Plano de Estudos
MAC 5701 - Tópicos em Ciência da Computação

Implementações do Método de Aproximação Primal-Dual Aplicado ao Problema da Floresta de Steiner

Aluno: Rafael Pereira Luna
Orientadora: Cristina Gomes Fernandes

São Paulo, 24 de Setembro de 2003

Sumário

1. **Introdução** 1
2. **O Problema** 1
3. **O Algoritmo** 1
    4. Idéia geral do algoritmo 2
    5. Descrição do algoritmo 2
6. **Implementações** 3
    7. Implementação de Cole, Hariharan, Lewenstein e Porat 4
        8. Descrição da implementação 4
9. **Plano de Estudos** 6
10. **Bibliografia** 6

1. Introdução

Apresentaremos aqui um plano de estudos para as atividades que iremos desenvolver dentro do escopo da disciplina MAC 5701 - Tópicos em Ciência da Computação no decorrer deste semestre. Tais atividades estarão voltadas para o estudo de algumas das possíveis implementações do método de aproximação primal-dual aplicado ao problema da floresta de Steiner. Inicialmente, definiremos o problema em questão e daremos uma descrição do algoritmo primal-dual proposto por Goemans e Williamson para o problema [2]. Após isso, citaremos duas implementações distintas deste algoritmo, nos aprofundando mais na descrição de uma delas, proposta por Cole, Hariharan, Lewenstein e Porat [3]. Ao final, detalharemos o plano de estudos para as atividades que iremos realizar neste semestre.

2. O Problema

Nesta e nas próximas seções, denotaremos o conjunto dos vértices e o conjunto das arestas de um grafo G por V_G e E_G, respectivamente. Além disso, denotaremos por $Q_{\geq}$ o conjunto dos racionais não-negativos.

O **problema da floresta de Steiner** consiste do seguinte: dados um grafo G, uma função custo c de E_G em $Q_{\geq}$ e uma coleção R de subconjuntos de V_G, encontrar uma *R-floresta* F que minimize $c(F) = \sum_{e \in E_F} c_e$.

Cada conjunto da coleção R é chamado de *conjunto de terminais*. Uma *R-floresta* F é uma floresta geradora de G que tem a seguinte propriedade: cada conjunto de terminais em R se encontra inteiramente contido no conjunto dos vértices de alguma das componentes de F. Quando R está implícito, dizemos que uma floresta satisfazendo tal propriedade é uma *floresta de Steiner* de G. Note que podemos assumir que os conjuntos de R são dois a dois disjuntos (caso contrário, bastaria substituir cada par de conjuntos que não satisfaz esta restrição pela união dos conjuntos no par).

Não se conhece um algoritmo eficiente (polinomial no tamanho da entrada (G, c, R)) para resolver o problema acima. Se a coleção R consiste em apenas um conjunto, o problema se reduz ao conhecido *problema de Steiner em grafos*, o qual é NP-difícil.

3. O Algoritmo

As implementações que estudaremos são todas baseadas num algoritmo de aproximação projetado por Goemans e Williamson [2] para o problema. Tal algoritmo utiliza o método de aproximação primal-dual e é uma 2-aproximação para o Problema da Floresta de Steiner (uma explicação detalhada sobre algoritmos de aproximação, incluindo o método de aproximação primal-dual, pode ser encontrada em [1]).

4. Idéia geral do algoritmo

Um conceito fundamental no qual se apóia o algoritmo é o de *conjunto ativo*. Dados um grafo G e uma coleção R de conjuntos de terminais, um subconjunto S de V_G é dito *ativo* se, e somente se, existe um conjunto T em R tal que $T \cap S \neq \emptyset$ e $T \setminus S \neq \emptyset$. Quando S é o conjunto dos vértices de alguma componente de uma floresta F de G, dizemos que tal componente é uma componente *ativa* de F; caso contrário, a componente é dita *inativa*.

Note que uma floresta geradora F de G é uma R-floresta se, e somente se, $E_F \cap \delta(S) \neq \emptyset$ para todo conjunto S ativo, onde $\delta(S)$ é o corte determinado por S no grafo G. Assim, podemos enunciar o problema da floresta de Steiner na forma de um programa inteiro, como segue abaixo.

$$\begin{array}{ll} \text{minimize } cx & \\ \text{sob as restrições} & \sum_{e\in\delta(S)} x_e \geq 1, \text{ para todo } S \text{ ativo} \\ & x_e \in \{0,1\}, \text{ para toda aresta } e \text{ em } E_G. \end{array}$$

Neste programa, x é um vetor indexado por E_G e c é a função custo definida no problema. Note que toda floresta de Steiner do grafo G pode ser interpretada como uma solução viável do programa acima: dada uma floresta de Steiner F do grafo G, o vetor x tal que $x_e = 1$ para e em E_F e $x_e = 0$ para e em $E_G \setminus E_F$ é uma solução viável do programa.

O programa linear abaixo é uma relaxação linear do programa inteiro.

$$\begin{array}{ll} \text{minimize } cx & \\ \text{sob as restrições} & \sum_{e\in\delta(S)} x_e \geq 1, \text{ para todo } S \text{ ativo} \\ & x_e \geq 0, \text{ para toda aresta } e \text{ em } E_G. \end{array}$$

O dual do programa acima é dado a seguir, onde y é um vetor indexado pela coleção de todos os conjuntos ativos de G.

$$\begin{array}{ll} \text{maximize } \sum_S y_S & \\ \text{sob as restrições} & \sum_{S:e\in\delta(S)} y_S \leq c_e, \text{ para toda aresta } e \text{ em } E_G \\ & y_S \geq 0, \text{ para todo } S \text{ ativo.} \end{array}$$

Em linhas gerais, o algoritmo parte de uma solução viável y do programa dual e, iterativamente, tenta "melhorar" esta solução até obter uma solução viável inteira x do programa primal que tenha custo igual a, no máximo, duas vezes o custo de uma solução ótima do problema da floresta de Steiner. Na verdade, no algoritmo, tal solução x é representada implicitamente por uma floresta geradora de G. Cada uma das iterações começa com uma floresta geradora de G; o algoritmo aumenta o valor das componentes de y correspondentes às componentes ativas desta floresta, fazendo com que a restrição do programa dual seja satisfeita com igualdade para alguma das arestas do grafo e, quando isto ocorre, tal aresta é incluída na floresta. Quando nenhuma das componentes da floresta corrente for ativa o algoritmo termina, devolvendo uma sub–floresta de Steiner minimal da floresta corrente.

A subseção abaixo descreve, mais detalhadamente, esse algoritmo.

5. Descrição do algoritmo

Cada iteração do algoritmo começa com uma floresta geradora de G que contém pelo menos uma componente ativa (e, portanto, o vetor x correspondente a esta floresta não satisfaz uma ou mais restrições do programa primal) e um vetor y, solução viável do programa dual. Na verdade, y não é armazenado explicitamente; ao invés disso, guarda-se o valor da função objetivo dual em y e, para cada vértice v do grafo, o número $d(v)$ que é a soma das componentes de y correspondentes às componentes às quais v pertenceu nas florestas das iterações anteriores (incluindo a floresta corrente). A primeira iteração começa com uma floresta que não contém arestas, sendo composta por n componentes, onde n é o número de vértices do grafo (ou seja, cada vértice é uma componente da floresta). Além disso, y é igual ao vetor nulo.

Em cada iteração, uma *aresta externa* é incluída na floresta; uma *aresta externa* nada mais é do que uma aresta que une duas componentes distintas da floresta. A aresta externa escolhida pelo algoritmo para ser incluída na floresta é aquela que apresenta a menor *folga* (na verdade, pode haver mais de uma). O valor da folga de uma aresta uv é calculado da seguinte forma:

- $(c_{uv} - d(u) - d(v))/2$, se u e v pertencem a componentes ativas;
- $c_{uv} - d(u) - d(v)$, se u ou v, mas não ambos, pertence a uma componente ativa;
- ∞, se u e v pertencem a componentes inativas.

Em caso de empate, é dada preferência para arestas que tenham cada um dos extremos em uma componente ativa.

Uma vez que foi escolhida uma aresta, atualiza-se o custo da floresta (somando-se o custo da aresta incluída), o valor da função objetivo dual (somando-se a folga da aresta incluída multiplicada pelo número de componentes ativas) e o valor de $d(v)$ para cada vértice v em uma componente ativa (somando-se a folga da aresta incluída).

O algoritmo pára quando todas as componentes da floresta estiverem inativas (o que ocorre após, no máximo, $n-1$ iterações), devolvendo uma floresta de Steiner minimal contida na floresta corrente.

Abaixo, apresentamos, em pseudo-código, uma descrição não muito detalhada do algoritmo.

Algoritmo *MinFS(G, R, c)*

$F \leftarrow (V_G, \emptyset)$
$E_{ext} \leftarrow E_G$
$d(v) \leftarrow 0$ para cada v em V_G
enquanto F tem alguma componente ativa faça
    escolha $\overline{a}$ tal que $folga(\overline{a}) = \min\{folga(a) \mid a \in E_{ext}\}$
    $d(v) \leftarrow d(v) + folga(\overline{a})$ para cada v em uma componente ativa
    inclua $\overline{a}$ em F
    $E_{ext} \leftarrow E_{ext} \setminus \{uv \in E_G \mid u$ e v pertencem a uma mesma componente de $F\}$
seja F' uma R-floresta minimal contida em F
devolva F'

6. Implementações

O algoritmo descrito na seção anterior pode ser implementado de diferentes maneiras. A implementação originalmente proposta Goemans e Williamson [2] consome tempo $O(n^2 \log n)$, onde n é o número de vértices do grafo de entrada, e é bastante adequada para grafos densos. Tal implementação foi, recentemente, alvo de nosso estudo [6]. Subseqüentemente, a complexidade de tempo desta implementação foi melhorada por outras implementações (como [3], [4] e [5]). Apresentaremos a seguir uma destas implementações.

7. Implementação de Cole, Hariharan, Lewenstein e Porat

Para grafos não muito densos, existem implementações que apresentam um melhor desempenho que a implementação de Goemans e Williamson. Um exemplo é a implementação proposta por Cole, Hariharan, Lewenstein e Porat que passaremos a descrever.

Esta implementação difere da implementação de Goemans e Williamson por tentar garantir que, durante a fase iterativa do algoritmo, cada aresta externa do grafo tenha no máximo um dos extremos em uma componente ativa da floresta (note que, desta forma, cada componente precisaria manter apenas um heap de arestas). Para isto, cada aresta do grafo é particionada dinamicamente ("dynamic edge spliting") através da inclusão de um ou mais vértices no grafo. A idéia aqui é a seguinte: sempre que uma aresta tem ambos os extremos em componentes ativas da floresta, divide-se esta aresta em duas arestas de igual comprimento através da inclusão de um novo vértice no grafo, o qual será considerado como uma nova componente inativa da floresta. Denominaremos um tal vértice de *s-vértice*. Durante toda execução da fase iterativa do algoritmo, uma aresta (do grafo original) é particionada no total $O(k \log n)$ vezes, onde k é uma dada constante fornecida como parâmetro. Se após isso alguma das arestas geradas através de uma dessas divisões da aresta original tiver ambos os extremos em componentes ativas, tal aresta é tratada como se apenas um de seus extremos fizesse parte de uma componente ativa. Isso introduz um erro no algoritmo, o qual é responsável por uma degradação de $\frac{1}{n^k}$ no fator de aproximação.

Para cada constante k, a implementação consome no total tempo $O(k(n+m)\log n)$, onde n e m são respectivamente o número de vértices e o número de arestas do grafo, e devolve uma floresta de Steiner de custo menor ou igual a $2(1+\frac{1}{n^k})$ vezes o custo de uma floresta de Steiner ótima.

8. Descrição da implementação

Como na implementação de Goemans e Williamson, faremos uso de heaps de arestas para reduzir o número de arestas examinadas em cada iteração do algoritmo, com a diferença de que aqui utilizaremos apenas um único heap por componente. A cada componente S da floresta estará associado um heap de mínimo $heap(S)$ contendo as arestas externas que possuem um dos extremos em S. A chave de uma dada aresta uv em $heap(S)$ é definida como segue:

$$chave(uv) = \begin{cases} c_{uv} - d(u) & \text{, se } u \in S \\ c_{uv} - d(v) & \text{, caso contrário.} \end{cases}$$

Esta definição é motivada pela seguinte propriedade garantida pela implementação: dada uma aresta uv do grafo original, cada nova aresta $u'v'$ gerada a partir de uv, com exceção de uma única aresta, a qual será chamada *aresta terminal* (mais adiante, ficará clara a motivação por trás desta terminologia), é tal que ou u' ou v' faz parte de uma componente inativa da floresta e, além isto, tal componente consiste de um único s-vértice. Assim, toda aresta externa $u'v'$ que não é terminal é tal que

$$folga(u'v') = \min\{c_{u'v'} - d(u'), c_{u'v'} - d(v')\}$$

pois $d(w) = 0$ para todo s-vértice w que é extremo de uma aresta externa. As arestas terminais podem possuir ambos os extremos em componentes ativas e, neste caso, a folga de tais arestas não pode ser calculada deste modo. Apesar disto, tais arestas serão tratadas como se apenas um de seus extremos fizesse parte de uma componente ativa e, assim, também consideraremos a folga de uma aresta terminal como sendo dada pela igualdade acima.

Logo, em cada iteração, para determinar a aresta externa que será incluída na floresta, serão examinadas as arestas que se encontram na primeira posição de cada heap associado a uma componente ativa e, dentre elas, será selecionada uma que possua a menor chave.

Vamos apresentar agora uma descrição não muito detalhada do algoritmo correspondente a esta implementação. Inicialmente, cada aresta do grafo é subdividida em duas arestas de igual comprimento através da inclusão de um novo vértice no grafo, sendo que cada vértice incluído corresponde a uma nova componente inativa da floresta (lembre-se que cada vértice em um conjunto da coleção R é considerado uma componente ativa e que cada vértice de Steiner do grafo original é considerado uma componente inativa). Além disso, a partir das arestas geradas através desta operação, é criado o heap associado a cada componente.

Em cada uma das iterações seguintes, o algoritmo executa os passos descritos abaixo:

1. *Escolha da próxima aresta*: inicialmente, é escolhida uma aresta externa que apresente a menor folga dentre todas as outras. Seja uv a aresta escolhida neste passo. Para cada vértice z pertencente a uma componente ativa da floresta, $d(z)$ é incrementado de $folga(uv)$. Além disso, uv é incluída na floresta. Por fim, é realizada a união dos conjuntos de vértices das componentes da floresta correspondentes aos vértices u e v.

2. *Divisão de arestas*: note que, no início desta iteração, pelo menos uma das componentes correspondentes aos vértices u e v era ativa. Se essas duas componentes eram ativas, este passo não é executado e o algoritmo passa a executar o próximo passo. Caso contrário, uma das componentes era inativa e, mais do que isso, tal componente era formada por apenas um único vértice, o qual é um s-vértice. Vamos supor, sem perda de generalidade, que esse vértice seja v. Como v é um s-vértice, existe apenas uma única aresta incidente em v, além de uv. Seja w o vértice que se encontra na outra extremidade desta aresta. Vamos dividir a aresta vw em pedaços subseqüentes através da adição de novos vértices ao grafo, os quais deverão se encontrar às distâncias $\lceil \frac{c_{vw}}{2} \rceil$, $\lceil \frac{c_{vw}}{4} \rceil$, $\lceil \frac{c_{vw}}{8} \rceil$, ..., $\lceil \frac{c_{vw}}{2^i} \rceil$ de v, onde i é o menor índice tal que uma das seguintes condições é satisfeita:

   - $c_{vw} - \lceil \frac{c_{vw}}{2^i} \rceil > d(w)$.
   - $\lceil \frac{c_{vw}}{2^i} \rceil \leq \lceil \frac{c_{v'w'}}{n^k} \rceil$, onde $v'w'$ é a aresta do grafo original a partir da qual vw foi gerada.

   A segunda condição acima é usada para garantir que cada aresta do grafo original seja dividida não mais que $O(k \log n)$ vezes. Sempre que tal condição for satisfeita, todas as arestas geradas como descrito acima, com exceção da última aresta (cujo comprimento é menor ou igual a $\lceil \frac{c_{v'w'}}{n^k} \rceil$), passarão a fazer parte da floresta. Note que esta última aresta não pode mais ser subdivida, mesmo que, em alguma iteração futura, ela possa ter ambos os extremos em componentes ativas (estas são as tais arestas terminais às quais nos referimos anteriormente).

   Agora, suponha que i seja tal que a primeira condição é satisfeita e seja z o vértice adicionado que se encontra a distância $\lceil \frac{c_{vw}}{2^i} \rceil$ de v. Note que, neste caso, todos os novos vértices adicionados ao grafo, com exceção de z, estarão a uma distância de w menor ou igual a $d(w)$. Deste modo, todos eles passarão a fazer parte da componente que contém w, ao passo que z irá formar uma nova componente inativa da floresta (para a qual será criado um heap contendo inicialmente duas arestas). Além disso, todas as novas arestas que foram criadas neste passo, com exceção daquelas incidentes no vértice z, serão incluídas na floresta. Por fim, as duas arestas incidentes em z serão inseridas em $heap(S_v)$ e $heap(S_w)$, onde S_v e S_w são, respectivamente, as componentes que contém v e w, ao passo que a aresta vw será removida destes dois heaps.

3. *Fusão de heaps*: sejam S_u e S_v as componentes de u e v respectivamente. Se tanto S_u quanto S_v são ativas (e, neste caso, uv é uma aresta terminal) então $heap(S_u)$ e $heap(S_v)$ são combinados para formar um único heap e, além disso, o estado (com relação à atividade) da nova componente formada pela união de S_u e S_v é determinado.

9. Plano de Estudos

Inicialmente, pretendemos concluir o programa correspondente a implementação proposta por Goemans e Williamson, o qual ainda requer alguns ajustes. Após isso, iremos implementar a estratégia de Cole, Hariharan, Lewenstein e Porat (em CWEB) e em seguida iniciar o estudo da implementação sugerida por Klein [5] (caso haja tempo, tal estratégia também será implementada). Ao final, faremos um estudo comparativo experimental envolvendo todas as implementações realizadas.

10. Bibliografia

Referências

[1] M.H. Carvalho, M.R. Cerioli, R. Dahab, P. Feofiloff, C.G. Fernandes, C.E. Ferreira, K.S. Guimarães, F.K. Miyazawa, J.C. Pina Jr., J. Soares e Y. Wakabayashi, *Uma Introdução Sucinta a Algoritmos de Aproximação*, Publicações Matemáticas do IMPA, 2001.

[2] M. Goemans e D. Williamson, *A General Approximation Technique for Constrained Forest Problems*, SIAM Journal on Computing 24 (2), pp. 296–317, 1995.

[3] R. Cole, R. Hariharan, M. Lewenstein e E. Porat, *A Faster Implementation of the Goemans-Williamson Clustering Algorithm*, SODA 2001.

[4] H. Gabow, M. Goemans e D. Williamson, *An Efficient Approximation Algorithm for the Survivable Network Design Problem*, IPCO 1993.

[5] P.N. Klein, *A Data Structure for Bicategories with Applications to Speeding Up an Approximation Algorithm*, Information Processing Letters 52, pp. 303–307, 1994.

[6] R.P. Luna, *Implementações do Método de Aproximação Primal-Dual Aplicado ao Problema da Floresta de Steiner*, Trabalho de Formatura Supervisionado (MAC 499), 2002, `http://www.linux.ime.usp.br/~cef/mac499-02/monografias/luna-nova/`.