Instituto Tecnológico de Aeronáutica
Divisão de Ciência da Computação

CTC-12

Laboratório 04

Paradigma SSP

Aluno:

Daniel Araujo Cavassani

Professor:

Luiz Gustavo Bizarro Mirisola

1) Problema SSP

O problema de SSP foi resolvido com o uso de 2 algoritmos diferentes, sendo que ambos foram comparados entre si e entre uma aplicação usada profissionalmente (DD), que aparecerá nos gráficos a título de comparação. Os algoritmos utilizados foram o de Backtracking e o de Programação Dinâmica (PD).

a. Programação Dinâmica

A implementação foi feita utilizando-se do Princípio de Otimalidade de Bellman:

> "em uma sequência ótima de escolhas ou de decisões, cada subsequência também deve ser ótima. Ela reduz drasticamente o número total de verificações, pois evita aquelas que sabidamente não podem ser ótimas: a cada passo são eliminadas subsoluções que certamente não farão parte da solução ótima do problema."

Nesse caso, o problema se resume a preencher uma tabela de n por v, em que n representa o número de itens disponíveis e v representa o valor que se deseja chegar ao somar os pesos dos tais itens. Ao preenchermos a tabela, o último valor preenchido é uma booleana que diz se é ou não possível de se resolver o problema, sendo que são necessários n*v passos, indiscutivelmente, para chegar à conclusão final. Em suma, o algoritmo:

1 – Cria uma matriz de programação dinâmica com dimensões (n+1) x (v)

2 – Inicializa-se a primeira coluna da matriz com True

3 – Para cada elemento do conjunto, percorre-se cada célula da matriz. Se o elemento for maior do que o valor atual da célula, copia-se o valor da célula acima. Caso contrário, verifica-se se é possível formar o valor atual com ou sem incluir o elemento. Se uma das condições for verdadeira, marca-se a célula com True, caso contrário, marca-se com False.

4 – Ao fim, o resultado é dado pelo valor da última célula da última linha da matriz

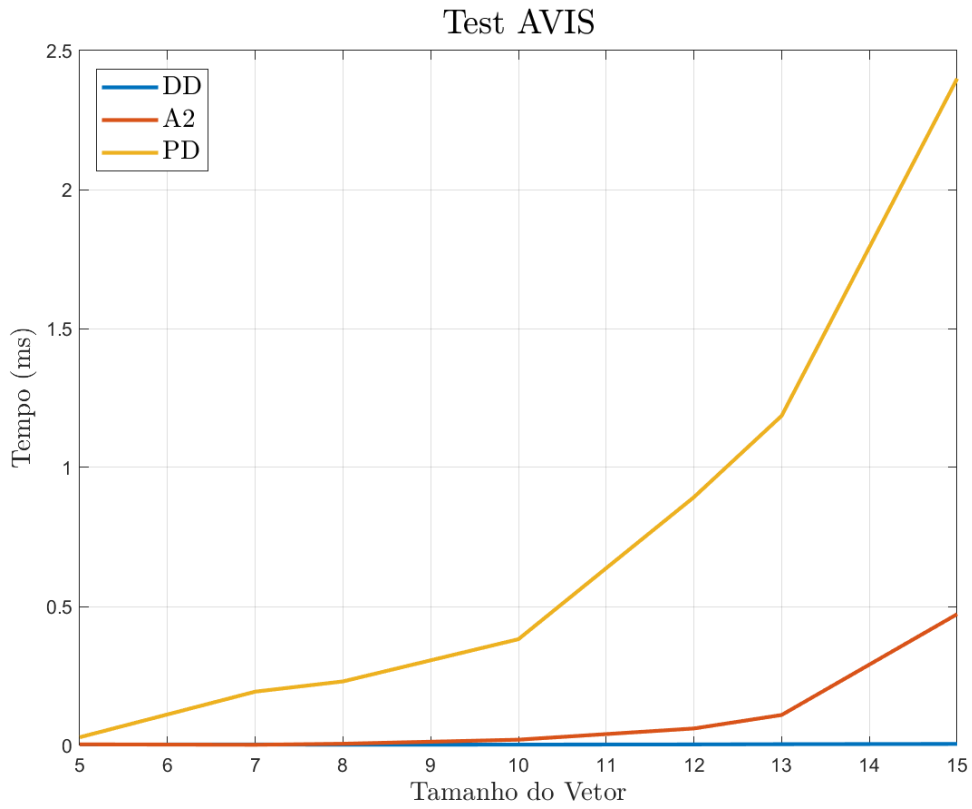
b. Backtracking

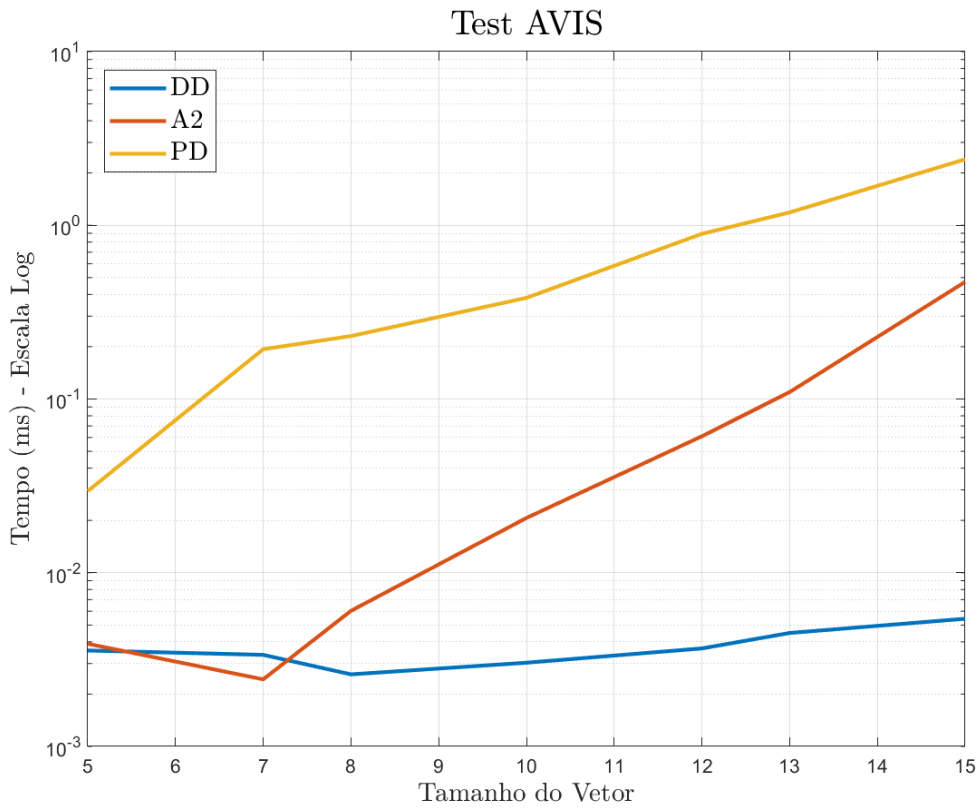
A implementação foi feita utilizando-se um método recursivo. No caso, o problema consiste em andar numa árvore de possibilidades em que, a priori, todas as possibilidades serão testadas (2^n, pois há n itens que podem ou não estarem inclusos na resposta). Todavia, o algoritmo testa constantemente, em todas as recursões, se aquele caminho é ou não possível de ser tomado. Para isso, verificam-se duas condições se a soma restante dos valores não verificados mais a soma atual é menor do que o valor desejado (indicando que não podemos alcançar o valor desejado com os itens restantes); ou se a soma atual já excedeu o valor desejado (indicando que já passamos do alvo e podemos parar de explorar esse caminho.

2) RESULTADOS COMPARATIVOS ENTRE OS ALGORITMOS

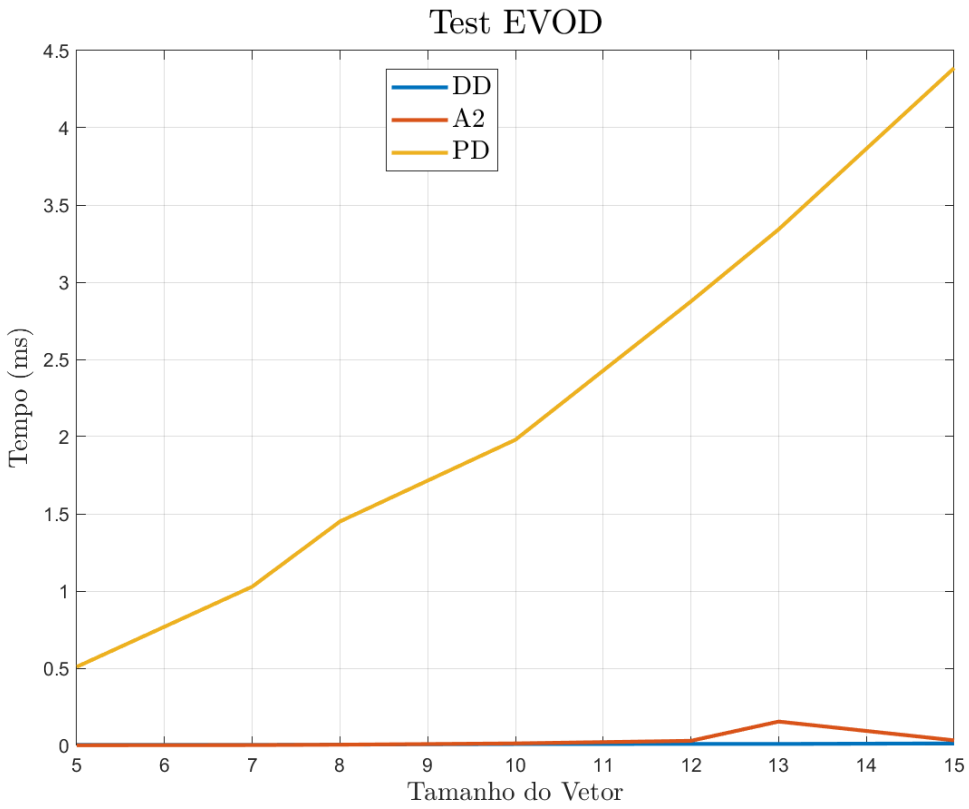
Nesse tópico, podemos comparar os diferentes testes realizados.

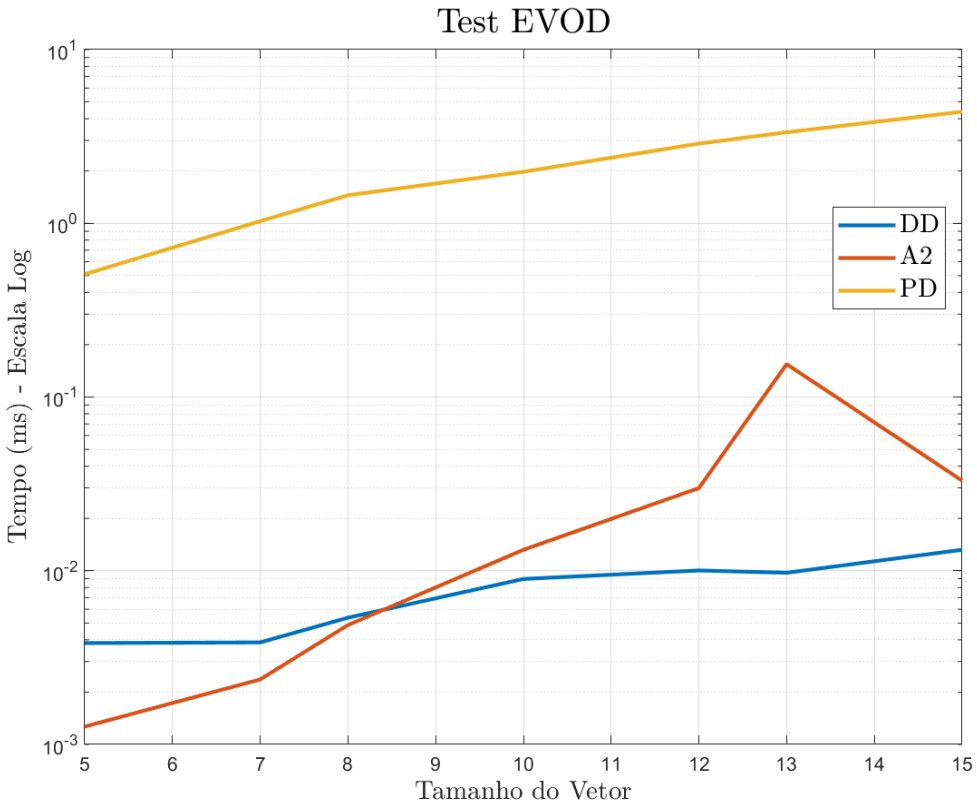
a. Test AVIS



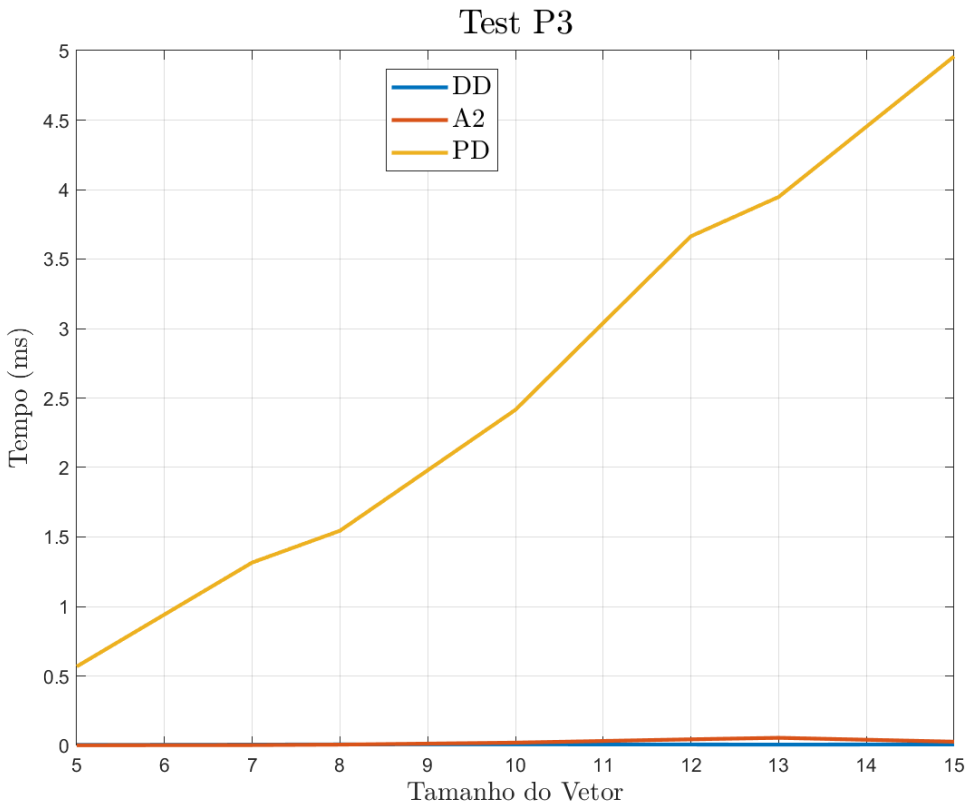


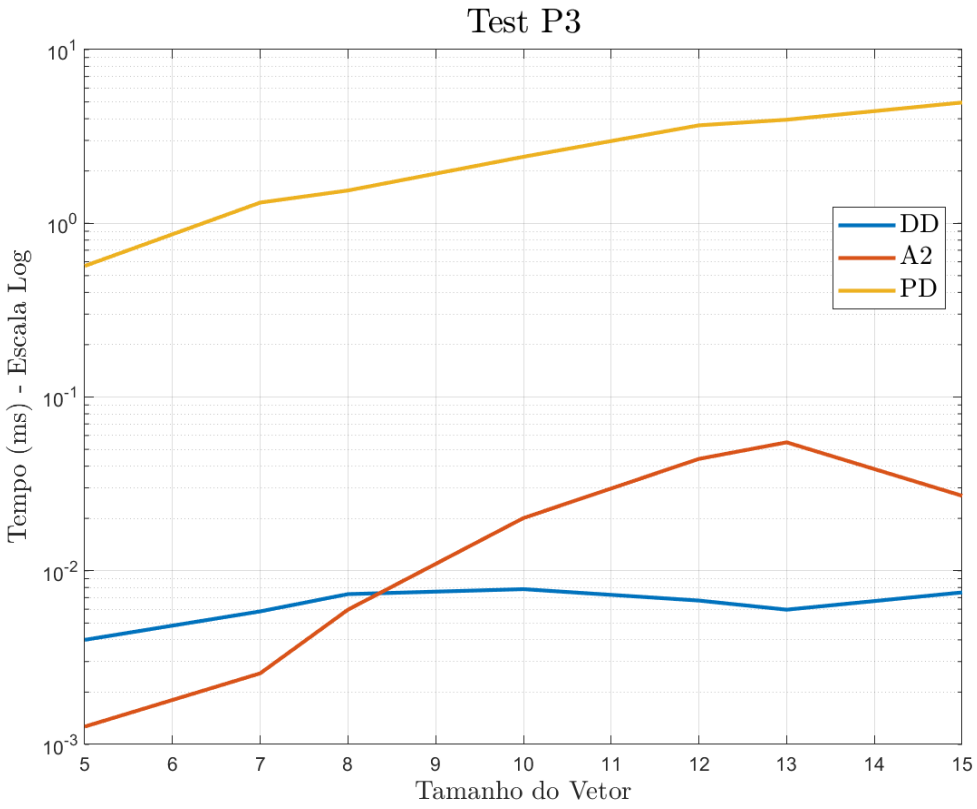
b. Test EVOD


Test EVOD
DD
A2
PD
Tempo (ms)
Tamanho do Vetor
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
6
7
8
9
10
11
12
13
14
15

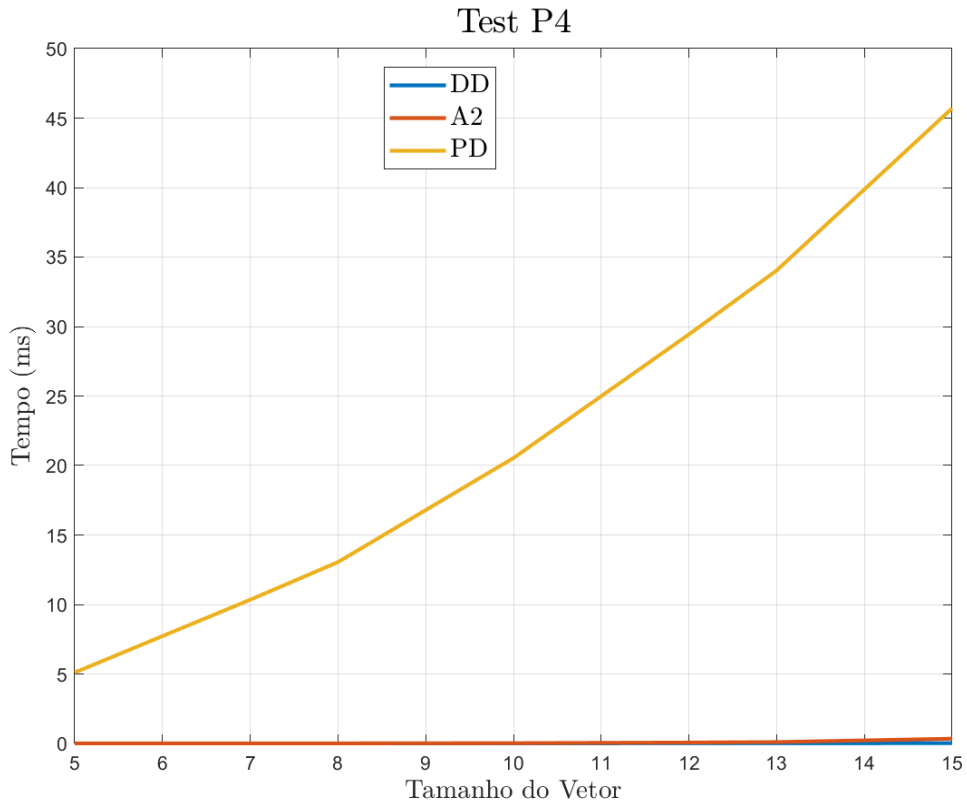

Test EVOD
DD
A2
PD
Tempo (ms) - Escala Log
Tamanho do Vetor
10^{1}
10^{0}
10^{-1}
10^{-2}
10^{-3}
5
6
7
8
9
10
11
12
13
14
15

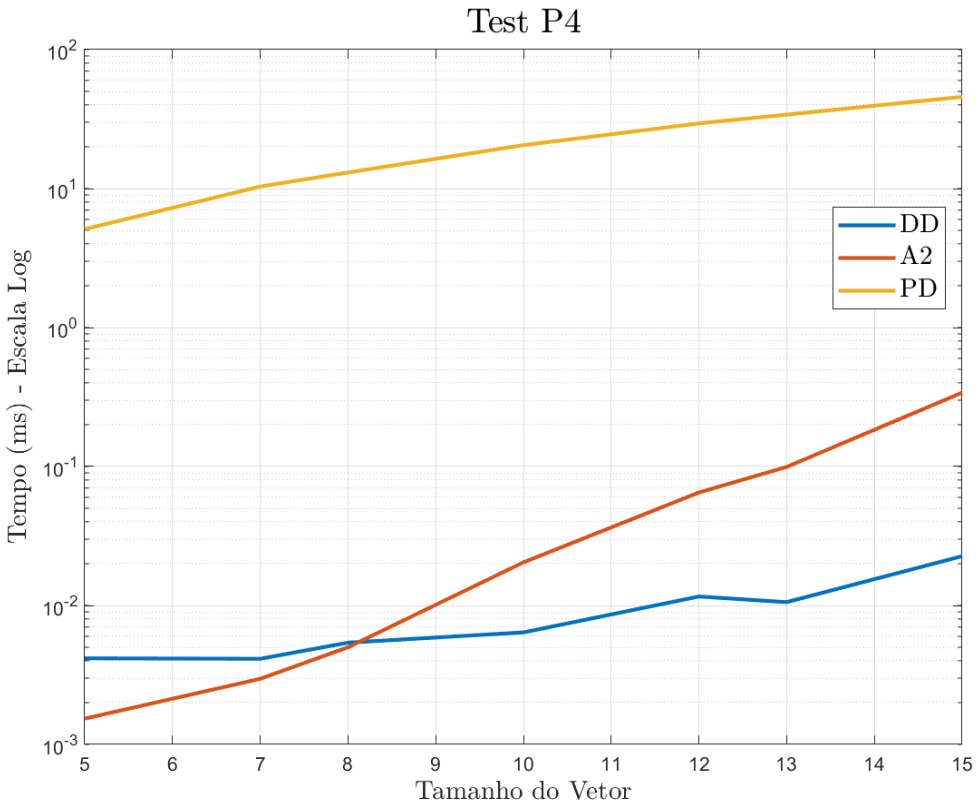
c. Test P3


Test P3
DD
A2
PD
Tempo (ms)
Tamanho do Vetor
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
5
6
7
8
9
10
11
12
13
14
15

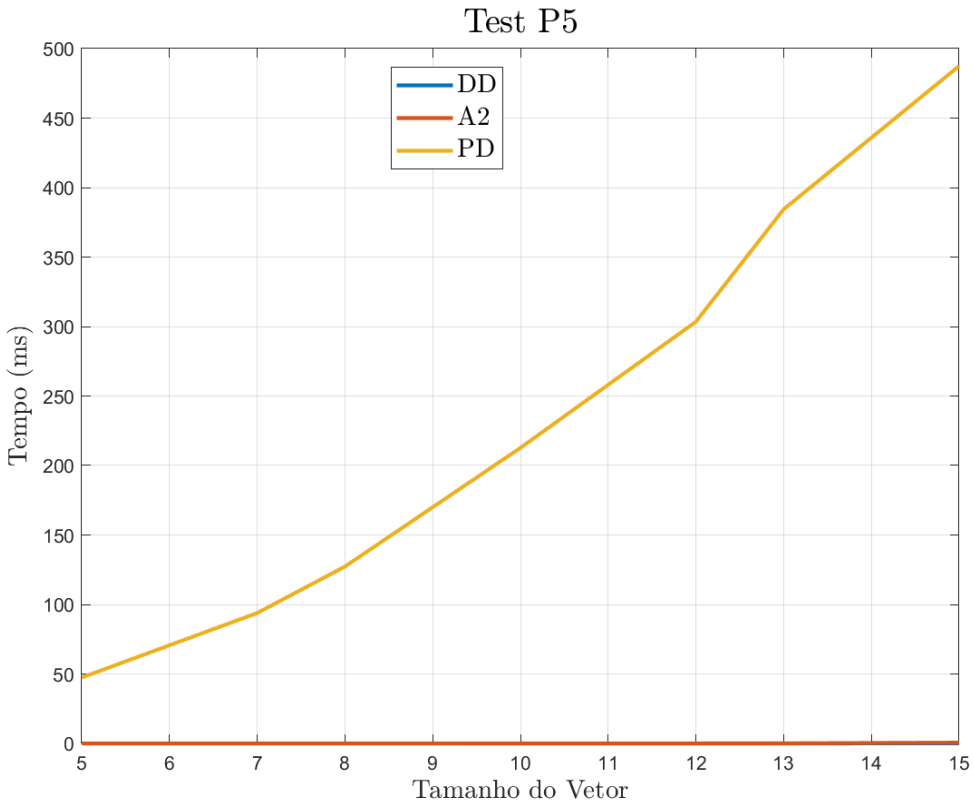

Test P3
DD
A2
PD
Tempo (ms) - Escala Log
Tamanho do Vetor
10^{1}
10^{0}
10^{-1}
10^{-2}
10^{-3}
5
6
7
8
9
10
11
12
13
14
15

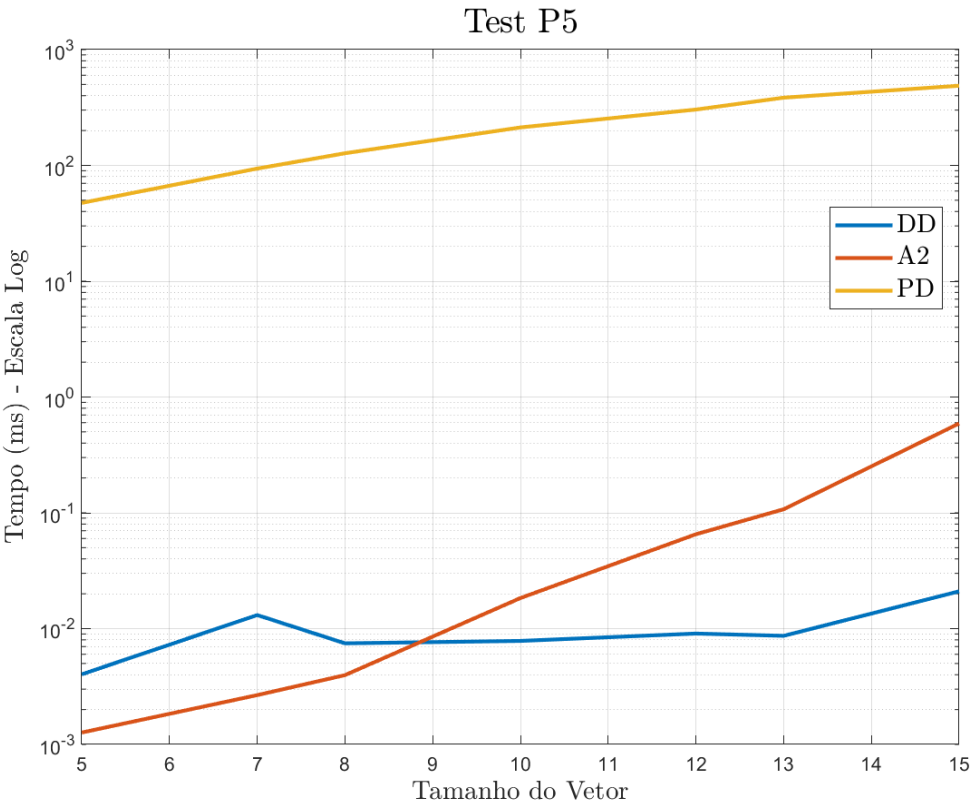
d. Test P4


Test P4
DD
A2
PD
Tempo (ms)
Tamanho do Vetor
0
5
10
15
20
25
30
35
40
45
50
5
6
7
8
9
10
11
12
13
14
15

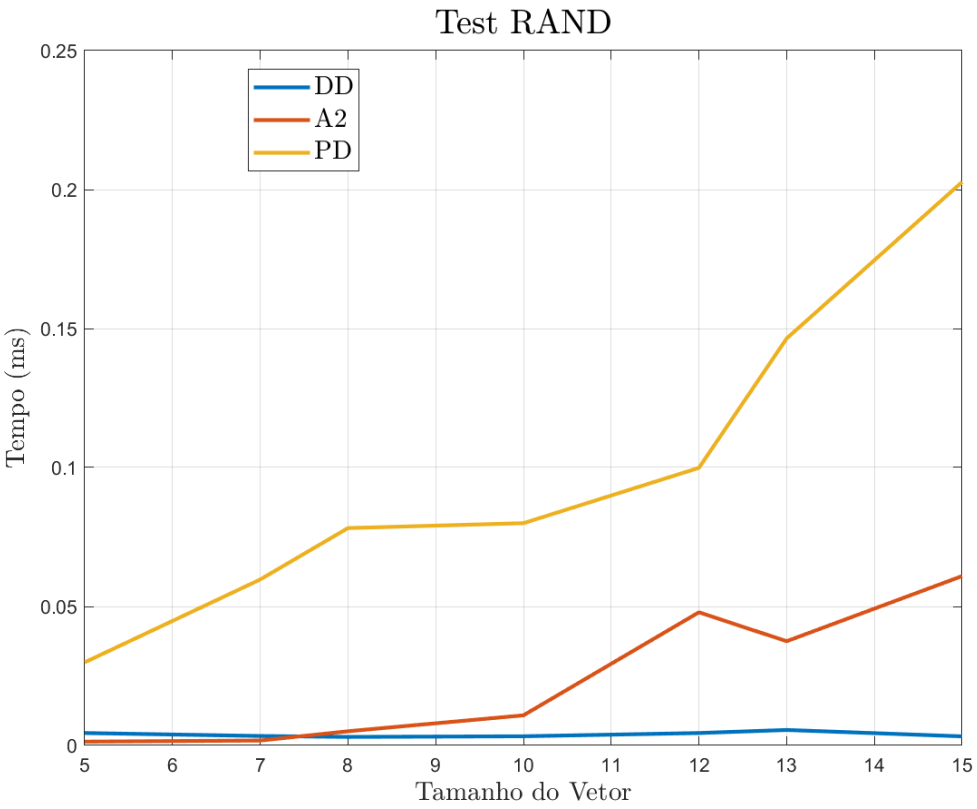

Test P4
DD
A2
PD
Tempo (ms) - Escala Log
Tamanho do Vetor
10^{2}
10^{1}
10^{0}
10^{-1}
10^{-2}
10^{-3}
5
6
7
8
9
10
11
12
13
14
15

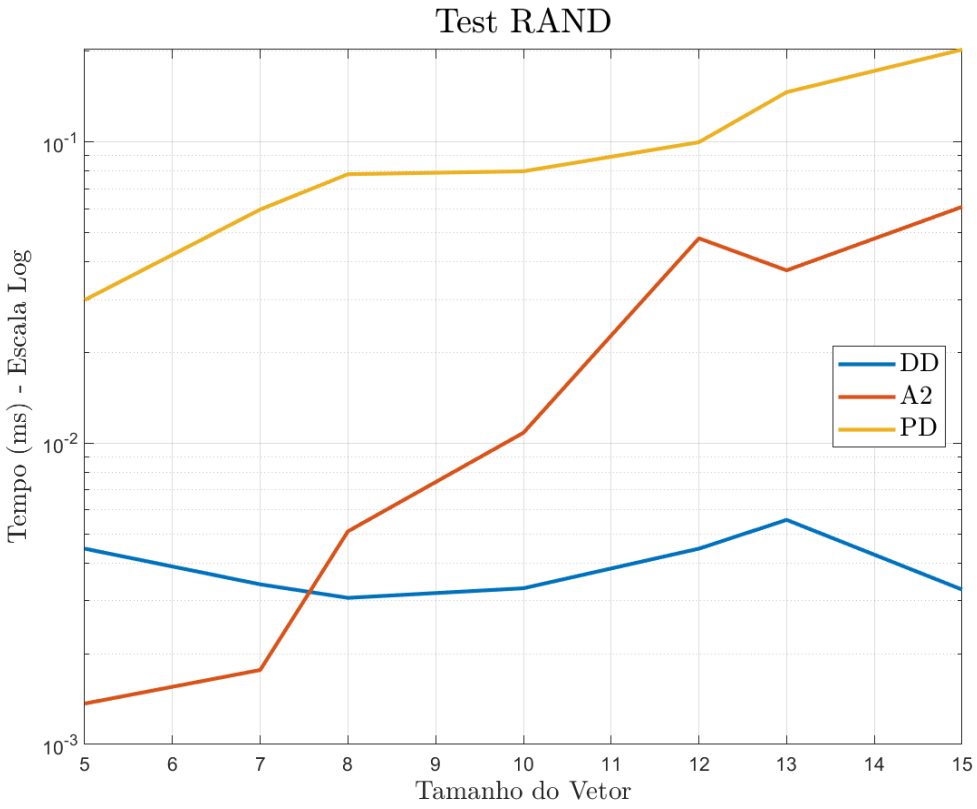
e. Test P5


Test P5
DD
A2
PD
Tempo (ms)
Tamanho do Vetor
0
50
100
150
200
250
300
350
400
450
500
5
6
7
8
9
10
11
12
13
14
15


Test P5
DD
A2
PD
Tempo (ms) - Escala Log
Tamanho do Vetor
10^{3}
10^{2}
10^{1}
10^{0}
10^{-1}
10^{-2}
10^{-3}
5
6
7
8
9
10
11
12
13
14
15

f. Test RAND


Test RAND
DD
A2
PD
Tempo (ms)
Tamanho do Vetor
0.25
0.2
0.15
0.1
0.05
0
5
6
7
8
9
10
11
12
13
14
15


Test RAND
DD
A2
PD
Tempo (ms) - Escala Log
Tamanho do Vetor
10^{-1}
10^{-2}
10^{-3}
5
6
7
8
9
10
11
12
13
14
15

g. Conclusões

Os resultados obtidos mostram que, em praticamente todos os testes, a programação dinâmica é muito pior em tempo do que a aplicação feita com Backtracking (A2) e a aplicação profissionalmente utilizada (DD), sendo que essa afirmação só não se mostrou totalmente verdade no caso randômico (item f), em que, apesar de ser pior, a programação dinâmica não é muito pior que a aplicação do Backtracking. Esse resultado é esperado principalmente pelo fato de que o backtracking evitar de testar todos os casos possíveis, já que em sua implementação, é constantemente testado os casos impossíveis de serem realizados (a cada chamada recursiva), enquanto que o método com programação dinâmica acaba por precisar computar todos os casos possíveis antes de chegar no último elemento da tabela, que é a resposta.

Como esperado, a DD ainda se mostrou extremamente bem otimizada em todos os testes, sendo que perdia para a Backtracking em casos com poucos itens (até cerca de 8 itens), todavia, piorando o tempo muito menos quando se aumentavam os itens, o que mostrou que é uma aplicação melhor em tempo para diversos casos distintos e lida bem com todos muito melhor do que a backtracking ou a programação dinâmica, que apenas pioram o tempo rapidamente conforme os itens aumentam.