Projeto de MC548: Problema do Ciclo de Steiner

Prof. Flávio K. Miyazawa - 2s2011

1 Introdução

O projeto pode ser feito por um ou por dois participantes e consiste em uma implementação em C++ (compilável em linux/gcc por Makefile e linha de comando) de um programa para o problema do Ciclo de Steiner. Deverão ser entregues por email, um arquivo .tgz contendo todos os fontes do programa e os diretórios dos pacotes utilizados para a compilação do mesmo. Além disso, deve ser entregue também um relatório. Este projeto pode ser feito tanto pelos alunos da turma MC548A como MC548#.

2 Definição do problema

Seja um grafo não-orientado $G = (V, E)$ com custos $c_e \in \mathbb{Q}^+$ associados às arestas $e \in E$ e um subconjunto de vértices $T \subseteq V$, chamados *vértices terminais*. Os vértices $V \setminus T$ são conhecidos como *vértices de Steiner*. Um ciclo C de tamanho k é uma sequência de vértices $C = (v_1 \dots v_k)$ tal que $v_i \neq v_j$ para todo $1 \leq i < j \leq k$ e a aresta $(v_i, v_{(i+1 \mod k)+1}) \in E$ para todo $1 \leq i \leq k$. O custo de um ciclo $c(C)$ é dado pela soma dos custos das arestas que o formam, ou seja, $c(C) = \sum_{i=1}^{k} c(v_i, v_{(i+1 \mod k)+1})$.

O problema do Ciclo de Steiner consiste em encontrar um ciclo C de menor custo tal que todo vértice de T pertence a C. Note que o ciclo também pode conter alguns vértices de Steiner embora não sejam necessários. Este é um problema bastante recorrente em projeto de redes de transmissão e comunicação, bem como roteamento de veículos em armazéns de grande porte.

3 Datas e Prazo

O prazo de entrega é 21 de novembro as 9hs por e-mail, tanto para o professor da disciplina como para o PED. Não deixe o projeto para os últimos dias. O prazo de vários dias é justamente para que o grupo possa investir em uma boa implementação e obter bons resultados. O projeto possui peso diferenciado na composição da nota final e os melhores trabalhos serão valorizados.

4 Sobre estratégias de resolução e tamanho do grupo

Considere as seguintes denominações:

$\mathcal{H}$: Um conjunto de no mínimo duas heurísticas, que podem ser construtivas, gulosas, de melhoria, busca local, etc. Estas heurísticas devem ser na prática heurísticas razoavelmente rápidas. Note que a implementação de mais heurísticas com características diferentes pode levar a soluções melhores.

Ao fazer várias heurísticas, é recomendado que o grupo procure combiná-las na obtenção de uma heurística que produza as melhores soluções. Por exemplo, uma heurística poderia simplesmente chamar (todas) as heurísticas simples ou de busca local e devolver a melhor solução. Outra alternativa seria fazer um Multi-Start Local Search, que itera até um limite de tempo e em cada iteração gera soluções

iniciais das heurísticas construtivas e executa heurísticas de busca local. Heurísticas de melhoria/busca local poderão ser úteis na elaboração dos algoritmos exatos e meta heurísticas. Caberá ao grupo decidir como combinar as rotinas de maneira a dar os melhores resultados.

Excepcionalmente este item pode ser de apenas uma heurística, se a mesma envolver grande trabalho de implementação. Para isso, justifique o algoritmo/heurística e sua implementação e estruturas de dados para o professor.

$\mathcal{M}$: uma metaheurística (GRASP, Busca Tabu, Algoritmo Genético, Colônia de Formigas ou outra a escolha do grupo). Para isto, o grupo deve usar também as rotinas feitas no item $\mathcal{H}$ e caberá ao grupo calibrar o tempo/jeito para executar estas rotinas dentro da metaheurística.

$\mathcal{E}$: um algoritmo exato. Para isto, o grupo deve usar também as rotinas feitas no item $\mathcal{H}$ e caberá ao grupo calibrar o tempo/jeito para executar estas rotinas dentro do algoritmo exato.

Cada grupo (de 1 ou 2 alunos) deve escolher a implementação de: $\mathcal{E} + \mathcal{H}$ ou $\mathcal{M} + \mathcal{H}$. Note que, em geral, as heurísticas dos itens $\mathcal{H}$ oferecem suporte para implementações dos itens $\mathcal{M}$ ou $\mathcal{E}$.

Observações:

- Se você for fazer o projeto sozinho, poderá fazer apenas a implementação de $\mathcal{H}$, ou $\mathcal{E}$ ou $\mathcal{M}$. Naturalmente as rotinas de $\mathcal{E}$ e $\mathcal{M}$ necessitam de subrotinas e heurísticas, porém estas poderão ser mais simples no caso de fazer o projeto sozinho. No caso de implementar $\mathcal{H}$, estas deverão já estar combinadas em apenas uma heurística que chama todas as heurísticas implementadas, podendo ser também uma heurística estilo Multi-Start Local Search ou algo nessa linha.
- Na implementação do algoritmo exato ($\mathcal{E}$), é esperado que se use o resultado das heurísticas em $\mathcal{H}$ para obter soluções boas e podar parte do processo de enumeração. No relatório do projeto, descreva como foi feito o processo de enumeração, poda, etc.
- Na implementação da metaheurística ($\mathcal{M}$), é esperado que se use o resultado das heurísticas e busca locais $\mathcal{H}$. Caso o grupo tenha implementado metaheurística não apresentada em aula, o grupo deve também descrever no relatório o funcionamento da metaheurística no geral, além de descrever a implementação da mesma.
- Caso o grupo opte pela implementação de $\mathcal{E} + \mathcal{H}$ ou $\mathcal{M} + \mathcal{H}$, o programa a ser entregue deve ser único com apenas uma chamada e caberá ao grupo juntar tudo em um programa de maneira a dar os melhores resultados. O algoritmo exato ou heurísticas devem obedecer as normas descritas no esqueleto divulgado.

5 Sobre os programas

Todos os programas devem ser feitos na linguagem `C++`, usando compilador `Gnu GCC/G++` e ferramentas de compilação padrão (make, ar, ranlib e cia). O esqueleto básico para implementação pode ser encontrado no link

`http://www.ic.unicamp.br/~fkm/disciplinas/mc548/2011s2/projeto/exemplo.tgz`

Ele consiste de uma classe base chamada `SteinerCycleSolver` que contém uma interface comum para todos os projetos, além de algumas estruturas básicas. O aluno/grupo deverá criar um diretório com o RA do

aluno em questão (se é uma dupla, deve ser o RA de ambos concatenados, por exemplo, RA123456_RA654321). Dentro deste diretório, deverá residir todo seu código, bibliotecas e outras aparatos necessários para execução do algoritmo. Sua classe derivada deverá ter o mesmo nome/RA do diretório, com arquivos de cabeçalho (.hpp) e implementação (.cpp) de mesmo nome (note que RA será sempre em maiúsculo).

Seu subdiretório deverá ter um Makefile capaz de gerar uma biblioteca que será linkada ao programa principal na raiz do projeto (vejam o Makefile de exemplo no diretório RA038508).

Detalhes sobre a interface

Para a realização do projeto, será utilizado a biblioteca Lemon para representar grafos e já possui várias facilidades e subrotinas implementadas, descrita mais abaixo.

A classe `SteinerCycleSolver` contém a assinatura de 3 métodos virtuais que *necessariamente* deverão ser implementados nas classes derivadas, além de estruturas básicas de armazenamento.

Os atributos básicos são:

`static ListGraph graph:` guarda o grafo usando ferramentas da biblioteca Lemon;

`static ListGraph::EdgeMap<double> length:` mapeamento dos custos das arestas;

`static ListGraph::NodeMap<bool> terminal:` mapeamento indicando os vértices terminais;

`std::list<ListGraph::Node> best_solution:` lista de vértices de uma solução. Estes devem estar *na ordem* em que aparecem no ciclo;

`double lb:` Limitante inferior. Apenas para algoritmos exatos que não encontraram solução ótima, volta com o limitante inferior que delimita solução ótima. Caso não tenha sido encontrado nenhum limitante, volta com valor `numeric_limits<double>::min()`;

`double ub:` Limitante superior. Volta com o custo do circuito encontrado. Caso não tenha sido encontrado com nenhum circuito, volta com o valor `numeric_limits<double>::max()`.

Note que o grafo e seus elementos são dados estáticos, ou seja, compartilhados por todos os objetos que são derivados desta classe. Estes dados, em geral, são usados apenas para consultas. Informações mais sensíveis ao contexto, são reservadas a este e devem ser implementadas nas classes derivadas.

As classes derivadas devem implementar obrigatoriamente:

`ResultType solve(const double max_time):` método que implementará o algoritmo exato ou metaheurística principal de seu trabalho. Seu algoritmo deve respeitar o tempo máximo passado como parâmetro. A execução deve retornar seu status de acordo com a enumeração `SteinerCycleSolver::ResultType`;

`ResultType solveFast(const double max_time):` idem o item anterior, mas para heurísticas rápidas de tempo polinomial;

Algumas funções de suporte já estão providas como:

`loadInstance():` carrega uma instância. Embora proceda carregmento, este método está parcialmente implementado, fazendo checagens de erros básica. Entretanto, não é preciso reimplementá-lo;

`displayInstance():` mostra na saída padrão a instância no formato de Lemon;

`checkSolution()`: checa se uma dada solução é válida ou não;

`displaySolution()`: mostra na saída padrão uma solução.

Particularidades de seu algoritmo

De modo geral, cada estratégia de resolução tem um conjunto de parâmetros que controla o processo de otimização. Por exemplo, algorimos de *simulated anneling* mantêm uma temperatura inicial e uma constante k. Algoritmos genéticos tem como parâmetros, em geral, taxa de *crossover*, mutação, etc. Você *deve* setar estes parâmetros como constantes em seu arquivo de cabeçalho `RAXXXXXX.hpp`. Embora esta estratégia de hardcode não seja a mais indicada, ela facilita a modificação e correção de seu código.

Estruture seu código de maneira adequada e clara. Faça comentários em alto nível sobre estratégias usadas em cada parte de seu código, e para trechos mais "tricks", seja bastante claro e explicativo.

Para comentários gerais sobre métodos, atributos e funções, é uma boa ideia usar comentários do tipo doxygen. Esta ferramenta é capaz de criar boas documentações sobre seu código. Se você tiver o doxygen em seu sistema, execute `make doc` (ou use o doxywizard) no diretório raiz do projeto de exemplo. Ele criará um subdiretório `doc` que conterá toda documentação em HTML do exemplo. Inclusive, você poderá ver a documentação da classe principal.

Programa principal e outras classes de suporte

O programa principal deve instanciar um objeto de sua classe derivada, carregar e instância e resolvê-la. Você deve implementar uma função para carregamento da instância nas estruturas do Lemon, que posteriormente serão carregadas em seu solver.

Para medir o tempo de execução, provemos uma classe `Timer` (arquivo `timer.hpp` na raiz do projeto), que mede o tempo real de execução. Ela também *deve* ser usada para limitar o tempo de execução de seu algoritmo. Um exemplo pode ser encontrado em `main.cpp`. De fato, recomendamos fortemente que rotinas de execução e apresentação de resultados estejam implementadas no arquivo `main.cpp`. Caso deseje algo diferente, você deverá modificar o `Makefile` e deixar estas modificações explicitamente descritas. Também é necessário que tais modificações sejam brevemente descritas em seu relatório.

Se sua implementação usar escolhas probabilísticas, *recomendamos fortemente* a utilização do Mersenne Twister [**?**], tido como estado da arte na geração de números aleatórios. Uma implementação está provida no arquivo `mtrand.hpp` na raiz do projeto, e um exemplo simples de utilização está no arquivo `main.cpp`.

Na raiz, ainda temos o arquivo `Makefile.inc`, que contem as configurações gerais para compilação do projeto. Em especial, temos as configurações do software CPLEX. Caso não use este, comente toda seção. O arquivo `Makefile` é o principal e deverá ser usado para compilação total do projeto. Você *deve setar* a variável `DIRS` para o diretório que contêm seu código, e a variável `EXE` para seu RA, ou seja, o nome de seu programa executável será seu RA.

Como já foi dito, seu diretório *deverá* conter um `Makefile` de compilação local. Você deve setar corretamente a variável `OBJLIB` para o nome de sua futura lib. Siga rigorasamente o padrão, para evitar problemas de compilação na hora da correção. A variável `OBJS` contém uma listagem de todos seus códigos. A rigor, você pode mudar este Makefile de maneira que atenda melhor suas necessidades, mas sempre deve gerar a lib como indicado.

De modo geral, o comando `make` na raiz do projeto irá carregar as configurações de `Makefile.inc`, realizar as compilações locais, executar o comando `make` nos subdiretórios usando para isso o `Makefile` local. Veja

o exemplo provido. Ele foi compilado com sucesso no MacOS X 10.6.8 (Gnu GCC 4.2.1, GNU Make 3.81 e ferramentas associadas) e Ubuntu 10.04 (Gnu G++ 4.4.3, GNU Make 3.81 e ferramentas associadas). Os testes serão feitas no Ubuntu citado.

Parâmetros

Seu programa obedecerá a seguinte sintaxe de entrada

`RAXXXXXX.e <`*entrada*`> <`*tempomax*`>`

onde *entrada* deve ser o nome do arquivo de entrada e *tempomax* deve ser o tempo máximo de execução em segundos. Por exemplo, a execução seguinte:

```
RA038508.e entrada.txt 60 > saida.txt
```

executa o programa RA038508, que lê o arquivo de entrada `entrada.txt` e executa durante 60 segundos no máximo, direcionando toda informação apresentada por seu programa para o arquivo `saida.txt`.

Formato dos arquivos de entrada

As instâncias a serem testadas foram extraídas da SteinLib Testdata Library [**?**] e TSPLIB [**?**]. Como o formato provido por esta base de dados é um pouco "chato", foi aplicada uma simplificação. O arquivo de entrada é um arquivo texto, onde cada linha apresenta números (separados por pelo menos um espaço em branco) e tem a seguinte cara:

- Na primeira linha, são dados o número total de vértices n, número total de arestas m e o número de terminais t. Os vértices são numerados de $\{0, \ldots, n-1\}$ e as arestas $\{0, \ldots, m-1\}$.
- Em seguida, aparecem m linhas, uma para aresta. Cada linha contém dois números inteiros indicando os vértices ligados pela aresta e um número real indicando o peso da mesma (algumas instâncias tem pesos inteiros, que devem ser convertidos para números reais);
- Após as m linhas, temos t linhas contendo, cada uma, um número inteiro representando um vértice terminal.

Exemplo de arquivo:

```
6 8 3
0 1 1.0
0 3 1.0
0 5 1.0
1 2 1.0
2 3 1.0
3 4 1.0
4 5 1.0
5 6 1.0
6 1 1.0
0
2
4
6
```

No exemplo acima temos 7 vértices ligados entre si por 9 arestas, todas de peso 1. Temos como terminais os vértices 0, 2, 4 e 6.

O professor disponibilizará várias instâncias com centenas e até milhares de vértices e terminais. O objetivo destas instâncias será avaliar o desempenho de cada classe de programa e ver os limites de cada programa para os diferentes tamanhos de instâncias. Assim, não quer dizer que o algoritmo exato deva resolver bem as instâncias grandes, mas ele será comparado com os outros algoritmos exatos implementados pela turma até tamanhos razoáveis para se testar os algoritmos exatos. Naturalmente se espera que heurísticas rápidas possam dar soluções para instâncias grandes.

Formato da saída

Seu programa *sempre deve* escrever na saída padrão para que, como no exemplo, possamos redirecionar os resultados para um arquivo específico. Esta saída é livre no início, mas deve obedecer algumas regras nas linhas finais.

Por exemplo, seja o seguinte exemplo de um algoritmo iterativo de busca:

```
1  Improvement Iter. 1 / Current Time: 0.02 / Best Current: 1038.74 / Best Overall: 0.00
2  Improvement Iter. 2 / Current Time: 0.05 / Best Current: 1241.21 / Best Overall: 1038.74
3  Improvement Iter. 5 / Current Time: 0.20 / Best Current: 1513.32 / Best Overall: 1241.21
4  Improvement Iter. 10 / Current Time: 0.35 / Best Current: 1613.74 / Best Overall: 1513.32
5  ...
6  ...
7
8  Best cycle
9  3 8 5 0 2 4 1
10 endcycle
11
12 Solution & Time & Best Solution Value
13 Y & 1.34 & 3961.74
```

As primeiras linhas são específicas do algoritmo, podendo conter informações gerais, de depuração, entre outras. Este início é livre, entretanto recomendamos desligar as informações de depuração para evitar geração de saídas grandes.

Deve existir uma seção chamada `Best Cycle` em linha única, que iniciará uma sequência de vértices do melhor ciclo encontrado. Essa sequência aparecerá na linha seguinte (recomendamos usar apenas uma linha). O fim da seção é dado por `endcycle` na linha seguinte (também em linha única). Caso nenhum ciclo seja encontrado, a seção deve conter a sequência `NoCycle`.

Rigorosamente, a última linha da saída deve ser a seguinte sequência:

Solution (`Y`, `N` ou `O`): indica se uma solução foi encontrada e se a mesma é ótima. Utilize `Y` ou `N` para indicar que uma solução foi encontra ou não. Se estiver usando um algoritmo exato e este conseguir provar a otimalidade, este campo deverá conter `O`;

Time: tempo real em segundos que o algoritmo gastou em sua execução. Utilize a classe Timer fornecida para calcular este tempo. Precisão de 2 casas decimais;

Best Solution Value: valor da melhor solução encontrada. Utilize precisão de 2 casas decimais.

Dica: para gravar sua saída em um arquivo, execute seu programa da seguinte maneira:

```
RA038508.e teste.scp 50 > saida.log
```

Pacotes e resolvedores

O projeto prevê o uso da biblioteca LEMON (Library for Efficient Modeling and Optimization in Networks) que pode ser encontrada no link

`http://lemon.cs.elte.hu`

Esta biblioteca contém uma série de rotinas de estruturas de dados e algoritmos para grafos, com alta eficiente e relativa facilidade de uso. Você deve instalar em seu sistema. Note que se não for instalada em local padrão, você deverá alterar o arquivo `Makefile.inc` incluindo os novos caminhos dos "includes" e "libs".

Os programas poderão usar pacotes disponíveis para resolução de programas lineares e lineares inteiros, como o GLPK (Gnu Linear Programming Kit), disponível livremente. Para aqueles que quiserem usar o resolvedor profissional CPLEX, basta acessar o site

`http://www-01.ibm.com/software/integration/optimization/cplex-optimizer`

e solicitar uma licença de uso acadêmico (você deverá provar seu vínculo com a Unicamp por um email institucional ou página pessoal hospedada na Unicamp). Será da responsabilidade do grupo aprender a sintaxe e como usar o resolvedor. Pode ser interessante utilizar o CPLEX, uma vez que ele pode ajudar a gerar programas com mais possibilidades de se obter soluções melhores e com isso obter bonus. Note que o CPLEX já tem uma estrutura toda pronta de programação linear inteira e para isso um algoritmo exato poderá ser feito inserindo a formulação inteira e colocando as heurísticas para serem chamadas durante a execução do *branch and bound* do próprio CPLEX.

Além do mais, a biblioteca Lemon provê uma ótima interface tanto com o CPLEX como com o GLPK. Recomendamos fortemente sua utilização.

Ferramentas Adicionais

Recomendamos fortemente a utilização de depurados de código como DDD (Data Display Debugger) e Valgrind. Ambos são excelente ferramentas para depuração e localização de vazamentos de memória, acessos indevidos, utilização de ponteiros inválidos, entre outras coisas interessantes.

O projeto de exemplo foi construído usando Eclipse com plugin CDT, que facilita completação de código e navegação por cabeçalhos e fontes. Basta importar o projeto para dentro do mesmo.

Para visualizar os grafos e ciclos gerados, recomendamos utilizar o Graphviz, uma excelente ferramenta de vizualização de grafos.

6 Avaliação

Os programas que não compilarem ou não derem soluções viáveis para instâncias pequenas terão nota limitada a 3.0. Falta de documentação/comentários ou modularização limitarão a nota do programa a 5.0.

Alguns programas que tiverem bom desempenho sobre um grupo de instâncias, no geral (a ordem será feita pelo professor), poderão ter nota adicional no projeto (neste caso a nota não será truncada). A nota adicional será dada nos seguintes casos:

- O melhor programa exato terá 1 ponto adicional.

- O segundo melhor programa exato terá 0,8 ponto adicional.
- O melhor programa dentre as metaheurísticas e conjunto de heurísticas terá 1 ponto adicional.
- O segundo melhor programa dentre as metaheurísticas e conjunto de heurísticas terá 0,8 ponto adicional.
- Pelo menos dois programas não considerados nos itens acima que tiverem características que os destaque (p.ex., bom desempenho mas não suficiente para destacar em primeiro ou segundo lugar; implementações de idéias inovadoras e não triviais que levaram a um bom desempenho; demonstrações teóricas realizadas pelo próprio grupo de propriedades importantes) serão beneficiados com pelo menos 0,5 ponto adicional.
- Se um programa ganhar ponto adicional, então este ponto será dado a todos os integrantes do grupo.

7 Relatório

Além dos códigos fontes, o arquivo `.tgz` (tareado e gzipado) também deve conter um relatório.

Este relatório deve descrever o algoritmo em alto nível e descrito em linguagem algorítmica de maneira que o professor possa entender bem como o(s) algoritmo(s) por trás do programa resolve(m) ou trata(m) o problema.

O relatório deve conter também uma seção de experimentos computacionais, contendo a configuração do computador onde o grupo fez os experimentos (cpu, memória, tipo de processador) e uma tabela contendo os valores e tempos que os programas implementados obtiveram para um conjunto de instâncias a ser disponibilizado posteriormente. A tabela deve destacar as soluções onde o programa obteve solução ótima (se souber).

Referências

[1] T. Koch and A. Martin and S. Voß SteinLib: An Updated Library on Steiner Tree Problems in Graphs Konrad-Zuse-Zentrum für Informationstechnik Berlin Takustr. 7, Berlin, 2000. `http://steinlib.zib.de`

[2] Makoto Matsumoto and Takuji Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comput. Simul.*, 8:3–30, January 1998. ISSN 1049-3301. doi: 10.1145/272991.272995.

[3] Gerhard Reinelt TSPLIB. Universität Heidelberg, Institut für Informatik. Im Neuenheimer Feld 368, D – 69120 Heidelberg, Germany. `http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95`

[4] B. Dezso, A. Juttner, P. Kovacs. LEMON - an OPEN Source C++ Graph Template Library. *Electronic Notes in Theoretical Computer Science*, 264: 23–45, 2011. doi: 10.1016/j.entcs.2011.06.003.
Elsevier: `http://www.sciencedirect.com/science/article/pii/S1571066111000740`
URL: `http://lemon.cs.elte.hu/`
Slides: `http://lemon.cs.elte.hu/pub/doc/lemon-intro-presentation.pdf`