Implementações do Fluxo a Custo Mínimo

Caio Dias Valentim
Otimização Combinatória
Departamento de Informática - PUC-Rio

8 de dezembro de 2010

Resumo

Neste trabalho investigamos implementações para algoritmos clássicos de fluxo a custo mínimo. A partir das implementações, comparamos a eficiência prática desses algoritmos.

1 Introdução

O problema de fluxo a custo mínimo é problema bastante estudado na literatura de algoritmos. Além de ser um problema conceitualmente interessante, ele modela diversas situações práticas. Dessa forma, diversos algoritmos foram desenvolvidos ao longo dos anos para o problema.

Nesse trabalho, estudamos os algoritmos: *Cycle Canceling* e *Successive Shortest Path*. Para o *Cycle Canceling* estudamos duas variações, uma em que o ciclo escolhido é arbitrário e a outra onde o ciclo de média é sempre escolhido. Da mesma forma, para o *Sucessive Shortest Path* estudamos a versão em que os caminhos são arbitrários e a que caminhos são escolhidos a partir de um limite Δ.

Nas seções abaixo, definimos o problema formalmente, detalhamos os algoritmos estudados, mostramos os detalhes da nossa implementação e das instâncias usadas para teste e, por fim, mostramos os experimentos realizados e tiramos conclusões dos mesmos.

2 Formalização do Problema

O problema do fluxo a custo mínimo consiste, basicamente, em encontrar a forma mais barata distribuir fluxo em uma rede de forma a satisfazer um conjunto de demandas e ofertas. Um fator de dificuldade é que a rede em questão tem restrições de com capacidades e custos nas arestas.

Para formalizar o problema precisamos de um terminologia:

Dado um grafo direcionado $G(V, E)$, definido por um conjunto de vértices(nós) V e um conjunto de arestas(arcos) E. Para todo arco $(i, j) \in E$ existe um capacidade associada μ_{ij} que determina a maior quantidade de fluxo que pode passar por essa aresta. Além disso,

cada aresta $(i, j) \in E$ apresenta um custo c_{ij} que representa o custo de passar uma unidade de fluxo por essa aresta.

Cada vértice $i \in V$ tem associado um número b_i. Esse número representa a demanda oferta do vértice. Se $b_i > 0$, o nó i é um nó de oferta, se $b_i < 0$ o nó i é de demanda.

Com a terminologia em mente, o problema do fluxo a custo mínimo pode ser representado formalmente pelo problema de otimização abaixo:

$$\min z(x) = \sum_{(i,j)\in E} c_{ij}x_{ij}$$

sujeito:

$$\sum_{j:(i,j)\in E} x_{ij} - \sum_{j:(j,i)\in E} x_{ji} = b_i \text{ ,para todo } i \in V$$

$$0 \leq x_{ij} \leq \mu_{ij}, \text{ para todo } (i, j) \in E$$

3 Algoritmos

Nessa seção apresentamos uma visão dos algoritmos implementados.

3.1 Cycle Canceling

Antes de explicar o algoritmo, vamos enunciar o lema principal que garante sua corretude.

Lemma 3.1 (Critério de Otimalizadade de Ciclos Negativos)**.** *Seja x* uma solução viável para o problema do fluxo a custo mínimo. Então, x* é ótima se, e somente se, a rede residual G_{x*} não contém ciclos negativos.*

Bom, em vista do lema acima, o algoritmo consiste em começar uma solução viável x para o problema e, a cada passo, tentar achar um ciclo negativo na rede residual G_x. Se um ciclo negativo for encontrado, passamos o máximo de fluxo possível por esse ciclo e atualizamos x de acordo. Se não existir um ciclo, então x é uma soluções ótima.

Note que na descrição do algoritmos dois pontos não totalmente definidos: Como achar uma solução viável inicial e como achar um ciclo negativo. Bom, para primeira parte questão, podemos achar uma solução viável com um algoritmo de fluxo máximo. Agora, o ponto mais interessante é como encontrar os ciclos negativos. De fato, a complexidade do algoritmo é diretamente ligada a esse passo.

Nesse contexto, chamaremos de forma genérica de *Cycle Canceling* o algoritmo que a cada passo encontra um ciclo arbitrário. E chamaremos de *Mean Cycle Canceling* o algoritmo que a cada passo encontra o ciclo de média mínima.

4 Implementação

Todos os algoritmos foram implementados em `C++` por ser uma linguagem de alto desempenho e por ser um denominador comum de conhecimento entre os integrantes do grupo.

Nas seções abaixo detalhamos a implementação dos algoritmos citados.

4.1 Cycle Canceling

A implementação do *Cycle Canceling* assume os seguintes fatos sobre a rede de entrada:

4.1.1 Restrições

1. *Todas capacidades, custos e demandas são números inteiros.* Como esses valores são guardados em variáveis do tipo inteiro de `C++` os seus valores são devem estar no intervalo *aberto* $[-2^{31}, 2^{31})$. Apesar desse intervalo, as capacidades e custos na nossa implementação devem ser inteiros positivos.

2. *Não existem arcos paralelos.* A capacidade, fluxo e custos são mantidos em uma matriz. Dessa forma, como precisamos guardar os arcos reversos não podem existir arcos paralelos.

3. *A rede é direcionada.* Essa é uma restrição do modelo. De forma geral, podemos simplesmente transformar uma rede não-direcionada em uma direcionada. Porém, essa transformação cria arcos paralelos que não são permitidos na nossa implementação.

As restrições mais importantes citatas mais algumas estruturais fazem parte dos possíveis códigos de erro do programa:

```
MCF_SUCCESS - Fluxo a custo mínimo calculado com sucesso.
MCF_ERROR_PARALLEL_EDGE - Arestas paralelas na rede entrada.
MCF_ERROR_BAD_INBALANCE - A soma das demandas mais ofertas não é zero.
MCF_ERROR_NEGATIVE_EDGE_COST - Alguma aresta tem custo negativo.
MCF_ERROR_NO_FEASEABLE_SOLUTION - A rede não apresenta uma solução viável.
```

4.1.2 Solução viável

Para achar a solução viável inicial resolvemos um problema de fluxo máximo. Criamos dois novos nós `SINK` e `SOURCE` e para todo vértice de i, se i for um nó de oferta, criamos uma aresta do nó `SOURCE` para ele com capacidade igual a b_i. Por outro lado, se i for um nó de demanda criamos uma aresta dele para o `SINK` com capacidade $-b_i$.

Uma vez feita a transformação acima, rodamos o algoritmo de *Ford-Fulkerson* para achar o fluxo máximo entre o `SOURCE` e o `SINK`. Se o fluxo máximo tiver valor 0, retornamos o código de erro `MCF_ERROR_NO_FEASEABLE_SOLUTION`, caso contrário removemos os dois nós criados e continuamos com os próximos passos com a solução viável dada pelo fluxo máximo.

4.1.3 Ciclo Negativo

Uma vez com uma solução viável x, devemos saber se existe ao menos um ciclo de custo negativo na rede. Para fazer isso rodamos o algoritmo de *Bellman-Ford* na rede residual G_x e verificamos se há ciclos negativos. O algoritmo indica isso de forma direta com um porém. O Bellman-Ford, originalmente, acha os menores caminhos de um de partida para todos os outros. Dessa forma, ele só detecta um ciclo negativo se esse for alcançável pelo nó de partida. Com isso, a princípio, deveríamos rodar o Bellman-Ford para cada nó na

rede para ter certeza que não há ciclos negativos. Isso adicionaria um custo de $|V|$ a um algoritmo que já não é barato.

Contudo, existe uma forma mais eficiente para detectar os ciclos negativos a partir do Bellman. Basta criar um novo nó na rede e ligar esse nó a todos os outros nós. É fácil ver que qualquer ciclo negativo na rede é alcançável por esse novo nó. Desta forma, podemos simplesmente executar o Bellman a partir desse nó para detectar possíveis ciclos.

4.1.4 Complexidade

A implementação usando Ford-Fulkerson para achar uma solução viável tem complexidade:

$$O(C|V|(|V| + |E|) + |V|^2|E|UC)$$

onde C é a maior capacidade de um arco na rede e U é o maior custo de um arco. A parte, $C|V|(|V| + |E|)$ vem da primeira fase de achar uma solução viável.

4.2 Mean Cycle Canceling

A implementação do *Mean Cycle Canceling* só difere da anterior na parte de achar o ciclo de média mínima.

Uma vez que temos uma solução viável x e uma rede residual G_x queremos achar o ciclo de média mínima. Para achar o ciclo fazemos uma busca binária pelo ciclo média mínima.

De forma detalhada, primeiro observe que podemos testar se existe um ciclo com de valor até l da seguinte forma: Modifique todas os custos c_{ij} da rede residual para $c_{ij} - l$. Assim, existe um ciclo de média até l no rede original se, e somente se, existir um ciclo negativo na rede residual modificada. Podemos novamente utilizar o Bellman-Ford para verificar se há um ciclo negativo. Se existir um ciclo, tente valores menores para l, caso contrário, tentar valores maiores.

Como o valor de l é limitado por $-U \leq l \leq U$, achar o ciclo mínimo custará $O(log(U) * |V|^2)$. Como é provado que o número de interações máximo desse algoritmo é um polinômio, a complexidade total será polinomial.

5 Instâncias

6 Experimentos

7 Conclusão