Universidade Federal de Mato Grosso do Sul
Campus Ponta Porã
Análise de Algoritmos I

Trabalho Prático II

Problema da Mochila Booleana

Aluno: Daniel de Leon Bailo da Silva
Professor: Eduardo Theodoro Bogue

Maio
2019

Sumário

Resumo

Este trabalho consiste em mostrar os resultados obtidos a partir da execução do algoritmo da *Mochilha Boolena* ou *Knapsack 0/1*, em suas versões dinâmicas.

Feito isso, dada as instâncias contento os dados necessários para aplicar os algoritmos, foi comparado o tempo de execução para cada instância nas suas versões dinâmicas, *Top-Down* e *Bottom-Up*.

Considerar o seguinte ambiente para a obtenção dos resultados:

- Processador: Intel Core™ i5-8250U
  - Número de núcleos 4
  - Número de threads 8
  - Frequência baseada em processador 1.60 GHz
  - Frequência turbo max 3.40 GHz
- Memória: 8GB RAM

Este trabalho foi armazenado num repositório *GitHub* para melhor controle do versionamento do programa.
https://github.com/danbailo/T2-Analise-Algoritmos-I

1 Introdução

Supondo que uma mochila com capacidade total de W e n itens distintos, em que cada um dos itens possui um peso p_i e um valor v_i , tal que $1 \leq i \leq n$. O *Problema da Mochila 0/1* consiste em determinar o valor máximo que podemos obter colocando um subconjunto de itens na mochila, tal que o somatório do peso dos itens não ultrapasse a capacidade W da mochila.

Visto que na *Programação Dinâmica* os problemas podem ser abordados de duas formas, o *Top-Down* e o *Bottom-Up* , devemos saber que entre eles existem vantagens e desvantagens quando comparados um com o outro.

Top-Down

Simplesmente uma recursão normal com a adição de uma tabela *memoization*.

Bottom-Up

- Prepare uma tabela com o tamanho do número de estados do problema.
- Comece a preencher a tabela através de casos triviais.
- Preencha a tabela de acordo com a ordem topológica do problema.

Vantagens e Desvantagens

- Top-Down
  - Transformação natural através da recursão.
  - Apenas computa um subproblema se ele for necessário.
  - Mais lento se ocorrerem muitas chamadas a subproblemas devido ao overhead recursivo.
- Bottom-Up
  - Mais rápido se muitos subproblemas são computados.
  - Pode ter uma economia de espaço em alguns casos (não precisar criar uma tabela com todos os subproblemas).
  - Não é tão intuitivo.
  - Computa todos os subproblemas.

2 Knapsack Top-Down

Código do algoritmo *Knapsack 0/1 Top-Down* que foi utilizado para obter os resultados onde o mesmo foi escrito na linguagem de programação *Python*.

```
1  def Knapsack(number_items, weight_max, values_items,
       weight_items):
2      if number_items == 0 or weight_max == 0: return 0
3      if weight_items[number_items-1] > weight_max:
4          return Knapsack(number_items-1,weight_max,values_items,
       weight_items)
5      if mem[number_items][weight_max] is not False:
6          return mem[number_items][weight_max]
7      temp = max(Knapsack(number_items-1,
8      weight_max-weight_items[number_items-1],values_items,
       weight_items)+values_items[number_items-1], Knapsack(
       number_items-1, weight_max, values_items, weight_items))
9      mem[number_items][weight_max] = temp
10     return temp
```

Algoritmo 1: Top-Down

2.1 Resultados

Após aplicar o algoritmo Top-Down nas instâncias propostas pelo professor, os seguintes resultados foram obtidos.

Tempo total para executar todas instâncias:	28.5181 segundos
Média de tempo de execução por instância:	0.9199 segundos
Maior tempo gasto por uma das instâncias:	1.8031 segundos
Menor tempo gasto por uma das instâncias:	0.25080 segundos
Instância com maior tempo de execução:	s025.kp
Instância com menor tempo de execução:	s011.kp

Tabela 1: Resultados Top-Down para 1 execução

Amplitude:	1.5523 segundos
Erro:	0.1190 segundos
Variância:	0.4251 segundos
Desvio Padrão:	0.6520 segundos
Desvio Absoluto:	0.2629 segundos

Tabela 2: Estatísticas

Nos gráficos representados pelas figuras 1 e 2, podemos ver o tempo que cada instâncias demorou para antingir tal resultado e qual resultado foi obtido, respectivamente. Na figura 3 conseguimos vizualizar o tempo de execução para cada instância.

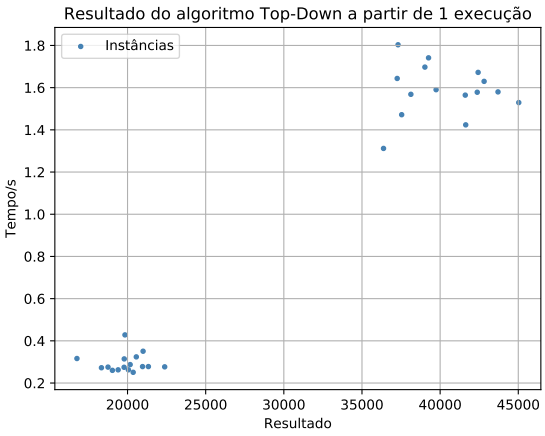

Figura 1: Tempo/Resultado.

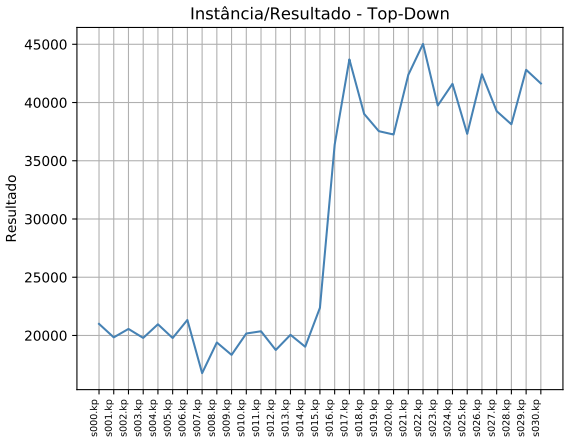

Figura 2: Resultado/Instância.

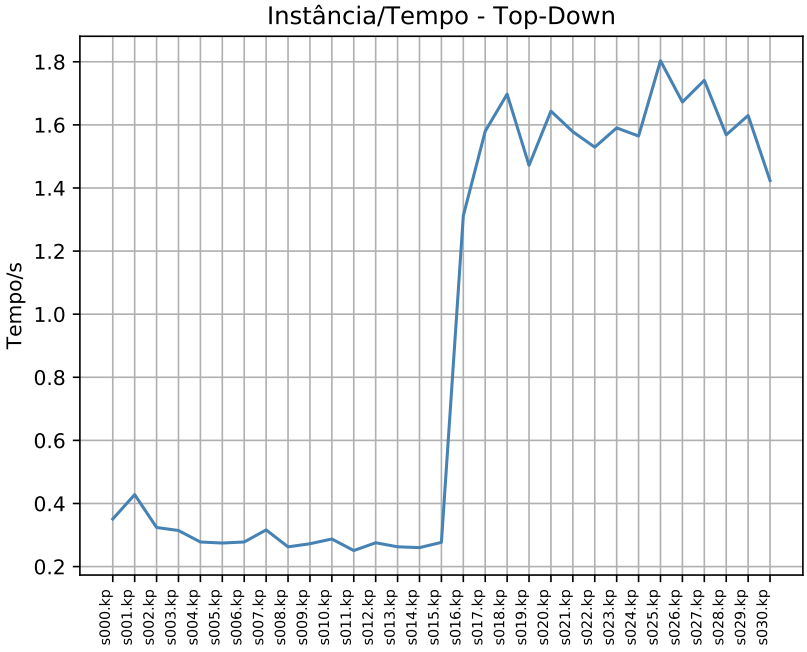

Figura 3: Gráfico do Tempo/Instância.

3 Knapsack Bottom-Up

Código do algoritmo *Knapsack 0/1 Bottom-Up* que foi utilizado para obter os resultados onde o mesmo foi escrito na linguagem de programação *Python*.

```
1 def Knapsack(number_items, weight_max, values_items,
    weight_items):
2     K = [[0 for x in range(weight_max + 1)] for x in range(
    number_items + 1)]
3     for i in range(number_items + 1):
4         for w in range(weight_max + 1):
5             if i == 0 or w == 0: K[i][w] = 0
6             elif weight_items[i-1] <= w:
7                 K[i][w] = max(values_items[i-1]+K[i-1][w-
    weight_items[i-1]], K[i-1][w])
8             else: K[i][w] = K[i-1][w]
9     return K[number_items][weight_max]
```

Algoritmo 2: Bottom-Up

3.1 Resultados

Após aplicar o algoritmo Bottom-Up nas instâncias propostas pelo professor, os seguintes resultados foram obtidos.

Tempo total para executar todas instâncias:	19.5897 segundos
Média de tempo de execução por instância:	0.6319 segundos
Maior tempo gasto por uma das instâncias:	1.2442 segundos
Menor tempo gasto por uma das instâncias:	0.1975 segundos
Instância com maior tempo de execução:	s025.kp
Instância com menor tempo de execução:	s014.kp

Tabela 3: Resultados Bottom-Up para 1 execução

Amplitude:	1.0466 segundos
Erro:	0.0763 segundos
Variância:	0.1750 segundos
Desvio Padrão:	0.4184 segundos
Desvio Absoluto:	0.1227 segundos

Tabela 4: Estatísticas

Nos gráficos representados pelas figuras 1 e 2, podemos ver o tempo que cada instâncias demorou para antingir tal resultado e qual resultado foi obtido, respectivamente. Na figura 3 conseguimos vizualizar o tempo de execução para cada instância.

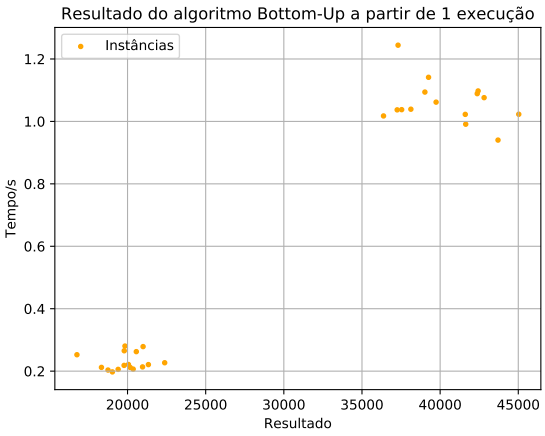

Figura 4: Tempo/Resultado.

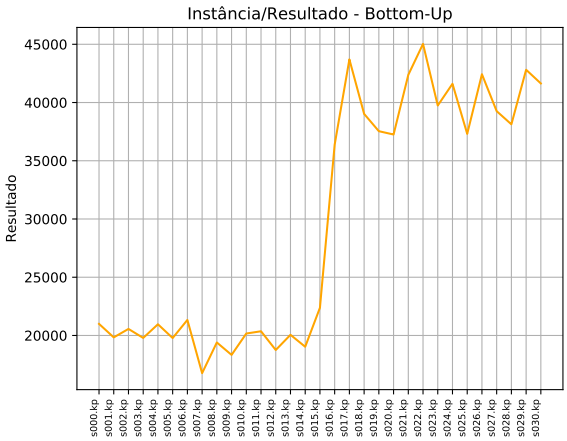

Figura 5: Resultado/Instância.

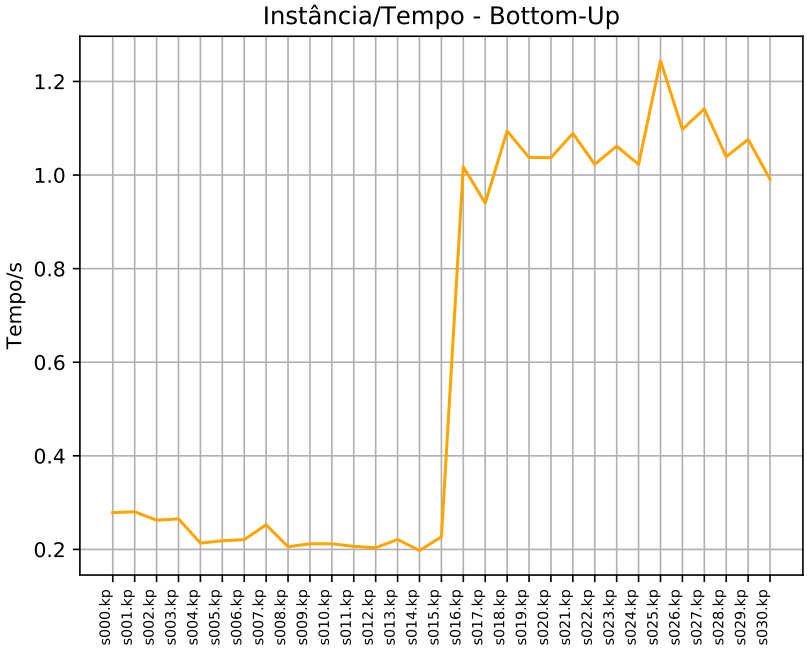

Figura 6: Gráfico do Tempo/Instância.

4 Análise dos Resultados

Como podemos ver abaixo, na figura 7, que é responsável por mostrar os resultados atingidos para cada instância, os gráficos estão sobrepostos e isso se deve pelo fato de que, independente dos algoritmos serem escritos e abordarem ideias diferentes, o objetivo final é o mesmo.

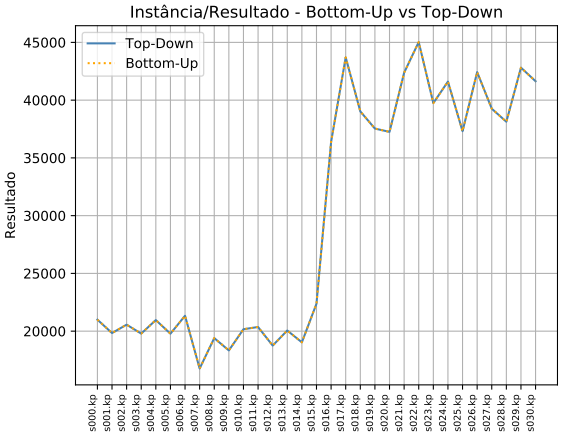

Figura 7: Gráfico comparativo dos resultados.

Abaixo, podemos ver que o tempo necessário para atingir os mesmos resultados são bem maiores quando comparamos o Knapsack Top-Down ao Knapsack Bottom-Up, computacionalmente falando. Porém, era de se esperar um resultado como esse, como vimos na página 2 as características de cada um.

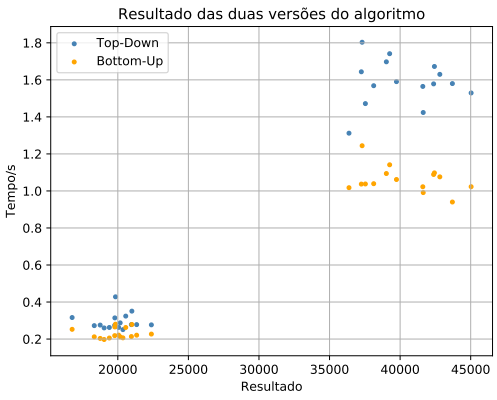

Figura 8: Gráfico do Tempo/Resultado

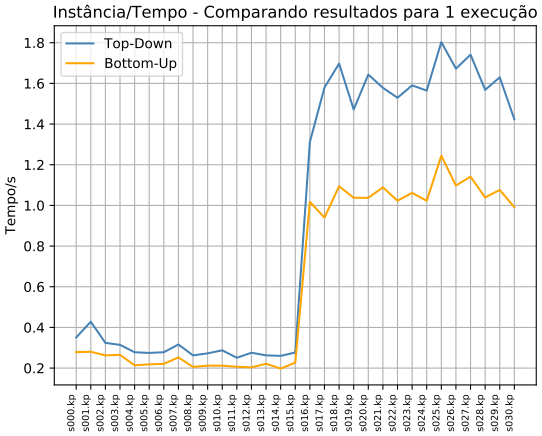

Figura 9: Gráfico do Tempo/Instância

4.1 Minimizando Resultados

A fim de comprovar o comportamento dos gráficos, decidi executar **100 vezes** as instâncias com o objetivo de minimizar a variação do tempo de execução de cada instância.

Para cada instância é armazenado seu tempo de execução individual, logo, foi necessário a cada execução armazenar e somar o tempo de cada instância para obter o tempo gasto para esse algoritmo para uma única execução. Feito isso, bastou repetir o mesmo processo para todas iterações. Assim, obtive o tempo total de execução para os dois algoritmos. O tempo médio é em relação ao tempo que uma única instância gastou em 100 execuções.

4.2 Top-Down

Tempo total de execução: 39 minutos
Tempo médio gasto por cada instância no total: 1 minuto e 25 seg

Tabela 5: Tempo total

Após coletar os dados de todas as execuções, foi calculado a média total para cada item e obtido um resultado mais exprimido a fim de minimizar a variação.

Tempo de execução: 23.4481 segundos
Média de execução por instância: 0.7563 segundos
Maior tempo gasto por uma das instâncias: 1.3748 segundos
Menor tempo gasto por uma das instâncias: 0.2420 segundos
Instância com maior tempo de execução: s027.kp
Instância com menor tempo de execução: s014.kp

Tabela 6: Resultados a partir de 100 execuções

Amplitude: 1.1327 segundos
Erro: 0.0907 segundos
Variância: 0.2472 segundos
Desvio Padrão: 0.4972 segundos
Desvio Absoluto: 0.1421 segundos

Tabela 7: Estatísticas

4.2.1 Ganhos

Ao compararmos os resultados das tabelas 1 e 2 com as tabelas 6 e 7, podemos ver qual é a diferença entre entre os seguintes resultados obtidos:

p/ Tempo de execução: 5.07 segundos
p/ Média de execução por instância: 0.1636 segundos
p/ Maior tempo gasto por uma das instâncias: 0.4282 segundos
p/ Menor tempo gasto por uma das instâncias: 0.0088 segundos

Tabela 8: Diferença de 1 execução para 100 execuções

p/ Amplitude: 0.4196 segundos
p/ Erro: 0.0282 segundos
p/ Variância: 0.1778 segundos
p/ Desvio Padrão: 0.1548 segundos
p/ Desvio Absoluto: 0.1208 segundos

Tabela 9: Estatísticas

4.3 Bottom-Up

Tempo total de execução: 29 minutos
Tempo médio gasto por cada instância no total: 55.9276 segundos

Tabela 10: Tempo total

Após coletar os dados de todas as execuções, foi calculado a média total para cada item e obtido um resultado mais exprimido a fim de minimizar a variação.

Tempo de execução: 17.3945 segundos
Média de execução por instância: 0.5611 segundos
Maior tempo gasto por uma das instâncias: 0.9881 segundos
Menor tempo gasto por uma das instâncias: 0.1927 segundos
Instância com maior tempo de execução: s027.kp
Instância com menor tempo de execução: s014.kp

Tabela 11: Resultados a partir de 100 execuções

Amplitude: 0.7954 segundos
Erro: 0.0638 segundos
Variância: 0.1221 segundos
Desvio Padrão: 0.3495 segundos
Desvio Absoluto: 0.1244 segundos

Tabela 12: Estatísticas

4.3.1 Ganhos

Ao compararmos os resultados das tabelas 3 e 4 com as tabelas 11 e 12, podemos ver qual é a diferença entre entre os seguintes resultados obtidos:

p/ Tempo de execução: 2.1952 segundos
p/ Média de execução por instância: 0.0707 segundos
p/ Maior tempo gasto por uma das instâncias: 0.2561 segundos
p/ Menor tempo gasto por uma das instâncias: 0.0047 segundos

Tabela 13: Diferença de 1 execução para 100 execuções

p/ Amplitude: 0.2512 segundos
p/ Erro: 0.0125 segundos
p/ Variância: 0.0528 segundos
p/ Desvio Padrão: 0.0689 segundos
p/ Desvio Absoluto: -0.0016 segundos

Tabela 14: Estatísticas

Como podemos ver, o objetivo de minimizar a variação foi atingido. Visto que, para o *Top-Down* teve um ganho de 5.07 segundos e para o *Bottom-Up* 2.1952 segundos, tempo que, computacionalmente falando, já é muito alto!

Em relação aos dados estatísticos, todos foram minimizados também, isso mostra que, quanto menor o erro, variação e o desvio, mais próximo do exato é o resultado final.

5 Resultados Obtidos

A figura abaixo representa o gráfico dos resultados obtidos após realizar a média de 100 execuções das instâncias, como foi visto nas tabelas 6 e 11.

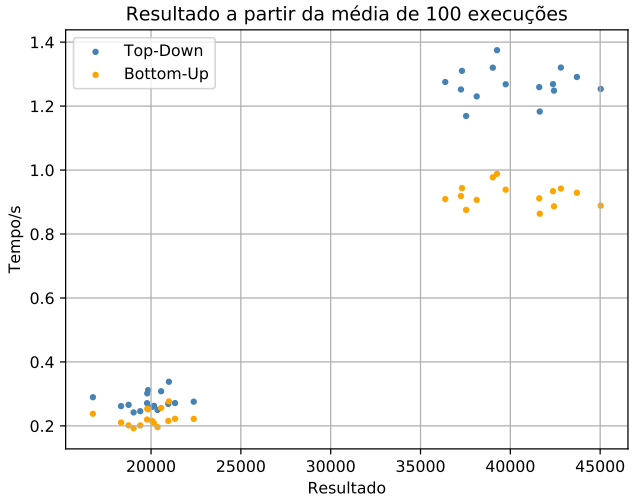

Figura 10: Gráfico do Tempo/Resultado

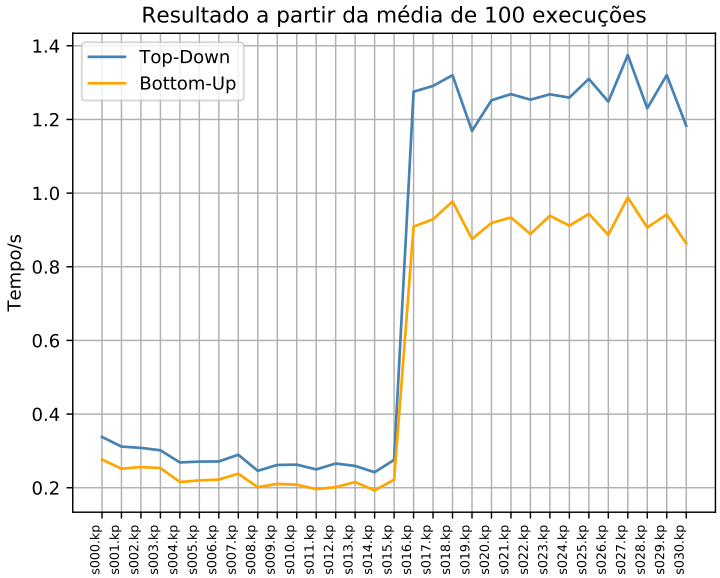

Figura 11: Gráfico do Tempo/Instância

5.1 Resultados Finais

Com isso, podemos concluir que, o objetivo de minimizar a variação do tempo de cada execução foi atingida.

Nos gráficos abaixo, temos ilustrado o tempo que demandou cada resultado, sendo que, as legendas que possuem *"AVG"* no nome, significa que são os gŕaficos após realizar a média entre as 100 execuções. Logo, é uma mesclagem das figuras 9 e 11.

Portanto, estas figuras representam a união de todos resultados mostrados até agora.

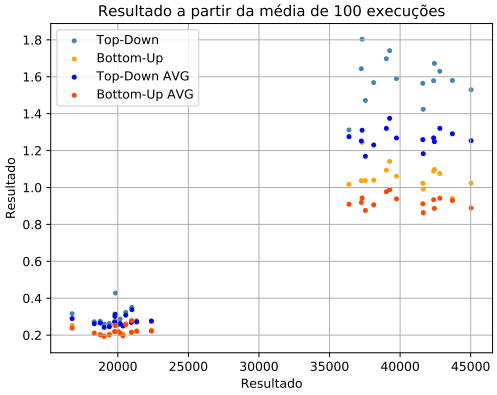

Figura 12: Resultados - Mesclado

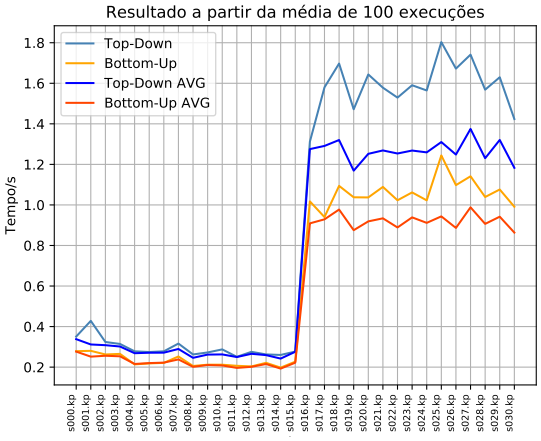

Figura 13: Resultados - Mesclado