Comparação de Algoritmos para o Problema de Corte Bidimensional: Força Bruta, Branch-and-Bound e Heurística

Relatório Técnico

Data: 21 de novembro de 2025

---

1 Introdução

O Problema de Corte de Estoque é um desafio clássico de otimização onde se busca cortar itens menores a partir de um material maior, minimizando o desperdício ou custo. Neste trabalho, abordamos uma variação bidimensional (2D), onde peças retangulares devem ser alocadas em placas de dimensões fixas (300×300).

O objetivo principal é minimizar a função de custo definida por:

$$Custo = (N_{placas} \times 1000) + (0.01 \times PerimetroTotal)$$

O trabalho implementa e compara três estratégias: Força Bruta (solução exata ingênua), Branch-and-Bound (solução exata otimizada) e Heurística Gulosa (solução aproximada), analisando o compromisso entre tempo de execução e qualidade da solução.

2 Solução Proposta

O problema foi modelado como uma busca pela melhor **sequência de entrada** (permutação) das peças. Uma vez definida a ordem, as peças são posicionadas na placa utilizando uma estratégia determinística *Bottom-Left* (primeiro encaixe possível, buscando y e depois x mínimos).

2.1 Algoritmos Utilizados

1. Força Bruta: Explora exaustivamente todas as $N!$ permutações possíveis das peças. Garante a solução ótima global, servindo de base para validação dos outros métodos.

Algoritmo 1 Força Bruta - Permutação Completa

Entrada: Lista de Peças P
Saída: Melhor Custo e Configuração das Placas

$MelhorCusto \leftarrow \infty$
for cada p em Permutações(P) **do**
    $Solucao \leftarrow EncaixarSequencialmente(p)$
    if $Custo(Solucao) < MelhorCusto$ **then**
        $MelhorCusto \leftarrow Custo(Solucao)$
        $MelhorSolucao \leftarrow Solucao$
    end if
end for

---

2. Branch-and-Bound: Utiliza uma árvore de recursão para construir permutações passo a passo. A eficiência vem da poda de ramos inviáveis.

- **Ramificação:** Em cada nível, escolhe qual peça, das restantes, será a próxima a ser encaixada.
- **Poda (Lower Bound):** Se $(CustoAtual + EstimativaRestante) \geq MelhorCustoEncontrado$, o ramo é cortado. A estimativa otimista assume que as peças restantes não exigirão novas placas (custo apenas do perímetro).

Algoritmo 2 Branch and Bound (Recursivo)

```
Função BB_RECURSIVO(PecasRestantes, SolucaoParcial)
    Bound ← Custo(SolucaoParcial) + Estimativa(PecasRestantes)
    if Bound ≥ MelhorCustoGlobal then
        return                                              ▷ Poda (Pruning)
    end if
    if PecasRestantes está vazio then
        MelhorCustoGlobal ← Custo(SolucaoParcial)
        return
    end if
    for cada peca em PecasRestantes do                      ▷ Ramifica nas permutações
        NovaSolucao ← Copiar(SolucaoParcial)
        Tentativa ← Encaixar(NovaSolucao, peca)
        if não Tentativa then
            AdicionarNovaPlaca(NovaSolucao)
            Encaixar(NovaSolucao, peca)
        end if
        NovasRestantes ← PecasRestantes − {peca}
        BB_RECURSIVO(NovasRestantes, NovaSolucao)
    end for
end Função
```

3. Heurística (First Fit Decreasing): Ordena as peças por **Área Decrescente** $(Altura \times Largura)$ e as encaixa sequencialmente. Tenta priorizar peças grandes para reduzir a fragmentação do espaço.

Algoritmo 3 Heurística Gulosa (First Fit Decreasing)

Entrada: Lista de Peças P

```
P ← OrdenarDecrescentePorArea(P)
Placas ← [NovaPlaca()]
for cada peca em P do
    Encaixou ← Falso
    for cada placa em Placas do
        if placa.TemEspaco(peca) then
            placa.Adicionar(peca)
            Encaixou ← Verdadeiro
            break
        end if
    end for
    if não Encaixou then
        Nova ← NovaPlaca()
        Nova.Adicionar(peca)
        Placas.Adicionar(Nova)
    end if
end for
```

3 Implementação

A solução foi desenvolvida em **Python**. O código (`trabalho.py`) está estruturado em classes `Peca` e `Placa` para melhor organização.

3.1 Detalhes Técnicos e Otimizações

- **Representação da Placa:** Matriz booleana 300×300 (`self.matriz`), permitindo verificação de colisão direta.
- **Deep Copy:** A função `copiar_placas` utiliza cópia profunda das listas e objetos para garantir que a recursão do Branch-and-Bound não gere efeitos colaterais entre ramos irmãos.
- **Função de Bound:** A função `calcular_bound` soma o custo das placas já abertas com o custo de corte das peças não alocadas. Isso evita descer em ramos que já custam mais que uma solução completa conhecida.
- **Visualização:** Foi utilizada a biblioteca `matplotlib` para gerar representações gráficas das placas e do posicionamento final das peças, facilitando a validação visual.

4 Relatório de Testes

Os testes foram realizados em um ambiente computacional padrão. Foram executados dois cenários principais variando a quantidade de peças para analisar o crescimento da complexidade.

4.1 Cenário 1: 5 Peças (Entrada Padrão)

Tabela 1: Comparativo de Desempenho (5 Peças)

Algoritmo	Custo (R$)	Placas	Nós/Permut.	Tempo (s)
Força Bruta	2031.40	2	120 (permutações)	13.0778
Branch and Bound	2031.40	2	91 (nós)	1.4598
Heurística	2031.40	2	1 (ordenação)	0.0916

4.2 Cenário 2: 6 Peças (Teste de Estresse)

Tabela 2: Comparativo de Desempenho (6 Peças)

Algoritmo	Custo (R$)	Placas	Nós/Permut.	Tempo (s)
Força Bruta	2030.00	2	720 (permutações)	46.0107
Branch and Bound	2030.00	2	772 (nós)	13.5469
Heurística	2030.00	2	1 (ordenação)	0.0736

4.3 Análise dos Resultados

- **Convergência e Correção:** Em ambos os cenários, todos os algoritmos encontraram a solução ótima (2 placas). O Branch-and-Bound (B&B) confirmou sua exatidão ao igualar o custo da Força Bruta.
- **Crescimento Exponencial:**
  - Ao aumentar de 5 para 6 peças, o espaço de busca da Força Bruta cresceu de 120 para 720 permutações ($6\times$ maior).
  - O tempo da Força Bruta triplicou (de 13s para 46s), evidenciando a inviabilidade para $N \geq 8$.
  - O B&B também sofreu aumento de tempo (de 1.4s para 13.5s), mas manteve-se significativamente mais rápido que a Força Bruta (cerca de 3.4 vezes mais rápido no cenário de 6 peças).
- **Comparação de Nós:** No cenário de 6 peças, o B&B explorou 772 nós. Embora este número seja maior que as 720 folhas da Força Bruta, o processamento foi mais rápido. Isso ocorre porque muitos nós do B&B são parciais e podados cedo, enquanto a Força Bruta avalia 720 soluções completas e custosas.

5 Conclusão

Os resultados confirmam empiricamente a teoria de complexidade. A **Força Bruta** ($O(N!)$) torna-se proibitiva com o aumento marginal da entrada. O **Branch-and-Bound**, apesar de também possuir complexidade de pior caso fatorial, demonstra na prática ser uma estratégia robusta para instâncias médias, reduzindo drasticamente o tempo de execução através da poda eficiente de ramos. Para instâncias maiores ou aplicações em tempo real, a **Heurística** mostrou-se imbatível em tempo ($< 0.1s$), embora não ofereça garantias matemáticas de otimalidade.

6 Bibliografia

CORMEN, T. H. et al. **Introduction to Algorithms**. 3rd ed. MIT Press, 2009.
ARENALES, M. et al. **Pesquisa Operacional**. Rio de Janeiro: Elsevier, 2007.
GOLDBARG, M. C.; LUNA, H. P. L. **Otimização Combinatória e Programação Linear**. Rio de Janeiro: Campus, 2005.