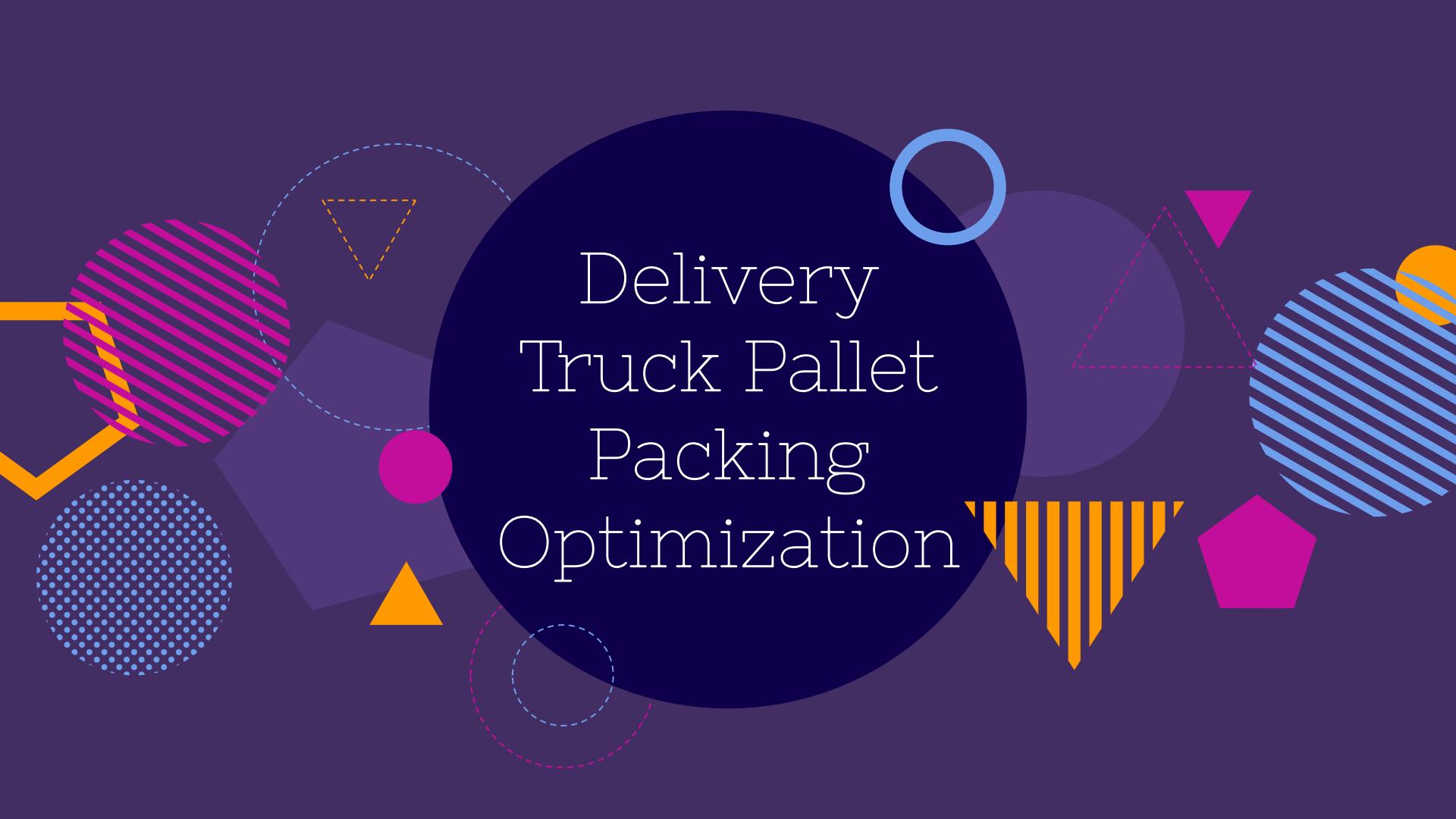


Delivery Truck Pallet Packing Optimization

The background features a dark purple gradient with various abstract geometric shapes in blue, yellow, and pink. These shapes include circles, triangles, and rectangles with different patterns like stripes and dots. Some shapes have dashed outlines, while others are solid. A large blue circle containing the main text is positioned in the center-right area.

Projeto 2

Design de Algoritmos

Amanda Araujo Silva - up202411286

João Pedro Câmara Furtado - up202305812

Objetivos

- Promover uma solução para uma variação do mundo real do problema 0/1 Knapsack
- Múltiplas abordagens:
 - Busca exaustiva (força bruta)
 - Programação dinâmica
 - Algoritmos de aproximação & heurística
 - Algoritmo guloso
 - Algoritmo Híbrido

Motivação

Problema real: em logística e gestão, carregar caminhões de entrega com eficiência é um desafio crítico.

- ⦿ Maximização do valor de remessa
- ⦿ Em respeito às restrições de peso



Motivação



Modelagem do problema

- ⦿ Knapsack ⇄ Caminhão
- ⦿ Itens ⇄ Pallets
 - Peso específico
 - Lucro baseado no valor dos produtos

Objetivo: selecionar subconjunto de pallets que maximiza o lucro total sem exceder a capacidade de carga do caminhão.

Motivação

Ganhar *insights* sobre os *trade-offs* em
problemas reais
optimalidade \leftrightarrow eficiência \leftrightarrow viabilidade



Leitura dos *Datasets*

Conjuntos de dados .csv representando instâncias do Problema de Delivery Truck Pallet Packing:

- **TruckAndPallets_X.csv**

Capacity,	Pallets
100,	9

- **Pallets_X.csv**

Pallet,	Weight,	Profit
1,	70,	10
2,	60,	5
3,	50,	10
...		

Funcionalidades e Estrutura

Desenvolvimento de uma ferramenta de otimização de Pallet Packing

→ Menu: seleção do *dataset*, abordagem algorítmica e visualização dos resultados

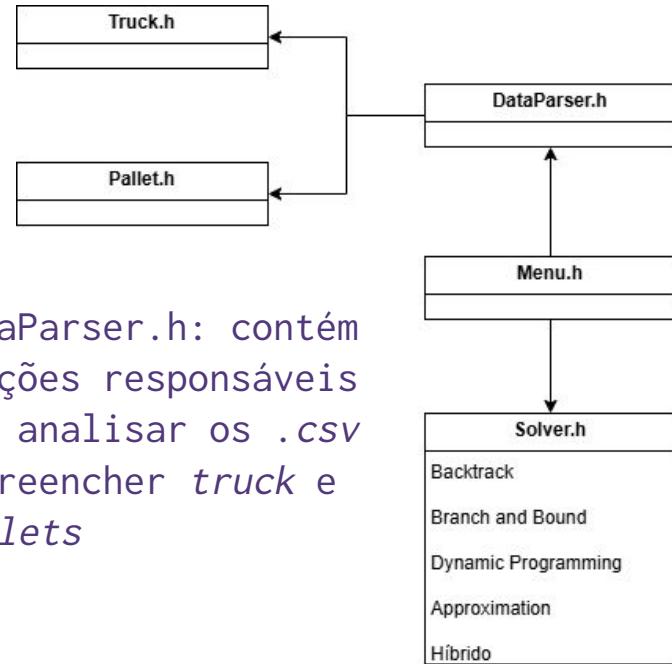
1. Busca exaustiva (força bruta)
2. Programação dinâmica
3. Aproximação gulosa
4. Algoritmo Híbrido

Funcionalidades e Estrutura

Organização da ferramenta com o uso de classes:

- **Truck.h**
 - maxCapacity
 - palletsCapacity
- **Pallet.h**
 - id ○ profit
 - weight ○ ratio
- **Solver.h**: funções dos algoritmos de solução
- **Menu.h**: interface UI

- **DataParser.h**: contém funções responsáveis por analisar os .csv e preencher *truck* e *pallets*



Interface usuário – Menu

Menu principal

```
==== MAIN MENU ====
```

1. Load Data Files
2. Select Solver Algorithm
3. Compare to optimal
4. View Results
6. Exit

Select option:

Interface usuário – Menu

Seleção do *dataset*

```
==== DATA LOADING ===

Enter dataset identifier (X): 05
Successfully loaded dataset 05!
Truck file: ../data/datasets-extra/TruckAndPallets_05.csv
Pallet file: ../data/datasets-extra/Pallets_05.csv
15 30

Press Enter to continue...
```

Interface usuário – Menu

Seleção da abordagem (*solver*)

```
==== SELECT SOLVER ===
```

- 1. Approximation → Aproximação gulosa
- 2. Backtracking ↗ Força bruta
- 3. Branch and Bound ↘
- 4. Dynamic Programming
- 5. Hybrid
- 6. Return to Main Menu

```
Select solver type:
```

Abordagem – Busca exaustiva Backtracking

Esta é uma estratégia de força bruta onde, em cada estado da árvore de decisões, decidimos se queremos ou não incluir o pacote atual.

Dado um conjunto de dados com tamanho considerável, o algoritmo pode nunca terminar, pois faz um número excessivo de cálculos – tornando-se, assim, inviável.

Ainda assim, é extremamente útil para conjuntos de dados pequenos.

Abordagem – Busca exaustiva Backtracking

Complexidade temporal: $O(2^n)$

→ Exploração de todos os estados possíveis

Complexidade espacial: $O(n)$ → *Call stack* recursivo

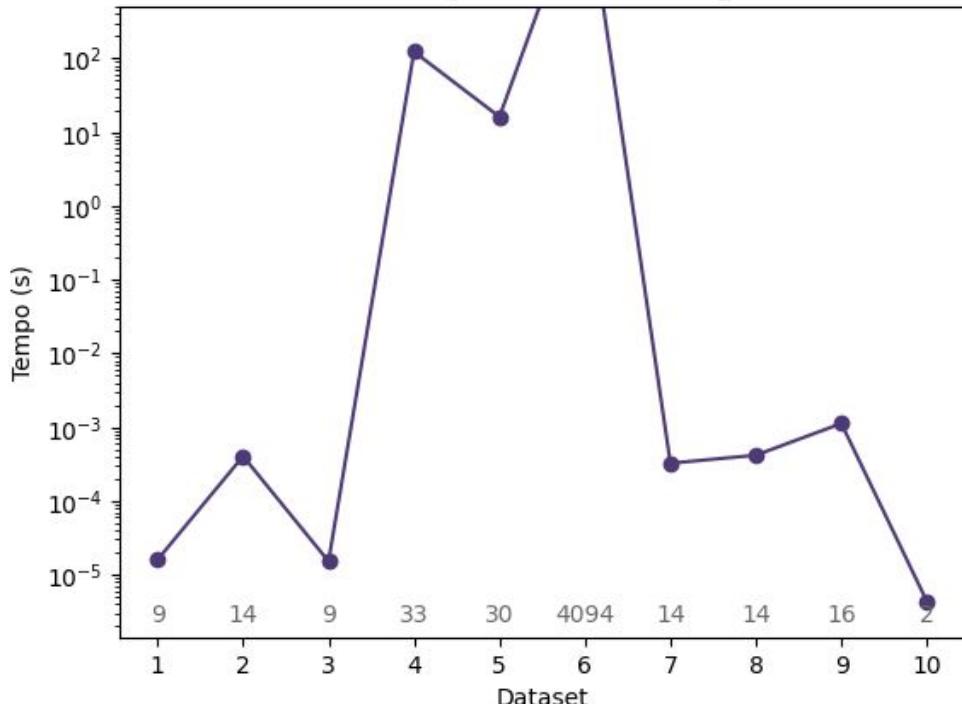


Devolve sempre a solução ótima para o problema;
Trade-off: eficiência por optimalidade.

- Como é exponencial, isto levanta problemas para *datasets* maiores.

Abordagem – Busca exaustiva Backtracking

Desempenho Backtracking



Abordagem – Busca exaustiva Branch and Bound

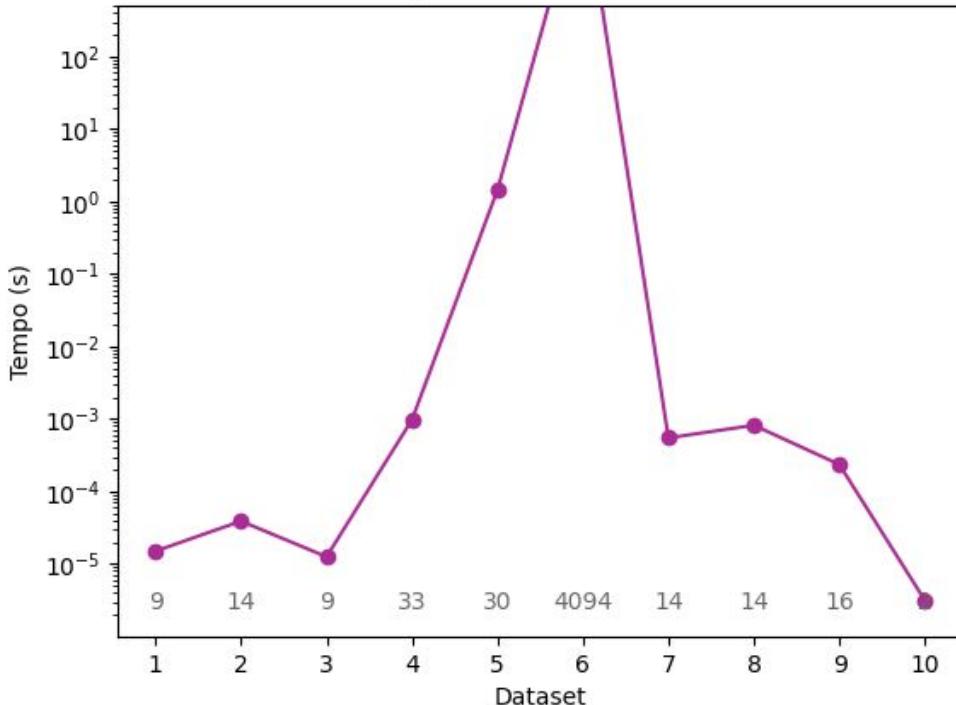
Estratégia de força bruta (*backtracking* melhorado): eliminação inteligente de *branches* inviáveis, ou seja, que não nos darão um resultado melhor do que o atual.

Complexidade teórica mantida, na prática: mais rápido.

- Num dado estado: soma atual e soma dos pacotes restantes;
- A ordenação das *pallets* restantes pela razão (v/w) permite o cálculo de um limite superior teórico para o estado;
- Fazemos um *Knapsack* fracionário, onde podemos incluir apenas parte de um pacote;
- Se o limite superior for inferior ao nosso melhor valor atual, então esse ramo é eliminado.

Abordagem – Busca exaustiva Branch and Bound

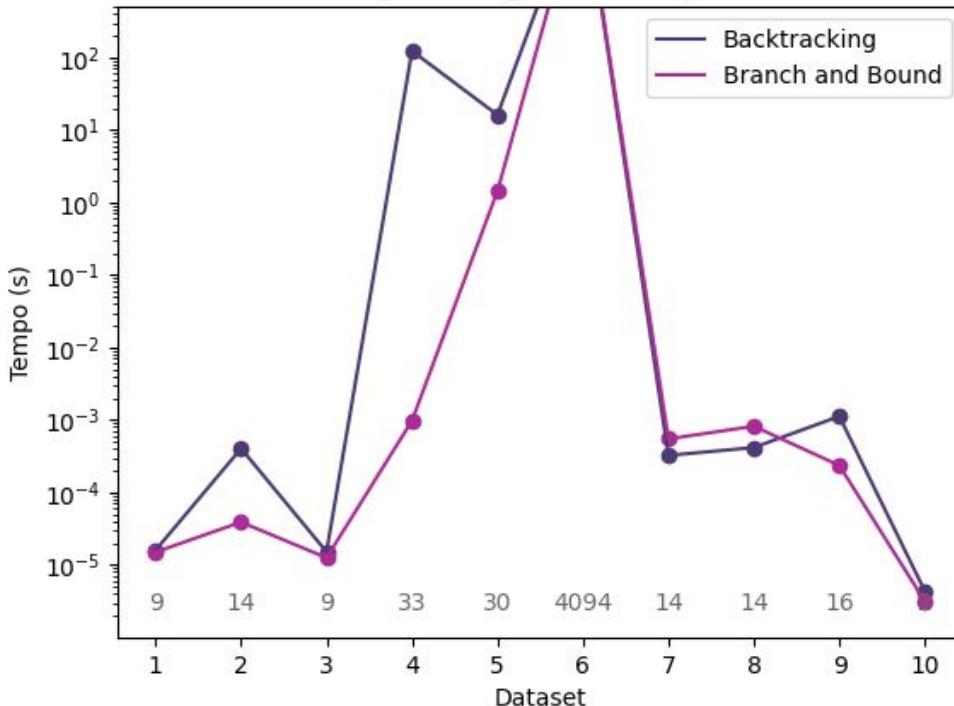
Desempenho Branch and bound



Abordagem – Busca exaustiva

Backtracking vs. Branch and Bound

Desempenho algoritmos força bruta



Abordagem 2 – Programação dinâmica

Esta estratégia utiliza uma abordagem *bottom-up* de programação dinâmica para resolver o problema de forma eficiente.

- Construção da tabela de memorização 2D $dp[w][k]$
 - $dp[i][j]$: lucro máximo possível com capacidade i ($i \leq w$), considerando os primeiros j ($j \leq k$) *pallets*
- Estrutura auxiliar: rastreio dos *pallets* usados em cada subproblema → reconstrução da solução ótima.

Para cada palete, decidimos se o incluímos ou não, comparando o lucro resultante em cada caso.

Abordagem 2 – Programação dinâmica

Complexidade temporal: $O(nW)$

→ Algoritmo pseudo-polynomial: polinomial no valor de W , mas exponencial no tamanho de W , pois são necessários $\log(W)$ bits para representar esse valor.

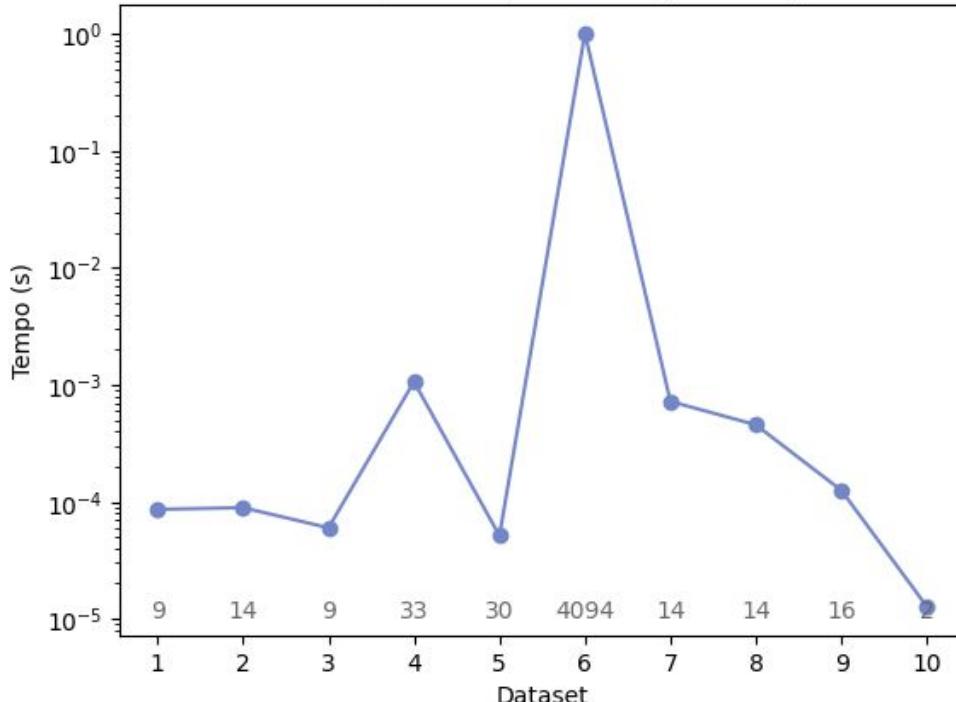
Complexidade espacial: $O(nW)$

→ Tabela dimensões $n \times W$



Abordagem 2 – Programação dinâmica

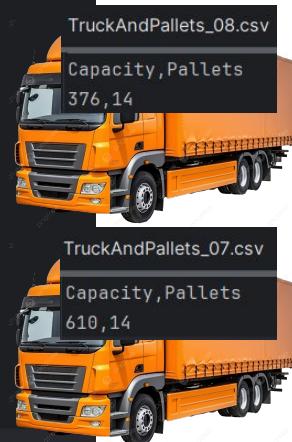
Desempenho Dynamic Programming



Abordagem 2 - Programação dinâmica

Pallet	Weight	Profit
1,	1,	1
2,	1,	2
3,	2,	3
4,	3,	5
5,	5,	8
6,	8,	13
7,	13,	21
8,	21,	34
9,	34,	55
10,	55,	89
11,	89,	144
12,	144,	233
13,	233,	377
14,	377,	610

Pallet	Weight	Profit
1,	1,	1
2,	1,	2
3,	2,	3
4,	3,	5
5,	5,	8
6,	8,	13
7,	13,	21
8,	21,	34
9,	34,	55
10,	55,	89
11,	89,	144
12,	144,	233
13,	233,	377
14,	377,	610



Caso interessante: dataset 07 e 08

Pseudo-polynomial $O(nW)$ | $W = 610$ vs. $W = 376$

Tempo de execução esperado: 07 ~2x 08 ✓

Dataset 07: $W = 610$, $n = 14$

```
Select solver type: 4
Best profit: 987
Pallets used (IDs): 1 2 4 6 8 10 12 14
Time taken by solveDP: 0.000711148 seconds
```

Dataset 08: $W = 376$, $n = 14$

```
Select solver type: 4
Best profit: 608
Pallets used (IDs): 1 2 3 4 5 6 7 8 9 10 11 12
Time taken by solveDP: 0.00035248 seconds
```

Abordagem 3 – Aproximação Knapsack

Esta estratégia utiliza duas heurísticas *greedy*:

- Escolha do elemento com maior proporção v/w
- Escolha sempre no elemento de maior valor v

$$\max(\text{ratioBased}, \text{profitBased})$$

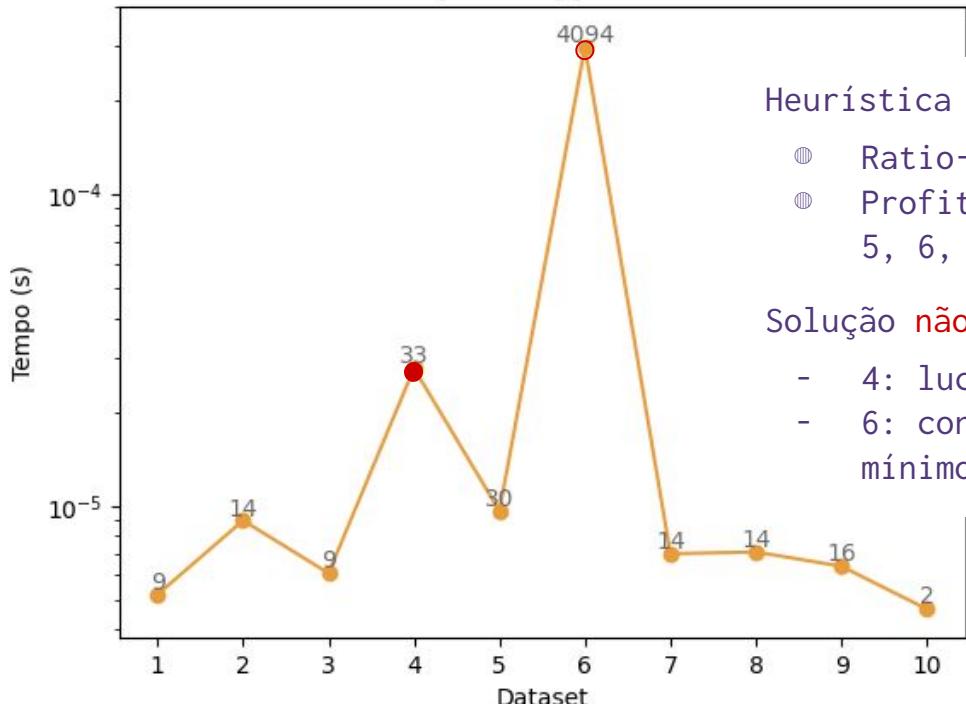
Complexidade temporal: $O(n \log(n))$

Complexidade espacial: $O(n)$

- Não garante o valor ótimo, mas é uma aproximação de factor 2, ou seja, o máximo que ele pode estar afastado do valor optimal é por um factor 2.

Abordagem 3 – Aproximação Knapsack

Desempenho Approximation



Heurística greedy:

- Ratio-based (v/w): 4, 9
- Profit-based (v): 1, 2, 3, 5, 6, 7, 8, 10

Solução não ótima:

- 4: lucro não máximo
- 6: conjunto de *pallets* não mínimo

Abordagem 4 – Algoritmo Híbrido

Casos de uso do algoritmo híbrido proposto:

- Para *datasets* menores ($n \leq 30$) usamos a estratégia de DP, para eficientemente encontrar a solução óptima.
- Para *datasets* maiores usamos uma estratégia de *Greedy Ratio* combinado com *Local Search Improvement* (LSI).
- Nesta nova estratégia, depois de calcular a solução do Greedy, vamos tentar melhorar a solução, invertendo cada *pallet*, ou seja, se o pacote foi incluído retiramos, se não foi incluímos e calcularmos a fim de ver se melhora a nossa solução. Fazemos isso para cada palete.

Abordagem 4 – Algoritmo Híbrido

Complexidade

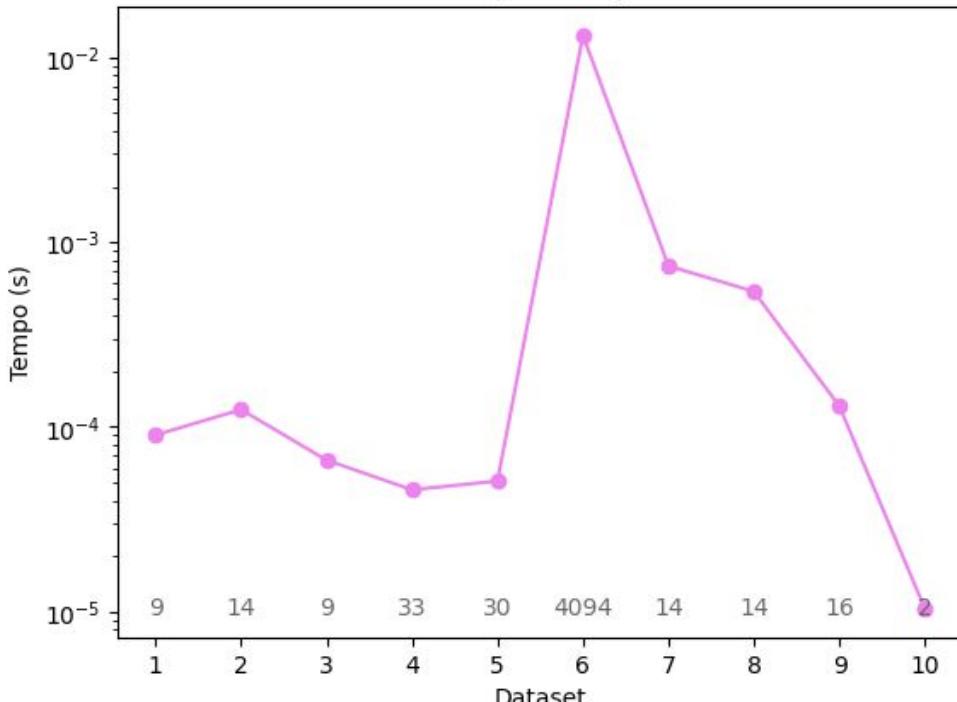
- *Datasets* pequenos: recai naquela do DP
- *Datasets* maiores: a complexidade espacial é $O(n^2)$, visto que o Greedy é $O(n \log(n))$, e quando tentamos melhorar a solução isso implica atravessar o vetor de *pallets*, para cada *pallet* e ver as soluções.

Ideia por trás do algoritmo híbrido proposto:

- Usamos este tipo de algoritmo porque para pequenos *datasets* utiliza uma estratégia eficiente para eles, e para grandes utiliza uma estratégia rápida e depois um algoritmo que consegue refinar resultados sub-optimais.

Abordagem 4 – Algoritmo Híbrido

Desempenho Hybrid



Casos Hybrid:

- DP ($n \leq 30$): 1, 2, 3, 5, 7, 8, 9, 10
- Greedy + Local search for large instance: 4, 6

Avaliação de desempenho

Comparação entre as diferentes soluções obtidas pela ferramenta de otimização de Pallet Packing

- Tempo de execução
- Complexidade espacial
- Acurácia da solução
 - ◆ Quão próxima a solução do algoritmo guloso está da solução ótima

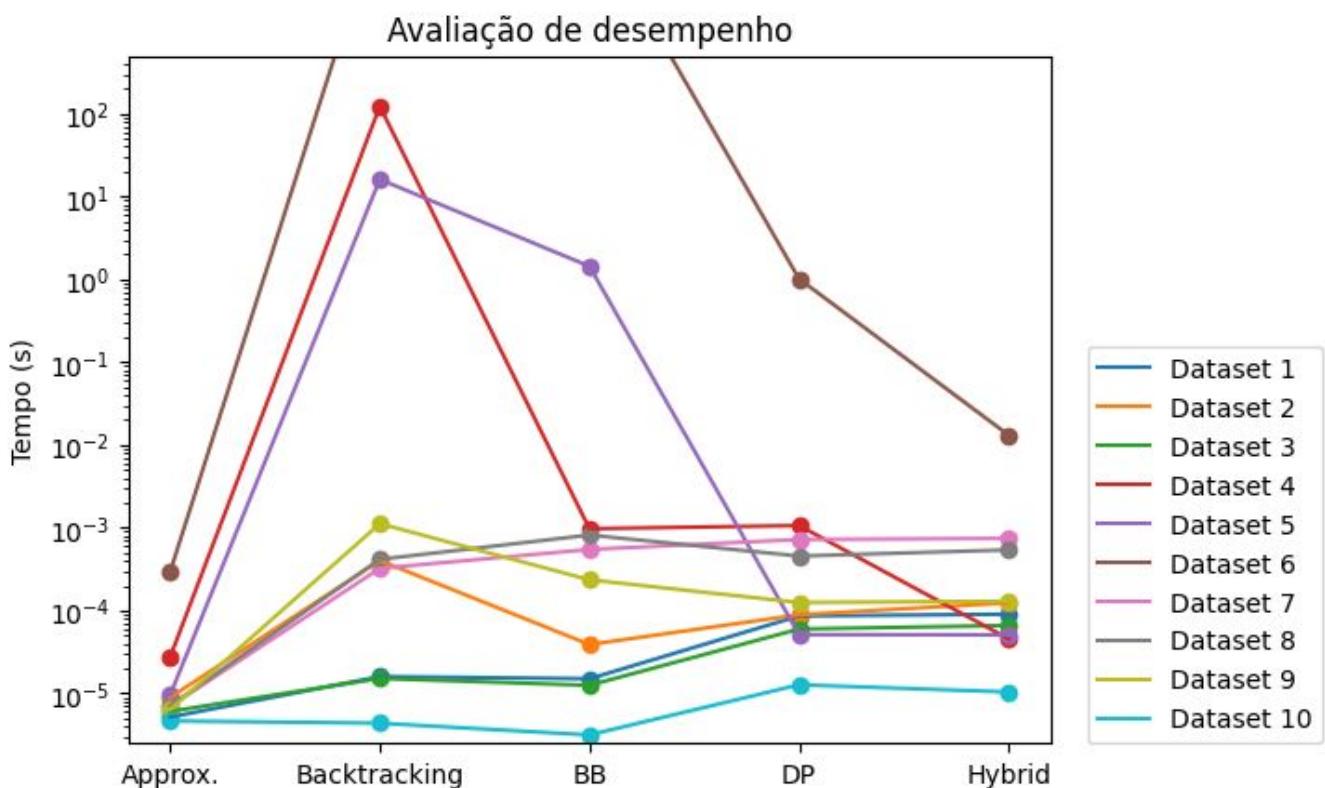
Justificação da estrutura de dados

Utilizamos o vector fornecido pela biblioteca de cpp <vector>.

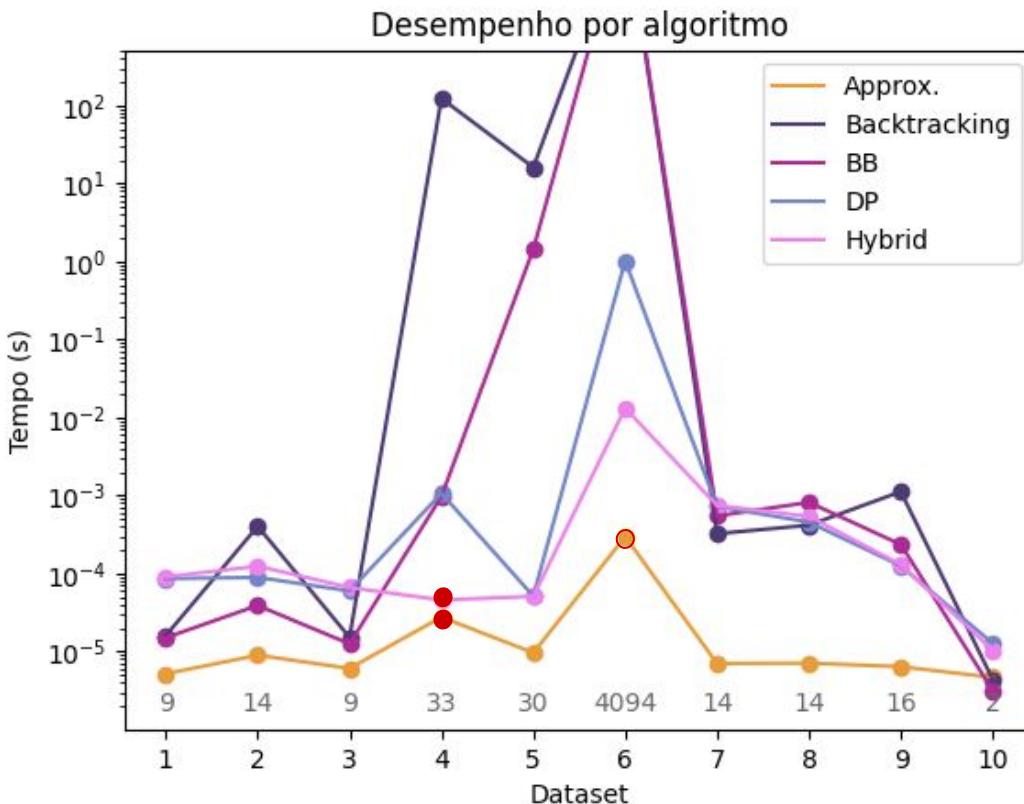
Esta E.D. foi extremamente útil para armazenar os IDs das paletes usados e assim depois formular a melhor escolha de paletes que resultam no maior resultado.

É de destacar que apresenta a capacidade de comparar dois vectores, dado que têm o operador “<” overloaded fazendo com que consiga comparar os conteúdos de vectores, o que nos ajudou no tie-breaking do projeto onde um dos critérios era selecionar o vector com menor tamanho, e, se tivessem igual, usava-se aquele que apresentava os menores IDs.

Tempo de execução



Tempo de execução



*Solução não
ótima: 4

Tempo de execução

- ◆ Dataset 06 ($n = 4094$, $W = 2047$) *Muito tempo!

Pallet	Weight	Profit
1,	1024,	1024
2,	1024,	1024
3,	512,	512
4,	512,	512
5,	512,	512
6,	512,	512
7,	256,	256
8,	256,	256
9,	256,	256
10,	256,	256
11,	256,	256
12,	256,	256
13,	256,	256
14,	256,	256
15,	128,	128
16,	128,	128

OptimalSolution_06.txt
1, 1024, 1024
3, 512, 512
7, 256, 256
15, 128, 128
31, 64, 64
63, 32, 32
127, 16, 16
255, 8, 8
511, 4, 4
1023, 2, 2
2047, 1, 1

- Algoritmos de força bruta não foram capazes de retornar uma solução em tempo hábil;
- Alta variância nos tempos de execução:
 - DP → 1.016s
 - Hybrid → 1.31e-02s
 - Approx. → 2.96e-04s

Solução ótima: lucro máximo 2047 ✓

Pallets: ✓✓●

Pallets used (IDs): 1 3 7 15 31 63 127 255 511 1023 2047

DP

Pallets used (IDs): 1 3 7 15 31 63 127 255 511 1023 2047

Hybrid

Pallets used (IDs): 2 3 7 15 62 79 142 272 526 1039 2062

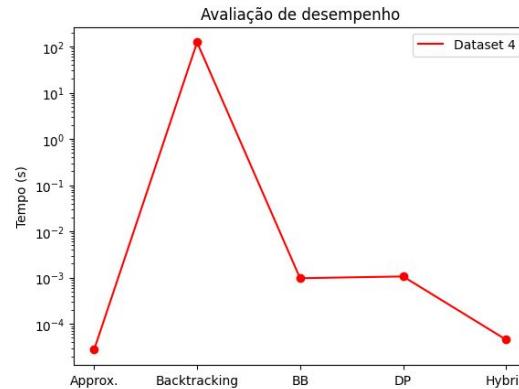
Approx.

Tempo de execução

- ◆ *Dataset 04* ($n = 33$, $W = 524$) *Solução não ótima

Pallet,Weight,Profit	
1,442,20	17,106,5
2,252,20	18,85,5
3,252,20	19,84,5
3,252,20	20,46,5 ←
4,252,25	21,37,5
5,252,25	22,37,8
6,252,30	23,12,8
7,252,30	24,12,8
8,252,22	25,12,2 ←
9,127,21	26,10,2
10,127,20	27,10,10
11,127,20	28,10,20
12,127,10	29,10,20
13,127,10	30,10,10
14,106,19	31,10,2
15,106,18	32,9,3
16,106,10	33,9,7

- Alta variância nos tempos de execução:



$10^{-5} - 10^2$

- Soluções não ótimas: Approx. & Hybrid (*greedy - ratio*)

Tempo de execução

- ⦿ Backtracking
- ⦿ Branch and bound
- ⦿ Dynamic Programming
- ⦿ Approximation



$$\begin{aligned}\mathcal{O}(2^n) \\ \mathcal{O}(nW) \\ \mathcal{O}(n\log(n))\end{aligned}$$

- ⦿ Hybrid
 - ⦿ Datasets ($n \leq 30$):
 - ⦿ Datasets grandes:

$$\begin{aligned}\mathcal{O}(nW) \\ \mathcal{O}(n^2)\end{aligned}$$

Comparação de Estratégias

Busca exaustiva

Backtracking & Branch and Bound

- Garante o resultado ótimo; para o nosso problema, mas para *datasets* maiores explode;
- Backtracking pode ser melhorado com Branch and Bound.

Programação Dinâmica

- Retorna o resultado ótimo;
- Funciona para *datasets* maiores, porém para W muito grande o algoritmo explode:
 - Pseudo-polynomial
- Costuma ser mais rápido do que o *Backtracking*.

Comparação de Estratégias

Approximation

- Muito mais rápido do que os outros algoritmos;
- Resultados sub-óptimos para alguns *datasets*;
- Factor 2 do resultado ótima;
- Funciona para *datasets* de qualquer tamanho.

Hybrid

- Capaz de adaptar-se para o tamanho do *dataset*;
- Eficiente para todos os tamanhos (diferentes complexidades);
- Datasets grandes: solução ótima não garantida → Uso heurística greedy, mas depois é melhorada no LSI.

Complexidade espacial

Busca exaustiva

Backtracking

$$\mathcal{O}(n)$$

Branch-and-bound

$$\mathcal{O}(n)$$

Programação dinâmica

$$\mathcal{O}(nW)$$

Aproximação

$$\mathcal{O}(n)$$

Híbrido

$$\mathcal{O}(nW) \quad \mathcal{O}(n)$$

Acurácia da solução

- ◆ Quão próxima a solução do algoritmo guloso está da solução ótima?

→ Comparação com backtracking

Dataset 1 ✓

```
== Approximation Result ==
Strategy used: Profit-based Greedy
Total profit: 29
Time Taken by Solver:5.442e-06
Pallets used (IDs): 3 5 7 9
```

```
Select solver type: 2
IDs of pallets used: 3,5,7,9,
Time taken by solveCase1: 1.3027e-05 seconds
Best Total Profit 29
```

Dataset 2 ✓

```
== Approximation Result ==
Strategy used: Profit-based Greedy
Total profit: 32
Time Taken by Solver:5.227e-06
Pallets used (IDs): 4 11 12 14 13
```

```
Select solver type: 2
IDs of pallets used: 4,11,12,13,14,
Time taken by solveCase1: 0.000286952 seconds
Best Total Profit 32
```

Dataset 3

```
== Approximation Result ==
Strategy used: Ratio-based Greedy
Total profit: 40
Time Taken by Solver: 5.773e-06 se
Pallets used (IDs): 9 8 6 7
```

```
Select solver type: 2
IDs of pallets used: 5,6,8,9,
Time taken by solveCase1: 1.8686e-05 seconds
Best Total Profit 40
```

Acurácia da solução

- ◆ Quão próxima a solução do algoritmo guloso está da solução ótima?

Dataset 5 ✓

```
== Approximation Result ==
Strategy used: Profit-based Greedy
Total profit: 15
Time Taken by Solver:8.055e-06
Pallets used (IDs): 1 3 7 15
```

OptimalSolution_05.txt		
1	1, 8, 8	
2	3, 4, 4	
3	7, 2, 2	
4	15, 1, 1	

Dataset 6

```
== Approximation Result ==
Strategy used: Profit-based Greedy
Total profit: 2047
Time Taken by Solver: 0.000352561
Pallets used (IDs): 2 3 7 15 62 79 142 272 526 1039 2062
```

OptimalSolution_06.txt	
1,	1024, 1024
3,	512, 512
7,	256, 256
15,	128, 128
31,	64, 64
	63, 32, 32
	127, 16, 16
	255, 8, 8
	511, 4, 4
	1023, 2, 2
	2047, 1, 1

Dataset 7 ✓

```
== Approximation Result ==
Strategy used: Profit-based Greedy
Total profit: 987
Time Taken by Solver:6.97e-06
Pallets used (IDs): 14 13
```

OptimalSolution_07.txt		
13,	233,	377
14,	377,	610

Dataset 8 ✓

```
== Approximation Result ==
Strategy used: Profit-based Greedy
Total profit: 608
Time Taken by Solver:5.532e-06
Pallets used (IDs): 13 11 9 7 5 3
```

OptimalSolution_08.txt		
3,	2,	3
5,	5,	8
7,	13,	21
9,	34,	55
11,	89,	144
13,	233,	377

Dataset 9 ✓

```
== Approximation Result ==
Strategy used: Ratio-based Greedy
Total profit: 945
Time Taken by Solver:2.5452e-05
Pallets used (IDs): 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
```

OptimalSolution_09.txt		
2	a	16

Dataset 10 ✓

```
== Approximation Result ==
Strategy used: Profit-based Greedy
Total profit: 15
Time Taken by Solver:3.188e-06
Pallets used (IDs): 2
```

OptimalSolution_10.txt		
2,	16,	15

Acurácia da solução

- ◆ Quão próxima a solução do algoritmo guloso está da solução ótima?

Dataset 04: resultado não ótimo

- ➊ Approximation & Hybrid (Greedy + LSI): 158
- ➋ Ótimo (Branch and Bound): 161

```
Select solver type: 1
===
Approximation Result ===
Strategy used: Ratio-based Greedy
Total profit: 158
Time Taken by Solver:1.0294e-05
Pallets used (IDs): 28 29 27 30 33 24 23 32 22 26 31 14 15 25 9
```

```
Select solver type: 5
[Hybrid] Using Greedy + Local Search for large instance (n = 33)
Hybrid Knapsack Solver Runtime: 3.4396e-05 seconds
Total Profit: 158
Selected Pallets (IDs): 28 29 27 30 33 23 24 32 22 26 31 14 15 25 9
```

```
Select solver type: 3
IDs of pallets used: 9,14,15,20,22,23,24,26,27,28,29,30,31,32,33,
Time taken by solveCase2: 0.000733478 seconds
Best Total Profit 161
```

*Falta pallet 20 (25 no lugar)!

Aproximação de fator 2:

$$\max\left(\frac{C^*}{C}, \frac{C}{C^*}\right) \leq 2 ; 161/158 < 2$$

Funcionalidade a destacar

Implementação de um recurso que armazena todos os resultados feitos para determinado *dataset* e permite ao usuário fornecer um id e verificar os cálculos feitos até o momento:

- *Solver*
- Resultado (lucro) encontrado

```
== MAIN MENU ==

1. Load Data Files
2. Select Solver Algorithm
3. View Results
4. Exit

Select option: 3
Enter dataset identifier to view results: 04

Results for Dataset: 04
Solver: Approximation
Result: 158
-----
Solver: Branch and Bound
Result: 161
-----
Solver: Dynamic Programming
Result: 161
-----
```

Conclusão

Este projeto revelou-se bastante educativo ao que resguarda a utilização de diferentes estratégias para resolver um problema NP, e os *trade-offs* entre optimalidade por complexidade.

A documentação apresentou-se relativamente complexa neste trabalho, quando chega a fazer os gráficos e a análise completa de cada estratégia e depois a comparação com todas as outras.

Implementação bem sucedida de uma ferramenta de Delivery Truck Pallet Packing Optimization com o uso de diferentes abordagens algorítmicas para resolução de problemas realísticos.



Obrigada!

== EXITING PROGRAM ==

Thank you for using Knapsack Solver!



