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 com conjuntos de penalidades

Neste trabalho será implementado um algoritmo exato *branch-and-bound* para a resolução do problema da mochila com conjuntos de penalidades e heurísticas para obter "boas" soluções em pouco tempo de processamento.
Seja $I = \{1, 2, \ldots, n\}$ um conjunto de n itens e $\mathcal{S} = \{S_1, \ldots, S_m\}$ uma coleção de subconjuntos de itens, ou seja, $S_j \subseteq I$, para cada $j \in J = \{1, \ldots, m\}$. Considere que cada item $i \in I$ tem um peso inteiro não-negativo p_i e um valor inteiro v_i. Adicionalmente, cada conjunto $S_j \in \mathcal{S}$ está associado a dois inteiros, d_j e h_j, que representam uma penalidade e um limite de isenção. Dado um subconjunto de itens U, uma penalidade d_j deve ser paga a cada unidade de $|U \cap S_j|$ que exceder h_j, para cada $S_j \in \mathcal{S}$.
O **problema da mochila com conjuntos de penalidades**, em inglês *knapsack problem with forfeit sets* (KPFS) consiste em, dados I, $\mathcal{S}$ e inteiros k e C, determinar um subconjunto U dos itens em I para serem transportados em uma mochila de capacidade C tal que a soma dos pesos dos itens em U não ultrapasse a capacidade C, o total de itens que violam os limites de isenção não exceda k e a soma dos valores dos itens transportados, descontando-se as penalidades, seja máxima. Ou seja, o KPFS procura $U \subseteq I$ tal que:

(i) $\sum_{i \in U} p_i \leq C$;

(ii) $\sum_{j \in J} \max\{0; |U \cap S_j| - h_j\} \leq k$;

(iii) $\sum_{i \in U} v_i - \sum_{j \in J} \max\{0; (|U \cap S_j| - h_j)\} d_j$ seja máxima.

1.1 Objetivo

O objetivo deste trabalho consiste em avaliar a qualidade das soluções geradas por heurísticas ingênuas para o KPFS. Para atingir esse objetivo será exigida a implementação de um algoritmo exato do tipo *branch-and-bound* usando o resolvedor SCIP [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 KPFS 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_i para cada item $i \in I$ e variáveis inteiras y_j, para cada conjunto $j \in J$, de modo que $x_i = 1$ se e, somente, se o item i é selecionado para ser transportado na mochila e y_j é o total de itens em S_j que são transportados na mochila e que excedem h_j.

O modelo de PLI para o KPFS é dado por:

$$\text{(F1)} \quad z = \max \sum_{i \in I} v_i x_i - \sum_{j \in J} d_j y_j \tag{1}$$

$$\text{sujeito a} \quad \sum_{i \in I} p_i x_i \leq C \tag{2}$$

$$\sum_{j \in J} y_j \leq k \tag{3}$$

$$\sum_{i \in S_j} x_i - y_j \leq h_j \qquad , \forall j \in J \tag{4}$$

$$x_i \in \{0, 1\} \qquad , \forall i \in I \tag{5}$$

$$y_j \in \mathbb{Z} \qquad , \forall j \in J. \tag{6}$$

Nesse modelo, a restrição (2) garante que a soma dos pesos dos elementos associados aos itens transportados na mochila não ultrapasse a sua capacidade e (3) que o total de violações não exceda k. Já as restrições (4) garantem que $y_j = |U \cap S_j| - h_j$. Por fim, as restrições (5) e (6) 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 KPFS, isso equivale a remover as restrições (5) e (6) e incluir as seguintes restrições:

$$0 \leq x_i \leq 1 \qquad , \forall i \in I, \tag{7}$$

$$0 \leq y_j \leq 1 \qquad , \forall j \in U. \tag{8}$$

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 KPFS devem ser propostas e implementadas. Testes computacionais devem ser realizados e os resultados analisados.

1.2 Breve revisão de literatura

O problema da mochila (KP) é um problema bastante estudado na literatura. Uma das variantes do KP bem conhecida é o problema da mochila múltipla (MKP). [6] propuseram um algoritmo *branch-and-bound* para o MKP. Um dos melhores algoritmos para o MKP foi proposto por [12], Mais recentemente, [9] modelaram um problema de escalonamento de máquinas como um MKP. Heurísticas têm sido propostas para o MKP, veja, por exemplo, [10]. Existem várias variantes do MKP estudadas recentemente na literatura. Enfim, trata-se de um tema bem estudado e ainda com bastante potencial de pesquisa.

Já o KPFS é um problema estudado mais recentemente. Ele foi introduzido em 2022 por D'Ambrosio et.al. [3] e ele generaliza o problema da mochila com penalidades, em inglês *knapsack problem with forfeits* (KPF), no qual, os subconjuntos $S_j \in \mathcal{S}$ têm cardinalidade 2 e $h_j = 1$, para todo $j \in J$. Ou seja, o KPF é um caso especial do KPFS em que a penalidade d_j, para cada conjunto $j \in J$, é paga se ambos os itens em S_j são colocados na mochila. O KPF foi introduzido em [3], no qual um modelo de PLI e duas heurísticas são propostos para o problema. Posteriormente, [2] propuseram uma heurística híbrida para o KPF. Já para o KPFS, [3] apresentaram um modelo PLI, três heurísticas e um conjunto de instâncias de teste. Mais recentemente, em 2024, [8] propuseram uma matheurística para o KPFS. As instâncias de testes usadas pelos autores estão disponíveis em `http://www.dipmat2.unisa.it/people/cdambrosio/www/DataSet/KPFS_instances.zip`.

1.3 Implementação

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

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

Atenção: Código parcial já implementado e disponível no AVA. O código está incompleto, pois não considera a restrição (3)**.**.

I2: Proponha e implemente pelo menos duas heurísticas ingênuas, incorporadas como *callbacks* do SCIP, para gerar boas soluções para o KPFS.

Dica: Código parcial de uma heurística aleatória para o KPFS está disponível no AVA. O código está incompleto, pois não considera a restrição (3)**. A ideia é completar essa heurística e propor outra heurística melhorada, por exemplo, uma gulosa, ou ainda propor uma outra heurística simples (sem o uso de matheurísticas)**.

1.4 Testes

Algumas instâncias de teste estão disponíveis na página da disciplina. Oportunamente, instâncias adicionais serão disponibilizadas.

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

```
n m C
v1 v2 ... vn
p1 p2 ... pn
hj dj nSj
i1 i2 ... inSj
              /* duas linhas para cada conjunto Sj */
k vk
```

no qual,

- n=$|I|$, m=$|\mathcal{S}|$, e C é a capacidade da mochila.
- v1 v2 ... vn são os valores dos itens 1 a n.
- p1 p2 ... pn são os pesos dos itens 1 a n.
- hj dj nSj é o limite de isenção, penalidade e cardinalidade de cada conjunto $S_j \in \mathcal{S}$.
- vk é o valor da constante k, que determina o total de itens que podem violar os limites de isenção.

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 v d p r t        /* z = valor da solucao
                    v = a soma dos valores dos itens transportados
                    d = a soma das penalidades pagas
                    p = a soma dos pesos dos itens transportados
```

```
                    r = total de itens transportados */
                    t = total de itens violando os limites de isenção */
i1 i2 ... ir     /* os itens transportados na mochila (no intervalo de 1 a n) */
```

ATENÇÃO: Se o nome do arquivo, contendo a instância teste de entrada, é `teste.txt` e o arquivo de configuração usado pelo programa é `heur.config`, o arquivo de saída a ser criado, contendo a solução para essa instância teste, deve ser `teste.txt-mochila-heur.config.sol`.

Arquivo resumido de desempenho. Além do arquivo de saída contendo a melhor solução obtida pelo algoritmo, o programa deve gerar um arquivo resumido de desempenho contendo uma única linha de informações no formato csv, para cada instância, conforme segue:

```
instance;n;m;C;LPiter;tempo;UB;LB;gap;status;rootUB;nodes;left;stime;ttime;mem;cols;
status;bestsolnode;bestsoltime;bestsoldepth;bestsolheur;heurtime;heurcalls;heurfound;
heurbestfound;heurname;configfile
```

ATENÇÃO: Se o nome do arquivo, contendo a instância teste de entrada, é `teste.txt` e o arquivo de configuração usado pelo programa é `heur.config`, o arquivo de saída a ser criado, contendo a solução para essa instância teste, deve ser `teste.txt-mochila-heur.config.out`.

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 2 minutos, sem habilitar nenhuma heurística. Para medir a qualidade da solução, calcule o *gap* de dualidade, que é dado por $100\%(UB - LB)/UB$, no qual UB é o melhor limitante dual e LB é o melhor limitante primal encontrado pelo algoritmo exato, dentro do limite de tempo. Quando o *gap* é zero, temos que a solução encontrada é ótima. Reporte o *gap* médio das instâncias não resolvidas na otimalidade, o total de instâncias de testes resolvidas na otimalidade e o tempo médio gasto nesses casos;

T2: *(60% da nota)* Verifique a qualidade das soluções geradas pelas heurísticas propostas e implementadas em I2, executando o algoritmo exato apenas no nó raiz. Calcule a qualidade das soluções, usando a fórmula do *gap* de dualidade e comparando o valor da solução gerada entre as diferentes heurísticas. 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;

T3: *(10% da nota)* Verifique o ganho de desempenho do algoritmo exato ao habilitar a melhor heurística proposta em I2 e avaliadas em T2. Ou seja, repita o experimento T1, desta vez, habilitando a melhor heurística. Para medir o ganho no desempenho, compare o *gap* de dualidade obtido quando a heurística é habilitada em relação aos resultados obtidos em T1. Reporte o *gap* médio das instâncias não resolvidas na otimalidade, o total de instâncias de testes em que o *gap* foi zero e o tempo médio gasto nesses casos.

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 **dois a três** integrantes.
- a programação deve ser feita em linguagem `C` compilável em uma instalação Linux padrão (com `gcc`) usando a opção `-std=c11`.

  Dica: Use o Makefile disponibilizado no AVA junto com o código parcial do trabalho..
- 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`, `output` 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.

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

  No subdiretório `output` deverão estar todos os arquivos de configurações usados e as respectivas saídas geradas pelo seu programa para cada instância de teste.

  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 2 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] Tobias Achterberg. Scip: Solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, 2009. `http://mpc.zib.de/index.php/MPC/article/view/4`.

[2] Giovanni Capobianco, Ciriaco D'Ambrosio, Luigi Pavone, Andrea Raiconi, Gaetano Vitale, and Fabio Sebastiano. A hybrid metaheuristic for the knapsack problem with forfeits. *Soft Computing*, 26(2):749–762, Jan 2022.

[3] Raffaele Cerulli, Ciriaco D'Ambrosio, Andrea Raiconi, and Gaetano Vitale. The knapsack problem with forfeits. In Mourad Baïou, Bernard Gendron, Oktay Günlük, and A. Ridha Mahjoub, editors, *Combinatorial Optimization*, pages 263–272, Cham, 2020. Springer International Publishing.

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

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

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

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

[8] Raka Jovanovic and Stefan Voß. Matheuristic fixed set search applied to the multidimensional knapsack problem and the knapsack problem with forfeit sets. *OR Spectrum*, Feb 2024.

[9] 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.

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

[11] 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.

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