Instituto Tecnológico de Aeronáutica
Divisão de Ciência da Computação

CTC-12

Laboratório 04

Paradigmas de Programação

SSP e Troco

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) PROBLEMA DO TROCO

O problema do Troco foi abordado através de 3 algoritmos distintos, sendo eles Divisão e Conquista (DC), Programação Dinâmica (PD) e Algoritmo Guloso/Greedy (G). Todos eles foram comparados entre si, conforme veremos nos gráficos mais pra frente.

a. Divisão e Conquista

Tal abordagem consiste em dividir o problema em subproblemas menores, resolver esses subproblemas de maneira independente e depois combinar as soluções a fim de obter o resultado. No caso do problema do Troco, a solução é obtida dividindo o valor do troco em partes menores e, assim, tentando dar o troco com a maior denominação possível primeiro. No fim, o objetivo é alcançado de forma a diminuir o número total de moedas usadas, mas isso pode resultar em uma execução mais demorada.

b. Programação Dinâmica

Esse método resolve um problema complexo ao dividi-lo em subproblemas e ir resolvendo do mais simples para o mais complicado. No problema do Troco, uma tabela é construída (ou dois vetores interconectados) de forma que nós resolvemos o problema do troco para valores menores e vamos aumentando até chegar no valor desejado. A tabela é preenchida de forma iterativa e a solução final é o valor encontrado na última célula da tabela. Tal método garante a solução ótima e é mais eficiente que o método anterior (DC), porém requer um maior uso da memória.

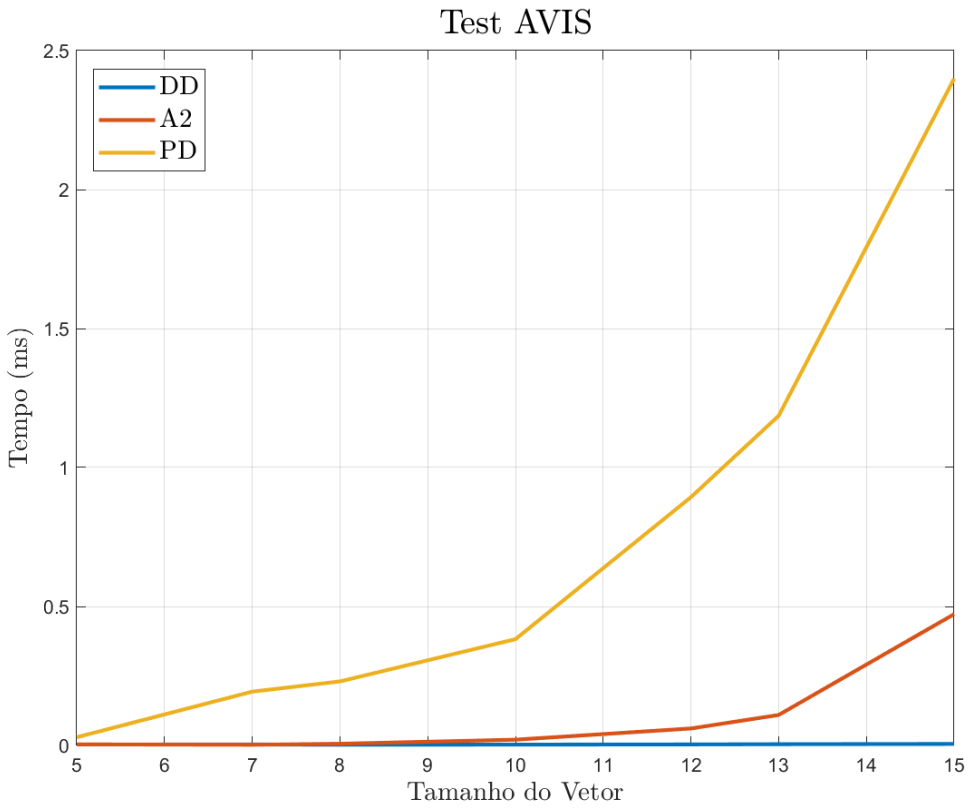
c. Greedy

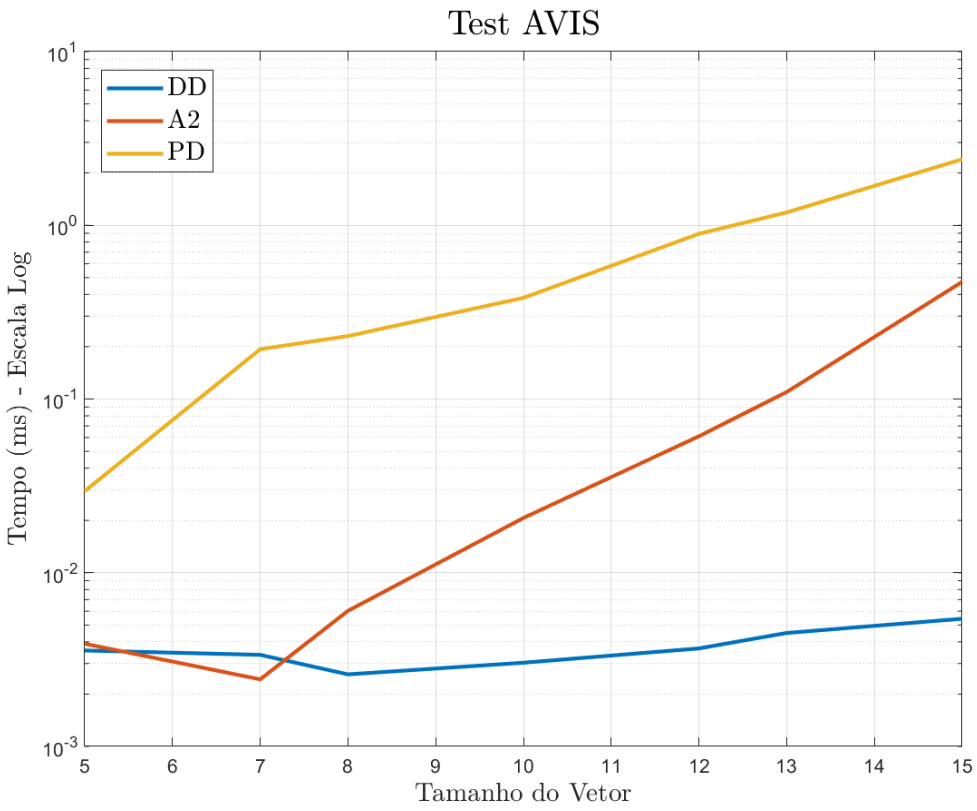
Por fim, o algoritmo guloso é uma forma de fazer escolhas localmente, de forma que a junção das melhores escolhas locais tenda à solução ótima. No contexto do problema do Troco, tal algoritmo tenta dar o troco com a maior denominação possível primeiro e, repetindo este processo, chega a uma solução. Embora este seja um método veloz (pelo menos em relação aos outros 2 acima), ele não garante a solução ideal.

3) RESULTADOS COMPARATIVOS ENTRE OS ALGORITMOS - SSP

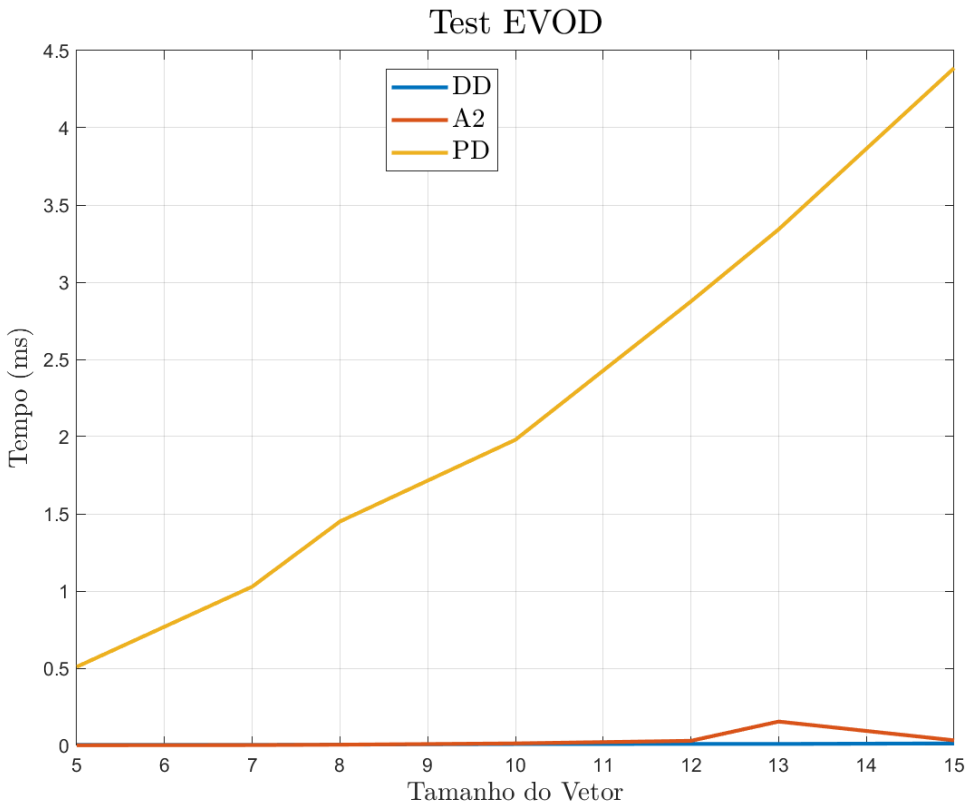
Nesse tópico, podemos comparar os diferentes testes realizados nos algoritmos de SSP.

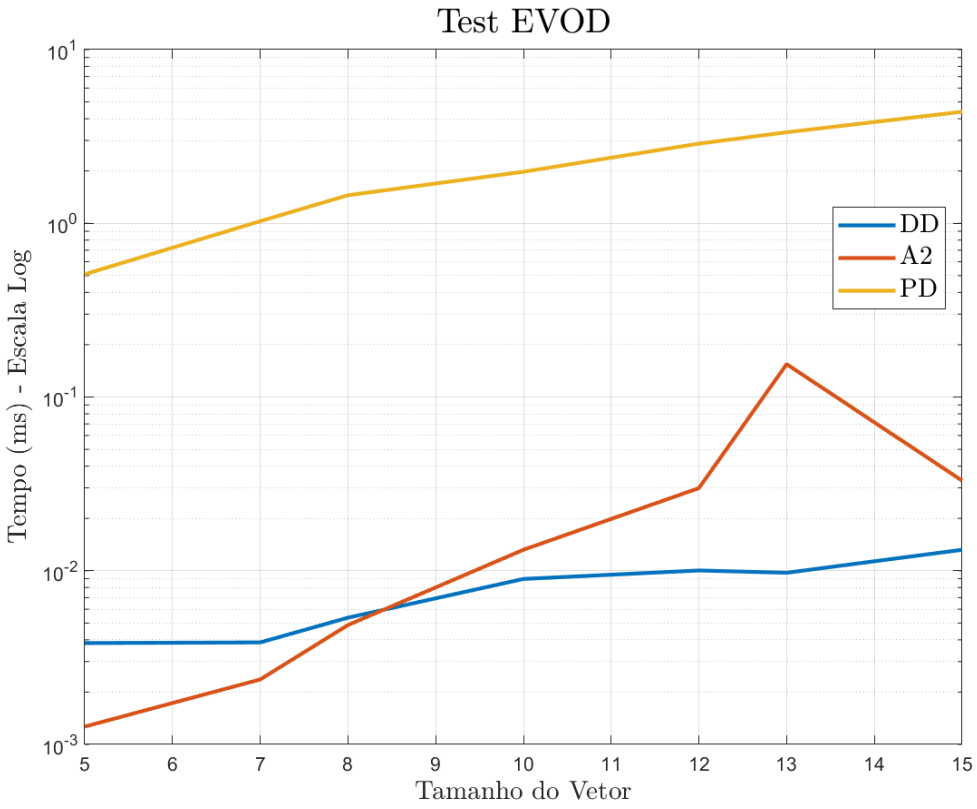
a. Test AVIS



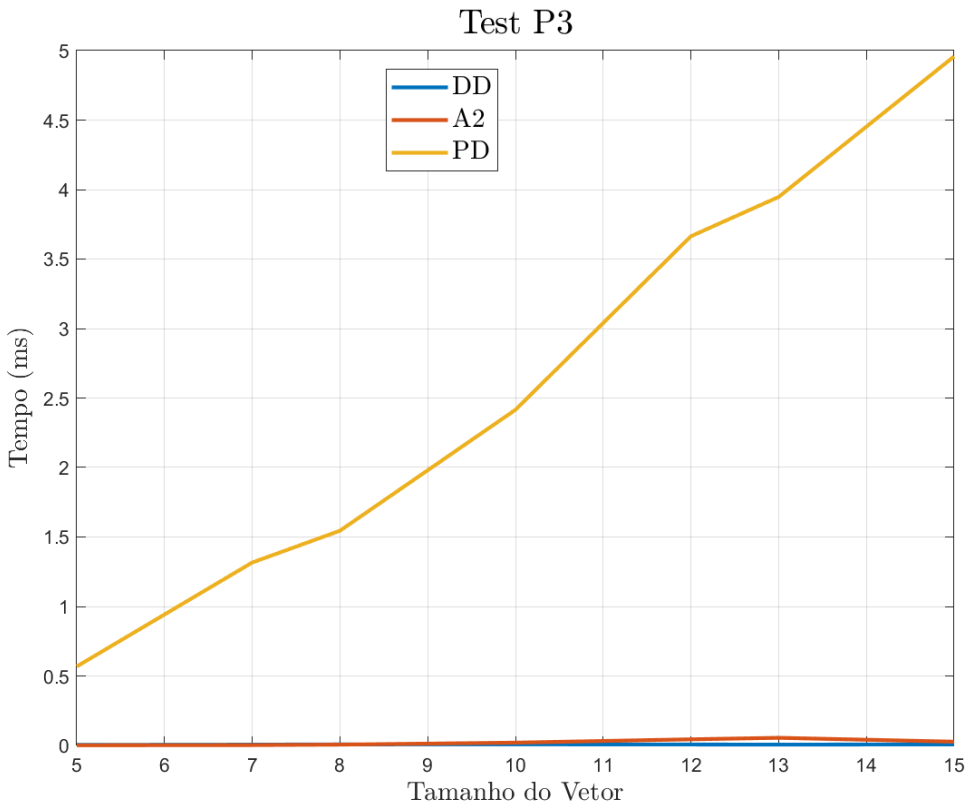


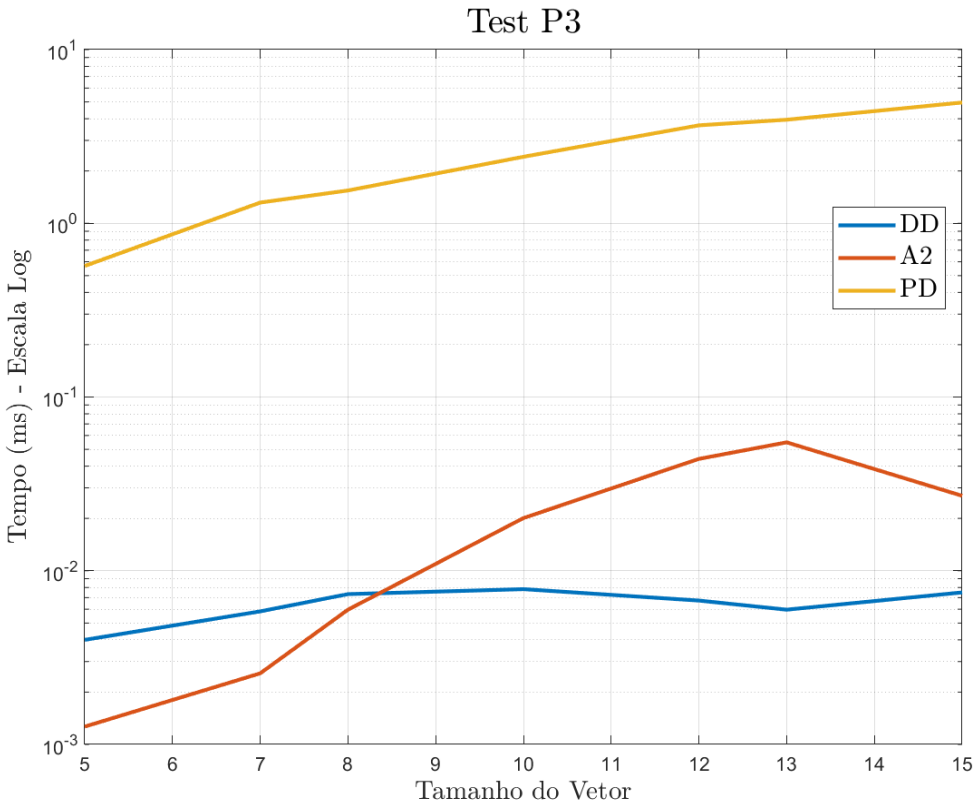
b. Test EVOD


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

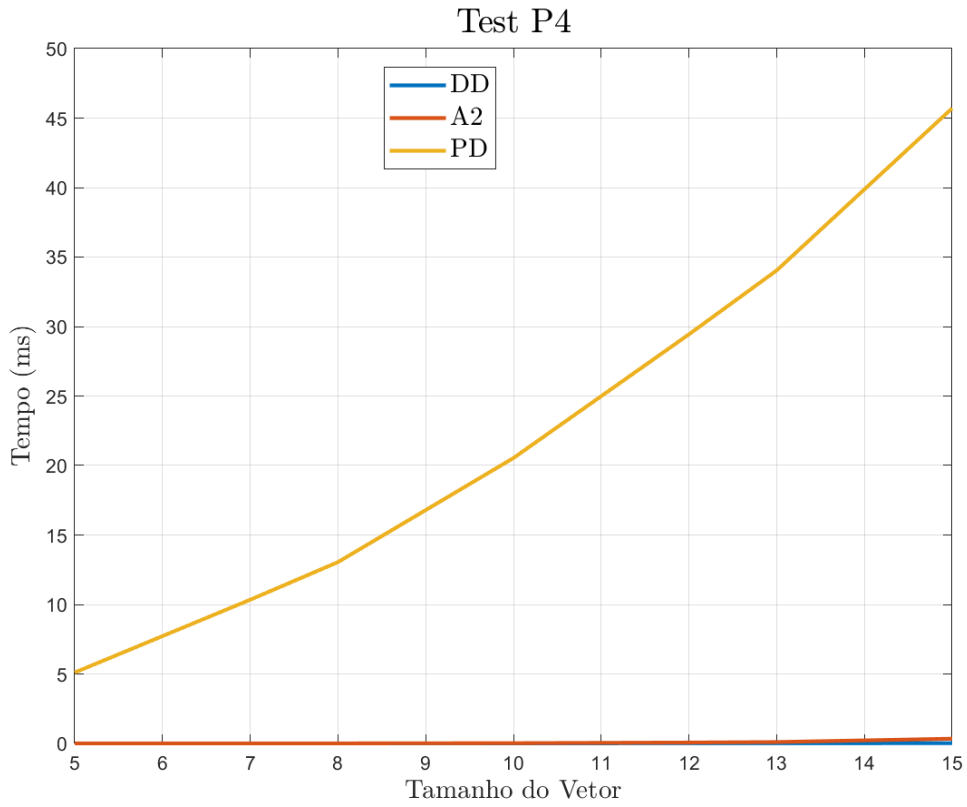

Test EVOD
DD
A2
PD
Tempo (ms) - Escala Log
Tamanho do Vetor
10^{1}
10^{0}
10^{-1}
10^{-2}
10^{-3}
5
6
7
8
9
10
11
12
13
14
15

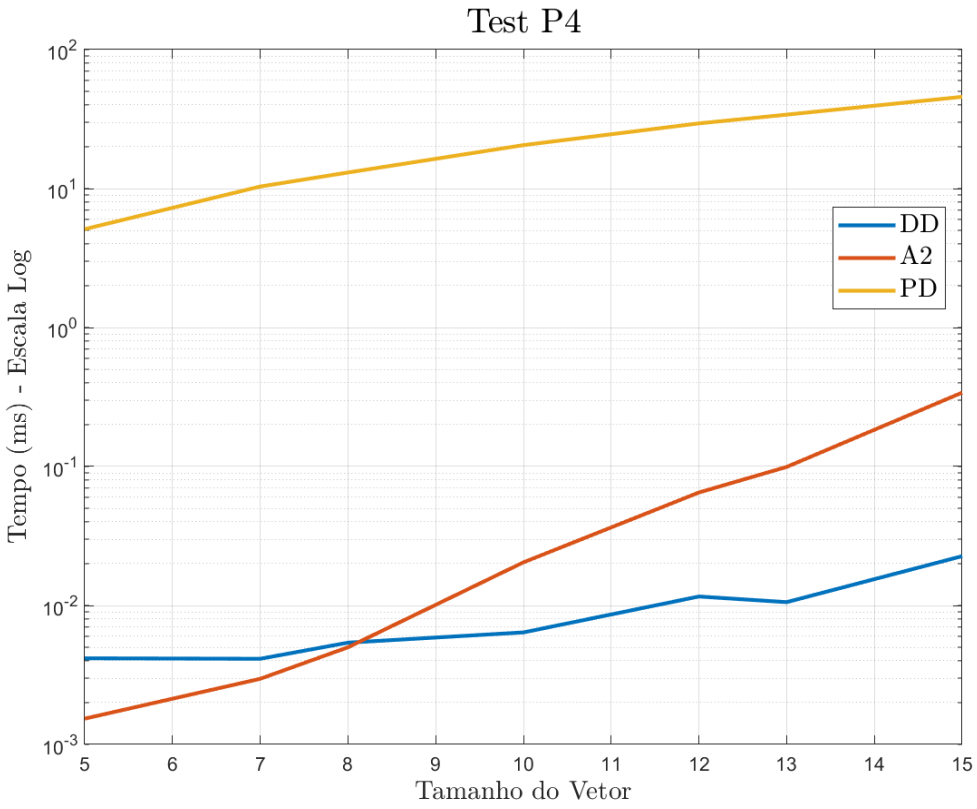
c. Test P3


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

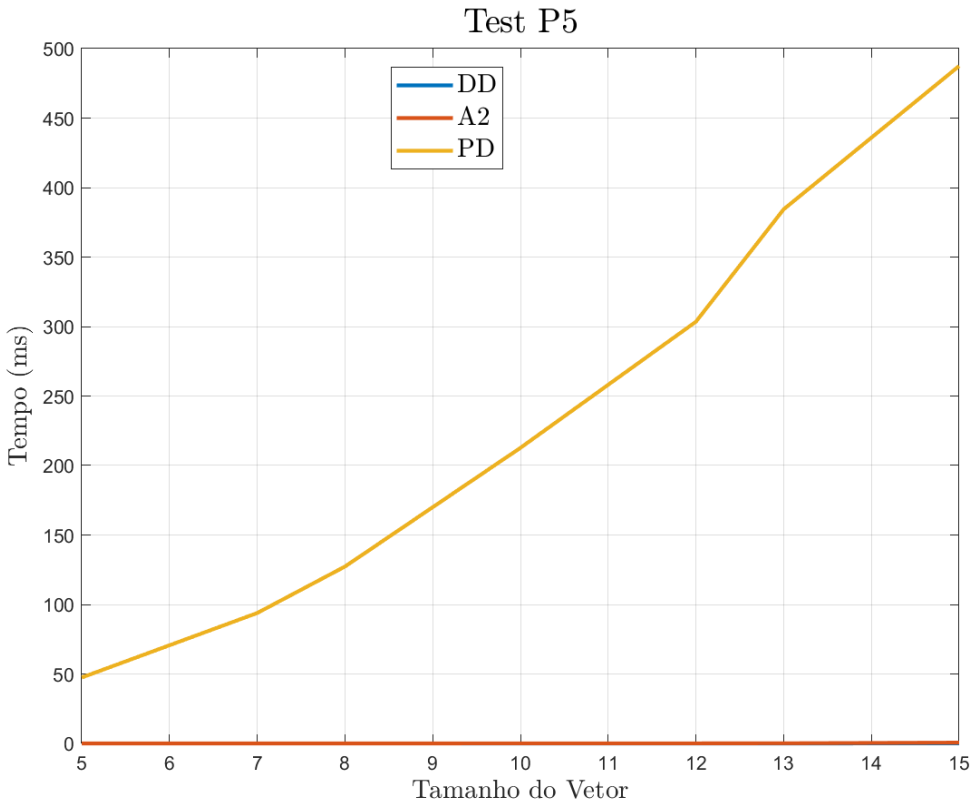

Test P3
DD
A2
PD
Tempo (ms) - Escala Log
Tamanho do Vetor
10^{1}
10^{0}
10^{-1}
10^{-2}
10^{-3}
5
6
7
8
9
10
11
12
13
14
15

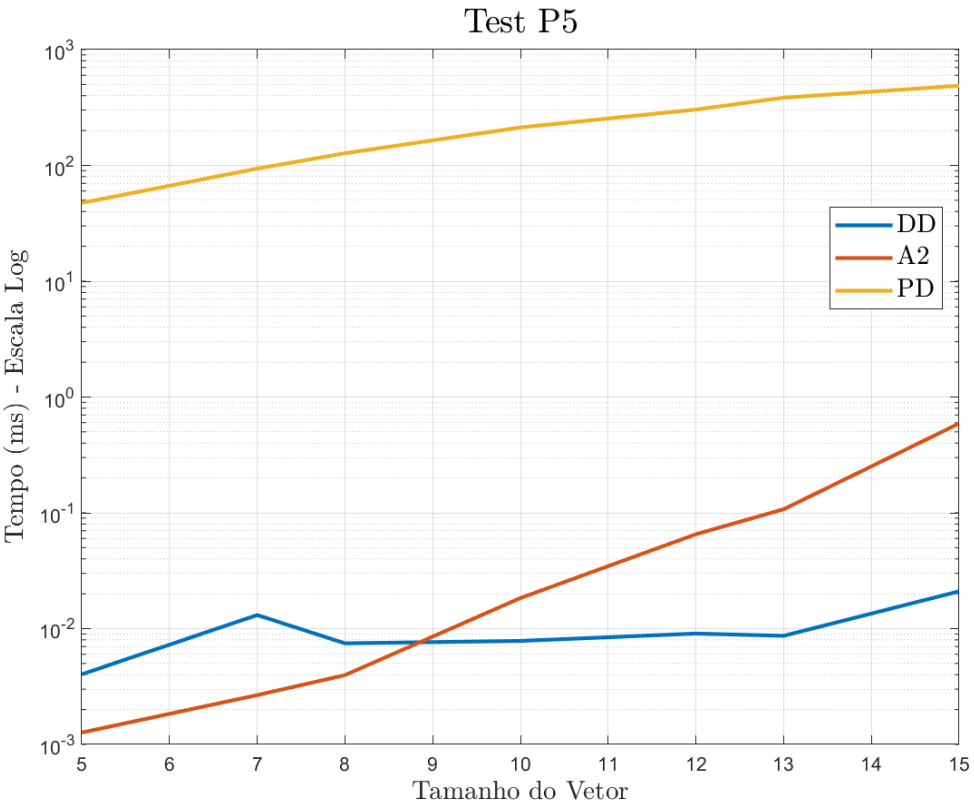
d. Test P4


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

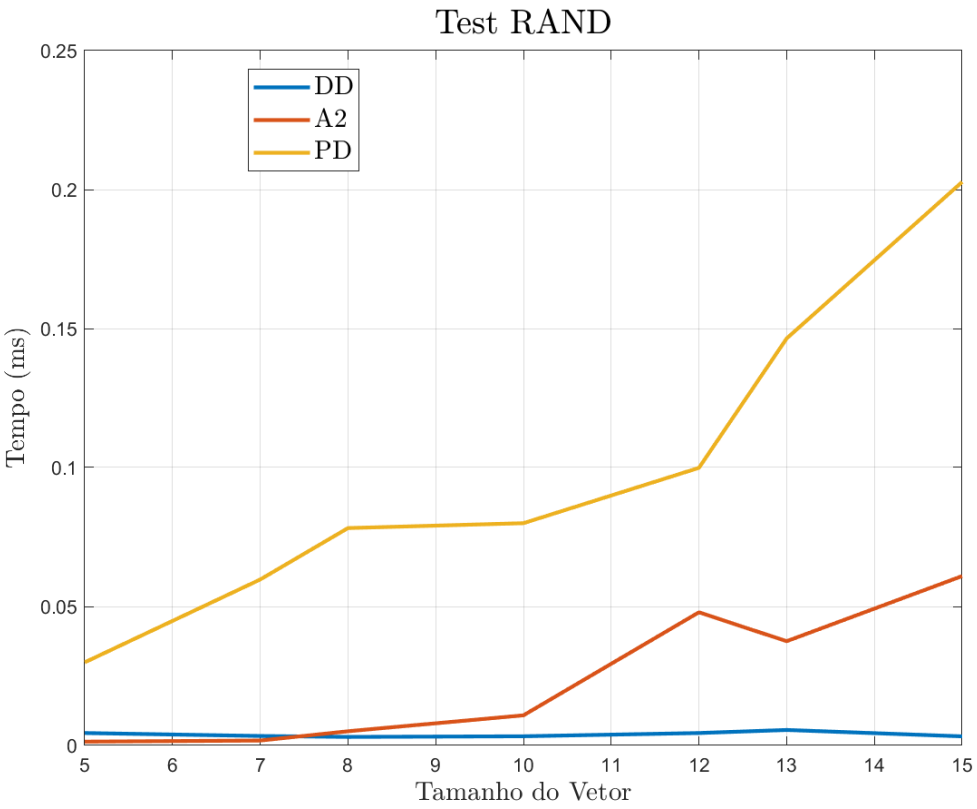

Test P4
DD
A2
PD
Tempo (ms) - Escala Log
Tamanho do Vetor
10^{2}
10^{1}
10^{0}
10^{-1}
10^{-2}
10^{-3}
5
6
7
8
9
10
11
12
13
14
15

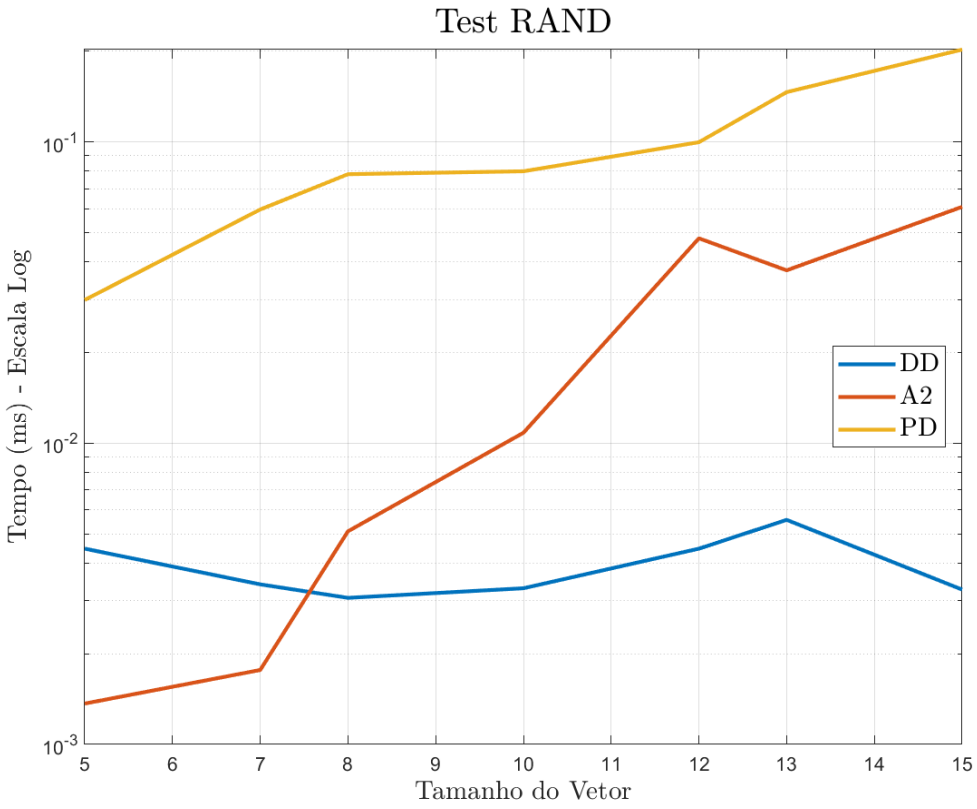
e. Test P5


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

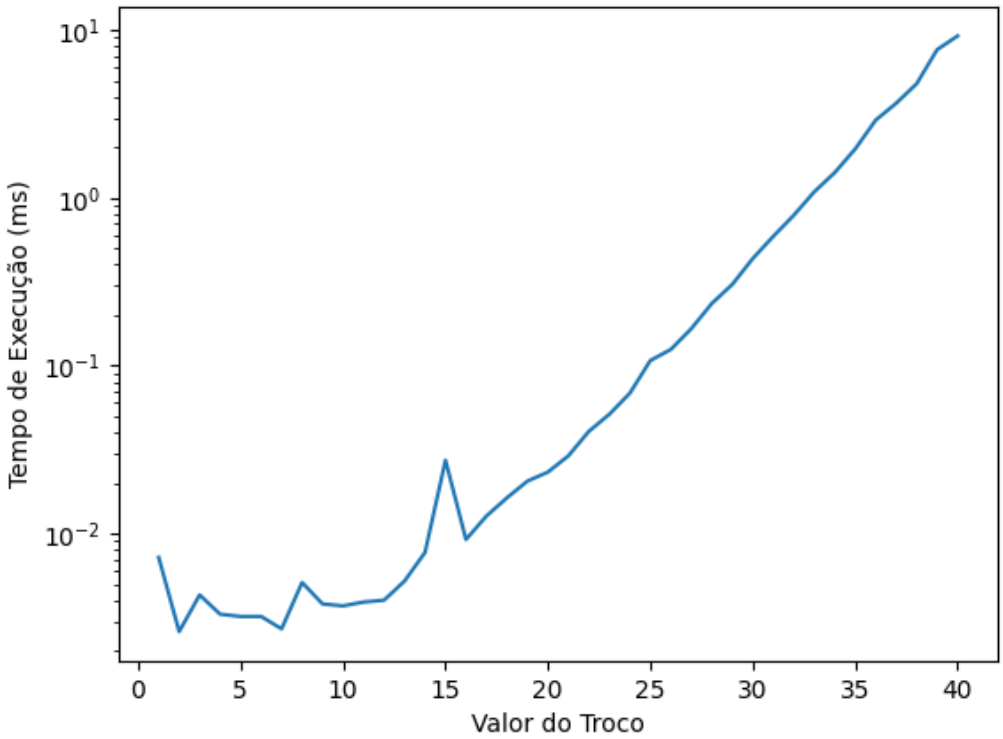

Test RAND
DD
A2
PD
Tempo (ms)
Tamanho do Vetor
0.25
0.2
0.15
0.1
0.05
0
5
6
7
8
9
10
11
12
13
14
15

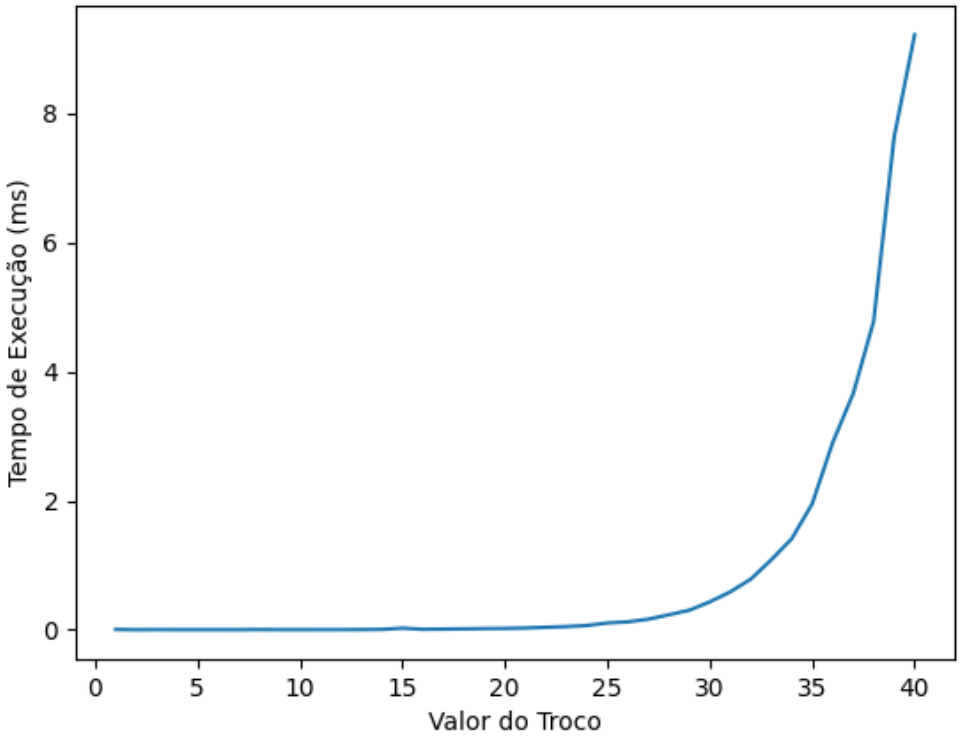

Test RAND
DD
A2
PD
Tempo (ms) - Escala Log
Tamanho do Vetor
10^{-1}
10^{-2}
10^{-3}
5
6
7
8
9
10
11
12
13
14
15

4) RESULTADOS COMPARATIVOS ENTRE OS ALGORITMOS - TROCO

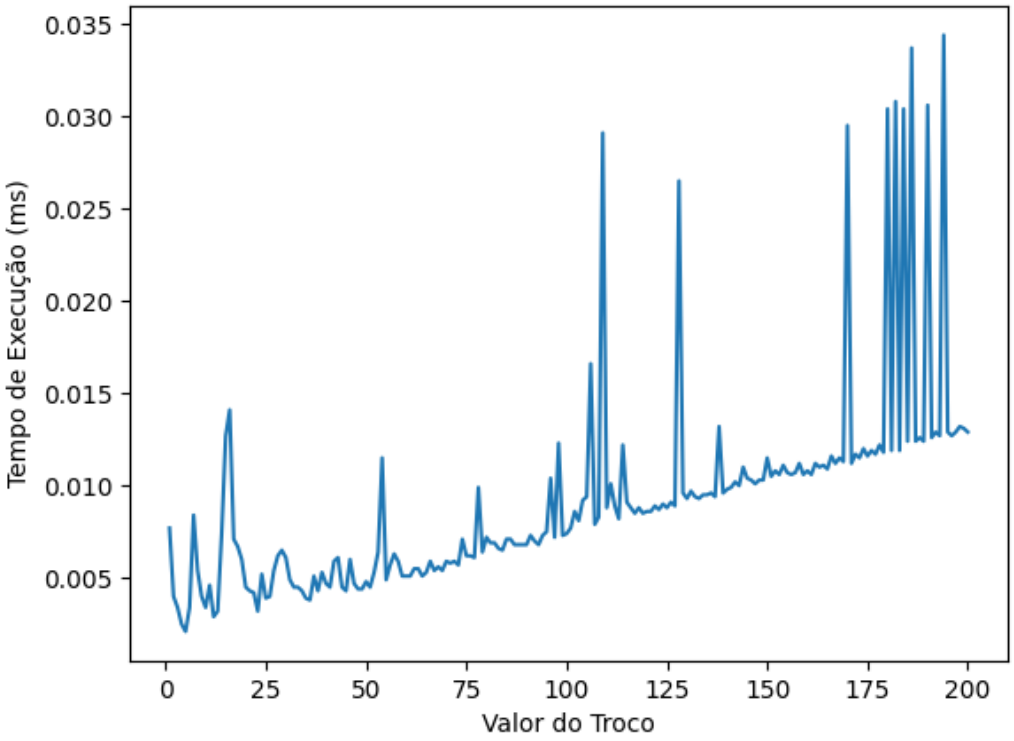
Nesse tópico, podemos comparar os testes realizados nos algoritmos do TROCO.

a. Divisão e Conquista (DC)

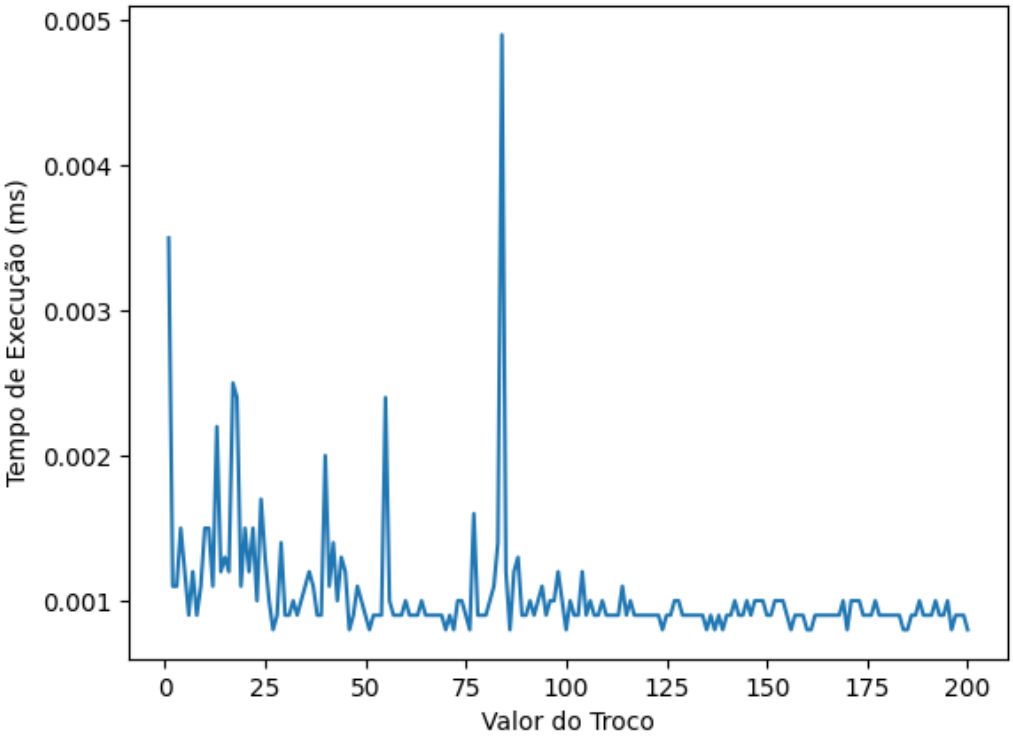


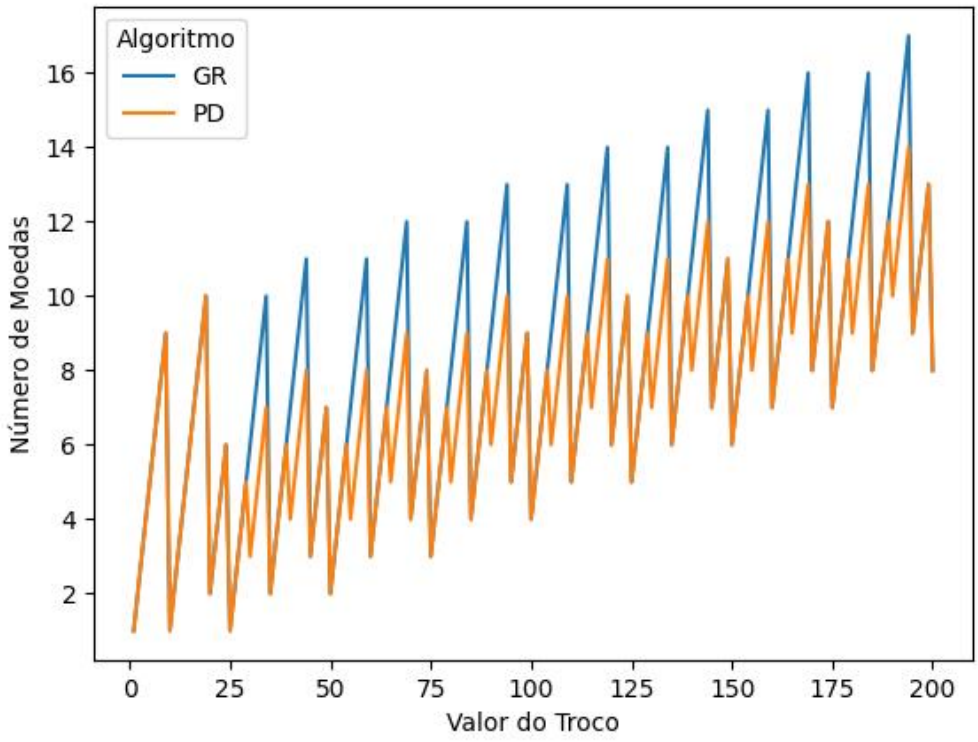


b. Programação Dinâmica (PD)



c. Algoritmo Guloso/Greedy (G)





5) CONCLUSÕES

a. Problema SSP

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 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.

b. Problema do Troco

Podemos notar, comparando os gráficos de tempo, que o algoritmo guloso é muito mais rápido em termos absolutos para vários valores de trocos diferentes em relação aos outros dois, enquanto o método de divisão e conquista (DC) é o mais lento, sendo que ele também sofre mais com o aumento do valor do troco.

Assim como visto anteriormente, este resultado é esperado pois o algoritmo guloso busca soluções locais rapidamente, sem se importar com o conjunto e sempre seguindo um mesmo passo a passo, na esperança de chegar no melhor lugar possível. Além disso, o algoritmo de divisão e conquista testa TODAS as soluções possíveis, o que o torna mais suscetível a longas demoras quanto maior as possibilidades (isto é, quanto maior os valores do troco). Embora o algoritmo de programação dinâmica também teste TODAS as soluções, ele guarda uma solução já calculada anteriormente, enquanto o de divisão e conquista tem que recalcular toda vez.

Por fim, o método de divisão e conquista se mostra rápido em casos específicos, visto que ele depende fortemente da natureza de estrutura do problema em questão. Por exemplo, numa busca binária ele consegue ser eficaz, mas não no problema do Troco, em que há muitas intersecções no problema.