belo Horizonte, Minas Gerais
thiadoamadocosta@gmail.com

***Abstract*— Esse artigo tem como objetivo aplicar as metodologias aprendidas em sala de aula para problemas de otimização, sendo eles o método de força-bruta, branch and bound e eurística,relacionados ao número de cortes de peças no ramo produtivo, analisando assim os seus respectivos desempenhos.**

I. Introdução

Ao nos depararmos com projeto e análise de algoritmos, temos como relevância o estudo para problemas de otimização. Sabendo-se disso empresas do ramo produtivo tendem a procurar cada vez mais a tecnologia para soluções eficientes de seu cotidiano. O trabalho tem como objetivo específico explorar os algoritmos de força-bruta, branch and bound e eurística e apresentar o algoritmo eficiente para aplicação e planejamento de recorte de peças de tecido para confecção de roupas, de modo a evitar ao máximo o desperdício de tecido.
Tratando o problema proposto, definimos métricas para as peças com altura igual ao do tecido (100 cm), todas elas sendo em formato trapezoidal e não podendo rotacionar, conseguindo impor determinados limites,na entrada de dados(input) para cálculos mais eficientes do algoritmo.
O artigo está organizado da seguinte maneira: a Seção II apresenta a solução proposta do problema e analisa a ordem de complexidade de cada algoritmo; a Seção III descreve a implementação, através dos detalhes dos programas implementados, como a melhor eficiência da solução e como foi feita a interface gráfica; a Seção IV apresenta os relatórios de teste, descrevendo os testes realizados e mostrando como será na prática os resultados descritos seção III; a Seção V apresenta a conclusão do trabalho comparando as soluções por força-bruta, branch-and-bound e heurística, quanto à sua ordem de complexidade e tempo de execução.

II. Solução proposta

Abordamos no nosso projeto três técnicas de problemas de otimização, a Força-bruta(brute force algorithm), branch-bound e Heurística(euristic),sendo eles posteriormente definidos em cada subtítulo, o retorno de cada função descrita é a melhor solução de acordo com a menor taxa de desperdício de cada tecido

A. Força-bruta

A Força Bruta, também conhecida como método exaustivo é baseado em uma técnica de gerar e testar todos os possíveis resultados da solução e através disso checar o resultado eficaz do algoritmo.
No trabalho, utilizamos a técnica de força-bruta para gerar todas as permutações e consequentemente todas as possibilidades de ordem de peças na tela, de acordo com o input. Para cada permutação gerada, seu tecido resultante é salvo em uma lista de tecidos. Dessa forma, a lista de tecidos é percorrida para encontrar o menor valor de desperdício, e ao encontrar, esse tecido é mostrado na tela.
Assim, como o algoritmo é feito gerando todas as permutações possíveis e depois percorrendo-as para encontrar a melhor solução, sua complexidade é de $O(n * n!)$, ou somente $O(n!)$, pois $n!$ domina assintoticamente n.

B. Branch-Bound

O método Branch-Bound, caracteriza-se por desenvolver uma enumeração inteligente das soluções candidatas à solução ótima inteira de um problema. O termo branch refere-se ao fato

de que o método efetua partições no espaço das soluções e o termo bound ressalta que a prova da otimalidade da solução utiliza-se de limites calculados ao longo da enumeração.
A lógica de implementação utilizada no trabalho baseia-se na geração de permutações candidatas para a ordem das coordenadas e a validação se a escolha é promissora para solução ou não. A técnica é executada através de uma função recursiva, cujo critério de parada é a verificação caso o vetor coords, que armazena as coordenadas dos trapezios lidos no arquivos de input, seja vazio, dado que neste caso o espaço amostral para expansão teria acabado. Dentro do critério de parada, temos ainda uma verificação se o gasto do Tecido atual é menor do que o armazenado na variável resposta, se verdadeiro atualizamos o valor de resposta e armazenamos a nova melhor ordem de coordenadas. Se caso o espaço amostral ainda puder ser expandido, isto é se ainda houver trapezios a serem inseridos na solução, temos um critério de poda que vai limitar inferiormente o melhor valor para o gasto do tecido. Nesse caso, comparamos o valor gasto pelo Tecido atual com a variável resposta, podando assim soluções que ainda que não expandiadas em sua totalidade não constituem uma solução melhor para o problem em questão. Se o cenário atual ainda for uma alternativa válida entramos em um loop que irá até a quantidade de coordenadas restantes, gerando as permutações de forma recursiva.
O método Branch-and-Bound contém em sua estrutura de dados uma árvore. A primeira árvore contém um único elemento chamado de pai (problema inicial) e as outras subárvores chamadas de nós filhos. Esta árvore vai se ramificando até chegar ao número de variáveis existentes do problema. Para um problema com n variáveis, há n! soluções a serem consideradas. O total de nós da árvore completa será $\sum_{i=0}^{b} 2^n$. Esta complexidade é formalizada como crescimento exponencial de ordem $O(2^k)$.

C. Heurística

A definição de Heurística consiste em regras de bom senso, baseadas na experiência e na forma como resolvemos problemas cotidianos, para o ramo da computação a Heurística são algoritmos que têm como objetivo fornecer soluções sem um limite formal de qualidade, em geral avaliado empiricamente em termos de complexidade (média) e qualidade de soluções. É projetada para obter ganho computacional ou simplicidade conceitual, possivelmente ao custo de precisão.
No trabalho, inicialmente dividimos a lista de coordenadas de acordo com a inclinação do trapezio, gerando assim, resultados visivelmente gráficos de trapezios inclinados à esquerda e trapezios inclinados à direita. Dessa forma, utilizamos o código de permutações para encotrar o melhor resultado tanto para os trapézios inclinados à esquerda quanto para os trapézios inclinados à direita, para finalmente juntar as duas melhores soluções para a solução final.

III. Implementação

A implementação do projeto é feita na linguagem python, sendo ela ultilizada para gerar toda a interface gráfica, utilizando a biblioteca *graphics.py*, além do módulo *time* para as medidas de tempo de execução.

A. Organização do Código

Como suporte, criamos classes para abstrair lógicas mais básicas do problema, como a classe Tissue, representando o tecido utilizado, e a classe Trapezium, representando a peça a ser inserida no tecido.

1) Classe do Trapézio: Na classe Trapezium, criamos o trapézio com base nas coordenadas $x1$, $x2$ e $x3$, além dos limites superiores e inferiores de x, que são utilizados na classe Tissue. As 4 coordenadas do trapézio são calculadas considerando os limites superior e inferior de X, de forma a evitar que um trapézio tenha coordenadas conflitantes com outro.
A lógica para o cálculo das coordenadas ocorre da seguinte forma: primeiro, é calculado a inclinação com base nos limites superiores e inferiores. Após isso, é calculado o primeiro ponto do trapézio, considerando a inclinação calculada anteriormente e os limites do trapézio anterior (caso o tecido estiver vazio, limites são ambos 0). Com isso, os 4 pontos do trapézio são calculados.

2) Classe do Tecido: Na classe Tissue, criamos o tecido inserindo peça por peça, atualizando atributos da classe a cada nova inserção, como o desperdício atual do tecido (área total do tecido - área de cada trapézio), largura, ordem de trapézios e ordem de coordenadas, além dos limites superiores e inferiores de X, utilizados para o tratamento de colisões dos trapézios, feito na classe Trapezium.

Também é na classe Trapezium que calculamos a area do trapézio, seguindo a formula $\frac{(B+b)*h)}{2}$, em que B é a base superior, b é a base inferior e h é a altura do tecido, constante em 100.

B. Criação de Casos de Teste

Para a criação de casos de teste, e de modo a facilitar a visualização das coordenadas de forma gráfica, foi implementado uma funcionalidade extra no código que recebe e exibe na tela, de forma sequencial, as coordenadas recebidas por um arquivo de input.
Segue abaixo, exemplo recebendo um input e sua representação na tela, utilizando a funcionalidade em questão.

Fig. 1. Representação do input 1

```
3
50 80 40
30 80 -40
50 20 -40
```

IV. Relatório de testes

Para nossos testes, utilizamos o módulo *time* para as medidas de tempo de execução.
Para cada algoritmo implementado (força bruta, branch and bound e método heurístico), medimos o tempo de execução para exemplos diversos, variando de 3 a 10 entradas de coordenadas.
Para cada input utilizado, seguem as respectivas execuções de cada algoritmo

A. Input com 3 coordenadas

Para 3 coordenadas, utilizamos o seguinte arquivo de input:

```
3
50 80 40
30 80 -40
50 20 -40
```

Fig. 2. Representação do input de 3 coordenadas

1) Execução Força Bruta: Com em média 0.0060 segundos de execução e menor desperdício de tecido sendo 4500, o método de força bruta encontra a solução, conforme a figura a seguir.

Fig. 3. Método de força bruta para 3 coordenadas

2) Execução Branch and Bound: Com em média 0.0060 segundos de execução e menor desperdício de tecido sendo 4500, o método de branch and bound encontra a solução, conforme a figura a seguir.

Fig. 4. Método de branch and bound para 3 coordenadas

3) Execução Heurística: Com em média 0.0050 segundos de execução e menor desperdício de tecido sendo 4500, o método heurístico encontra a solução, conforme a figura a seguir.

Fig. 5. Método heurístico para 3 coordenadas

B. Input com 5 coordenadas

Para 5 coordenadas, o arquivo de input:

```
5
50 80 30
30 80 -40
50 20 -40
80 50 20
89 10 -80
```

Fig. 6. Representação do input de 3 coordenadas

1) Execução Força Bruta: Com em média 0.04 segundos de execução e menor desperdício de tecido sendo 8450, o método de força bruta encontra a solução, conforme a figura a seguir.

Fig. 7. Método de força bruta para 5 coordenadas

2) Execução Branch and Bound: Com em média 0.015 segundos de execução e menor desperdício de tecido sendo 8450, o método de branch and bound encontra a solução, conforme a figura a seguir.

Fig. 8. Método de branch and bound para 5 coordenadas

3) Execução Heurística: Com em média 0.006 segundos de execução e menor desperdício de tecido sendo 8450, o método de branch and bound encontra a solução, conforme a figura a seguir.

C. Input com 7 coordenadas

Para 7 coordenadas, o arquivo de input:

```
7
80 80 40
```

Fig. 9. Método heurístico para 5 coordenadas

```
30 80 -40
50 20 -40
80 50 20
89 10 -80
50 40 40
30 80 -40
```

Fig. 10. Representação do input de 7 coordenadas

1) Execução Força Bruta: Com em média 2 segundos de execução e menor desperdício de tecido sendo 8450, o método de força bruta encontra a solução, conforme a figura a seguir.

Fig. 11. Método de força bruta para 7 coordenadas

2) Execução Branch and Bound: Com em média 0.2 segundos de execução e menor desperdício de tecido sendo 8450, o método de branch and bound encontra a solução, conforme a figura a seguir.

Fig. 12. Método de branch and bound para 7 coordenadas

3) Execução Heurística: Com em média 0.02 segundos de execução e menor desperdício de tecido sendo 8450, o método heurístico encontra a solução, conforme a figura a seguir.

D. Input com 10 coordenadas

Para 10 coordenadas, o arquivo de input:

```
10
50 80 40
30 80 -40
50 25 -40
80 50 20
89 10 -80
60 120 40
```

Fig. 13. Método heurístico para 7 coordenadas

```
30 100 -30
80 50 20
80 100 -60
30 100 -30
```

Fig. 14. Representação do input de 10 coordenadas

1) Execução Força Bruta: Com 10 coordenadas, a execução do algoritmo de força bruta já fica inviável, considerando que o algoritmo é $O(n!)$. Ou seja, o algoritmo executa $10! = 3628800$ vezes.

2) Execução Branch and Bound: Com uma média variando entre 60 - 120 segundos de execução e menor desperdício de tecido sendo 167000, o método de branch and bound encontra a solução, conforme a figura a seguir.

Fig. 15. Método de branch and bound para 10 coordenadas

3) Execução Heurística: Com em média 0.25 segundos de execução e menor desperdício de tecido sendo 25200, o método heurístico não encontra a melhor solução, porém é considerado uma solução de compromisso, uma vez que sacrifica a solução ótima por um melhor uso de memória e tempo.

Fig. 16. Método heurístico para 7 coordenadas

V. Conclusão

Através da análise dos resultados obtidos e plotagem dos mesmos em cada representação de tecido, o conteúdo apresentado em sala de aula foi muito bem consolidado. Á respeito de cada teste obtido no tópico anterior, podemos concluir que o método força-bruta é muito bem aplicável para números menores de trapézios, isso se deve pelo fato de que os cálculos de permutações é realizado em n fatorial, ou seja, quanto maior o nosso input mais demorado e custoso ele se torna, como visto no teste com 10 coordenadas, que executa 3628800 vezes. Ao analisar o branch and bound, percebemos que apesar da complexidade deste método ser exponencial e teoricamente difícil de ser resolvido para instâncias grandes, percebe-se que

na prática os resultados são satisfatórios visto que a poda da árvore restringe o universo de ramos a serem percorridos. Já o método Heurístico, foi verificado que ele atua muito bem para inputs específicos, como os testes com 3, 5 e 7 coordenadas. Porém, aumentando o número da entrada, a qualidade da heurística piora.

Para trabalhos futuros, será interessante realizar, também, simulações com diversos tipos de otimizações e não setando valores baixos de entradas, além de adicionar a funcionalidade de girar as peças no tecido, o que aumenta o número de possibilidades para todos os casos.

REFERENCES

[1] Graphics Reference (graphics.py v5) . Disponível em: https://mcsp.wartburg.edu/zelle/python/graphics/graphics.pdf

[2] [B] (livro-texto) N. Ziviani, Projeto de Algoritmos com Implementações em Java e C++. Editora Thompson, 2006.

[3] (livro-texto) T. H. Cormen, C. E. Leiserson, R. L. Rivest , C. Stein. Algoritmos. Campus, 2012.

[4] E. Horowitz, S. Sahni, Fundamentals of Computer Algorithms. Computer Science Press, 1978.