Comparação de desempenho de algoritmos para o Problema de Seleção de Atividades

Daniela E. Pralon[1], **Flávia N. de Matos** [1], **Maycon J. J. Amaro**[1]

[1]Instituto de Ciências Exatas e Biológicas – Departamentto de Computação
Universidade Federal de Ouro Preto (UFOP)

Abstract. *This meta-paper compares the performance of three algorithms for Activity Selection Problem, a combinatorial optimization problem also know as Interval Scheduling problem, being a general problem that describes an entire class of similar problems. The algorithms are greedy, dynamic and backtracking approaches.*

Resumo. *Este artigo propõe uma comparação de desempenho de três algoritmos para o Problema de Seleção de Atividades, um problema de otimização combinatória também conhecido como Agendamento de Intervalo (Interval Scheduling), que serve de generalização para uma classe de problemas similares. Os algoritmos utilizados seguem as abordagens gulosa, dinâmica e busca com retrocesso.*

1. Introdução

O problema de Seleção de Atividades concerne em encontrar o maior conjunto de atividades mutuamente compatíveis. Formalmente, seja $S = \{a_1, a_2, ..., a_n\}$ um conjunto composto por n atividades que desejam utilizar um recurso que só pode atender uma atividade de cada vez. Cada atividade possui um tempo de início s_i e um tempo de término f_i, tal que $0 \leq s_i < f_i < \infty$. Duas atividades a_i e a_j são compatíveis se elas não se sobrepõem, ou seja, se $s_i \geq f_j$ ou $s_j \geq f_i$ [Cormen et al. 2009]. Este problema serve de generalização para uma classe de problemas similares, que normalmente envolvem elementos competindo por um recurso limitado. Há ainda variações como o Problema de Seleção de Atividades Ponderado, em que cada atividade passa também a possuir um peso w_j, que pode ser tanto um valor que representa custo ou vantagem [Wayne 2013].

2. Desenvolvimento

No desenvolvimento deste trabalho, considera-se que uma instância do problema de seleção de atividades é uma lista, não necessariamente ordenada, de triplas (i, s_i, f_i), contendo o índice de cada atividade e seus tempos de início e término. Foram geradas X instâncias aleatoriamente para o problema, que podem ser divididas em três categorias:

- Otimista: instâncias que representam a situação onde todas as atividades são mutuamente compatíveis
- Pessimista: instâncias que representam a situação onde nenhum par de atividades é compatível
- Aleatório: instâncias com cenários aleatórios

2.1. Abordagem gulosa

A primeira abordagem utilizada é o método guloso, onde a lista é ordenada pelo tempo de término das atividades, e sequencialmente escolhe-se, de forma gulosa, a atividade compatível com o tempo de término mais próximo. O algoritmo 1 ilustra o procedimento. Ordenar uma lista pode ser feito em tempo computacional $\mathcal{O}(n \log n)$, utilizando-se algoritmos como MergeSort e QuickSort, já as escolhas gulosas acontecem em tempo $\mathcal{O}(n)$ no total, independente da situação da entrada. Desta forma, o algoritmo 1 executa em tempo computacional $\mathcal{O}(n \log n)$ em qualquer caso.

Algorithm 1 selecao (atividades, n)

```
1:  compativeis ← {}
2:  atividades ← sort(atividades)                ▷ Ordenação pelo tempo de término
3:  i ← 0
4:  compativeis ← compativeis ∪ {atividades[0].indice}
5:  para j ← 1 até n faça
6:      indice1, inicio1, fim1 ← atividades[j]
7:      indice2, inicio2, fim2 ← atividades[i]
8:      se inicio1 >= fim2 então
9:          compativeis ← compativeis ∪ {indice1}
10:         i ← j
11:     fim se
12: fim para
13: imprima compativeis
```

2.2. Abordagem de busca com retrocesso

A segunda abordagem utiliza o método da busca com retrocesso, também conhecido como *backtracking*. Nesta abordagem, considera-se o espaço de todas as possíveis soluções como uma árvore binária, em que cada ramificação indica a escolha positiva e negativa de se incluir uma determinada atividade na solução final. O algoritmo 2 ilustra o procedimento, em que se realiza uma busca em profundidade pela árvore, realizando retrocessos quando se encontra um nodo folha ou uma solução inconsistente. Para instâncias otimistas, o algoritmo 2 executa em tempo $\Theta(2^n)$, analisando todas as possibilidades, tal como uma busca exaustiva, já que não acontecerá nenhum retrocesso por inconsistência, sendo este o pior caso. Para instâncias pessimistas, o algoritmo executará em tempo $\mathcal{O}(n)$, já que acontecerá retrocessos por inconsistência em todos os níveis da árvore de busca, sendo este o melhor caso. Para instâncias com cenários aleatórios, o algoritmo executa em tempo $\mathcal{O}(2^n)$. Nota-se que na primeira chamada, a lista de compatíveis é vazia e a lista de atividades contém todas as atividades da instância em questão, e que em cada chamada recursiva, a solução retornada é aquele de maior cardinalidade.

Algorithm 2 selecao (atividades, compativeis)

1: $novaLista \leftarrow novaLista \cup \{atividades[0]\}$
2: **se** $|atividades| = 1$ **então**
3: retorne $novaLista$
4: **fim se**
5: $novoAtividades \leftarrow novoAtividades - \{atividades[0]\}$
6: **se** solucaoInconsistente($novaLista$) **então**
7: retorne $compativeis$
8: **senão**
9: $x \leftarrow$ selecao($novoAtividades$, $novaLista$)
10: **fim se**
11: $y \leftarrow$ selecao($novoAtividades$, $compativeis$)
12: retorne maxCardinalidade(x, y)

2.3. Abordagem dinâmica

A terceira abordagem utiliza programação dinâmica, como mostrado no algoritmo 3. A lista de atividades é ordenada pelo tempo de término (existem bons algoritmos para ordenação, com complexidade $\mathcal{O}(n \log n)$, conforme dito anteriormente) e é criada uma tabela com n linhas e m colunas, sendo n a quantidade de atividades e m o tempo máximo de término das atividades. A tabela é inicializada com -1, em tempo $\mathcal{O}(nm)$. Em seguida, itera-se na lista ordenada de atividades e, para cada atividade i, divide-se o problema atual em dois subproblemas, v_1 e v_2 que armazenam a quantidade de atividades selecionadas, sendo v_1 o caso onde não é seleciona a atividade i, então chama-se a função recursivamente para a próxima tarefa da lista $(i + 1)$ mantendo o tempo final atual; e v_2 quando é selecionada a atividade i, onde chama-se a função recursivamente para a próxima tarefa da lista $(i + 1)$ atualizando o tempo final para o tempo da atividade i e somando 1 na quantidade de atividades selecionadas.

É armazenado na tabela o valor máximo entre os dois subproblemas. Tal tabela é importante já que, caso a situação já tenha sido calculada, é retornado o valor já armazenado na tabela, evitando que o trabalho seja refeito.

Para saber quais atividades foram selecionadas, foi criada a função resultado com complexidade de tempo $\mathcal{O}(n)$ onde para cada atividade i em tempo ,j, compara-se o valor armazenado na posição (i, j) com o valor armazenado na posição $(i + 1, j)$. Caso o valor seja maior, significa que a atividade i foi selecionada. Então, ela é adicionada à solução e agora chama-se a função recursivamente para a posição da próxima atividade $(i + 1)$ atualizando o tempo final para o tempo final da atividade i. Caso o valor seja igual, significa que a atividade i não foi selecionada. Então chama-se a função recursivamente para a posição da próxima atividade $(i + 1)$ mantendo o tempo final atual. Independente do caso, o algoritmo possui complexidade $\mathcal{O}(n \log(n) + nm)$, ou seja, o tempo computacional assintótico do algoritmo é limitado pela ordenação ou pela inicialização, dependendo apenas se $\log n < m$ ou não.

Algorithm 3 dinamico (indice, tempo_final, atividades)

se $indce = 0$ **então**
    retorna 0
fim se
se $tabela\_quantidades[i][j] \neq -1$ **então**
    $retornatabela\_quantidade[i][j]$
fim se
$v1 = dinamico(indice + 1, tempo_final, atividades)$
se $atividades[indice].inicio > tempo_final$ **então**
    $v2 = dinamico(indice + 1), atividades[indice].termino, atividades) + 1$
fim se
$tabela\_quantidade[i][j] = max(v1, v2)$
$retornatabela_quantidade$

3. Resultados

Os algoritmos, implementados na linguagem Python, versão 2.7, foram executados em um notebook Dell Inspiron com processador Intel Core i5 (Quad-core) e 4 GB de memória RAM. As tabelas 1, 4 e 7 mostram o desempenho do algoritmo guloso nos três diferentes cenários, já as tabelas 3, 6 e 9 mostram o desempenho do algoritmo dinâmico e as tabelas 2, 5 e 8 mostram o desempenho do algoritmo de busca com retrocesso. Em cada tabela, mostra-se o tempo em segundos que o algoritmo demorou para executar na instância, informada pela linha, do tamanho de entrada informado pela coluna. Também mostram a média e os limites (extremidades) intervalo de confiança com 95% de coeficiente de confiança.

Tabela 1. Algoritmo guloso, cenário aleatório

	10 entradas	50 entradas	100 entradas
Entrada	Tempo	Tempo	Tempo
1	0,0000849	5,20E-05	6,20E-05
2	5,10E-05	4,01E-05	1,51E-04
3	4,51E-05	4,01E-05	1,09E-04
4	2,69E-05	4,60E-05	1,18E-04
5	9,39E-05	6,10E-05	5,98E-05
6	3,10E-05	4,79E-05	1,20E-04
7	9,61E-05	4,01E-05	1,24E-04
8	6,29E-05	7,10E-05	1,39E-04
9	3,41E-05	4,29E-05	7,61E-05
10	3,70E-05	6,48E-05	2,04E-04
Média	0,000056290626	5,07E-05	1,16E-04
Limite inferior	4,06E-05	4,41E-05	9,05E-05
Limite superior	7,20E-05	5,73E-05	1,42E-04

Tabela 2. Algoritmo backtracking, cenário aleatório

	10 entradas	50 entradas	100 entradas
Entrada	Tempo	Tempo	Tempo
1	0,0009239	1,17E+00	5,89E+01
2	6,78E-04	5,74E-01	2,58E+02
3	2,40E-04	1,68E-00	5,91E+01
4	4,06E-04	2,75E-01	5,07E+01
5	2,12E-04	1,16E-01	4,32E+01
6	8,51E-04	4,05E-01	3,25E+01
7	2,45E-04	2,76E-01	2,02E+01
8	2,44E-04	1,71E-01	3,72E+02
9	1,44E-04	7,84E-01	5,04E+01
10	3,80E-04	2,19E+00	7,13E+01
Média	0,000432419776	0,000432419776	1,31E+02
Limite inferior	2,66E-04	3,50E-01	5,38E+00
Limite superior	5,99E-04	1,18E+00	4,59E+02

Tabela 3. Algoritmo dinâmico, cenário aleatório

	10 entradas	50 entradas	100 entradas
Entrada	Tempo	Tempo	Tempo
1	0,0004401	4,95E-03	1,79E-02
2	3,02E-04	4,87E-03	2,03E-02
3	2,98E-04	4,42E-03	1,95E-02
4	2,40E-04	5,21E-03	2,04E-02
5	3,11E-04	5,86E-03	2,10E-02
6	2,85E-04	5,92E-03	2,08E-02
7	3,07E-04	9,18E-03	1,79E-02
8	3,18E-04	4,53E-03	1,96E-02
9	2,60E-04	5,20E-03	1,91E-02
10	2,37E-04	5,23E-03	1,83E-02
Média	0,000299811363	5,54E-03	1,95E-01
Limite inferior	2,66E-04	4,73E-03	1,88E-02
Limite superior	3,33E-04	6,34E-03	2,02E-02

Tabela 4. Algoritmo guloso, cenário otimista

	10 entradas	50 entradas	100 entradas
Entrada	Tempo	Tempo	Tempo
1	0,0000360	6,60E-05	7,61E-05
2	4,10E-05	8,61E-05	7,80E-05
3	5,01E-05	5,89E-05	6,51E-05
4	3,89E-05	5,32E-05	7,41E-05
5	9,68E-05	4,48E-05	6,51E-05
6	3,91E-05	5,51E-05	7,80E-05
7	3,50E-05	6,01E-05	8,11E-05
8	4,79E-05	6,29E-05	6,89E-05
9	2,91E-05	4,60E-05	6,79E-05
10	3,29E-05	4,51E-05	7,49E-05
Média	0,00004467964172	5,78E-05	7,29E-05
Limite inferior	3,33E-05	5,05E-05	6,95E-05
Limite superior	5,61E-05	6,51E-05	7,63E-05

Tabela 5. Algoritmo backtracking, cenário otimista

	10 entradas	50 entradas	100 entradas
Entrada	Tempo	Tempo	Tempo
1	0,0038171	> 1800	> 1800
2	4,70E-03	> 1800	> 1800
3	5,03E-03	> 1800	> 1800
4	4,15E-03	> 1800	> 1800
5	3,31E-03	> 1800	> 1800
6	3,56E-03	> 1800	> 1800
7	3,30E-03	> 1800	> 1800
8	3,09E-03	> 1800	> 1800
9	3,25E-03	> 1800	> 1800
10	3,23E-03	> 1800	> 1800
Média	0,003744006157	> 1800	> 1800
Limite inferior	3,35E-03	1800	1800
Limite supetior	4,14E-03	1800	1800

Tabela 6. Algoritmo dinâmico, cenário otimista

	10 entradas	50 entradas	100 entradas
Entrada	Tempo	Tempo	Tempo
1	0,0001040	1,86E-03	6,13E-03
2	2,72E-05	2,31E-03	1,03E-02
3	2,57E-04	2,83E-03	1,29E-02
4	2,27E-04	3,64E-03	1,40E-02
5	3,65E-04	3,73E-03	1,72E-02
6	2,61E-04	4,12E-02	1,80E-02
7	3,01E-04	5,14E-03	2,06E-02
8	3,37E-04	5,40E-03	2,30E-02
9	3,68E-04	6,20E-03	2,47E-02
10	3,92E-04	6,38E-03	2,32E-02
Média	0,0002639532089	7,87E-03	1,70E-02
Limite inferior	1,94E-04	9,24E-04	1,34E-02
Limite superior	3,34E-04	1,48E-02	2,06E-02

Tabela 7. Algoritmo guloso, cenário pessimista

	10 entradas	50 entradas	100 entradas
Entrada	Tempo	Tempo	Tempo
1	0,0000360	6,29E-05	7,70E-05
2	4,39E-05	6,89E-05	7,30E-05
3	3,10E-05	6,01E-05	7,70E-05
4	3,98E-05	5,01E-05	7,10E-05
5	8,89E-05	5,89E-05	8,01E-05
6	2,60E-05	5,29E-05	7,61E-05
7	2,79E-05	5,20E-05	1,82E-04
8	3,29E-05	5,89E-05	8,49E-05
9	3,00E-05	4,41E-05	7,70E-05
10	2,88E-05	5,20E-05	7,70E-05
Média	0,00003852844238	5,61E-05	8,75E-05
Limite infetior	2,76E-05	5,18E-05	6,89E-05
Limite superior	4,94E-05	6,03E-05	1,07E-04

Tabela 8. Algoritmo backtracking, cenário pessimista

	10 entrada	50 entradas	100 entradas
Entrada	Tempo	Tempo	Tempo
1	0,0001080	2,40E-03	8,12E-03
2	1,16E-04	1,77E-03	6,99E-03
3	8,51E-05	1,89E-03	6,77E-03
4	9,89E-05	1,76E-03	6,57E-03
5	9,30E-05	1,82E-03	7,01E-03
6	8,70E-05	1,55E-03	1,37E-02
7	1,07E-04	1,87E-03	6,39E-03
8	9,51E-05	1,75E-03	6,56E-03
9	1,01E-04	2,07E-03	9,88E-03
10	1,04E-04	1,48E-03	7,20E-03
Média	0,00009953975677	1,84E-03	7,92E-03
Limite inferior	9,38E-05	2,69E-03	6,58E-03
Limite superior	1,05E-04	1,99E-03	9,36E03

Tabela 9. Algoritmo dinâmico, cenário pessimista

	10 entradas	50 entradas	100 entradas
Entrada	Tempo	Tempo	Tempo
1	0,0002809	4,91E-03	1,67E-02
2	2,81E-04	4,90E-03	1,69E-02
3	2,31E-04	4,35E-03	1,83E-02
4	2,81E-04	4,91E-03	2,25E-02
5	2,81E-04	4,86E-03	2,25E-02
6	2,81E-04	4,91E-03	2,25E-02
7	2,81E-04	4,86E-03	1,95E-02
8	2,72E-04	4,35E-03	1,99E-02
9	2,81E-04	4,91E-03	1,86E-02
10	2,31E-04	4,91E-03	1,72E-02
Média	0,0002699702923	4,79E-03	1,95E-01
Limite inferior	2,58E-04	4,65E-03	1,81E-02
Limite superior	2,82E-04	4,92E-03	2,08E-02

A seguir estão representados graficamente a média, limite inferior e limite superior de cada algoritmo com 10 entradas (linhas azuis), 50 entradas (linhas vermelhas) e 100 entradas (linhas amarelas).

4. Conclusão

Analisando os resultados obtidos, conclui-se que para esse problema a abordagem gulosa possui o melhor desempenho no caso geral (caso médio). Para instâncias pessimistas, o algoritmo de busca com retrocesso possui uma complexidade melhor que o algoritmo guloso, porém nos tamanhos de instâncias observados, o ponto em que o tempo computacional do algoritmo guloso ultrapassa o da busca com retrocesso não foi atingido,

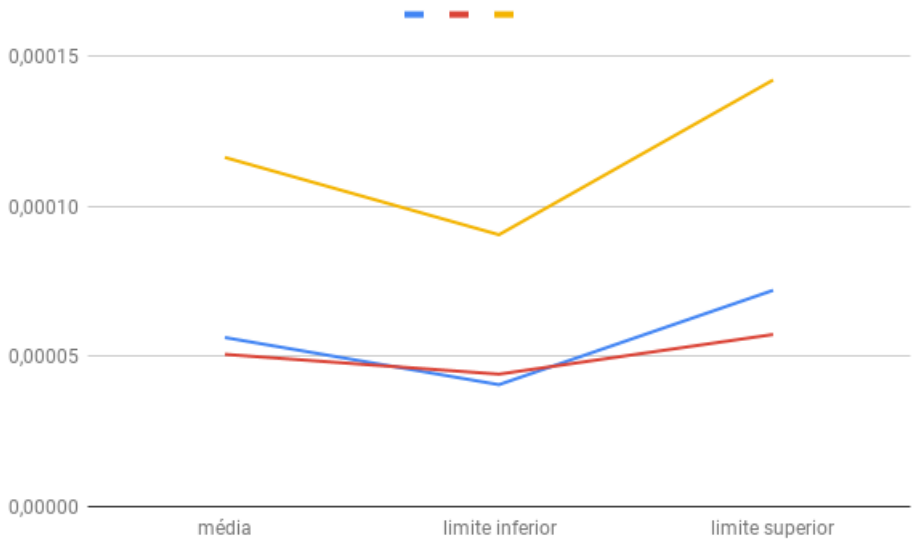

Figura 1. Algoritmo guloso, cenário aleatório

indicando que para instâncias de tamanho pequeno, o algoritmo guloso é o mais rápido independente do cenário. Dado que o algoritmo guloso resolve o problema de forma exata em tempo determinístico polinomial, o Problema de Seleção de Atividades está na classe de complexidade P. É interessante observar que mesmo o problema possuindo solução eficiente, ainda é possível obter soluções não eficientes para ele, como é o caso da busca com retrocesso (complexidade exponencial), e soluções sem garantia de eficiência, como o caso da programação dinâmica (pseudopolinomial); e por isso, ainda que não tenhamos encontrado soluções eficientes para muitos problemas em NP, não significa que elas não existem, a menos que se prove o contrário.

Referências

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). *Introduction to algorithms*. MIT press.

Wayne, K. (2013). Dynamic programming. `http://www.cs.princeton.edu/~wayne/cs423/lectures/dynamic-programming-4up.pdf`. acessado em: 01/12/2018.

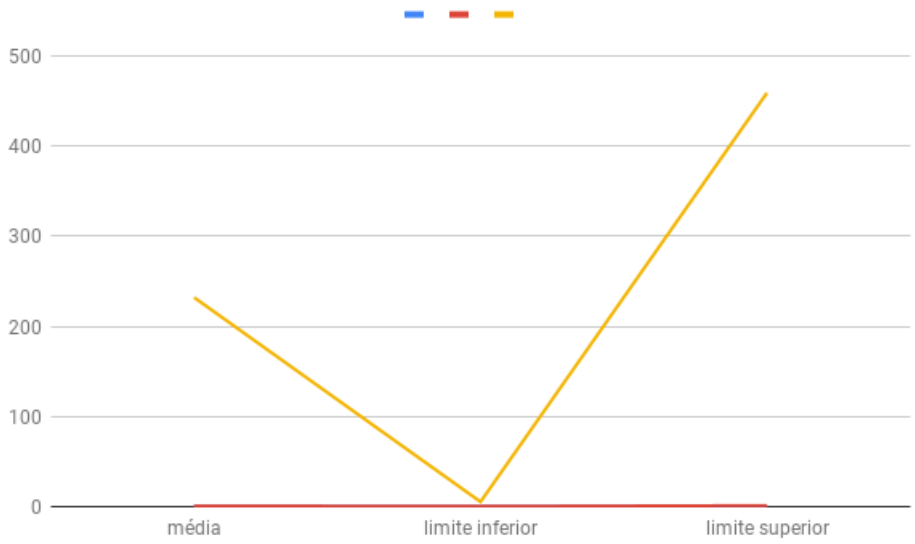

Figura 2. Algoritmo backtracking, cenário aleatório

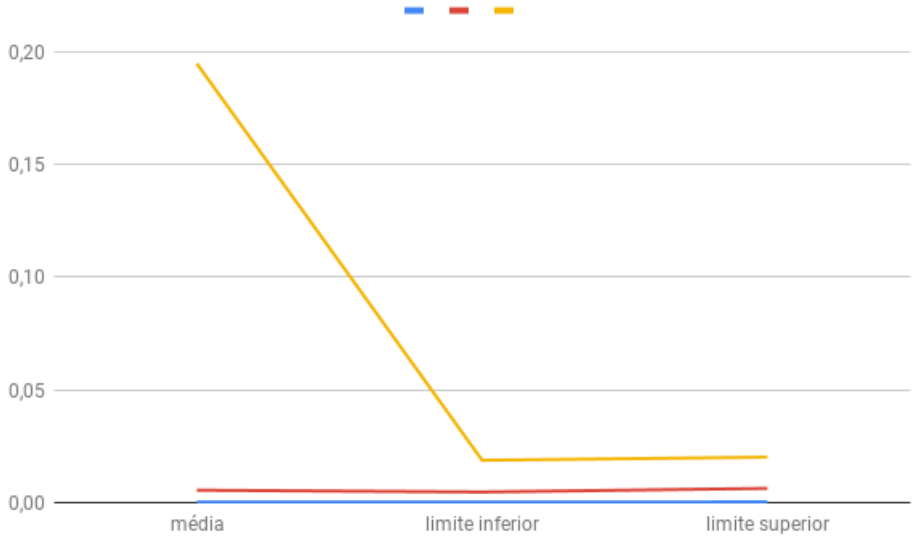

Figura 3. Algoritmo dinâmico, cenário aleatório

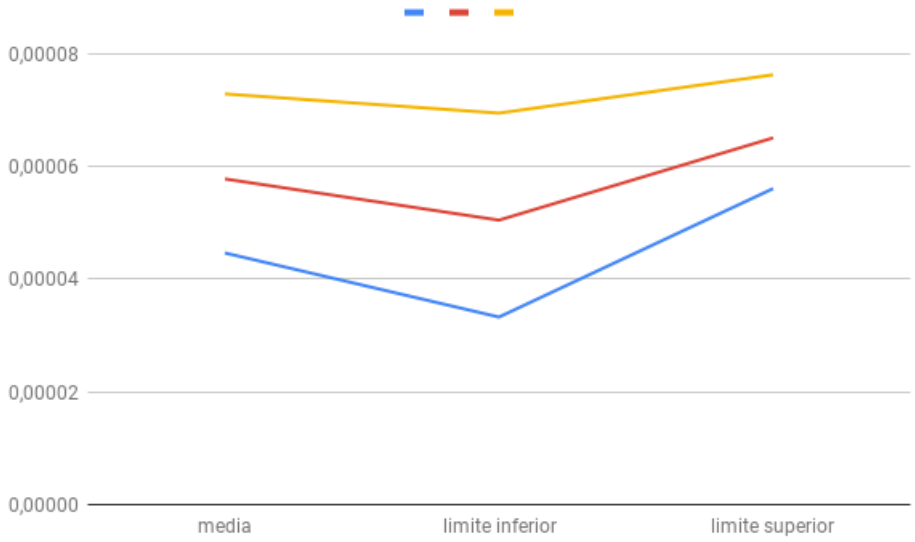

Figura 4. Algoritmo guloso, cenário otimista

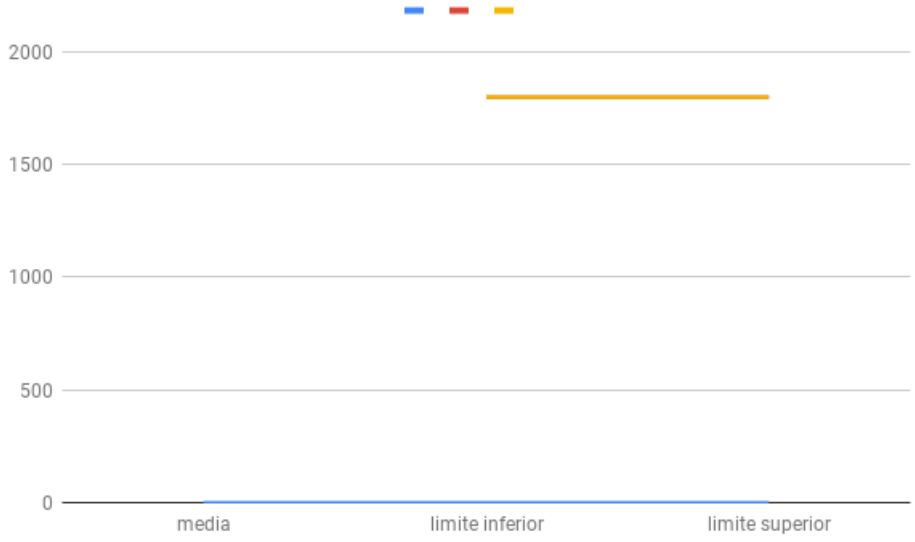

Figura 5. Algoritmo backtracking, cenário otimista

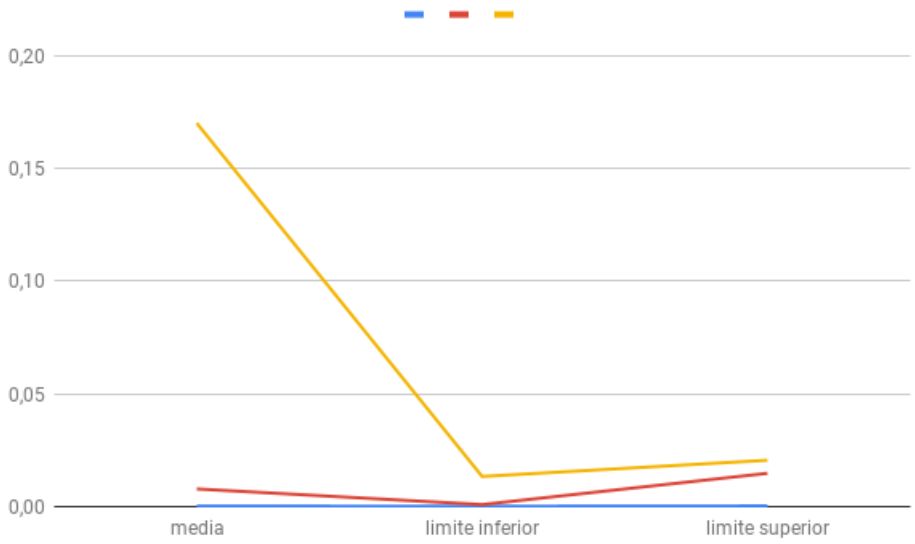

Figura 6. Algoritmo dinâmico, cenário otimista

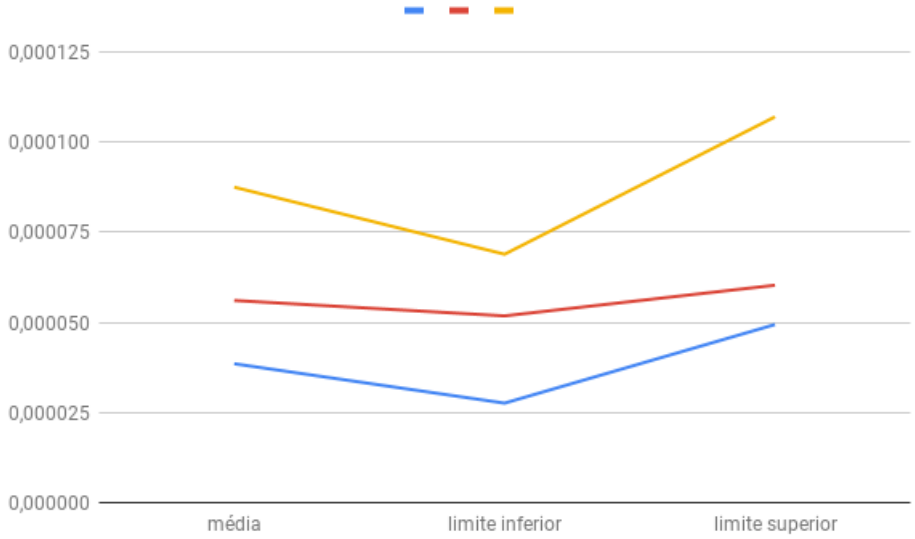

Figura 7. Algoritmo guloso, cenário pessimista

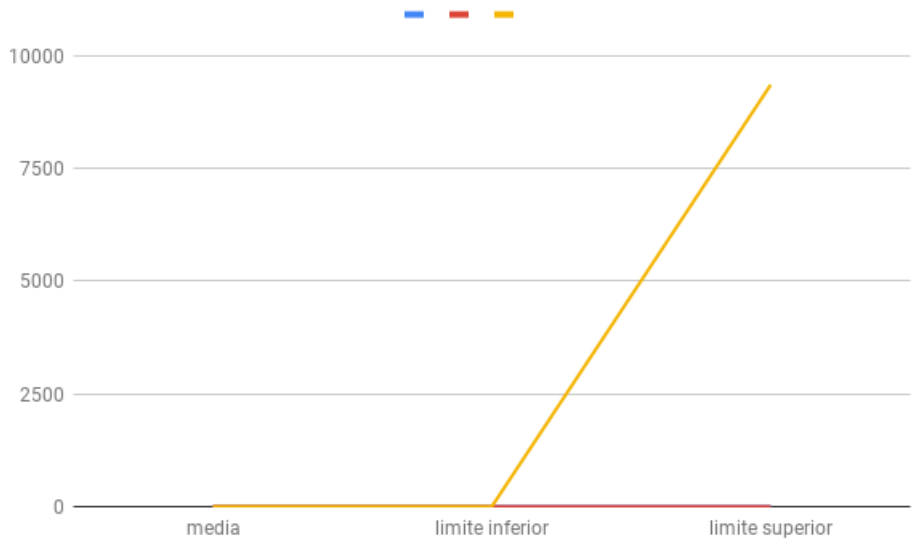

Figura 8. Algoritmo backtracking, cenário pessimista

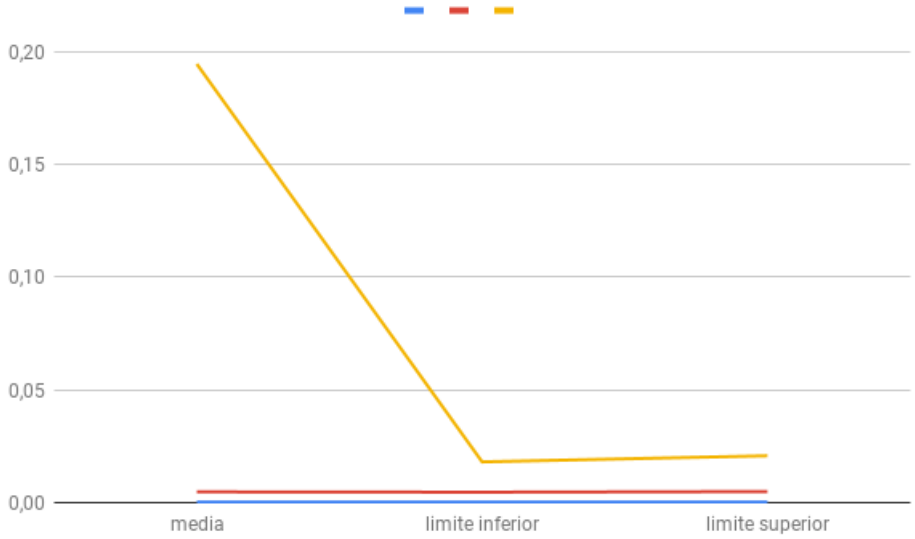

Figura 9. Algoritmo dinâmico, cenário pessimista