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 Heurísticas baseadas em programação matemática para o problema da mochila da união de conjuntos

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 [3], as metaheurísticas de interesse são as da classe de heurísticas de melhoria, de decomposição e as baseadas na relaxação.
Seja $I = \{1, 2, \ldots, n\}$ um conjunto de n itens e $U = \{1, \ldots, m\}$ um conjunto de m elementos. Considere que cada elemento j em U tem um peso inteiro não-negativo p_j e que cada item i em I tem um valor inteiro v_i. Cada item i está associado a um subconjunto R_i de elementos de U, ou seja, $R_i \subseteq U$. O **problema da mochila da união de conjuntos** (SUKP) consiste em determinar um subconjunto S dos itens em I para serem transportados em uma mochila de capacidade C tal que a soma dos valores dos itens transportados seja máxima e a soma dos pesos dos elementos por S não ultrapasse a capacidade C da mochila. Dizemos que um elemento é **coberto** por S se $i \in \bigcup_{i \in S} R_i$. Observe que um elemento pode estar associado a mais de um item sendo transportado na mochila, mas o seu peso é contabilizado apenas uma vez.

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, ou as de decomposição para o SUKP. 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] ou o SCIP [2]. A ideia é usar o algoritmo exato para obter a solução ótima, desta forma, medir a qualidade das soluções gerada pelas matheurísticas. Como o SUKP 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) final 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 binárias y_j, para cada elemento $j \in U$ de modo que $x_i = 1$ se e, somente, se o item $i \in S$, ou seja, se o item i for selecionado para ser transportado na mochila e $y_j = 1$ se e, somente, se o elemento j é coberto por S.
O modelo de PLI para o SUKP é dado por:

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

$$\text{sujeita a} \quad \sum_{j \in U} p_j y_j \leq C \tag{2}$$

$$y_j \geq x_i \qquad , \forall i \in I, \forall j \in R_i \tag{3}$$

$$x_i \in \{0, 1\} \qquad , \forall i \in I \tag{4}$$

$$y_j \in \{0, 1\} \qquad , \forall j \in U. \tag{5}$$

Nesse modelo, as restrições (2) garantem que a soma dos pesos dos elementos associados aos itens transportados na mochila não ultrapasse a sua capacidade. Já as restrições (3) garantem que um elemento seja

coberto pela solução se ele pertence ao conjunto R_i de algum item i transportado. Por fim, as restrições (4) e (5) 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 SUKP, isso equivale a remover as restrições (4) e incluir as seguintes restrições:

$$0 \leq x_i \leq 1 \qquad , \forall i \in I, \tag{6}$$

$$0 \leq y_j \leq 1 \qquad , \forall j \in U. \tag{7}$$

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 podem ser aplicadas em soluções geradas por heurísticas ingênuas (propostas no trabalho T1) ou ainda nas soluções obtidas por matheurísticas de decomposição ou baseadas em relaxação (como Relax&Fix) para o SUKP, que 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 [3] 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 de decomposição consistem em dividir o problema em dois ou mais subproblemas e resolver cada um deles por heurísticas ou por métodos da programação matemática. A ideia é obter modelos menores e independentes para cada subproblema, 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. Um dos métodos baseados em relaxação linear é a Relax&Fix que particiona as variáveis em K partes e, iterativamente, resolve cada parte, impondo a integralidade apenas em uma parte da partição por vez.
Como vimos no trabalho T1, o SUKP é um problema introduzido em 1994 por Goldschmidt et. al. [4] e provado ser um problema NP-difícil. Dentre os trabalhos recentes que propõem heurísticas para o SUKP, podemos citar os trabalhos de Wei e Hao [5, 6]. As instâncias de testes reportadas nestes trabalhos estão disponíveis em `https://leria-info.univ-angers.fr/~jinkao.hao/SUKP_KBTS.html`.

1.3 Implementação

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

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

I2: Proponha e implemente uma heurística baseada em modelo matemático da classe das heurística de melhoria e/ou da classe de heurística baseadas na relaxação ou na decomposição para gerar boas soluções para o SUKP. **Dica: discutimos, durante as aulas, algumas heurísticas dessas classes para o SUKP. A ideia é implementar alguma dessas heurísticas, ou propor uma outra heurística usando um dos métodos visto em aula.**.

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:

```
m n C
v1 v2 ... vm
p1 p2 ... pn
R11 R12 ... R1n
R21 R22 ... R2n
...
Rm1 Rm2 ... Rmn
/* uma linha para cada item i e uma coluna para cada elemento.
   Rij=1 se o conjunto Ri contem o elemento j. */
```

Aqui, $m = |I|$, $n = |U$, e C é a capacidade da mochila. $v_1 v_2 \dots v_m$ são os lucros dos itens 1 a m. $p_1 p_2 \dots p_n$ são os pesos dos elementos 1 a n.

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 p              /* z = valor da solucao, i.e. a soma dos valores dos itens transportados
                        nas mochilas
                   p = a soma dos pesos dos elementos cobertos pelos itens transportados */
i1 i2 ... ir     /* os itens transportados na mochila (no intervalo de 1 a n) */
e1 e2 ... ek     /* os itens transportados na mochila (no intervalo de 1 a m) */
```

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 deve ser `teste.mochila-h.out`, em que h=1 (relaxacao linear), h=2 (exato), h=3 em diante para cada heurística implementada.

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 matheurísticas 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 já disponibilizado 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 **dois a três** 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`.
- o relatório não pode ultrapassar 5 páginas (limite rígido).
- o trabalho deve ser entregue em um arquivo formato `tgz` ou `zip` 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 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 `output` deverão estar todas as saídas geradas pelo seu programa para todas as instâncias 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 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] 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`.

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

[4] Olivier Goldschmidt, David Nehme, and Gang Yu. Note: On the set-union knapsack problem. *Naval Research Logistics (NRL)*, 41(6):833–842, 1994.

[5] Zequn Wei and Jin-Kao Hao. Kernel based tabu search for the set-union knapsack problem. *Expert Systems with Applications*, 165:113802, 2021.

[6] Zequn Wei and Jin-Kao Hao. Multistart solution-based tabu search for the set-union knapsack problem. *Applied Soft Computing*, 105:107260, 2021.