Universidade Federal de Mato Grosso do Sul
Campus Ponta Porã
Teoria da Computação

Trabalho Prático I

Heurísticas para o Problema da Mochila Booleana

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

Setembro
2019

Sumário

Resumo

A proposta deste trabalho consistia em desenvolver uma *Heurística Gulosa* e uma *Heurística GRASP* para o *Problema da Mochila Boolena* e aplicá-las nas instâncias disponibilizadas para teste e comparar os resultados obtidos para as duas versões.

Porém, a fim realizar uma pesquisa mais concreta para realizar as comparações dos algoritmos, foram desenvolvidas três heurísticas gulosas e algumas variações de uma mesma heurística GRASP, alterando o tamanho da sua janela e o número de iterações do mesmo. Após comparar os resultados entre si, foi escolhido a melhor heurística gulosa e a melhor variação do GRASP, a partir do GAP obtido ao realizar a comparação desses resultados com os resultados exatos para cada instância. Logo, é possível saber exatamente o desempenho de cada heurística construída.

1 Introdução

O problema da Mochila(knapsack problem) pode ser enunciado da seguinte forma: Dados um número $W \geq 0$, um inteiro positivo n e, para cada i em $1, ..., n$, um número $v_i \geq 0$ e um número $w_i \geq 0$, encontrar um subconjunto S de $1, ..., n$ que maximize $v(S)$ sob a restrição $w(S) \leq W$. Onde, v(S) denota a soma $\sum_{i \in S} v_i$ e, analogamente, $w(S)$ denota a soma $\sum_{i \in S} w_i$.

Os números v_i e w_i podem ser interpretados como o valor e peso respectivamente de um objeto i. O número W pode ser interpretado como a capacidade de uma mochila, ou seja, o peso máximo que a mochila comporta. O objetivo do problema é então encontrar uma coleção de objetos, a mais valiosa possível, que respeite a capacidade da mochila.

Este problema vem sendo estudado deste o trabalho de *D.G. Dantzig* [2], devido a sua utilização imediata na Indústria, porém foi mais enunciado por razões teóricas, uma vez que este frequentemente ocorre pela relaxação de vários problemas de programação inteira. Toda a família de **Problemas da Mochila** requer que um subconjunto de itens sejam escolhidos, de tal forma que o somatório dos seus valores seja maximizado sem exceder a capacidade da mochila. Diferentes tipos de problemas da Mochila ocorrem dependendo da distribuição de itens e Mochilas como citado em [2]:

No problema da *Mochila 0/1(0/1 Knapsack Problem)*, cada item pode ser escolhido no máximo uma vez, enquanto que no problema da *Mochila Limitado(Bounded Knapsack Problem)* existe uma quantidade limitada para cada tipo de item. O problema da *Mochila com Múltipla Escolha(Multiple-choice Knapsack Problem)* ocorre quando os itens devem ser escolhidos de classes disjuntas, e se várias Mochilas são preenchidas simultaneamente este é o problema da *Mochila Múltiplo(Multiple Knapsack Problem)*. A forma mais geral é o problema da *Mochila com multirestrições (Multi-constrained Knapsack Problem)* o qual é basicamente um problema de *Programação Inteira Geral com Coeficientes Positivos*.

Todos os problemas da Mochila pertencem a família **NP-Hard** [2], significando que é muito improvável que possamos desenvolver algoritmos polinomiais para este problema. Porém, apesar do tempo para o pior caso de todos os algoritmos terem tempo exponencial, diversos exemplos de grandes instâncias podem ser resolvidos de maneira ótima em fração de segundos. Estes resultados surpreendentes vem de várias décadas de pesquisa que tem exposto as propriedades estruturais especiais do Pro-

blema da Mochila, que tornam o problema tão relativamente fácil de resolver.

Neste trabalho, foram desenvolvidas três heurísticas gulosas e algumas variações de uma mesma heurística GRASP que resolvem o Problema da Mochila 0/1, que consiste em escolher n itens, tais que o somatório dos preços(valor) é maximizado sem que o somatório dos pesos extrapolem a capacidade da Mochila. Logo, dada as instâncias para realizar os experimentos, foi comparado o resultado obtido para cada instância a partir das heurísticas com o resultado ótimo para cada instância, a fim de verificar o quão bom ficou cada heurística programada.

1.1 Tramento de Problemas NP-Hard

Para tratar um problema NP-Hard, deve-se sacrificar uma das seguintes características:

1. Resolver o problema na otimalidade.
2. Resolver o problema em tempo polinomial.

Para isto, pode-se desenvolver:

- Algoritmos Aproximados, nos quais estes sacrificam 1.
- Algoritmos Exatos, nos quais estes sacrificam 2.
- Algoritmos Heurísticos, nos quais estes sacrificam 1 e possivelmente 2.

1.2 Heurísticas

Algoritmos heurísticos são aqueles que não apresentam garantia de determinação da solução ótima para o problema estudado. Os algoritmos aproximados podem se enquadrar nesta categoria, acrescentando-se que, para estes casos, são conhecidas propriedades com garantia do pior caso. Entretanto, nem todo algoritmo heurístico é aproximado, ou seja, nem toda heurística tem uma razão de qualidade comprovada matematicamente ou prova formal de convergência, ou seja a heurística é um conjunto de regras e métodos que conduzem à descoberta, à invenção e à resolução de problemas. Por este motivo, em várias referências bibliográficas distingue-se os termos algoritmo aproximado e heurística:

- Aproximado é a denominação do algoritmo que fornece soluções dentro de um limite de qualidade absoluto ou assintótico, assim como um limite assintótico polinomial de complexidade (pior caso) comprovado matematicamente;
- Heurística e método heurístico são denominações para o algoritmo que fornece soluções sem um limite formal de qualidade, tipicamente avaliado empiricamente em termos de complexidade (média) e qualidade das soluções.

1.2.1 GRASP

GRASP (greedy randomized adaptive search procedure) é uma metaheurístisca multistart para problemas de otimização combinatória, na qual cada iteração consiste basicamente de duas fases: construção e busca local. A fase de construção cria uma solução viável cuja vizinhança é investigada até que um mínimo local seja encontrado durante a fase de busca local. A melhor solução geral é mantida como resultado [1].

2 Avaliação Experimental

Foram disponibilizadas 16 instâncias para realizar os testes e quantidade de itens dessas instâncias variam de 20 até 10000 itens. Todos os algoritmos abordados neste trabalho foram escritos na linguagem de programação *Python* e estão disponibilizados no seguinte repositório: https://github.com/danbailo/T1-Teoria-Computacao.

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 máx. 3.40 GHz.
- Memória: 8GB RAM.

2.1 Algoritmo Exato

O algoritmo utilizado para obter os resultados ótimos foi a versão *Bottom-up* do Problema da Mochila 0/1.

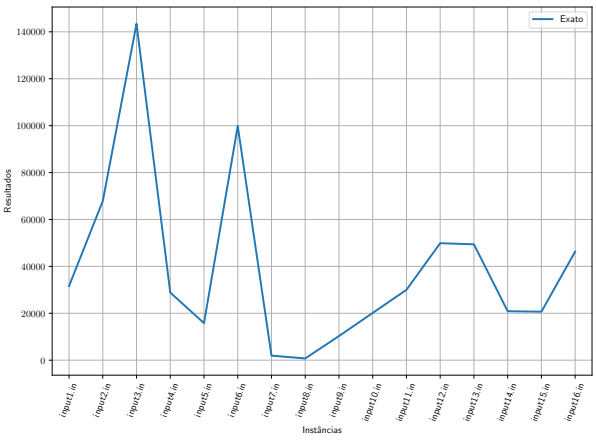

Figura 1: Resultados exatos.

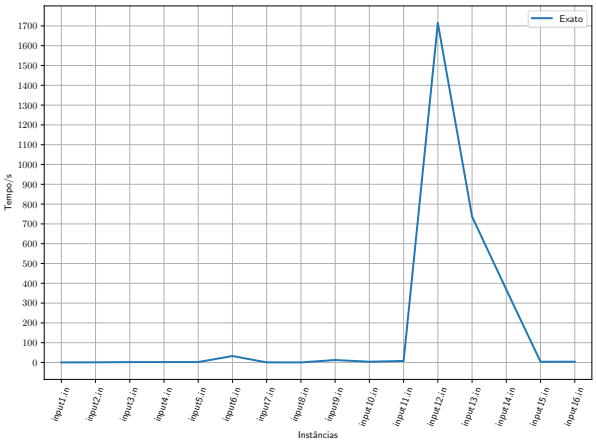

Figura 2: Tempo para obtenção dos resultados.

Como é possível observar na figura 2, para algumas instâncias, o algoritmo retornou o resultado ótimo instantaneamente, mas em contrapartida, para algumas instâncias o tempo para obtenção do mesmo foi muito alto. Por exemplo, a instância *"input12.in"*, demorou mais de *1700 segundos*, ou seja, aproximadamente *28 minutos*.

O objetivo então é tentar se aproximar o máximo possível desses resultados ou até mesmo chegar nesses, mas em um tempo ideal.

2.2 Heurísticas Gulosas

A fim de melhorar o tempo na aquisição dos resultados, foram programadas três heurísticas gulosas. Essas heurísticas foram nomeadas e desenvolvidas da seguinte forma:

- Crescente: O itens foram ordenados de acordo com seu respectivo peso, de forma crescente;
- Decrescente: O itens foram ordenados de acordo com seu respectivo peso, de forma decrescente;
- Eficiente: Os itens foram ordenados de forma decrescente de acordo com a razão $\frac{valor}{peso}$.

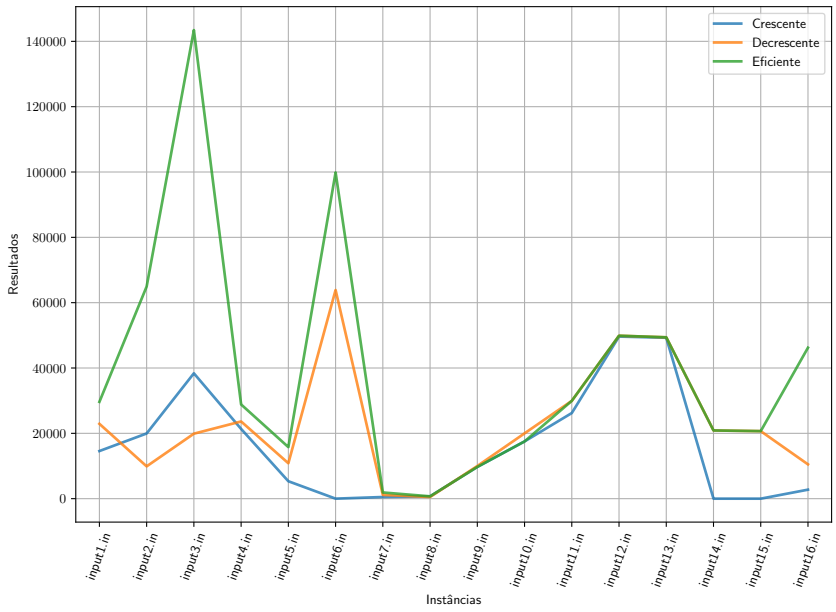

Figura 3: Comparação entre resultados das Heurísticas Gulosas.

Como pode-se ver na figura 3, existe uma grande discrepância entre os resultados obtidos de uma versão gulosa para outra e nesse caso, quanto maior o resultado obtido, melhor é a heurística quanto a proximidade aos resultados ótimos. Uma forma de garantir o quão boa é a heurística, é realizando o cálculo do **GAP médio**, que é dado como:

$$(\sum_{i=1}^{n}((x_i - y_i)/x_i) \cdot 100)/n \tag{1}$$

Onde x_i é o resultado exato na *i-ésima* instância e y_i o resultado obtido pela heurística na *i-ésima* instância e n a quantidade de instâncias. Logo, quanto mais proximo de zero for o resultado final, melhor. Supondo que alguma heurística alcance um GAP de zero, quer dizer que esta obteve os resultados ótimos para estas instâncias.

Heurísticas	**GAP**
Crescente	49.594312
Decrescente	27.347272
Eficiente	2.213083

Tabela 1: GAP médio - Heurísticas Gulosas

Portanto, a heurística gulosa Eficiente é a melhor dentre as três apresentadas neste experimento.

2.3 GRASP

Para a obtenção dos resultados utilizando a metaheurístisca GRASP, a *semente* para a geração dos resultados randômicos foi definida com o valor de 42. Ao definir um valor estático para a semente, os resultados que serão gerados na fase de construção, partirão do último elemento gerado na mesma cadeia a cada iteração do GRASP. Ou seja, supondo que na primeira iteração de uma janela qualquer o numero gerado foi k, então, na próxima iteração, após realizar a busca local, a cadeia de números geradas randomicamente, partirá de $k+1$, portanto, é uma forma de evitar que a cada execução para uma mesma janela, o GRASP gere resultados totalmente aleatórios.

Com a intenção de escolher um bom valor para o tamanho da janela e assim obter bons resultados locais, foi realizado um experimento selecionando oito valores distintos para o tamanho da janela, onde este tem total influência no algoritmo na fase de construção, ou seja, cada janela se remete a um "novo"algoritmo. Esses valores estão no intervalo de $[2, 9]$, logo, serão 8 variações diferentes somente considerando o tamanho da janela.

Além disso, também foi foram escolhidos quatro números diferentes como critério de parada nas iterações do GRASP, isto é, o número máximo de iterações, os valores são $10, 100, 1000$ e 10000, ou seja, para cada valor distinto, o algoritmo só irá parar ao realizar este número de iterações e para estes, cada número máximo de iterações, é também uma variação diferente do GRASP. Logo, é possivel afirmar que serão analisadas 24 versões a partir de um mesmo algoritmo GRASP.

Na figura 4, é possivel notar um comportamento interessante quanto aos resultados obtidos, isto porque para cada número máximo de iterações, os resultados se diferem um do outro quanto ao tamanho da janela quando o número de iterações é menor, ou seja, 10 e 100. Para os valores 100 e 1000, independente do tamanho da janela, os resultados obtidos são praticamente os mesmos, por isso se observa um comportamento "constante".

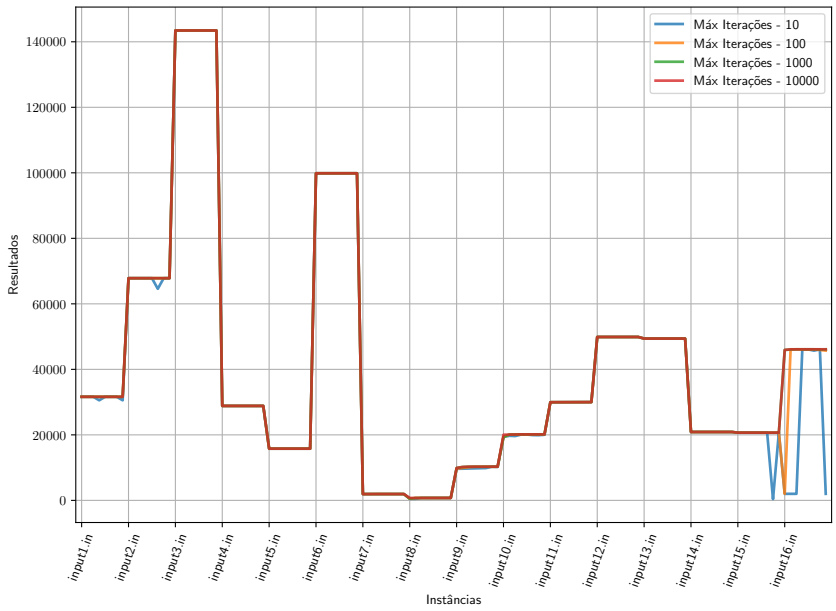

Figura 4: Resultados para as janelas e número de iterações específicos.

A fim de escolher o melhor número de iterações para posteriormente analisar o

melhor tamanho da janela para estes experimentos, a figura 5 mostra o cálculo do GAP/Instância, considerando que para cada instância, existem vários tamanhos de janela, por isso é possivel notar alguns "picos"sobre o eixo num intervalo de uma mesma instância; isso significa que para alguma janela deste mesmo número de iterações, o resultado divergiu mais da otimalidade.

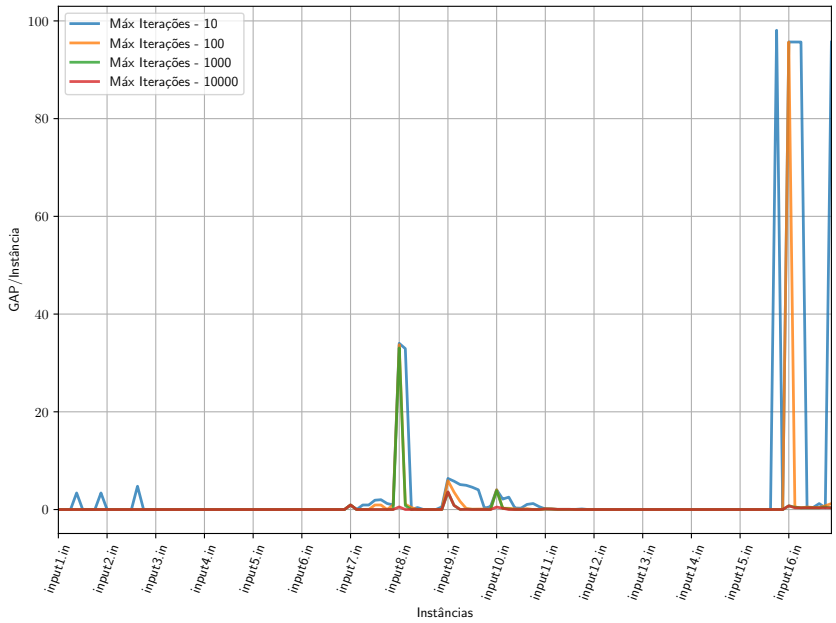

Figura 5: GAP/Instância.

Supostamente, o número de iterações que mostra ser o melhor dentre os testados, é o de 10000 iterações, a tabela 3, apresenta o GAP médio para dos resultados para todas as janelas/número de iteração.

Heurísticas	GAP médio
Máx Iterações - 10	4.811709
Máx Iterações - 100	1.217037
Máx Iterações - 1000	0.370746
Máx Iterações - 10000	0.080223

Tabela 2: GAP médio para todas as janelas e iterações - GRASP

Portanto, pode-se afirmar que, de fato, para um número de iterações mais reduzido, o GAP é maior, pois dependendo do tamanho da janela os resultados gerados podem se divergir muito e a tendência para um número maior de iterações é que os vizinhos e resultados gerados, sejam cada vez mais próximos e melhores.

A partir disso, foi selecionado somente os resultados gerados a partir de 10000 iterações e então foi desenvolvido um procedimento para escolher o melhor tamanho de janela dentre o intervalo $[2, 9]$.

O procedimento funciona da seguinte forma, para cada instância foi armazenado o índice das janelas que obtiveram o maior valor dentre todas as janelas daquela instância. Por exemplo, para a instância *"input1.in"*, todos as janelas obtiveram como resultado o mesmo valor, ou seja, o valor máximo era comum para todas as janelas nessa instância, então foi armazenado o índice de todas as janelas na *"input1.in"*.

```
function FIND_OCCURRENCES(grasp)
    all_windows_ocurr = []
    for inst in instancias:
        ocurr = np.where(grasp.loc[inst].values == grasp.loc[inst].values.max())
        all_windows_ocurr += list(ocurr)
    best_window = 0
    for i in range(len(janelas)):
        if all_windows_ocurr.count(i) > best_window:
            best_window = all_windows_ocurr.count(i)
    return best_window
end function
```

Este procedimento foi realizado para todas as instâncias e ao final deste, o índice da janela que teve mais ocorrências de valores máximos foi escolhida como o melhor tamanho de janela para prosseguir com os testes. Se ao final deste procedimento, houver mais de uma janela com a mesma quantidade de ocorrências de valor máximo, a janela de menor índice é selecionada. Portanto, foi definido que o melhor tamanho de janela para essa quantidade de iterações foi a **Janela de tamanho 7.**

Então, a seguinte questão foi levantada: **"Será mesmo necessário realizar 10000 iterações para obter bons resultados?"**

Isto se deu pelo fato de 10000 iterações ser uma alta quantidade de repetições e este foi escolhido pelo fato de seu GAP médio ser o menor dentre os outros testados, porém foi realizado o mesmo processo para escolher a melhor janela para os resultados que foram obtidos a partir de 1000 iterações, já que o GAP médio deste era pouco maior do que o de 10000 iterações.

Surpreendentemente, a janela que obteve os melhores resultados tambem foi a janela 7, nesse caso, foi calculado o GAP médio somente para esta janela, nas duas versões em questão.

Versões - GRASP	**GAP médio**
GRASP 1000 it. - janela 7	0.022221
GRASP 10000 it. - janela 7	0.021815

Tabela 3: GAP médio - GRASP

Ainda assim, mesmo para essa janela específica, o GAP da versão de 10000 iterações é pouca coisa menor que o de 1000 iterações. Entretanto, foi realizado uma análise do tempo de execução para essas duas versões do GRASP, para levar em consideração se vale pena sacrificar um milésimo de um resultado mais próximo da otimalidade, para selecionar o que demanda menos custo computacional.

Pode-se observar na figura 6 e na tabela 4 a diferença de tempo para obtenção de cada resultado por instância e o tempo total para cada variação do GRASP, respectivamente. É possivel afirmar que a disparidade de tempo na obtenção dos resultados existe e é considerável para uma diferença tão pequena no GAP médio, ou seja, é aceitável afirmar também que após 1000 iterações pouco alterou-se os resultados obtidos em busca de uma melhor solução local.

A tabela 4 também mostra o **Speedup** obtido da versão de 10000 iterações para a versão de 1000 iterações, e como pode-se ver, a versão de 1000 iterações é 10 vezes mais rápida que a versão de 10000 iterações, o que em recurso computacional é muito

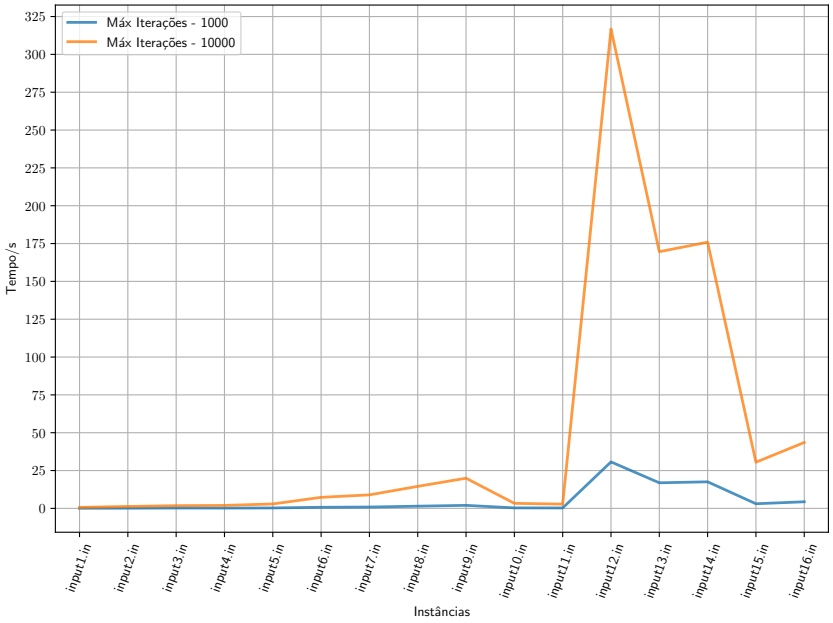

Figura 6: Comparação entre resultados das Heurísticas Gulosas.

Versões - GRASP	Tempo/s	Speed-up
Máx Iterações - 10000	801.770529	-
Máx Iterações - 1000	79.166350	10.1276

Tabela 4: Somatório do tempo para obtenção dos resultados.

relevante. Portanto, foi definido que a melhor versão do GRASP dentre as testadas, em relação a custo/benefício, é a versão de 1000 iterações com a janela de tamanho 7.

3 Conclusão

A figura 7 mostra a comparação final entre a melhor heurística gulosa e a melhor variação do GRASP dentre as testadas, com os resultados exatos. A figura 8 apresenta tempo em que cada algoritmo obteve tal resultado.

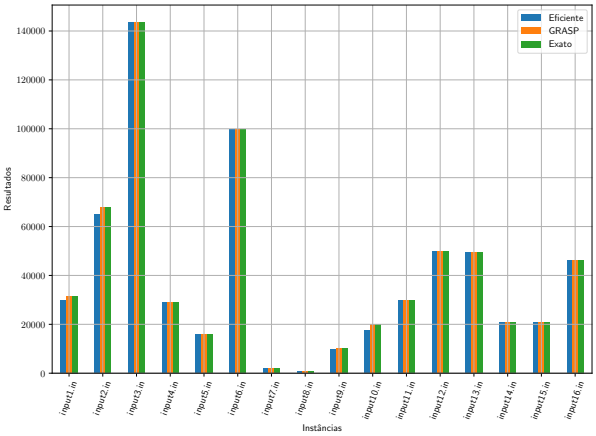

Figura 7: Comparação entre os resultados finais.

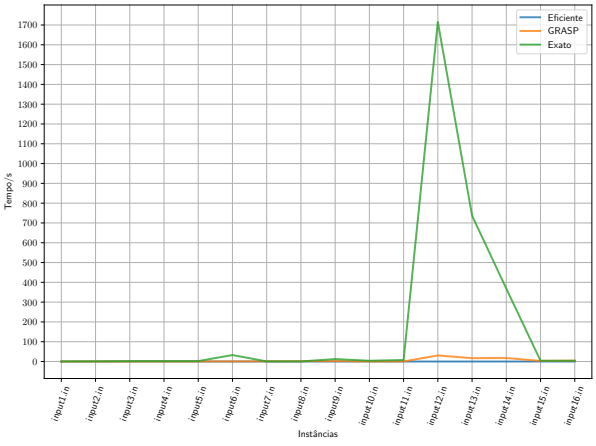

Figura 8: Comparação de tempo para obtenção dos resultados entre os algoritmos.

A tabela 5 mostra o tempo total em segundos, Speedup e o GAP para cada heurística programada. Como o algoritmo Exato é utilizado como base de comparação, o mesmo não possui Speedup e nem GAP.

Algoritmos	Tempo/s	Speedup	GAP médio
Exato	2893.456222	-	-
GRASP	79.166350	36.5490	0.022221
Eficiente	0.029479	98153.1334	2.213083

Tabela 5: GAP entre as melhores Heurísticas testadas.

Pode-se concluir que os resultados alcançados foram satisfatórios. As heurísticas não alcançaram a otimalidade , bem como visto na subseção 1.1, porém, foram bons resultados visto que o GAP é bem próximo de zero e o tempo de obtenção dos mesmos é mais reduzido. É importante notar que, as instâncias disponibilizadas não são tão grandes e mesmo assim, a atingir os resultados ótimos demandou bastante tempo, logo, é inviável utilizar um algoritmo exato para a obtenção de tal resultado num cenário onde as instâncias ultrapassam 100000 itens por exemplo.

Referências

[1] Thomas A Feo and Mauricio GC Resende. Greedy randomized adaptive search procedures. *Journal of global optimization*, 6(2):109–133, 1995.

[2] David Pisinger. Algorithms for knapsack problems. 1995.