PUC Minas
A B A K Ó S
Instituto de Ciências Exatas e Informática

Problema de Otimização de Cortes em Placas de Aço

Bruno Castanheira, brunoeliazar7@gmail.com
Davi Dias Magalhães, davi.magalhaes@sga.pucminas.br
Felipe Tadeu Góes Guimarães , floptadeu@gmail.com

Resumo

O Problema de Otimização de Cortes em Placas de Aço no setor metalúrgico, focando na minimização do desperdício durante o planejamento de recorte de peças em uma chapa de aço. Nos baseamos em diversas abordagens, incluindo algoritmos gulosos gerados por área e score, algoritmos de Branch and Bound e Programação Dinâmica. A proposta deste estudo é realizar uma análise meticulosa dos resultados obtidos por cada abordagem, visando entender as vantagens específicas que cada algoritmo oferece na resolução desse desafio específico. Ao explorar a eficácia dessas técnicas, buscamos identificar padrões de desempenho de cada um dos algoritmos.

Palavras-chave: Python, Branch And Bound, Busca Gulosa, Busca Dinâmica

1 INTRODUÇÃO

Problemas de otimização são bastante comuns em vários setores produtivos. Escolhemos um setor que sua utilização é indispensável, o setor metalúrgico, em que o planejamento de recorte de peças em uma chapa de aço requer minimizar o desperdício de acordo com o formato dos cortes na chapa pré-definidos. Uma vez que o custo total do processo depende do grau de utilização da matéria prima, um problema relevante consiste em definir a sequencia de peças a serem recortadas, de modo a evitar ao máximo o desperdício do aço. Este trabalho tem o objetivo de mostrar diferentes abordagens para este problema, serão abordadas duas implementações gulosas, uma implementação dinâmica, uma implementação branch and bound e uma implementação, para observar quais algoritmos performam melhor para o problema a seguir.

2 ENTENDIMENTO DO PROBLEMA

A maximização da eficiência no uso de recursos limitados durante o processo de corte de aço em uma chapa é um desafio complexo, abrangendo a otimização da utilização da matéria-prima, a minimização de resíduos e a redução dos custos associados. Nesse sentido, desenvolvemos algoritmos: gulosos, programação dinâmica, de branch and bound para solucionar o desafio de aprimorar os gastos relacionados aos cortes de chapas de aço, todos implementados em Python com o uso das bibliotecas Opencv2 e numpy, visando aperfeiçoar esse procedimento.

3 OBJETIVOS

3.1 Minimização de Desperdício

O custo associado durante o processo de corte está diretamente ligado à eficiência na utilização da matéria-prima. Isso destaca a importância de encontrar uma sequência de recorte que minimize o desperdício de aço e evite os gastos desnecessários.

3.2 Planejamento de Recorte de Peças

As peças a serem cortadas geralmente possuem formas e tamanhos variados. A otimização deve levar em consideração as restrições geométricas, como orientação, tamanho e posição das peças na chapa, para garantir cortes precisos e eficientes.

3.3 Otimização de Recursos Limitados

As empresas do setor metalúrgico enfrentam desafios significativos no gerenciamento eficiente de recursos, especialmente quando se trata da matéria-prima vital, como o aço. A busca por maneiras inovadoras de reduzir custos e minimizar desperdícios tem levado muitas dessas empresas a adotarem estratégias avançadas, com destaque para a otimização no recorte de chapas de ferro. O foco deste trabalho é desenvolver um algoritmo de otimização que maximize a produção e, consequentemente, reduza os gastos associados ao processo.

4 METODOLOGIA

A otimização de cortes de chapa de aço trapezoidais é um problema complexo que se enquadra na categoria de problemas de otimização de corte e embalagem (Cutting and Packing Problems). Este problema é particularmente desafiador devido à forma irregular dos trapezóides e à necessidade de minimizar o desperdício de material. Para resolver este problema fizemos algumas convenções : Consideramos o rolo de metal que gera as chapas como infinito sem nos importarmos se ele poderia acabar ou não. Definimos uma altura constante para o cortes. Definimos todos os formatos das chapas de aço como trapezoidais e resumimos suas medidas em x1,x2 e x3 como pode ser observado na imagem a seguir. O vértice x1 é sempre positivo; x2 e x3 podem ser negativos, mas x2 está sempre à direita de x3. Por fim para esse trabalho precisamos então definir como podemos conseguir os melhores encaixes para gerarmos o menor desperdício possível utilizando algoritmos de otimização para não precisarmos fazer todas as permutações de encaixe possíveis para acharmos a melhor solução. Pode se notar também que uma solução razoável é aceitável para este problema contanto que isso diminua seu gasto computacional.

4.1 Implementação dos Algoritmos

4.1.1 Algoritmo Branch and Bound

O método Branch and Bound é uma técnica de otimização usada para encontrar soluções ótimas em problemas complexos, como a minimização de desperdício em conjuntos de trapézios. Ele explora sistematicamente todas as possíveis configurações dos trapézios, podendo descartar eficientemente certas configurações que não levarão à solução ótima. A abordagem começa dividindo o problema em vários subproblemas menores, o "branching". Cada subproblema representa um possível arranjo de trapézios. Para cada subproblema, o método calcula um limite inferior ou superior do desperdício possível, o "bounding". Se este limite mostrar

que o subproblema não pode produzir uma solução melhor do que a melhor já encontrada, esse ramo é descartado, evitando cálculos desnecessários.

4.1.2 Algoritmo Guloso Gerado por Área

Nos algoritmos greedy geralmente nos preocupamos com a melhor solução local sem se preocupar com a melhor solução global por tanto resultados ótimos podem ser obtidos e resultados razoáveis serão garantidos.Para este algoritmos utilizamos a lógica de que quanto maior a área trapézio menor será desperdício gerado por ela.

4.1.3 Algoritmo Guloso por Score

Para esse algoritmo greedy para arranjo de trapézios, visando minimizar o desperdício, focamos na escolha de trapézios com uma menor diferença entre os cantos inferior direito e superior direito, e com o valor de x3 mais próximo de zero gerando assim um Score. Isso indica trapézios mais regulares, resultando potencialmente em menor desperdício.

4.1.4 Algoritmo Programação Dinâmica

A programação dinâmica é uma técnica eficaz para otimização, utilizada para calcular o desperdício mínimo em conjuntos de trapézios. Ela divide o problema em subproblemas menores, armazenando seus resultados em uma tabela DP (Dynamic Programming). Cada entrada dp[i] na tabela representa o desperdício mínimo para os primeiros i trapézios. Este método calcula o desperdício mínimo uma única vez para cada subconjunto, reutilizando esses resultados, o que aumenta a eficiência do algoritmo, especialmente para grandes conjuntos de trapézios.

5 ANÁLISE DOS RESULTADOS

5.1 Algoritmo Branch and Bound x Programação Dinâmica

Percebe-se na Figura 1 que o algoritmo de programação dinâmica realiza o processo com o mesmo gasto de recursos e com um melhor tempo do que o algoritmo Branch and Bound. É possível observar que o processo sem otimização há quase 4000cm de desperdício com relação aos algoritmos dinâmico e Branch and Bound.

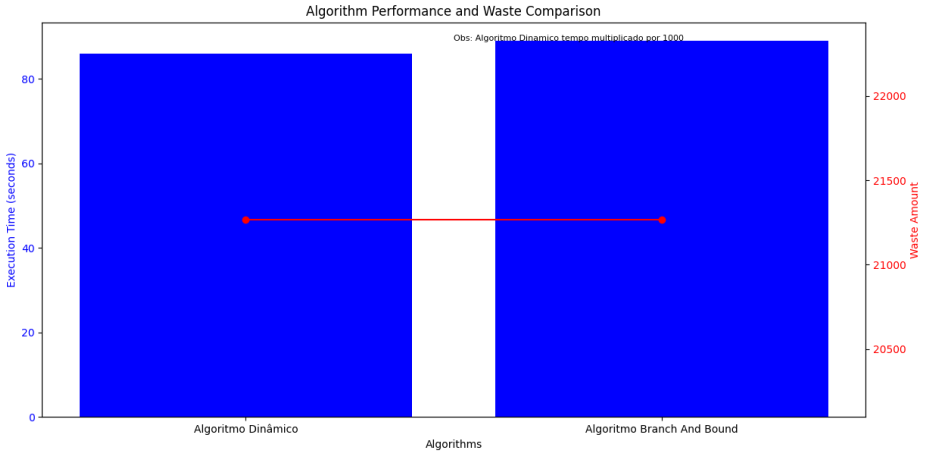

Figura 1 – Tempo de execução x gasto de recurso(cm)

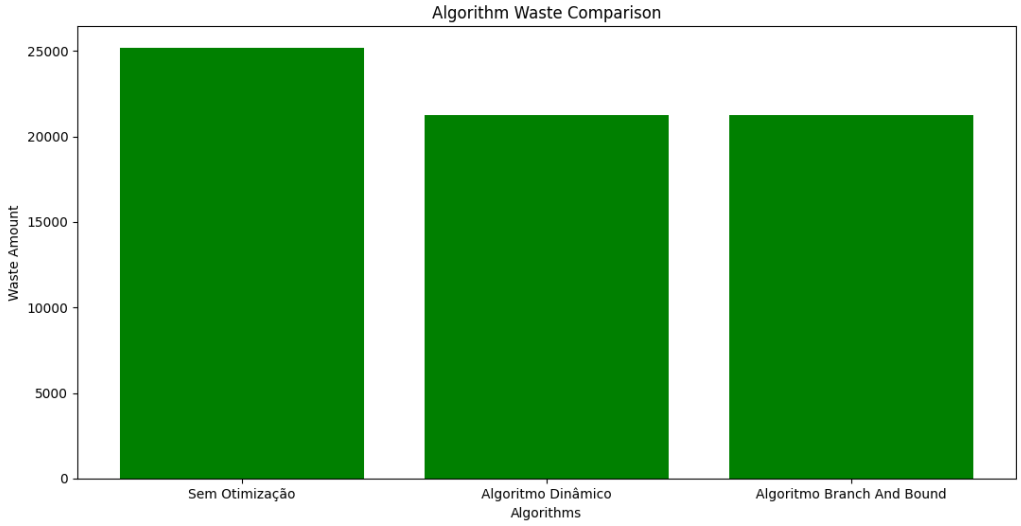

Figura 2 – Gasto de recurso sem otimização; Dinâmico; Branch and Bound

5.2 Algoritmo Guloso Gerado por Área

O algoritmo guloso gerado por área se comportou de forma quase linear, como pode ser observado na figura 3.

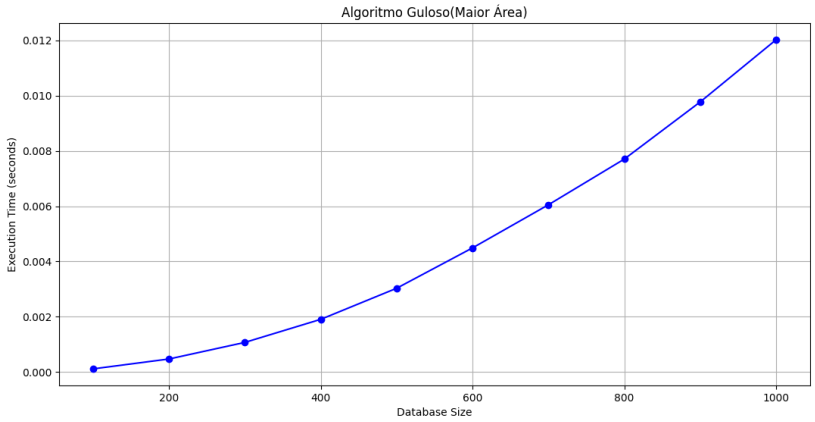

Figura 3 – Tempo de execução x tamanho do database(número de interação de peças))

5.3 Algoritmo Guloso (Gerado por Score)

O algoritmo guloso gerado por score tem alguns picos que revelam certa queda de desempenho com o aumento do tamanho de interação de trapézios.

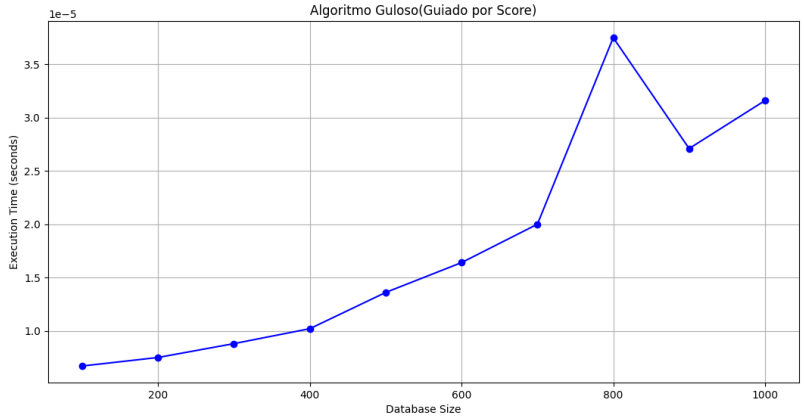

Figura 4 – Tempo de execução x tamanho do database(número de interação de peças)

É possível perceber na figura 5 que houve o mesmo gasto de recurso tanto para ambos os algoritmos gulosos.

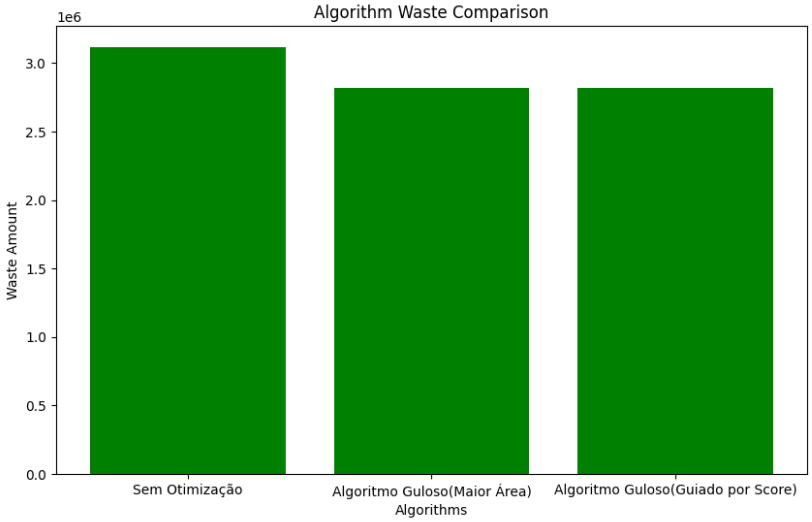

Figura 5 – Gasto de recurso sem otimização; Guloso(Maior Àrea); Guloso Score

5.4 Algoritmo Programação Dinâmica

À medida que o tamanho do banco de dados aumenta, o número de subproblemas (o espaço de estados) que precisam ser considerados também pode aumentar, muitas vezes de forma não linear. Isso pode levar a um aumento exponencial do tempo de execução.Algoritmos de programação dinâmica podem ter uma "transição de fase"em determinados tamanhos de problema, onde um pequeno aumento no tamanho do problema leva a um grande aumento na quantidade de trabalho necessário para encontrar a solução. Isso geralmente está relacionado com a estrutura interna do problema e como os subproblemas interagem.

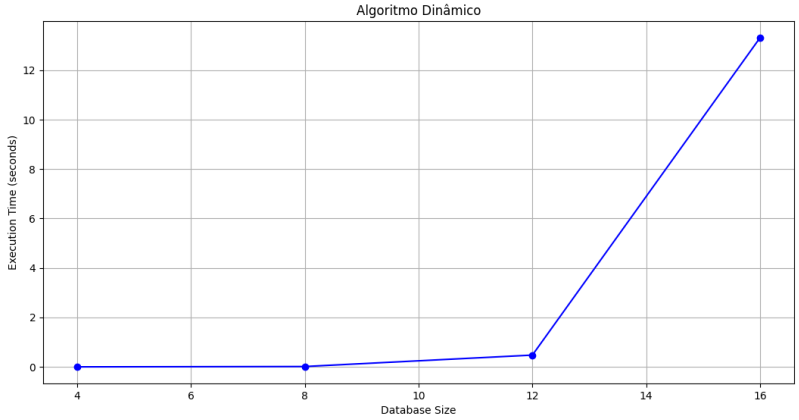

Figura 6 – Tempo de execução x tamanho do database(número de interação de peças)

6 CONCLUSÃO

Podemos concluir que o projeto explorou diferentes algoritmos de otimização para resolver o problema de minimizar o desperdício ao cortar chapas de aço trapezoidais. Os algoritmos testados incluem abordagem gulosa, programação dinâmica e branch and bound.

As imagens mostram os resultados de desempenho e desperdício de cada algoritmo. Os gráficos de linha destacam como o tempo de execução aumenta com o tamanho do conjunto de dados para cada algoritmo. Por exemplo, a programação dinâmica mostra um aumento exponencial no tempo de execução à medida que o tamanho da base de dados cresce, o que é típico para essa abordagem devido à complexidade do problema.
Os gráficos de barras comparam o desperdício gerado por cada algoritmo. Eles ilustram que, embora o algoritmo guloso possa não garantir sempre a solução mais ótima, ele tende a oferecer uma boa relação entre tempo de execução e eficiência, o que pode ser preferível em cenários onde o tempo é um fator crítico.

Por fim, a abordagem gulosa baseada na heurística de área e score (onde um score menor é mais desejável) mostrou ser uma estratégia eficaz para reduzir o desperdício e manter o tempo de execução gerenciável, mesmo para conjuntos de dados maiores. Em suma, cada método de otimização tem seus méritos e a escolha entre eles deve ser baseada em um equilíbrio entre precisão e eficiência computacional, de acordo com as necessidades específicas do problema e os recursos disponíveis.

REFERÊNCIAS

Pessoa, Sadykov, Uchoa, & Vanderbeck. (2020). *Solver for Vehicle Routing and Related Problems*. [Online]. Available: <https://consensus.app/papers/solver-vehicle-routing-related-problems-pessoa/f7b2986454395737afbb3a451827fa91>

Munien & Absalom. (2021). *Algorithms for Binpacking Problems: A Survey of Advances*. [Online]. Available: <https://consensus.app/papers/algorithms-binpacking-problems-survey-advances-munien/43a59beb07b65c648748fc0b369e19ed>

Oliveira & Wäscher. (1995). *Cutting and Packing*. [Online]. Available: <https://consensus.app/papers/cutting-packing-oliveira/80ccb7855f295fb7af309061cf2c821e>

Pessoa, Sadykov, Uchoa, & Vanderbeck. (2019). *Ng-Path Relaxation and Rank-1 Cuts for Vehicle Routing Problems*. [Online]. Available: <https://consensus.app/papers/solver-vehicle-routing-related-problems-pessoa/4e1f2af7bec9518e93496d64c326b1c5>

Bennell & Oliveira. (2009). *Shape Packing Problems*. [Online]. Available: <https://consensus.app/papers/shape-packing-problems-bennell/d8363541d467577992e191841e8fa77b>

Perboli, Gobbato, & Perfetti. (2014). *Packing Problems in Transportation and Supply Chain*. [Online]. Available: <https://consensus.app/papers/packing-problems-transportation-supply-chain-problems-perboli/8384e30a94fd55f2b65877cf7fb54c1a>

7 REPOSITÓRIO DO PROJETO

Os códigos e dados utilizados neste projeto podem ser encontrados no repositório do GitHub:

<https://github.com/floptadeu/Paa_Final>
<https://shorturl.at/hLX07>