Universidade Federal de Mato Grosso do Sul
Faculdade de Computação
Tópicos em Computação I - Heurísticas baseadas em modelos matemáticos
Profa. Edna Ayako Hoshino
2º Trabalho Prático

1 Matheurísticas para o problema da mochila múltipla

Este trabalho consiste no uso de uma classe de metaheurísticas baseadas em modelos matemáticos para o problema da mochila múltipla. Segundo a classificação definida em [2], as metaheurísticas de interesse são as da classe de heurísticas de melhoria e da classe de heurísticas baseadas na relaxação.
Seja $I = \{1, 2, \ldots, n\}$ um conjunto de n itens e $J = \{1, \ldots, m\}$ 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 de melhoria baseadas em modelos matemáticos ou baseadas na relaxação para o MKP. Para atingir esse objetivo, os resultados obtidos serão comparados com os resultados obtidos por uma 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:

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

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

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

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

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 de melhoria baseadas em modelos matemáticos, como o LNS e a partição por linhas/colunas, para o MKP devem ser propostas e implementadas. Testes computacionais devem ser realizados e os resultados analisados.

1.2 Breve revisão de literatura

A classificação de metaheurísticas baseadas em modelos matemáticos feita por Ball [2] em 2011 é uma das mais usadas na literatura. Nesse trabalho, além de reunir e classificar os métodos, Ball faz uma revisão de artigos relevantes da área. Dentre os diferentes métodos, os algoritmos classificados como metaheurísticas de melhoria consistem no uso de programas que resolvem modelos matemáticos para melhorar soluções geradas por heurísticas. Os algoritmos Large Neighborhood Search (LNS) expandem o espaço de busca graças ao uso de modelos matemáticos. Uma dessas ideias consiste em destruir parte de uma solução inicial e usar modelos matemáticos para reparar a solução destruída para gerar uma solução vizinha melhor. Já os métodos da partição por linhas e da partição por colunas são, geralmente, usados em modelos de partição ou de cobertura de conjuntos. Por exemplo, a partição de linhas consiste em dividir as linhas do modelo e dividindo apropriadamente as colunas (ou variáveis), de modo a obter dois modelos menores e independentes, cada um deles podendo ser resolvido na otimalidade. As heurísticas de melhoria baseadas em modelos matemáticos podem resolver um modelo matemático uma única vez para melhorar a solução ou múltiplas vezes para sucessivamente melhorar as soluções geradas. As metaheurísticas classificadas como heurísticas baseadas na relaxação reúnem aquelas que fazem uso do resultado de uma relaxação do problema para guiar a construção de uma "boa" solução. Dentre as relaxações, em geral, as mais usadas são a relaxação linear e a relaxação lagrangeana. Outras metaheurísticas também usam informações dos duais fornecidos durante a resolução do modelo matemático. Um exemplo clássico e simples é a heurística de arredondamento, que gera uma solução inteira a partir do arredondamento da solução ótima da relaxação linear. O sucesso desta técnica depende do problema e do modelo matemático.

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 uma heurística baseada em modelo matemático da classe das heurística de melhoria ou da classe de heurística baseadas na relaxação para gerar boas soluções para o MKP. **Dica: discutimos, durante as aulas, algumas heurísticas dessas classes para o MKP. A ideia é implementar alguma dessas heurísticas, ou propor uma outra heurística usando um dos métodos dessa classe visto em aula.**.

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 [3]. Para obter instâncias difíceis foram usadas as configurações com base no estudo de [4].

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-h.sol`, no qual `h` é um inteiro que represente o algoritmo que gerou a solução (adote h=0 para soluções geradas pelo algoritmo exato e h=1, 2, 3, assim sucessivamente para cada uma das heurísticas propostas).

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 cada implementação deve ser `teste.mochila-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: *(80% 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, que é computado pela fórmula $100(UB - LB)/UB$ e dá uma garantia para a qualidade da solução. Quando o *gap* é zero, temos que a solução encontrada é ótima. Nessa fórmula, LB é o valor da solução gerada pela heurística e UB é o valor da relaxação linear (ou o valor do melhor limitante dual obtido pelo algoritmo exato na implementação I1). 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.

T2: *(20% da nota)* Compare os resultados de desempenho e de qualidade das soluções geradas pela implementação I2 em relação à qualidade e desempenho das heurísticas ingênuas implementadas no primeiro trabalho prático. Caso o grupo não tenha implementado as heurísticas ingênuas, fica a critério do grupo implementá-las nesta etada ou usar o código das heurísticas ingênuas a ser disponibilizado pelo professor no Moodle.

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 e T2, 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] Michael O. Ball. Heuristics based on mathematical programming. *Surveys in Operations Research and Management Science*, 16(1):21–38, 2011.

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

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