
Atividade 7 – Heurísticas e PLI aplicadas ao Problema do Empacotamento Unidimensional

F.C. Pereira, Mestrando em Ciência da Computação com o RA 230214.

J.T. Belotti, Doutorando em Ciência da Computação com o RA 230260.

1. Introdução

O presente relatório descreve a sétima atividade avaliativa da disciplina de Tópicos em Otimização Combinatória, ministrada pelo Prof. Dr. Fábio L. Usberti. O objetivo da atividade consistiu na implementação e experimentação de algoritmos para o problema do Empacotamento Unidimensional (BPP, do inglês *Bin-Packing Problem*). Na experimentação, um conjunto de dez instâncias foi submetido a um resolvedor comercial de programação Programação Linear Inteira (PLI), a três algoritmos aproximados e à uma heurística baseada na metaheurística GRASP.

2. Descrição do problema

O BPP é um problema NP-difícil que pode ser descrito da seguinte forma: dados n itens e n mochilas, onde w_j é o peso do item j e c é a capacidade de cada mochila, deseja-se alocar cada item em uma mochila tal que a soma dos pesos dos itens em cada mochila não ultrapasse a capacidade c e o número de mochilas utilizadas seja mínimo [1].

Podemos construir uma formulação matemática para o problema utilizando dois conjuntos de variáveis binárias. O primeiro conjunto é o da alocação dos itens nas mochilas, onde cada variável x_{ij} representa alocação do item j na mochila i. Se $x_{ij} = 1$, então j foi alocado em i, caso contrário $x_{ij} = 0$. O segundo conjunto é o do uso das mochilas, onde cada variável y_i indica se a mochila i foi usada, isto é, pelo menos um item foi alocado nela. Se $y_i = 1$ então a mochila i foi utilizada, caso contrário $y_i = 0$ [1].

Minimizar:

$$z = \sum_{i=1}^{n} y_i \tag{2.1}$$

sujeito a:

$$\sum_{j=1}^{n} w_j x_{ij} \leq cy_i \qquad \forall\, i \in N = \{1, \cdots, n\} \tag{2.2}$$

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad \forall\, j \in N \tag{2.3}$$

$$x_{i,j} \in \{0, 1\} \qquad \forall\, i, j \in N \tag{2.4}$$

$$y_j \in \{0, 1\} \qquad \forall\, j \in N \tag{2.5}$$

A função objetivo (2.1) minimiza a quantidade de mochilas utilizadas. A restrição (2.2) garante que a soma dos pesos dos itens alocados em uma mochila não exceda sua capacidade. Além disso, esta restrição determina que uma mochila seja utilizada caso pelo menos um item seja alocado nela. A restrição (2.3) estabelece que cada item deve ser alocado em uma, e somente uma, mochila. Por fim, as restrições (2.4) e (2.5) determinam a característica binária das variáveis [1].

3. Metodologia

Esta seção trata das estratégias e algoritmos utilizados neste trabalho para atacar o BPP.

3.1. Modelo PLI

O modelo de PLI utilizado para solucionar o BPP de forma exata consiste exatamente no modelo de otimização combinatória disposto na seção 2. Observe que naquele modelo há uma grande multiplicidade de soluções, uma vez que as capacidades das mochilas são idênticas. Dessa forma, para melhorar o modelo de PLI, eliminando uma grande quantidade de soluções múltiplas, foi adicionada a seguinte restrição: $y_i \leq y_i + 1\ \forall i \in \{1, \cdots, n-1\}$.

3.2. Heurística GRASP

A meta-heurística GRASP é caracterizada pela obtenção de soluções através de duas etapas distintas. Na fase de construção, uma solução factível do problema é construída por uma combinação de critérios guloso e randômico. Na segunda fase, uma busca local é realizada a partir de operações sobre a solução obtida na fase anterior. As realização duas etapas compõem uma única iteração [2].

Cada solução válida foi considerada como um vetor em que cada elemento é um item que deve ser empacotado, além disso o vetor deve conter todos os n itens. Para calcular a quantidade de mochilas necessárias e em qual mochila vai cada item basta executar a Heurística construtiva NFD sobre o vetor da solução. Essa abordagem permitiu que os processos de construção e de busca local fossem simplificados.

Em cada iteração, a fase de construção faz uso da Lista Restrita de Candidatos (RCL, do inglês *Restricted Candidate List*) para selecionar o próximo elemento a ser adicionado na solução em construção. Por sua vez, a Lista de Candidatos (CL, do inglês *List of Candidates*) contém todos os elementos que podem ser inseridos na solução em construção, de modo que nenhum elemento pode fazer parte da RCL se o mesmo não fizer parte antes da CL [2].

A forma padrão de construção da RCL é através da utilização de um parâmetro $\alpha \in [0, 1]$, de modo que um elemento $e \in$ CL será inserido na RCL se e somente se $c(e) \in [c_{\min}, c_{\min} + \alpha(c_{\max} - c_{\min})]$, onde $c(e)$ é o custo de adicionar o elemento e na solução, $c_{\min}$ é o menor custo dentre todos os elementos da CL e $c_{\max}$ o maior.

Uma forma de construção alternativa é chamada de *Sampled Greedy*. Nesta, $\min\{p, |CL|\}$ elementos da CL de forma aleatória para compor a RCL, por sua vez, o elemento da RCL que entrará na solução parcial é escolhido de forma gulosa.

Neste trabalho, a construção de uma solução foi implementada considerando os itens como elementos a serem adicionados na solução parcial. O custo da inserção de um item na solução parcial é calculado pelo capacidade residual que permanece na mochila mais recentemente aberta caso o item seja alocado nela. Ou seja, Quanto menor a capacidade residual, melhor é o custo. Essa é a característica gulosa da etapa de construção.

Quanto à busca local, foi definida apenas uma forma de vizinhança: a troca de posição entre dois itens do vetor de solução. Os resultados dessa troca de posição são a diminuição do número de mochilas, o aumento do número de mochilas ou a manutenção desse número. Nos dois primeiros casos é fácil dizer se o vizinho é melhor que a solução atual, basta verificar se o número de mochilas diminuiu. Já no terceiro caso é necessário olhar para o menor espaço residual entre as duas novas mochilas dos itens trocados, se esse valor for menor que o da solução anterior então o vizinho é melhor. Como estratégia de busca foi utilizada o *Best Improvement*, ou seja, para cada solução foram explorados todos os vizinhos e escolhido o melhor.

3.3. Algoritmos aproximados

Muitos algoritmos aproximados foram propostos na literatura para o BPP. Neste trabalho foram realizadas implementações de três deles: o Next-fit decreasing (NFD), o First-fit decreasing (FFD) e o Best-fit decreasing (BFD). Nestes algoritmos cada um dos itens é indexado de acordo com uma ordem não crescente dos respectivos pesos [1].

No NFD, seguindo a ordem de indexação dos itens, verifica-se se o próximo item cabe na mochila mais recentemente aberta. Se couber, o item é alocado, caso contrário, uma nova mochila é aberta e o item é alocado na nova mochila. O procedimento é repetido para os demais itens. O algoritmo NFD provê uma solução

de no máximo duas vezes o valor de uma solução ótima [1].

No FFD, cada uma das mochilas também são indexadas. Diferentemente do NFD, a alocação de um item é realizada na mochila menos recentemente aberta em que o item couber. Novamente, caso não haja mochila com capacidade disponível, uma nova mochila é aberta. O algoritmo FFD provê uma solução de no máximo $\frac{17}{10}$ vezes o valor ótimo [1].

Por fim, no BF, a alocação de um item é realizada na mochila já aberta em que a capacidade residual pós alocação seja mínima. Caso não haja mochila com capacidade disponível uma nova mochila é aberta. O algoritmo FFD provê uma solução de no máximo $\frac{11}{9}$ vezes o valor ótimo mais um fator constante de valor 4 [1].

Neste trabalho, foi realizada uma implementação de complexidade linear no tamanho da entrada para NFD. Já para o FFD e BFD, implementações de complexidade quadrática em relação ao tamanho da entrada foram feitas, embora haja na literatura implementações de complexidade $O(n \log n)$, onde n é o número de itens e de mochilas [1].

4. Testes computacionais

Para realização dos testes computacionais, foram disponibilizadas pelo docente 10 instâncias do BPP. Seis diferentes algoritmos foram executados sobre cada uma das instâncias sob limite de tempo de 10 minutos. Em cada execução foi colhido o melhor limitante superior, isto é, a melhor solução factível obtida. Para a abordagem exata, também foi coletado o valor do melhor limitante inferior.

Os seis algoritmos utilizados foram:

- **PLI**: utilização do modelo descrito na seção 3.1 por meio do resolvedor comercial de programação linear inteira Gurobi;
- **NFD**: implementação linear do NFD;
- **FFD**: implementação quadrática do FFD;
- **BFD**: implementação quadrática BFD;
- **M1**: heurística baseada em GRASP com busca local padrão e $\alpha = 0,05$;
- **M2**: heurística baseada em GRASP com busca local *Sampled Greedy* com $p = 0,1 \cdot n$, onde n é o número de itens.

Com exceção do modelo PLI, que foi implementado na linguagem C++, todos os algoritmos foram implementados na linguagem Java. Para execução dos testes, foi utilizada uma máquina com 8 GB de RAM e processador Intel® Core™ i7-7500U CPU @ 2.70GHz, com o sistema operacional Linux Mint versão 19.1 x64. Os *outputs* das execuções e implementações estão disponíveis em `https://github.com/jonatastbelotti/MO824A_Atividade7.git`.

4.1. Resultados

A Tabela 1 apresenta os resultados obtidos por cada um dois seis algoritmos para cada instancia do BPP. Nela também é possível verificar qual o tamanho de cada instancia.

Tabela 1: Resultados dos testes computacionais.

		Modelo		NFD	FFD	BFD	M1	M2
Instância	Tam.	LI	LS_1	LS_2	LS_3	LS_4	LS_5	LS_6
instance0	201	66	66	85	66	66	69	83
instance1	201	65	66	84	66	66	70	81
instance2	402	132	133	166	133	133	144	176
instance3	402	132	133	168	133	133	145	180
instance4	600	198	199	250	199	200	217	275
instance5	600	198	199	255	199	199	217	280
instance6	801	265	266	332	266	266	291	374
instance7	801	265	266	335	266	267	292	376
instance8	1002	332	349	420	333	334	365	471
instance9	1002	332	341	418	333	334	368	464

Observando os dados da Tabela 1 nota-se que os limitantes inferior e superior para os dois modelos de PLI obtiveram valores semelhantes, salvo para as duas últimas instancias onde os valores tiveram uma maior diferença.

Se tratando das heurísticas construtivas os dados da Tabela 1 indicam que que o algoritmo NFD obteve as piores soluções para todas as instancias testadas, o que já era esperado uma vez que como mencionado na Seção 3.3 suas soluções são no máximo 2 vezes o valor da solução ótima. A surpresa entretanto ficou por conta do FFD, que mesmo tendo uma taxa de aproximação pior que o BFD obteve as mesmas soluções ou ainda soluções melhores em todos os casos de teste. A razão para esse fato pode ser as instancias selecionadas para os testes, a posição e tamanho de cada item dentro das instancias pode ter favorecido a execução do FFD, entretanto, vale lembrar que mesmo sem o favorecimento das instancias o BFD obteve bons resultados.

Por fim, os dados da Tabela 1 mostram que o GRASP padrão foi melhor que o GRASP com busca local *Sampled Greedy* em todas as instancias testadas.

As figuras 1 e 2 trazem os gráficos Performance Profile com a comparação da melhor solução obtida entre as heurísticas construtivas e entre as duas variações do GRASP respectivamente. Como se trata de um problema de minimização a taxa de performance $t_{s,i}$ para o algoritmo s na instancia i foi calculada como:

$$t_{s,i} = \frac{c_{s,i}}{\min_{\forall s' \in S} \{c_{s',i}\}}$$

, sendo $c_{s,i}$ a melhor solução obtida pelo algoritmo s para a instancia i e S o conjunto de algoritmos testados.

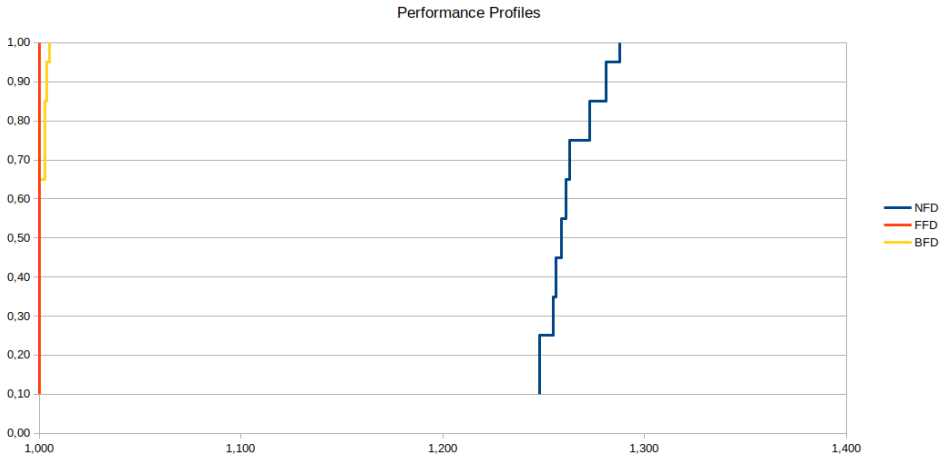

Figura 1: Gráfico Performance profile para as heurísticas construtivas: NFD, FFD e BFD.

O gráfico da Figura 1 confirma o que já foi discutido a respeito dos resultados da Tabela 1, que os dois melhores algoritmos aproximados foram o FFD e o BFD. Observando as curvas de performance verifica-se que o algoritmo FFD tem probabilidade de 60% de obter a melhor solução dentre os três algoritmos, enquanto que o BFD tem 100% de probabilidade. Em contrapartida o algoritmo NFD foi o mais distante da melhor resposta em todos os pontos, indicando que o mesmo foi o com pior desempenho em todos os casos de teste.

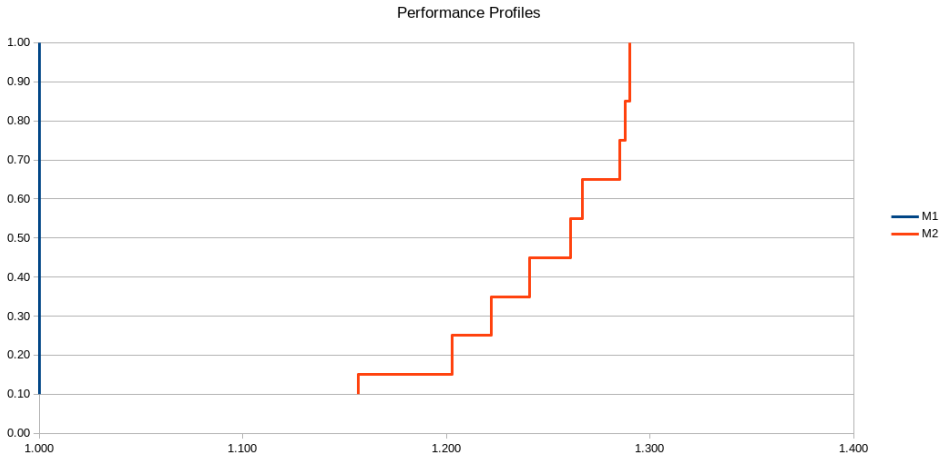

Figura 2: Gráfico Performance profile para as duas variações do GRASP.

Por sua vez, o gráfico performance profile da Figura 2 reforça a superioridade do GRASP padrão em relação ao *Simpled Greedy* para todas as instancias testadas, tendo 100% de probabilidade de alcançar a melhor solução, enquanto que o *Simpled Greedy* teve 0%.

Referências

[1] S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations*. New York, NY, USA: John Wiley & Sons, Inc., 1990.

[2] M. G. Resende and C. C. Ribeiro, *Greedy Randomized Adaptive Search Procedures: Advances, Hybridizations, and Applications*, pp. 283–319. Boston, MA: Springer US, 2010.