MO420/MC908: Programação Linear Inteira
Prof. Cid Carvalho de Souza - IC/UNICAMP
2º Trabalho Prático – 1º semestre de 2018

1 Um algoritmo *branch-and-cut* para o Problema da Árvore Geradora de Custo Mínimo com Conflitos

Assim como no trabalho anterior, será estudado o Problema da Árvore Geradora de Custo Mínimo com Conflitos (AGMCC), cujo enunciado é relembrado abaixo.
Seja um grafo $G = (V, E)$, não direcionado, com $|V| = n$ e $|E| = m$. A cada aresta e de E está associado um custo c_e. Além disso, é dado um conjunto C de pares de arestas, chamado de *conjunto de conflito*. Cada elemento de $\{e, f\}$ em C é chamado de um *par (de arestas) conflitante*. Uma árvore geradora T de G é dita ser *livre de conflitos* se nenhum par de arestas em T é conflitante. No AGMCC o que se deseja é encontrar uma árvore geradora de custo mínimo que seja livre de conflitos. Foi visto no primeiro trabalho prático que este problema é $\mathbb{NP}$-difícil e que vários artigos recentes têm se dedicado a tentar resolvê-lo, inclusive de forma exata. Em um destes artigos [3], Samer e Urrutia propõem um algoritmo *branch-and-cut* para o AGMCC. O objetivo deste trabalho prático é justamente a implementação de algoritmo deste tipo, reproduzindo **parcialmente** o que foi feito por aqueles autores.
Antes de prosseguir é conveniente introduzir o seguinte conceito. Dados G e o conjunto de conflitos $C \subset E \times E$, define-se o *grafo de conflitos* $\hat{G}$ cujos vértices correspondem ao conjunto de arestas do grafo G e cujas arestas são exatamente aquelas associadas a pares conflitos presentes em C (i.e., $(e, f) \in E(\hat{G})$ *se e somente se* $(e, f) \in C$). Seja $x \in \mathbb{B}^{|E|}$ um vetor que representa uma solução do AGMCC e P_{tree} e P_{stab} respectivamente, as envoltórias convexas das árvores geradoras mínimas de G e dos conjuntos independentes (ou estáveis) de $\hat{G}$. Tem-se então que $x \in P_{\text{tree}} \cap P_{\text{stab}}$.

2 Modelagem PLI do AGMCC

Pela última observação da seção anterior, pode-se formular o AGMCC usando os modelos **PLI** conhecidos para os problemas da árvore geradora e do conjunto independente mínimos em grafos. Usando o vetor x definido anteriormente, um modelo para árvores geradoras seria:

$$\sum_{e \in E(S)} x_e \leq |S| - 1, \qquad \forall\ S \subset V, S \neq \emptyset, \tag{1}$$

$$\sum_{e \in E} x_e = n - 1, \tag{2}$$

$$x_e \in \mathbb{B}, \qquad \forall\ e \in E. \tag{3}$$

Usando o mesmo vetor binário x, um modelo simples para conjuntos independentes seria:

$$x_e + x_f \leq 1, \qquad \forall\ (e, f) \in C. \tag{4}$$

Estes modelos são apresentados na seção 3.1 de [3]. A partir deles, uma formulação para o AGMCC seria

$$\min \left\{ \sum_{e \in E} c_e x_e \ :\ x \text{ satizfaz a (1), (2), (3) e (4)} \right\}. \tag{5}$$

Um resultado conhecido é que o sistema linear composto pelas inequações (1), (2) e as relaxações lineares de (3) ($0 \leq x_e \leq 1, \forall\ e \in E$) descreve completamente a envoltória convexa das árvores geradoras de G (ou seja, P_{tree}). No caso do problema do conjunto independente, a seção 3.1 de [3] exibe duas desigualdades válidas conhecidas na literatura. São elas: a *desigualdade do ciclo ímpar* (OCI) (equação (5) de [3]) e a *desigualdade clique* (CLIQUE) (equação (6) de [3]). Repare que o número de desigualdade OCI e CLIQUE são em número exponencial em $|E|$ e, da mesma forma, as desigualdades (1) são em número exponencial em $|V|$. Portanto, mesmo sem fazer uso das desigualdades OCI e CLIQUE, a única forma de usar a formulação para resolver o AGMCC é mediante um algoritmo *branch-and-cut*.
Em relação às desigualdades CLIQUE, sabe-se que elas só definem *facets* se o grafo suporte da mesma for uma clique maximal. Assim, quando uma aresta (e, f) de $\hat{G}$ está em uma clique maximal Q de tamanho maior que 2 neste grafo, a restrição correspondente a (e, f) em (4) é dominada pela desigualdade CLIQUE associada a Q. Em [2] propõe-se um algoritmo *branch-and-cut* para o problema do conjunto estável de peso mínimo. Naquele artigo, com intuito de obter uma formulação ao mesmo tempo forte e compacta para o P_{stab} os autores sugerem usar a seguinte estratégia. Primeiramente, por meio da heurística `CLQ1` (ver pg. 445, § 3), encontra-se uma *cobertura das arestas* de $\hat{G}$ *por cliques maximais* (ou seja, um conjunto $\{Q_1, Q_2, \ldots, Q_p\}$ de cliques maximais tal que, para toda aresta (e, f) de $\hat{G}$, existe $i \in \{1, \ldots, p\}$ satisfazendo $e \in Q_i$ e $f \in Q_i$). A partir daí, uma nova formulação para o AGMCC é obtida substituindo-se as restrições (4) pelas desigualdades CLIQUE associadas a $Q_1, Q_2, \ldots, Q_p$, ou seja:

$$\min\{\sum_{e \in E} c_e x_e \ :\ x \text{ satizfaz a (1), (2), (3) e } \sum_{g \in Q_i} x_g \leq 1, \forall\ i = 1, \ldots, p\}. \tag{6}$$

3 Implementação

Deve ser preparado um (único) programa em `C` ou `C++` que use as bibliotecas providas pelo CPLEX para implementar o algoritmo *branch-and-cut*. As seguintes rotinas deverão ser codificadas para poder rodar as variantes do algoritmo que serão testadas:

1. a rotina de separação das desigualdades (1) conforme descrita na pg. 47 de [3];
2. a rotina de separação das desigualdades OCI conforme descrita nas pgs. 47–48 de [3];
3. uma versão da rotina `CLQ1` pg. 445 de [2] para construir uma cobertura de cliques para as aresta de $\hat{G}$;

Além destas rotinas, na Seção 5 são discutidas outras implementações que poderão levar à concessão de bônus na nota do trabalho.

3.1 Testes a serem feitos

As instâncias de teste bem como seu formato de entrada são as mesmas do primeiro trabalho prático. Usando-as como *benchmark* rode os experimentos listados a seguir.

1. Faça uma análise comparativa do desempenho dos algoritmos *branch-and-cut* usando os modelos (5) e (6) sem que sejam usadas as desigualdades OCI;

2. Faça uma análise comparativa do desempenho dos algoritmos *branch-and-bound* usando os modelos 5 e 6 no qual sejam usadas também as desigualdades OCI;

Dê um tempo limite para execução de uma instância pelos algoritmos *branch-and-cut* (sugestão: 20 minutos). Suas análises devem ser feitas no relatório (parte escrita). Gráficos e tabelas devem ser empregados para auxiliá-lo neste processo. Procure tirar conclusões do seus experimentos e não apenas encher o relatório de gráficos e tabelas sem comentá-los. Inspire-se em alguns dos artigos que estão na lista de referência, notadamente [3].

4 Forma de entrega do trabalho

O texto do trabalho não deve ultrapassar 12 páginas. Nele você deve reportar os resultados obtidos pelos algoritmos, fazendo uma análise comparativa das diferentes versões algorítmicas testadas. Isso inclui as comparações discutidas na Seção 3.1.
Note que alguns dados são essenciais para fundamentar suas conclusões, incluindo: *(i)* o tempo de computação; *(ii)* os limitantes duais e primais tanto ao término do nó raiz da enumeração (o que equivale à execução da relaxação linear naquele instante) quanto ao término da execução (interrompida por prova de otimalidade ou por tempo limite de computação); *(iii)* o número de nós explorados na enumeração, e *(iv)* a quantidade de desigualdades de cada tipo que foram separadas pelo algoritmo. Assim, sempre que possível, gráficos e tabelas devem ser usados com essas (e outras) informações para descrever os seus resultados e justificar suas afirmações.
O trabalho deve ser entregue por `email` enviado ao docente no seguinte formato: arquivo `tgz` que ao ser aberto deverá criar (no diretório corrente) um subdiretório RAXXXXXX-RAYYYYYY, onde XXXXXX e YYYYYY são, respectivamente, o número do RA do primeiro e do segundo integrante do grupo. Este diretório deverá conter dois subdiretórios:

1. TEXTO, contendo um arquivo `raxxxxxx-rayyyyyy-texto.pdf` (nenhum outro formato será aceito) com o relatório reportando os resultados que você obteve nos seus testes e as suas análises sobre os mesmos, e

2. CODIGO, contendo os programas fonte que você implementou e um arquivo `makefile` que permita sua compilação através do comando `make` (a ausência deste arquivo será penalizada **com rigor**).

   O arquivo executável gerado pelo `makefile` deverá **obrigatoriamente** se chamar `bnc`. Para executar o programa, a seguinte linha de comando deverá ser dada:

   `bnc <tipo-exec> <model> <time-limit> <heur-primal> <arq-input>`,

onde

- `tipo-exec` é o tipo de execução dado por um caracter tal que: "*e*" equivale a um algoritmo *branch-and-bound* puro (só usado no caso de implementação do bônus, ver Seção 5) e "*c*" equivale a um algoritmo *branch-and-cut*;

- `model` é o tipo do modelo utilizado, sendo "*n*" o modelo (5) e "*c*" o modelo (6);

- `time-limit` é o tempo máximo de execução do algoritmo **em segundos** dado por um número inteiro positivo.
- `heur-primal` é um valor binário que vale "1" se a heurística primal for usada e "0" caso contrário (só usado no caso de implmentação do bônus, ver Seção 5).
- `arq-input` é o nome do arquivo de entrada.

Considerando o arquivo de entrada correspondente à instância `arq-input`, o seu programa deverá gravar na saída dois arquivos: `<arq-input>.sol` e `<arq-input>.est`. Esses arquivos devem ter o seguinte formato:

- `<arq-input>.sol`: uma linha contendo os parâmetros de entrada, na mesma ordem em que aparecem na linha de comando descrita acima, sendo todos os parâmetros separados por um (caracter) *branco*; $n-1$ linhas contendo todos os pares de vértices i e j, com $i < j$, tais que a aresta (i, j) de G está na árvore geradora de menor custo encontrada pelo algoritmo; uma linha contendo o valor desta solução.
  NOTA: mesmo no caso do ótimo não ter sido provado, os valores reportados devem ser referentes à melhor solução primal encontrada. Se nenhuma solução primal for encontrada, todos os pares de valores inteiros nas $n-1$ linhas correspondentes às arestas da solução devem ser iguais a "`-1 -1`" e a linha de custo da solução deverá conter o valor "`-1`".
- `<arq-input>.est`: este arquivo deve ter 9 linhas contendo as seguintes informações **nesta ordem**: uma linha contendo os parâmetros de entrada, na mesma ordem em que aparecem na linha de comando descrita acima, sendo todos os parâmetros separados por um (caracter) *branco*;
  1. total de desigualdades (1) que foram separadas durante a execução do algoritmo (tipo: inteiro),
  2. total de desigualdades OCI que foram separadas durante a execução do algoritmo (tipo: inteiro),
  3. valor da FO da primeira relaxação (tipo: double),
  4. valor da FO no nó raiz (tipo: double),
  5. total de nós explorados (tipo: inteiro),
  6. nó onde encontrou a melhor solução inteira (tipo: inteiro),
  7. valor da melhor solução inteira (tipo: inteiro),
  8. melhor limitante dual (double) e
  9. o tempo de execução em segundos (tipo: double)

Todos os valores `double` devem ser impressos com 6 casas decimais.
Importante: os arquivos de saída devem conter **apenas** os valores pedidos e nada mais ! Em particular, não coloque textos explicativos sobre os valores impressos (exemplo do que não fazer: imprimir algo como "`Valor da FO=4.000000`" ao invés de simplesmente imprimir "`4.000000`").

5 Bônus

Dois bônus são oferecidos que poderão acrescentar pontos ao trabalho (ver Seção 6):

1. projete e implemente uma heurística primal qualquer, a qual deverá ser executada após a resolução de cada relaxação linear ou após a conclusão do algoritmo de planos de corte em cada nó da árvore de enumeração, sempre durante de um algoritmo *branch-and-cut*.

2. uma formulação compacta para árvores geradoras é dada em [1] (pg 534, equações (3.38)–(3.45)). No modelo (6), substitua as desigualdades que descrevem a árvore geradora (equações (1) e (2)) pelo modelo compacto dado em [1]. Note que, agora, o AGMCC pode ser resolvido por um algoritmo *branch-and-bound* puro, já que não é mais necessário gerar cortes (o modelo está correto).

Os experimentos adicionais que você deverá fazer para concorrer ao bônus são:

1. inclua no melhor algoritmo *branch-and-cut* que você achou nos testes da Seção 3.1 a heurística primal que você projetou e reporte os resultados comparativos das versões com e sem a heurística, indicando se houve ou não melhora no desempenho do algoritmo;

2. compare o desempenho do algoritmo de *branch-and-bound* puro usando o modelo compacto com aquele do melhor algoritmo *branch-and-cut* que você achou nos testes da Seção 3.1, indicando quem foi melhor.

Duas páginas adicionais poderão ser acrescentadas ao seu relatório caso você faça um dos bônus.

6 Pontuação

A pontuação de cada parte do trabalho será distribuída da seguinte forma:

1. programa não roda: nota final *ZERO*.

2. algoritmo *branch-and-cut* usando os modelos (5) e sem as desigualdades OCI está rodando e os resultados por ele obtidos foram bem documentados no relatório: $4,0$ pontos.

3. algoritmo *branch-and-cut* usando os modelos (5) e com as desigualdades OCI está rodando e os resultados por ele obtidos foram bem documentados no relatório, inclusive comparando-os com modelo do(s) item(ns) anterior(es): $+2,0$ pontos.

4. algoritmo *branch-and-cut* usando os modelos (6) e sem as desigualdades OCI está rodando e os resultados por ele obtidos foram bem documentados no relatório, inclusive comparando-os com modelo do(s) item(ns) anterior(es): $+2,0$ pontos.

5. algoritmo *branch-and-cut* usando os modelos (6) e com as desigualdades OCI está rodando e os resultados por ele obtidos foram bem documentados no relatório, inclusive comparando-os com modelo do(s) item(ns) anterior(es): $+2,0$ pontos.

No caso dos bônus, a pontuação será a seguinte:

1. heurística primal foi implementada e resultados comparando uma versão do algoritmo *branch-and-cut* com e sem o uso da heurística foram bem reportados no relatório: $+1,0$ ponto.

2. algoritmo *branch-and-bound* do modelo compacto foi implementado e resultados comparativos com uma versão do algoritmo *branch-and-cut* foram bem reportados no relatório: $+1,0$ ponto.

IMPORTANTE: você pode concorrer aos bônus mesmo não tendo feito todas implementações requeridas pelos testes da Seção 3.1.

7 Considerações finais

Algumas considerações importantes para serem levadas em conta:

- Os trabalhos deverão ser feitos em grupos de dois alunos.
- Não serão aceitos trabalhos entregues fora do prazo.
- Os programas devem atender às especificações contidas neste documento. Qualquer desvio em relação às mesmas acarretará em descontos na nota final e poderão, inclusive, resultar em nota ZERO em casos mais graves (p.ex., programas que não compilam ou que geram saídas em formatos incompatíveis com aqueles especificados na seção 4).
- O texto do trabalho deve especificar claramente o ambiente computacional onde o código foi testado.
- Rotinas auxiliares poderão ser baixadas da rede e usadas em seu programa desde que o texto do trabalho mencione explicitamente o uso deste código e cite o endereço completo do *site* de onde ele foi baixado.
- A nota final do trabalho terá uma componente comparativa, ou seja, será considerado qual grupo fez mais coisas e melhor.

Referências

[1] T. L. Magnanti and L. A. Wolsey. Chapter 9 optimal trees. In C. M. M.O. Ball, T.L. Magnanti and G. Nemhauser, editors, *Network Models*, volume 7 of *Handbooks in Operations Research and Management Science*, pages 503 – 615. Elsevier, 1995.

[2] G. L. Nemhauser and G. Sigismondi. A strong cutting plane/branch-and-bound algorithm for node packing. *The Journal of the Operational Research Society*, 43(5):443–457, May 1992.

[3] P. Samer and S. Urrutia. A branch and cut algorithm for minimum spanning trees under conflict constraints. *Optimization Letters*, 9(1):41–55, 2015.