K-corte mínimo

Tiago Montalvão

Universidade Federal do Rio de Janeiro

13 de junho de 2017

Sumário

UFRJ

Problema

Formulação 1

Dado um grafo ponderado $G = (V, E)$ e um inteiro $k = 2, 3, \ldots, |V(G)|$, encontre um subconjunto $S \subseteq E$ de arestas de tal forma que o grafo $G' = (V, E \setminus S)$ tenha exatamente k componentes conexas e S tenha custo mínimo.

UFRJ

Problema

Formulação 2

Dado um grafo ponderado $G = (V, E)$ e um inteiro $k \in \{2, 3, \dots, |V(G)|\}$, particione os vértices em k conjuntos $(C_1, C_2, \dots, C_k)$, de tal forma a minimizar

$$\sum_{i=1}^{k-1} \sum_{j=i+1}^{k} \sum_{\substack{u \in C_i \\ v \in C_j}} w(u, v),$$

sendo $w(u, v)$ o peso da aresta (u, v).

Problema

Formulação 3

Dado um grafo ponderado $G = (V, E)$ e um inteiro $k \in \{2, 3, \ldots, |V(G)|\}$, particione os vértices em k conjuntos $(C_1, C_2, \ldots, C_k)$, de tal forma a maximizar

$$\sum_{i=1}^{k} \sum_{u \in C_i} \sum_{\substack{v \in C_i \\ v \neq u}} w(u, v),$$

sendo $w(u, v)$ o peso da aresta (u, v).

UFRJ

Exemplo

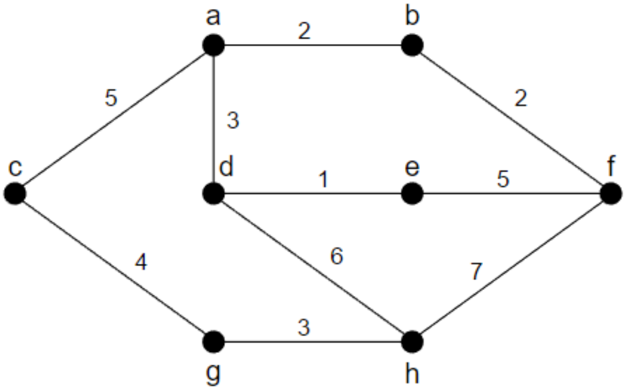

Figura: Grafo de exemplo. Suponha k = 3.

Exemplo

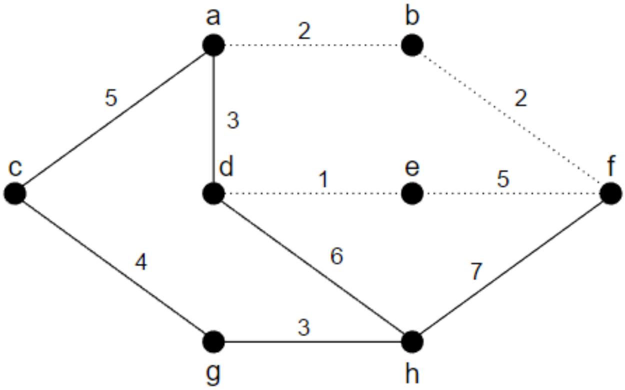

Figura: Grafo de exemplo. Suponha $k = 3$.

Exemplo

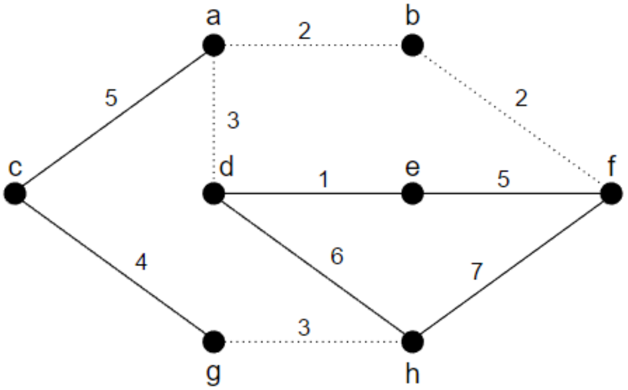

Figura: Grafo de exemplo. Suponha k = 3.

Implementações

- C++11, apenas com as bibliotecas padrão e a flag de compilação -O2.
- Aproximadamente 350 linhas cada
- Implementações em `github.com/tiagomontalvao/minkcut`

UFRJ

Algoritmo exato

Branch and Bound

- bestCost $\leftarrow \infty$
- lower, upper $\leftarrow$ calculaLimites()
- $H \leftarrow \{$(lower, upper, cost = 0, last = -1, classEdges)$\}$
- Enquanto $H \neq \emptyset$:
  - node $\leftarrow$ extrai(H)
  - se solucao(node):
    - se node.cost $<$ bestCost e node.cntComponents = k: bestCost $\leftarrow$ node.cost
  - senão se node.cntComponents $>$ k ou node.lower $\geq$ bestCost:
    - continua
  - senão se node.lower = node.upper:
    - se node.cost $<$ bestCost e node.cntComponents = k: bestCost $\leftarrow$ node.cost

UFRJ

Algoritmo exato

Branch and Bound

- senão:
  - nextNode $\leftarrow$ fixaAresta(node)
  - se nextNode.lower $<$ bestCost:
    - $H \leftarrow H \cup \{nextNode\}$
  - nextNode $\leftarrow$ removeAresta(node)
  - se nextNode.lower $<$ bestCost:
    - $H \leftarrow H \cup \{nextNode\}$

Algoritmo exato

calculaLimites(node, classEdges)

- C $\leftarrow$ nConnectedComponents(classEdges)
- lower $\leftarrow$ node.cost + S, sendo S a soma das $k - C$ arestas mais baratas disponíveis
- upper $\leftarrow$ greedy(classEdges)

UFRJ

Algoritmo aproximativo

Implementação do GRASP

- $f(s^*) \leftarrow \infty$
- $Elite \leftarrow \{\}$
- Para k = 1,2,. . . , MaxIterações:
  - Solução gulosa aleatória s com ideia parecida ao Prim
  - Aplicação de busca local para obtenção de s', tendo $f(s') < f(s)$
  - Uso de path relinking a partir da metade das iterações para obtenção de uma possível solução melhor s''
  - $s^* \leftarrow \text{argmin}\{f(s^*), f(s'), f(s'')\}$ e atualiza *Elite*

UFRJ

Algoritmo guloso

Guloso aleatório

- $K \subseteq V$ com k vértices aleatórios
- Criação da lista de vizinhos LRC dos vértices já explorados, levando em conta a qualidade de cada um em relação ao vizinho mais caro
- Seleção de um vizinho aleatório

UFRJ

Busca local

A busca local recebe como entrada uma solução do algoritmo guloso anterior:

Busca local

- Para $u \in V$, percorridos em ordem aleatória:
  - Se u está sozinho em uma componente conexa, **continua**
  - Senão, verifica todas as componentes conexas e transfere para a que causar uma maior diminuição no custo total

UFRJ

Path relinking

- Uso do conjunto *Elite* para escolha aleatória de uma boa solução anterior.
- Utilização do *path relinking* após metade das iterações do GRASP.
- Começa da melhor solução para a pior e vai mudando arestas aleatórias. Das configurações válidas (grafo com k componentes conexas), é guardada a melhor e verificado se ela é boa o suficiente para entrar no conjunto *Elite*.

UFRJ

Resultados

$n = 10, m = 27, 1 \leq c_e \leq 30$

k	BnB	GRASP	s^*_{GRASP}/s^*_{BnB}
2	0m0.642s	**0m0.052s**	1
3	0m6.160s	**0m0.057s**	1
4	0m34.460s	**0m0.056s**	1
5	>10m	**0m0.052s**	-

UFRJ

Resultados

$n = 20, m = 42, 1 \leq c_e \leq 30$

k	BnB	GRASP	s^*_{GRASP}/s^*_{BnB}
2	**0m0.101s**	0m0.129s	1
3	0m1.495s	**0m0.137s**	1
4	0m25.544s	**0m0.134s**	1
5	>10m	**0m0.140s**	-

UFRJ

Resultados

$n = 30, m = 58, 1 \leq c_e \leq 30$

k	BnB	GRASP	s^*_{GRASP} / s^*_{BnB}
2	0m1.904s	**0m0.202s**	1
3	>10m	**0m0.211s**	-
4	>10m	**0m0.235s**	-
5	>10m	**0m0.234s**	-

UFRJ

Resultados

$n = 512, m = 39373, 1 \leq c_e \leq 50, k = 2$

GRASP		Execução 1		Execução 2		Execução 3	
α	Elite	Custo	Tempo	Custo	Tempo	Custo	Tempo
1	10	3130	7.990s	**3087**	7.939s	3260	7.967s
2	10	**5667**	17.666s	5800	17.213s	5819	19.355s
3	10	**6362**	19.461s	6500	19.939s	6676	18.501s
1	25	**3130**	8.806s	3326	8.690s	**3130**	9.608s
1	40	3159	9.550s	**3130**	9.490s	3333	10.671s

UFRJ

Resultados

$n = 512, m = 39373, 1 \leq c_e \leq 50, k = 5$

GRASP		Execução 1		Execução 2		Execução 3	
α	Elite	Custo	Tempo	Custo	Tempo	Custo	Tempo
1	10	19628	41.576s	**18572**	41.338s	18923	40.500s
2	10	22274	44.070s	**17470**	41.758s	21845	43.319s
3	10	26669	46.826s	**21695**	42.486s	22532	41.701s
1	25	20339	48.924s	**18567**	43.289s	19758	46.621s
1	40	19621	47.598s	18938	48.367s	**17900**	51.389s

UFRJ

Referências

EEL857, Otimização em Grafos, UFRJ, Prof. Luidi Simonetti
http://cos.ufrj.br/luidi/eel857/otim_grafo.html