Universidade Federal de Mato Grosso do Sul
Faculdade de Computação
Tópicos em Computação I - Heurísticas baseadas em modelos matemáticos
Profa. Edna Ayako Hoshino
1º Trabalho Prático

1 Um algoritmo exato *branch-and-bound* e heurísticas para o problema da mochila múltipla

Neste trabalho será implementado um algoritmo exato *branch-and-bound* para a resolução do problema da mochila múltipla e heurísticas para obter "boas" soluções em pouco tempo de processamento.
Seja $I = \{1, 2, \dots, n\}$ um conjunto de n itens e $J = \{1, \dots, m\}$ um conjunto de m mochilas. Considere que cada item i em I tem um peso inteiro não-negativo p_i e um valor inteiro v_i. Além disso, cada mochila j em J tem uma capacidade C_j. O **problema da mochila múltipla** (MKP) consiste em determinar um subconjunto S dos itens em I para serem transportados nas mochilas, não violando suas capacidades, tal que a soma dos valores dos itens transportados seja máxima. Observe que um item selecionado em S só pode ser transportado em uma das mochilas.

1.1 Objetivo

O objetivo deste trabalho consiste em avaliar a qualidade das soluções geradas por heurísticas ingênuas para o MKP. Para atingir esse objetivo será exigida a implementação de um algoritmo exato do tipo *branch-and-bound* usando o resolvedor GLPK [1]. A ideia é usar o algoritmo exato para obter a solução ótima e, desta forma, medir a qualidade das soluções gerada pelas heurísticas. Como o MKP é NP-difícil, pode ser muito improvável obter a solução ótima para todas as instâncias testes em um limite de tempo razoável. Nesses casos, a comparação da qualidade pode ser feita usando-se o limitante dual (superior) que é obtido pela relaxação linear do problema.
O modelo de programação linear inteira (PLI) que deve ser considerado no desenvolvimento do algoritmo exato utiliza as variáveis de decisão binárias x_{ij} para cada item $i \in I$ e $j \in J$ de modo que $x_{ij} = 1$ se e, somente, se i for selecionado para estar na mochila j.
O modelo de PLI para o MKP é dado por:

$$\text{(F1)} \qquad z = \max \sum_{i \in I} \sum_{j \in J} v_i x_{ij} \tag{1}$$

$$\text{sujeita a} \qquad \sum_{i \in I} p_i x_{ij} \leq C_j \qquad , \forall j \in J \tag{2}$$

$$\sum_{j \in J} x_{ij} \leq 1 \qquad , \forall i \in I \tag{3}$$

$$x_{ij} \in \{0, 1\} \qquad , \forall i \in I, \forall j \in J. \tag{4}$$

Nesse modelo, as restrições (2) garantem que a soma dos pesos dos itens em cada mochila não ultrapasse a sua capacidade máxima. Já as restrições (3) evitam que um item seja colocado em mais de uma mochila. Por fim, as restrições (4) são as restrições de integralidade.

Relaxação linear. A **relaxação linear** de um modelo de PLI consiste na remoção das restrições de integralidade do modelo. Ou seja, as variáveis de decisão inteiras (ou binárias) passam a aceitar qualquer valor contínuo. No MKP, isso equivale a remover as restrições (4) e incluir as seguintes restrições:

$$0 \leq x_{ij} \leq 1 \qquad , \forall i \in I, \forall j \in J. \tag{5}$$

O valor de z obtido pela relaxação linear é um limitante superior para a solução ótima de (F1).
Heurísticas ingênuas, como heurísticas gulosas e aleatórias, para o MKP devem ser propostas e implementadas. Testes computacionais devem ser realizados e os resultados analisados.

1.2 Breve revisão de literatura

Diferentes modelos de PLI e desigualdades válidas definindo *facets* foram propostas na literatura para o problema da mochila simples. Para mais informações sobre essas desigualdades veja, por exemplo, [8]. De forma generalizada, [5] provaram que todas as desigualdades não-triviais que definem *facet* para o politopo da mochila simples também definem *facet* para o MKP. [3] estudaram o politopo do MKP e apresentaram novas desigualdades válidas que definem *facet* para o MKP. Por exemplo, os autores propõem desigualdades de cobertura múltipla que considera a cobertura em várias mochilas simultaneamente. [2] apresentam um algoritmo de planos de corte para o MKP, usando as desigualdades propostas em [3] e avalia os resultados computacionais em várias instâncias. [4] propuseram um algoritmo *branch-and-bound* para o MKP. Um dos melhores algoritmos para o MKP foi proposto por [9], cujo código e instâncias podem ser obtidas em `http://www.diku.dk/~pisinger/codes.html`. Mais recentemente, [6] modelaram um problema de escalonamento de máquinas como um MKP. Heurísticas têm sido propostas para o MKP, veja, por exemplo, [7]. Existem várias variantes do MKP estudadas recentemente na literatura. Enfim, trata-se de um tema bem estudado e ainda com bastante potencial de pesquisa.

1.3 Implementação

As seguintes tarefas de programação devem ser realizadas, usando as rotinas da **biblioteca GLPK**:

I1: implemente um algoritmo de *branch-and-bound* para o MKP usando a formulação (F1);

I2: Proponha e implemente pelo menos duas heurísticas ingênuas para gerar boas soluções para o MKP. **Dica: discutimos, na aula prática, duas heurísticas simples, uma gulosa e outra aleatória, para gerar soluções para o MKP. A ideia é propor alguma melhoria em alguma dessas heurísticas, ou propor uma outra heurística simples (sem o uso de metaheurísticas).**.

1.4 Testes

Algumas instâncias de teste estão disponíveis na página da disciplina. Oportunamente, instâncias adicionais serão disponibilizadas. As instâncias de testes foram geradas pelo gerador disponibilizado em `http://www.diku.dk/~pisinger/codes.html` e usadas por [9]. Para obter instâncias difíceis foram usadas as configurações com base no estudo de [10].

Formato dos arquivos de entrada O formato dos arquivos de entrada para as instâncias testes consiste em:

```
n m           /* n = |I| e m = |J| */
L1
L2
...
Lm
1 p1 v1           /* uma linha para cada item i. Aqui, i deve ser 1, 2, ..., n*/
2 p2 v2
...
n pn vn
```

Formato dos arquivos de saída

Arquivo de solução. O formato dos arquivos de saída contendo a solução obtida pelo algoritmo para uma instância teste é:

```
z k                /* z = valor da solucao, i.e. a soma dos valores dos itens transportados
                          nas mochilas
                      k = total de mochilas usadas */
mochila j1 r      /* j1 = indice da mochila ( 1 <= j1 <= m )
                     r  = total de itens transportados na mochila */
i1 i2 ... ir      /* os itens transportados na mochila (no intervalo de 1 a n) */
mochila j2 s
i1 i2 ... is
...
mochila jk t
i1 i2 ... it
```

ATENÇÃO: Se o nome do arquivo, contendo a instância teste de entrada, for `teste.mochila`, o arquivo de saída a ser criado, contendo a solução para essa instância teste, deve ser `teste.mochila.sol`.

Arquivo resumido de desempenho. Além do arquivo de saída contendo a melhor solução obtida pelo algoritmo, o programa implementado deve gerar um arquivo resumido de desempenho contendo uma única linha de informações no formato csv, descrita a seguir:

```
instance;gerador;tempo;LB;UB;status
```

- instance = nome do arquivo contendo a instância teste;
- gerador = 1:relaxação linear; 2:branch-and-bound; 3=heuristica (e outros)
- tempo = tempo total, em segundos, gastos pelo programa;
- LB = valor da melhor solução encontrada (limitante inferior);
- UB = limitante superior para a instância. No caso da implementação I2, deixe UB vazio;
- status = status final do resolvedor (5=tempo limite atingido, 10=sucesso, entre outros). Caso seja gerado pela heurística indique status=10.

ATENÇÃO: Se o nome do arquivo, contendo a instância teste de entrada, for `teste.mochila`, o arquivo de resumo a ser criado para uma determinada implementação Ik (k=1 ou k=2) deve ser `teste.mochila-k-h.out`. Se mais de uma heurística for proposta, use h=1, h=2, etc. No caso da implementação I1 use h=0.

Testes. Faça os experimentos listados a seguir usando todas as instâncias testes disponibilizadas no EAD, reporte e analise os seus resultados, sempre que possível, fazendo uso de gráficos e tabelas nas suas explicações.

T1: *(30% da nota)* Verifique a qualidade das soluções geradas pela implementação I1, limitado a um tempo limite de 5 minutos. Para medir a qualidade da solução, calcule o *gap* de dualidade, que é dado por $100\%(UB - LB)/UB$ e dá uma garantia para a qualidade da solução. Quando o *gap* é zero, temos que a solução encontrada é ótima. Reporte o *gap* médio, o total de instâncias de testes resolvidas na otimalidade e o tempo médio gasto nesses casos;

T2: *(10% da nota)* Verifique a qualidade dos limitantes superiores dados pela relaxação linear, usando a implementação I1. Para medir a qualidade dos limitantes UB obtidos pela relaxação linear, calcule o *gap* de otimalidade, que é dado por $100\%(UB - z*)/z*$ e dá uma medida para a qualidade do limitante considerando-se o valor da melhor solução conhecida como o valor de $z*$. O valor de $z*$ a ser usado nesta análise deve ser o valor da melhor solução gerada pela implementação I1 no experimento T1. Quanto mais próximo de zero o *gap* está, melhor é a qualidade do limitante gerado pela relaxação linear. Reporte o *gap* médio, o total de instâncias de testes em que o *gap* foi zero e o tempo médio gasto pela relaxação linear em todas as instâncias testes;

T3: *(60% da nota)* Verifique a qualidade das soluções geradas pelas heurísticas propostas e implementadas em I2. Calcule a qualidade das soluções, usando a fórmula do *gap* de dualidade e considerando o valor da relaxação linear como UB da instância. Reporte o *gap* médio de cada heurística, o total de instâncias testes em que cada heurística obteve melhor resultado dentre as heurísticas propostas e o tempo médio gasto pela heurística.

1.5 Observações importantes

Para entregar o seu trabalho corretamente, observe os itens listados abaixo:

- o trabalho deve ser feito em grupos e cada grupo deve ter de **três a quatro** integrantes.

- a programação deve ser feita em linguagem `C` compilável em uma instalação Linux padrão (com `gcc`) usando a opção `-STD=C99`. Adcionalmente será permito o uso da linguagem `C++` compilável em Linux com `g++` com opção `-STD=C++17`. Opcionalmente será permita a utilização da linguagem Python usando Python v.3.9.2.

- o relatório não pode ultrapassar 5 páginas (limite rígido).

- o trabalho deve ser entregue em um arquivo formato `tgz` enviado via *EAD* até as 23:55 horas da data fixada para a entrega na página da disciplina. Ao ser descompactado, este arquivo deve criar um diretório chamado `grupo` contendo os subdiretórios `src`, `testes` e `texto`, onde grupo deve ser o nome dado ao grupo no EAD. Inclua os nomes dos componentes do grupo no relatório.

  O diretório `grupo` deve conter o `makefile` para compilar todo o código ou outras instruções caso use outra linguagem ou o pacote `Pyomo`.

  No subdiretório `src` deverão estar todos os programas fonte. Se os programas não compilarem a nota do trabalho será *ZERO*.

  No subdiretório `testes` deverão estar todos as instâncias testadas e as respectivas saídas geradas pelo seu programa.

  No subdiretório `texto` deverá estar o arquivo com o seu relatório em formato `pdf`. Se o arquivo não estiver nesse formato, a nota do trabalho será *ZERO*.

- Para cada instância, o algoritmo exato não deve ultrapassar o limite de tempo **máximo** de 5 minutos.

  Nota: *estes parâmetros poderão vir a ser modificados. Caso isto ocorra, você será notificado via EAD!.*

- O relatório deve incluir uma descrição das heurísticas propostas e reportar os resultados dos testes T1, T2 e T3, sendo fundamental que seja acompanhado de uma análise dos resultados. Testes adicionais ou gráficos ilustrativos podem ser incluídos. Além dos resultados dos testes descritos na Seção 1.4, reporte conclusões gerais dos experimentos e do trabalho desenvolvido.

Referências

[1] *GNU Linear Programming Kit. Reference manual for GLPK Version 4.45*, dezembro 2010.

[2] C. E. Ferreira, A. Martin, and R. Weismantel. Solving multiple knapsack problems by cutting planes. *SIAM Journal on Optimization*, 6(3):858–877, 1996.

[3] Carlos E Ferreira, Alexander Martin, and Robert Weismantel. *Facets for the multiple knapsack problem.* ZIB, 1993.

[4] Alex S. Fukunaga. A branch-and-bound algorithm for hard multiple knapsack problems. *Annals of Operations Research*, 184(1):97–119, 2011.

[5] Elsie Sterbin Gottlieb and M. R. Rao. The generalized assignment problem: Valid inequalities and facets. *Mathematical Programming*, 46(1):31–52, 1990.

[6] Y. Laalaoui and R. M'Hallah. A binary multiple knapsack model for single machine scheduling with machine unavailability. *Computers & Operations Research*, 72:71 – 82, 2016.

[7] Yacine Laalaoui. *Improved Swap Heuristic for the Multiple Knapsack Problem*, pages 547–555. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[8] George L Nemhauser and Laurence A Wolsey. Integer programming and combinatorial optimization. *Wiley, Chichester. GL Nemhauser, MWP Savelsbergh, GS Sigismondi (1992). Constraint Classification for Mixed Integer Programming Formulations. COAL Bulletin*, 20:8–12, 1988.

[9] David Pisinger. An exact algorithm for large multiple knapsack problems. *European Journal of Operational Research*, 114(3):528 – 541, 1999.

[10] David Pisinger. Where are the hard knapsack problems? *Computers & Operations Research*, 32(9):2271 – 2284, 2005.