EXAME DE QUALIFICAÇÃO
MESTRADO EM CIÊNCIA DA COMPUTAÇÃO

k-árvores de custo mínimo

MARCIO TAKASHI IURA OSHIRO
ORIENTADOR: JOSÉ COELHO DE PINA

DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO
INSTITUTO DE MATEMÁTICA E ESTATÍSTICA
UNIVERSIDADE DE SÃO PAULO

1 Introdução

Uma **k-árvore** de um grafo (V, E) é uma árvore com pelo menos k vértices. Se existe uma função de custos c de E para $\mathbb{Z}_+$, dizemos que o custo de uma k-árvore T é a soma dos custos de suas arestas e é denotado por $c(T)$.

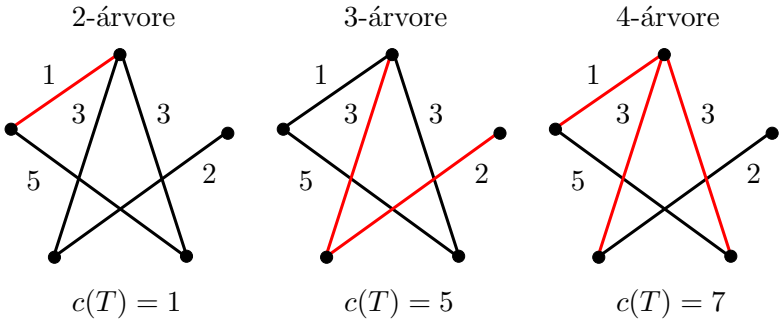

Figura 1: Exemplo de k-árvores com custo nas arestas.

Um problema clássico em otimização combinatória é o de encontrar uma árvore geradora de custo mínimo em um grafo (MST). Ou seja, se n é o número de vértices do grafo, esse problema consiste em encontrar uma n-árvore com o menor custo possível. Existem algoritmos eficientes para resolver o MST [11, 13].

O problema de encontrar uma k-árvore de custo mínimo (kMST) é uma generalização do MST, pois a quantidade de vértices da árvore que queremos encontrar é um dos parâmetros do problema. Uma descrição mais precisa é dada a seguir.

Problema $k\text{MST}(V, E, c, k)$: *dado um grafo conexo (V, E), um número inteiro k e custos não-negativos c_e para cada aresta e, encontrar uma k-árvore de custo mínimo.*

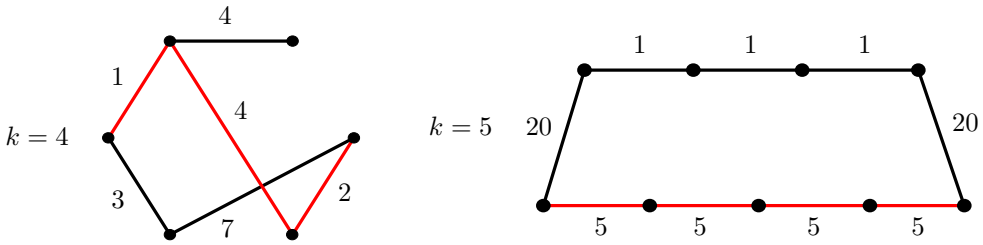

Figura 2: Exemplo de k-árvores de custo mínimo.

É claro que $1 \leq k \leq |V|$, caso contrário o problema não faz sentido.

Uma variante equivalente do kMST, chamada **enraizada** ou **com raiz**, é descrita a seguir.

> **Problema** $k\text{MST}(V, E, c, k, r)$: *dado um grafo conexo (V, E), custos não-negativo c_e para cada aresta e, um número inteiro k e um vértice r, encontrar uma k-árvore de custo mínimo que contém r.*

As duas versões do kMST são equivalentes, pois um algoritmo eficiente que resolve uma pode ser usado como subrotina de um algoritmo eficiente para a outra. De fato, podemos resolver o $k\text{MST}(V, E, c, k)$ resolvendo o $k\text{MST}(V, E, c, k, r)$ para cada r em V e tomando dentre as k-árvores obtidas uma de menor custo. Já o $k\text{MST}(V, E, c, k, r)$ pode ser resolvido através do $k\text{MST}(V, E, c, k)$ adicionando $n := |V|$ vértices conectados a r por arestas de custo 0. Neste caso o que procuramos é uma $(k+n)$-árvore de custo mínimo. Pela quantidade de vértices adicionados, certamente r estará em uma $(k + n)$-árvore de custo mínimo e removendo os vértices adicionados, obtemos uma k-árvore de custo mínimo para a instância original.

Consideraremos tanto a versão sem raiz como a com raiz, dependendo de qual for mais conveniente para a situação.

Apesar de existirem casos particulares do kMST para os quais existem algoritmos polinomiais, ele é um problema NP-difícil. Isso significa que sob a hipótese de que P $\neq$ NP, sabemos que não existem algoritmos polinomiais para resolvê-lo. Assim, uma das maneiras de tratá-lo eficientemente é por meio de algoritmos de aproximação.

2 Complexidade

Para verificar que o kMST é NP-difícil vamos considerar a versão com raiz. Mostraremos que um outro problema NP-difícil, que chamaremos de ST, poder ser reduzido polinomialmente a ele. Isso quer dizer que dada uma instância do ST, podemos gerar, em tempo polinomial, uma instância para kMST cuja solução pode ser transformada, em tempo polinomial, em uma solução para a instância do ST.

A descrição do problema ST é dada a seguir. Sejam (V, E) um grafo conexo e R um subconjunto de V. Chamamos R de **conjunto de vértices terminais**. Uma **árvore de Steiner** é uma árvore que contém todos os vértices terminais.

Problema ST(V, E, R): *dado um grafo conexo* (V, E) *e um conjunto* $R \subseteq V$, *encontrar uma árvore de Steiner com número mínimo de arestas.*

O problema ST é conhecidamente NP-difícil [6].

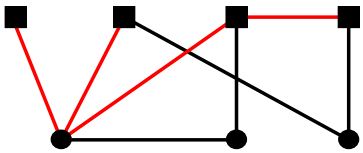

Figura 3: Exemplo de árvore de Steiner mínima. Vértices quadrados são terminais.

Teorema 1. O problema ST pode ser reduzido polinomialmente ao kMST.

3 kMST em árvores

Apesar do kMST ser um problema NP-difícil, existem casos em que podemos resolvê-lo em tempo polinomial. Um desses casos ocorre quando restringimos o grafo (V, E) da instância do kMST a árvores. Ou seja, quando queremos encontrar uma k-árvore mínima em uma árvore.

Maffioli Francesco [12] propôs um algoritmo de programação dinâmica para resolver o kMST com raiz em árvores. Esse algoritmo, no entanto, apenas devolve o custo da k-árvore mínima. Christian Blum [4] mostra as estruturas de dados necessárias para que o algoritmo também devolva a árvore. A idéia do algoritmo é a seguinte.

Sejam T uma árvore e r sua raiz. Denotaremos por T_v a árvore com raiz v de T tal que um vértice v' de T está em T_v se e somente se o caminho de r a v' contém v.

Para usar programação dinâmica dividimos o problema de encontrar uma k-árvore mínima em subproblemas. O subproblema consiste em achar, para cada vértice v de $T = (V, E)$, uma l-árvore mínima, $0 \leq l \leq \min\{k, |V|\}$, em T_v.

Para resolver os subproblemas calculamos os seguintes valores:

- $f_{-v,l}$: custo da l-árvore mínima de T_v que não contém v.
- $f_{+v,l}$: custo da l-árvore mínima de T_v que contém v.

Dessa forma o custo de uma l-árvore em T_v é $f_{v,l} = \min\{f_{-v,l}, f_{+v,l}\}$.

Esses valores são calculados segundo a recorrência abaixo. Se v é um vértice de T com q filhos, então denotaremos os filhos de v por $v_1, v_2, \ldots, v_q$.

$$f_{-v,l} = \begin{cases} \infty & \text{se } v \text{ é uma folha} \\ \min\{f_{v_i,l} | i = 1, 2, \ldots, q\} & \text{caso contrário} \end{cases}$$

$$f_{+v,l} = \begin{cases} 0 & \text{se } v \text{ é uma folha} \\ \min\{\sum_{i=1}^{q} (\delta(\alpha_i) \times (c_{vv_i} + f_{+v_i,\alpha_i})) | \sum_{i=1}^{q} \alpha_i = l - q, \alpha_i \geq -1\} & \text{caso contrário} \end{cases}$$

O valor $\alpha_i = -1$ indica que o T_{v_i} não contribui com nenhum vértice. A função $\delta : \mathbb{Z} \to \{0, 1\}$ é tal que $\delta(x) = 0$ se $x < 0$ e $\delta(x) = 1$ caso contrário. Note que uma implementação ingênua do cálculo de $f_{+v,l}$ pode consumir tempo exponencial, pois o número de $\alpha_1, \alpha_2, \ldots, \alpha_q$ tais que $\sum_{i=1}^{q} \alpha_i = l - q$ e $\alpha_i \geq -1$ é exponencial. Em [4] Christian Blum mostra como fazer esse cálculo em tempo polinomial.

4 Algoritmos de aproximação

Um **problema de otimização** é definido em três partes:

- conjunto de **instâncias**
- conjunto de **candidatos a solução**, também chamado de conjunto de **soluções viáveis**, para cada instância
- função não negativa que define o **valor**, ou **custo**, de um candidato a solução

Problemas de otimização podem ser de **maximização** ou **minimização**. Em problemas de maximização queremos encontrar um candidato a solução de valor máximo. De maneira análoga, em problemas de minimização queremos encontrar candidatos a solução de valor mínimo. O kMST, MST e ST são problemas de minimização. Dizemos que a solução do problema é um candidato de custo máximo, no caso de problemas de maximização, ou de custo mínimo, no caso de problemas de minimização.

No caso do kMST com raiz, uma instância é dada por um grafo conexo com custos não negativos nas arestas, um número inteiro k e um vértice r. Um candidato a solução é uma k-árvore que contém r e seu valor é o custo dessa k-árvore. Como o grafo é conexo, se $1 \leq k \leq n$, onde n é o número de vértices do grafo, então sempre existe um candidato a solução. Denotaremos por T^* uma solução do kMST, ou seja, uma k-árvore mínima e por $\text{OPT} := c(T^*)$ o custo de T^*.

Muitos problemas de otimização são NP-difíceis, ou seja, sob a hipótese de que P $\neq$ NP, sabemos que não existem algoritmos polinomiais para eles. Existem pelo menos duas maneiras de tratar problemas NP-difíceis: buscar a solução exata apesar

da demora para obtê-la, ou obter eficientemente um candidato a solução razoavelmente bom. Os chamados algoritmos de aproximação são do segundo tipo.

Um **algoritmo de aproximação** é um algoritmo polinomial para problemas de otimização. Ao invés de procurar por uma solução do problema, ele procura um candidato a solução cujo valor não está longe do valor de uma solução. Isto é, se S é uma solução do problema e C é um candidato a solução devolvido pelo algoritmo de aproximação, então, no caso de problemas de maximização temos

$$\text{valor}(C) \geq \alpha \times \text{valor}(S)$$

e no caso de problemas de minimização temos

$$\text{valor}(C) \leq \alpha \times \text{valor}(S),$$

onde α é um número que pode depender da instância. Em ambos os casos dizemos que C é uma **α-aproximação**. Também usamos a nomenclatura α-aproximação para denotar algoritmos de aproximação que devolvem uma α-aproximação para toda instância do problema.

Chamamos α de **fator de aproximação** do algoritmo. Note que em problemas de maximização $0 < \alpha \leq 1$ e em problemas de minimização $\alpha \geq 1$, pois $\alpha = 1$ corresponde a um algoritmo que encontra a solução exata do problema. Portanto quanto mais próximo de 1 for o fator de aproximação, melhor será o algoritmo de aproximação.

Vários algoritmos de aproximação foram desenvolvidos para o kMST. O primeiro foi apresentado por Ramamurthy Ravi, Ravi Sundaram, Madhav V. Marathe, Daniel J. Rosenkrantz e Sekharipuram S. Ravi [15] e tem fator de aproximação $2\sqrt{k}$. Baruch Awerbuch, Yossi Azar, Avrim Blum e Santosh Vempala [2] obtiveram um algoritmo melhor, com fator de aproximação $O(\log^2 k)$. Sridhar Rajagopalan e Vijay V. Vazirani [14] apresentaram um algoritmo com fator de aproximação $O(\log k)$. Um algoritmo com fator de aproximação 17, o primeiro constante, foi proposto por Avrim Blum, Ramamurthy Ravi e Santosh Vempala [3]. Naveen Garg [7] mostrou em um mesmo artigo um algoritmo com fator de aproximação 5 e como refiná-lo para obter um fator de aproximação 3. Sanjeev Arora e George Karakostas [1] apresentaram um algoritmo com fator de aproximação $2 + \varepsilon$. Atualmente o algoritmo com menor fator de aproximação para o kMST foi obtido por Naveen Garg [8] e tem fator de aproximação 2.

5 Algoritmos combinatórios

O conhecido problema da árvore geradora mínima é um caso particular do kMST, onde k é igual ao número de vértices do grafo. Existem algoritmos polinomiais para resolver o problema da árvore geradora mínima. Um exemplo é o algoritmo de Kruskal.

$2\sqrt{k}$	Ravi, Sundaram, Marathe, Rosenkrantz e Ravi (1994)
$O(\log^2 k)$	Awerbuch, Azar, Blum e Vempala (1995)
$O(\log k)$	Rajagopalan e Vazirani (1995)
17	Blum, Ravi e Vempala (1995)
5 e 3	Garg (1996)
$2 + \varepsilon$	Arora e Karakostas (2000)
2	Garg (2000)

Tabela 1: Evolução do desenvolvimento de algoritmos de aproximação para o kMST.

Algoritmo Kruskal (V, E, c)

```
1  suponha c(e_1) ≤ c(e_2) ≤ ··· ≤ c(e_m), onde {e_1, ..., e_m} = E
2  F ← (V, ∅)
3  para i ← 1 até m faça
4      se F + e_i não forma circuito então F ← F + e_i
5  devolva F
```

Basicamente o algoritmo de Kruskal começa com uma floresta F na qual cada vértice é um componente. A cada iteração tentamos inserir uma aresta em F, juntando dois componentes, de modo que F continue sendo uma floresta.

5.1 Algoritmo kMST-Kruskal

Um algoritmo de aproximação simples para o kMST é dado a seguir. O algoritmo basicamente segue os mesmos passos do algoritmo de Kruskal. A diferença está no fato de que ele pára assim que encontrar uma k-árvore.

Algoritmo kMST-Kruskal (V, E, c, k)

```
1  suponha c(e_1) ≤ c(e_2) ≤ ··· ≤ c(e_m), onde {e_1, ..., e_m} = E
2  F ← (V, ∅)
3  i ← 1
4  enquanto F não tem componente com pelo menos k vértices faça
5      se F + e_i não forma circuito então F ← F + e_i
6      i ← i + 1
7  devolva o componente T_0 de F que tem pelo menos k vértices
```

Podemos supor sem perda de generalidade que T_0 tem exatamente k vértices, caso contrário podemos remover folhas da árvore, sem aumentar o custo de T_0, até que T_0

tenha exatamente k vértices.

Teorema 2. O fator de aproximação do algoritmo kMST-Kruskal é $k-1$.

Teorema 3. Seja T^* uma k-árvore mínima com custo OPT $:= c(T^*)$. Suponha que ao final do algoritmo os vértices de T^* estão contidos em $t+1>1$ componentes de F. Então $c(T_0) \leq \frac{k-1}{t}\text{OPT}$.

O teorema 3 implica que quanto mais espalhados pelos componentes de F estiver os vértices de uma k-árvore mínima, melhor é o candidato a solução encontrado pelo algoritmo. No entanto, se os vértices de uma k-árvore mínima estiverem concentrados em poucos componentes de F, então o candidato a solução não será muito bom.

5.2 Algoritmo RSMRR

Ramamurthy Ravi, Ravi Sundaram, Madhav V. Marathe, Daniel J. Rosenkrantz e Sekharipuram S. Ravi [15] desenvolveram um algoritmo de aproximação para o kMST baseado no algoritmo kMST-Kruskal. A idéia do algoritmo é tentar encontrar uma k-árvore a cada iteração do kMST-Kruskal e no final devolver a mais barata delas.

Algoritmo RSMRR (V, E, c, k)

1 suponha $c(e_1) \leq c(e_2) \leq \cdots \leq c(e_m)$, onde $\{e_1, \ldots, e_m\} = E$
2 $F \leftarrow (V, \emptyset)$
3 $i \leftarrow 0$
4 enquanto F não tem componente com pelo menos k vértices faça
5 $\quad i \leftarrow i + 1$
6 $\quad$ se $F + e_i$ não forma circuito então
7 $\quad\quad F \leftarrow F + e_i$
8 $\quad\quad$ obtenho a melhor k-árvore T_i, conectando os componentes de F
9 $T_0 \leftarrow$ componente de F que tem pelo menos k vértices
10 dentre as k-árvores T_0, T_1, ..., T_i, devolva a de menor custo

A linha 8 do algoritmo acima funciona da seguinte forma. A partir de cada componente C de F executamos um algoritmo de caminhos mínimos para conectar os componentes. Seja dC a menor distância tal que, dentre os componentes que estão a uma distância menor ou igual a dC, existe um conjunto de no máximo $\sqrt{k}$ componentes com a quantidade de vértices somando pelo menos k. Conecte esses componentes para obter uma k-árvore, removendo arestas incidentes a folhas para que a k-árvore tenha exatamente k vértices. T_i será a k-árvore obtida dessa forma para o componente C com menor dC. Se dC não estiver bem definido na iteração atual, marque que T_i tem custo infinito.

Teorema 4. O fator de aproximação do algoritmo RSMRR é $2\sqrt{k}$.

6 Algoritmos primais-duais

Os algoritmos de aproximação com fator de aproximação constante, conhecidos atualmente, são primais-duais. Eles usam como subrotina um algoritmo de aproximação para o problema da árvore de Steiner com coleta de prêmios (*prize-collecting Steiner tree*), ou simplesmente (PCST). Esse algoritmo foi desenvolvido por Michel X. Goemans e David P. Williamson [9, 10].

6.1 Árvore de Steiner com coleta de prêmios

O problema PCST é o seguinte.

Problema PCST(V, E, c, π, r): *dado um grafo conexo (V, E), custos não-negativo c_e para cada aresta e, penalidades não-negativa π_v para cada vértice v e um vértice r, encontrar uma árvore que minimize o custo das arestas mais as penalidades dos vértices fora da árvore.*

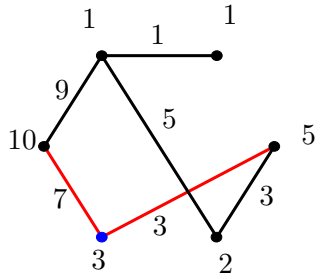

Figura 4: Exemplo de solução para o PCST. Vértice azul é r.

Uma observação interessante é que quanto maior o valor da penalidade dos vértices mais vértices terá a solução. Em particular, se a penalidade nos vértices for 0, então a solução será a árvore que contém apenas r. Se as penalidades forem suficientemente grandes (por exemplo, maiores que a soma dos custos de todas as arestas), então a solução será um árvore geradora mínima.

Uma formulação como programa inteiro para o PCST é dada a seguir. Seja c o vetor que representa os custos das arestas e π o vetor que representa as penalidades dos vértices. Tomaremos como variáveis os vetores x indexado por E e z indexado por 2^V. Queremos que x_e, para e em E, tenha valor 1 se e pertencer a solução e 0 caso contrário. E z_S tem valor 1 se S for o subconjunto de V contendo todos os vértices que

não fazem parte da solução e 0 caso contrário. Para um subconjunto S de V, usaremos as seguintes notações $\pi(S) = \sum_{v \in S} \pi_v$ e $\delta(S)$ é o corte de S.

$$\begin{array}{lllll} \text{minimize} & cx + \sum_{S \subseteq V \setminus \{r\}} \pi(S) z_S & & & \\ \text{sujeito a} & \sum_{e \in \delta(S)} x_e + \sum_{X : S \subseteq X} z_X & \geq & 1 & \text{para cada } S \subseteq V \setminus \{r\} \\ & x_e & \in & \{0,1\} & \text{para cada } e \in E \\ & z_S & \in & \{0,1\} & \text{para cada } S \subseteq V \setminus \{r\} \end{array} \tag{1}$$

O dual para a relaxação linear de (1) é

$$\begin{array}{lllll} \text{minimize} & \sum_{S \subseteq V \setminus \{r\}} y_S & & & \\ \text{sujeito a} & \sum_{S : e \in \delta(S)} y_S & \leq & c_e & \text{para cada } e \in E \\ & \sum_{X : X \subseteq S} y_X & \leq & \pi(S) & \text{para cada } S \subseteq V \setminus \{r\} \\ & y_S & \geq & 0 & \text{para cada } S \subseteq V \setminus \{r\}. \end{array} \tag{2}$$

Um algoritmo para o PCST que denotaremos por $\mathrm{GW}(V, E, c, \pi, r)$ ou simplesmente GW foi criado por Goemans e Williamson por volta de 1995. Ele devolve (T, A), onde $T = (V_T, E_T)$ é um candidato a solução do PCST e A é o conjunto dos vértices que não fazem parte de T. Também é devolvido candidato a solução y para o dual (2). Esse algoritmo não será muito discutido neste texto, sendo usado como "caixa preta".

Pela dualidade fraca e o fato de y ser um candidato a solução de (2), sabemos que $\sum_{S \subseteq V \setminus \{r\}} y_S$ é um limitante inferior para o valor da solução do PCST.

Teorema 5. (Goemans e Williamson [9]) Os candidatos a solução para o PCST $((V_T, E_T), A)$ e, para o seu dual, y devolvidos pelo algoritmo $\mathrm{GW}(V, E, c, \pi, r)$ satisfazem a seguinte relação.

$$\sum_{e \in E_T} c_e + \left(2 - \frac{1}{|V| - 1}\right) \pi(A) \leq \left(2 - \frac{1}{|V| - 1}\right) \sum_{S \subseteq V \setminus \{r\}} y_S$$

Corolário 6. Suponha que $\pi_v = \lambda$ para todo $v \in V$, onde $\lambda \geq 0$ é uma constante. Então para $((V_T, E_T), A)$ e y obtidos de $\mathrm{GW}(V, E, c, \pi, r)$, vale o seguinte.

$$\sum_{e \in E_T} c_e + 2|A|\lambda \leq 2 \sum_{S \subseteq V \setminus \{r\}} y_S$$

6.2 Uma 5-aproximação para o kMST

A 5-aproximação obtida por Naveen Garg [7] usa o algoritmo GW como sub-rotina. A idéia de usar um algoritmo para o PCST está no fato das relaxações lineares dos programas inteiros para o kMST e para o PCST serem similares.

Uma formulação do problema kMST(V, E, c, k, r) como programa linear inteiro é a seguinte. Considere c como um vetor indexado por E. Tomaremos como variáveis os vetores x indexado por E e z indexado por 2^V. Queremos que x_e, para e em E, tenha valor 1 se e pertencer a solução e 0 caso contrário. E z_S tem valor 1 se S for o subconjunto de V contendo todos os vértices que não fazem parte da solução e 0 caso contrário. A formulação abaixo é uma relaxação linear do programa inteiro.

$$\begin{array}{lrcll} \text{minimize} & cx & & & \\ \text{sujeito a} & \sum_{e\in\delta(S)} x_e + \sum_{X:S\subseteq X} z_X & \geq & 1 & \text{para cada } S \subseteq V \setminus \{r\} \\ & \sum_{S:S\subseteq V\setminus\{r\}} |S| z_S & \leq & n-k & \\ & x_e & \geq & 0 & \text{para cada } e \in E \\ & z_S & \geq & 0 & \text{para cada } S \subseteq V \setminus \{r\} \end{array} \tag{3}$$

Na formulação acima o primeiro tipo de restrição serve para garantir que todo candidato a solução seja uma árvore que contém r. O segundo tipo de restrição garante que todo candidato a solução tem pelo menos k vértices. Os dois últimos servem para garantir que o valor atribuido às variáveis sejam inteiros, mais especificamente, sejam 0 ou 1. Note que essa formulação é apenas uma possível formulação. Existem outras formulações possíveis e igualmente corretas.

A relaxação linera para o PCST é a seguinte.

$$\begin{array}{lrcll} \text{minimize} & cx + \sum_{S\subseteq V\setminus\{r\}} \pi(S) z_S & & & \\ \text{sujeito a} & \sum_{e\in\delta(S)} x_e + \sum_{X:S\subseteq X} z_X & \geq & 1 & \text{para cada } S \subseteq V \setminus \{r\} \\ & x_e & \geq & 0 & \text{para cada } e \in E \\ & z_S & \geq & 0 & \text{para cada } S \subseteq V \setminus \{r\} \end{array} \tag{4}$$

Note que (3) e (4) diferem apenas na função objetivo e em uma restrição a mais que aparece na formulação do kMST. Dessa forma, se usarmos a relaxação lagrangeana nessa restrição obtemos (5).

$$\begin{array}{lrcll} \text{minimize} & cx + \lambda\big(\sum_{S:S\subseteq V\setminus\{r\}} |S| z_S & - & (n-k)\big) & \\ \text{sujeito a} & \sum_{e\in\delta(S)} x_e + \sum_{X:S\subseteq X} z_X & \geq & 1 & \text{para cada } S \subseteq V \setminus \{r\} \\ & x_e & \geq & 0 & \text{para cada } e \in E \\ & z_S & \geq & 0 & \text{para cada } S \subseteq V \setminus \{r\} \end{array} \tag{5}$$

Agora (4) e (5) diferem apenas na função objetivo, tendo o mesmo conjunto de candidatos a solução. Logo, queremos interpretar o candidato a solução T para o PCST devolvido pelo algoritmo GW, como um candidato a solução para o kMST. O algoritmo GW também devolve um candidato a solução y para o dual, do qual podemos extrair um limitante inferior para o valor de uma solução para o kMST.

Para entender isso melhor, considere o dual de (5).

$$\begin{array}{lrcll} \text{maximize} & \sum_{S\subseteq V\setminus\{r\}} y_s & - & (n-k)\lambda & \\ \text{sujeito a} & \sum_{S:e\in\delta(S)} y_S & \leq & c_e & \text{para cada } e \text{ em } E \\ & \sum_{X:X\subseteq S} y_X & \leq & |S|\lambda & \text{para cada } S\subseteq V\setminus\{r\} \\ & y_S & \geq & 0 & \text{para cada } S\subseteq V\setminus\{r\} \end{array} \tag{6}$$

Considerando $\pi_v = \lambda$ para todo vértice v, então (2) e (6) também diferem apenas na função objetivo. Logo o y devolvido por GW, também é um candidato a solução de (6). Pela dualidade fraca, sabemos que o valor do candidato a solução y não será maior do que o valor de uma solução do kMST.

Sejam (V, E, c, k, r) uma instância do kMST e OPT o valor de uma solução para essa instância. Denotaremos por n a cardinalidade de V e por $(\lambda)_V$ um vetor indexado por V com o valor constante λ em todas as suas entradas.

Lema 7. Sejam $((V_T, E_T), A)$ e y os candidatos a solução devolvidos por $\mathrm{GW}(V, E, c, (\lambda)_V, r)$. Em relação ao kMST temos que

$$\sum_{e\in E_T} c_e + 2\lambda(|A| - (n-k)) \leq 2\left(\sum_{S\subseteq V\setminus\{r\}} y_S - \lambda(n-k)\right) \leq 2\mathrm{OPT}.$$

Se o $((V_T, E_T), A)$ devolvido por GW for tal que $|A| = n - k$, isto é, V_T possuir exatamente k vértice, então o lema 7 garante que (V_T, E_T) é uma 2-aproximação para o kMST. No entanto, se $|A| < n - k$ o lema 7 não fornece nenhuma informação sobre limitantes. E o caso $|A| > n - k$ não ocorre, pois (V_T, E_T) é um candidato a solução, logo $|A| \leq n - k$.

Baseando-se nessas idéias e no fato de que as penalidades nos vértices influenciam a quantidade de vértices de uma solução do PCST, o algoritmo proposto por Garg tenta encontrar um valor suficientemente bom para as penalidades de forma a garantir que o candidato a solução seja uma 5-aproximação.

Sejam (V, E, c, k, r) uma instância do kMST e OPT o valor de uma solução para essa instância. Denotaremos por n a cardinalidade de V e por $(\lambda)_V$ um vetor indexado por V com o valor constante λ em todas as suas entradas. Para facilitar a explicação do algoritmo, vamos supor algumas hipóteses sobre a instância do kMST, sem perda de generalidade.

1. Os custos das arestas satisfazem a desigualdade triangular.

2. A distância entre um vértice v qualquer e r é no máximo OPT.

3. OPT$\geq c_{\min}$, onde $c_{\min}$ é o menor custo não zero das aresta em E.

Como dito anteriormente o algoritmo tentará encontrar um valor λ suficientemente bom para as penalidades dos vértices e através do algoritmo $\mathrm{GW}(V, E, c, (\lambda)_V, r)$ encontra uma k-árvore. A busca é feita usando a idéia da busca binária com várias chamadas ao algoritmo $\mathrm{GW}(V, E, c, (\lambda)_V, r)$, cada vez com uma constante λ diferente para as penalidades.

Como observado na seção 6.1, se tomarmos $\lambda_1 = 0$ como penalidade para cada vértice, a solução para o PCST será a árvore $(\{r\}, \emptyset)$. Se tomarmos $\lambda_2 = \sum_{e \in E} c_e$ como penalidade para cada vértice, a solução será uma árvore geradora mínima. Portanto a busca pelo valor de λ começará no intervalo $[\lambda_1, \lambda_2]$.

A condição de parada para a busca é satisfeita quando encontramos uma árvore com exatamente k vértices ou obtemos $\lambda_1 < \lambda_2$ tal que

- $\lambda_2 - \lambda_1 \leq \frac{c_{min}}{2n(2n+1)}$ e
- para $i = 1, 2$, o algoritmo $\mathrm{GW}(V, E, c, (\lambda_i)_V, r)$ devolve (T_i, A_i) e y^i tal que T_i tem exatamente k_i vértices e $k_1 < k < k_2$.

Se encontramos uma árvore com exatamente k vértices, então o lema 7 garante que tal árvore é uma 2-aproximação para o kMST. Se a busca parar sem encontrar uma árvore com exatamente k vértices, então o algoritmo combinará (T_1, A_1, y^1) e (T_2, A_2, y^2) para obter uma 5-aproximação, onde (T_i, A_i, y^i) são os candidatos a solução devolvidos por $\mathrm{GW}(V, E, c, (\lambda_i)_V, r)$.

Algoritmo GARG5(V, E, c, k, r)

```
1   λ1 ← 0      λ2 ← Σ c_e
2   enquanto a condição de parada não é satisfeita faça
3       λ ← (λ1+λ2)/2
4       (T, A, y) ← GW(V, E, c, (λ)_V, r)
5       se T tem exatamente k vértices então devolve T
6       se T tem mais do que k vértices então λ2 ← λ
7       senão λ1 ← λ
8   (T1, A1, y^1) ← GW(V, E, c, (λ1)_V, r)
9   (T2, A2, y^2) ← GW(V, E, c, (λ2)_V, r)
10  devolve combinação(T1, T2)
```

Figura 5: Pseudo-código do algoritmo de Garg.

A combinação de T_1 com T_2 funciona da seguinte maneira. Sejam k_1 e k_2 o número de vértices de T_1 e T_2 respectivamente. Do algoritmo GARG5(V, E, c, k, r) sabemos que $k_1 < k < k_2$. Se sabemos que T_2 é uma 5-aproximação para o kMST, então podemos devolvê-la. Caso contrário, adicionamos alguns vértices de T_2 e T_1 para obter uma k-árvore. Para isso, primeiro duplicamos as arestas de T_2 (6). Dessa forma todos os vértices de T_2 têm grau par e, assim, T_2 têm uma trilha euleriana fechada. Usando a técnica dos atalhos (*shortcuts*), bastante usada em algoritmos de aproximação para o problema do caixeiro viajante [16], podemos reduzir T_2 a um circuito contendo apenas os vértices de T_2 que não estão em T_1. Seja P um caminho nesse circuito com $k - k_1$ vértices e com o menor custo possível. Conectamos P a r através de um caminho de menor custo entre r e algum vértice de P. Assim, obtemos uma árvore com exatamente k vértices.

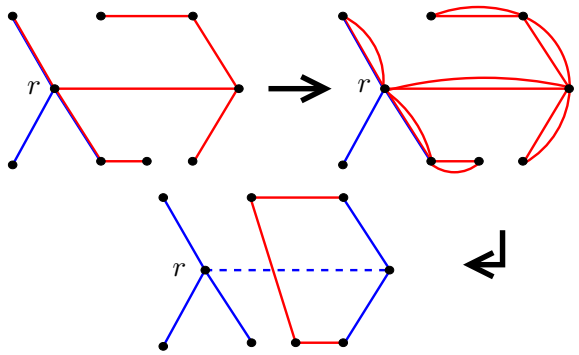

Figura 6: Ilustração da combinação de árvores. Arestas azuis representam T_1, arestas vermelhas representam T_2 e $k = 7$.

Teorema 8. O fator de aproximação do algoritmo GARG5 é 5.

7 Projeto de mestrado

Neste projeto de mestrado pretendemos estudar vários algoritmos de aproximação para o problema kMST e tentar reescrevê-los de maneira mais simples. Também tentaremos reescrevê-los de forma unificada, ou seja, deixá-los o mais semelhantes possível para destacar onde eles diferem.

7.1 Tópicos estudados

Comecei os estudos pela 5-aproximação de Garg por um artigo de Fabián A. Chudak, Tim Roughgarden e David P. Williamson [5]. Apresentei um seminário sobre esse

algoritmo na disciplina MAC5727 - Algoritmos de Aproximação. Também estudei um pouco sobre o problema PCST.

Em seguida passei a estudar a 2-aproximação de Garg [8], mas achei o algoritmo muito complicado e resolvi deixar para terminar o estudo desse algoritmo mais tarde.

Passei a estudar a $\sqrt{k}$-aproximação de Ravi, Sundaram, Marathe, Rosenkrantz e Sekharipuram [15]. Apresentei um seminário sobre esse algoritmo na disciplina MAC5700 - Seminários em Ciência da Computação.

Estudei também um algoritmo polinomial para o kMST em árvores [12].

Atualmente estou estudando a $O(\log^2)$-aproximação de Awerbuch, Azar, Blum e Vempala [2].

7.2 Estrutura da dissertação

Inicialmente pretendemos que a dissertação siga uma estrutura semelhante a deste projeto, apresentando os seguintes capítulos.

Introdução

Neste capítulo, basicamente, descreveremos o problema kMST.

Preliminares

Aqui pretendemos apresentar alguns conceitos básicos como grafos, árvores, programação linear. Também apresentaremos algumas terminologias e notações que aparecerão frequentemente pela dissertação. Alguns resultados básicos também podem ser incluidos.

Complexidade

Neste capítulo discutiremos a complexidade computacional do kMST, provando que ele é um problema NP-difícil.

Algoritmos exatos para casos particulares

Mostraremos neste capítulo alguns casos em que o kMST pode ser resolvido em tempo polinomial.

Algoritmos de aproximação

Apresentaremos o conceito de algoritmos de aproximação e algumas terminologias e notações.

Algoritmos combinatórios

O primeiros algoritmos de aproximação são baseados no algoritmo de Kruskal. Neste capítulo descreveremos os algoritmos de Ramamurthy Ravi, Ravi Sundaram, Madhav V. Marathe, Daniel J. Rosenkrantz e Sekharipuram S. Ravi [15] e de Baruch Awerbuch, Yossi Azar, Avrim Blum e Santosh Vempala [2].

Algoritmos primais-duais

Apresentaremos aqui alguns dos algoritmos primais duais que utilizam o algoritmo de Michel X. Goemans e David P. Williamson [9, 10] para o PCST. Dentre esses algoritmos estão o de Avrim Blum, Ramamurthy Ravi e Santosh Vempala [3], de Naveen Garg [7, 8] e de Sanjeev Arora e George Karakostas [1].

7.3 Planejamento

Atividade	AGO	SET	OUT	NOV	DEZ	JAN	FEV
[2]	√						
[3]	√	√					
[8]			√	√	√		
escrever e revisar dissertação	√	√	√	√	√	√	√

Referências

[1] Sanjeev Arora e George Karakostas, A $2 + \varepsilon$ approximation algorithm for the k-MST problem, *SODA '00: Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms* (Philadelphia, PA, USA), Society for Industrial and Applied Mathematics, 2000, pp. 754–759.

[2] Baruch Awerbuch, Yossi Azar, Avrim Blum e Santosh Vempala, New approximation guarantees for minimum-weight k-trees and prize-collecting salesmen, *SIAM Journal on Computing* **28** (1998), no. 1, 254–262.

[3] Avrim Blum, Ramamurthy Ravi e Santosh Vempala, A constant-factor approximation algorithm for the k-MST problem, *J. Comput. Syst. Sci.* **58** (1999), no. 1, 101–108.

[4] Christian Blum, Revisiting dynamic programming for finding optimal subtrees in trees, *European Journal of Operational Research* **177** (2007), no. 1, 102–115.

[5] Fabián Ariel Chudak, Tim Roughgarden e David Paul Williamson, Approximate k-MSTs and k-Steiner trees via the primal-dual method and Lagrangean relaxation, *Math. Program.* **100** (2004), no. 2, 411–421.

[6] Michael Randolph Garey e David Stifler Johnson, *Computers and Intractability: A guide to the theory of NP-completeness*, W. H. Freeman & Co., New York, NY, USA, 1990.

[7] Naveen Garg, A 3-approximation for the minimum tree spanning k vertices, *FOCS '96: Proceedings of the 37th Annual Symposium on Foundations of Computer Science* (Washington, DC, USA), IEEE Computer Society, 1996, pp. 302–309.

[8] ———, Saving an epsilon: a 2-approximation for the k-MST problem in graphs, *STOC '05: Proceedings of the thirty-seventh annual ACM symposium on Theory of computing* (New York, NY, USA), ACM Press, 2005, pp. 396–402.

[9] Michel Xavier Goemans e David Paul Williamson, A general approximation technique for constrained forest problems, *SODA '92: Proceedings of the third annual ACM-SIAM symposium on Discrete algorithms* (Philadelphia, PA, USA), Society for Industrial and Applied Mathematics, 1992, pp. 307–316.

[10] ———, The primal-dual method for approximation algorithms and its application to network design problems, In *Approximation algorithms for NP-hard problems* (Boston, MA, USA), PWS Publishing Co., 1997, pp. 144–191.

[11] Joseph Bernard Kruskal Jr., On the shortest spanning subtree of a graph and the traveling salesman problem, *Proceedings of the American Mathematical Society* **7** (1956), 48–50.

[12] Francesco Maffioli, *Finding a best subtree of a tree*, Tech. Report 91.041, Politecnico di Milano, Dipartimento di Elettronica, Itália, 1991.

[13] Robert Clay Prim, Shortest connection networks and some generalizations, *Bell System Technical Journal* **36** (1957), 1389–1401.

[14] Sridhar Rajagopalan e Vijay V. Vazirani, Logarithmic approximation of minimum weight k trees, *Manuscrito não publicado* (1995).

[15] Ramamurthy Ravi, Ravi Sundaram, Madhav Vishnu Marathe, Daniel J. Rosenkrantz e Sekharipuram S. Ravi, Spanning trees short or small, *SODA '94: Proceedings of the fifth annual ACM-SIAM symposium on Discrete algorithms* (Philadelphia, PA, USA), Society for Industrial and Applied Mathematics, 1994, pp. 546–555.

[16] Vijay V. Vazirani, *Approximation algorithms*, Springer-Verlag New York, Inc., New York, NY, USA, 2001.