Multiway Cut, k-Cut, and k-Center[1]

[1] Chapter 4 of Approximation Algorithm, Vijay V. Vazirani, 2003

Multiway Cut

Definition (Cut)

Given a connected, undirected graph $G = (V, E)$ with an assignment of weights to edges, $w : E \to \mathcal{R}^+$, a *cut* is defined by a partition of V into two sets, say V' and $V - V'$, and consists of all edges that have one endpoint in each partition.

Clearly, the removal of the cut from G disconnects G.

Definition (s-t cut)

Given terminals $s, t \in V$, consider a partition of V that separates s and t. The cut defined by such a partition will be called an *s-t cut*.

The problems of finding a minimum weight cut and a minimum weight st cut can be efficiently solved using a maximum flow algorithm.

Problem (Multiway cut)

Given a set of terminals $S = \{s_1, s_2, \ldots, s_k\} \subseteq V$, a multiway cut is a set of edges whose removal disconnects the terminals from each other. The multiway cut problem asks for the minimum weight such set.

Problem (Minimum k-cut)

A set of edges whose removal leaves k connected components is called a k-cut. The k-cut problem asks for a minimum weight k-cut.

Multiway Cut and k-Cut

Problem (Multiway cut)

Given a set of terminals $S = \{s_1, s_2, \ldots, s_k\} \subseteq V$, a multiway cut is a set of edges whose removal disconnects the terminals from each other. The multiway cut problem asks for the minimum weight such set.

Remark

The problem of finding a minimum weight multiway cut is NP-hard for any fixed $k \geq 3$. Observe that the case $k = 2$ is precisely the minimum s-t cut problem. The minimum k-cut problem is polynomial time solvable for fixed k; however, it is NP-hard if k is specified as part of the input.

In this following, we will obtain factor $2 - \frac{2}{k}$ approximation algorithms for both problems.

Definition (Isolating cut)

Define an *isolating cut* for s_i to be a set of edges whose removal disconnects s_i from the rest of the terminals.

Algorithm 0.1: $\textsc{Multiway cut}(G)$

for $i = 1, \ldots, k$
$\{$compute a minimum weight isolating cut for s_i, say C_i.
Discard the heaviest of these cuts, and output the union of the rest, say C.

Multiway Cut

Each computation in step 1 can be accomplished by identifying the terminals in $S - \{s_i\}$ into a single node, and finding a minimum cut separating this node from s_i; this takes one max-flow computation. Clearly, removing C from the graph disconnects every pair of terminals, and so is a multiway cut.

Theorem

The above algorithm achieves an approximation guarantee of $2 - \frac{2}{k}$.

Proof.

Let A be an optimal multiway cut in G. We can view A as the union of k cuts as follows:

- The removal of A from G will create k connected components, each having one terminal (since A is a minimum weight multiway cut, no more than k components will be created). Let A_i be the cut separating the component containing s_i from the rest of the graph. Then $A = \bigcup_{i=1}^{k} A_i$.

Since each edge of A is incident at two of these components, each edge will be in two of the cuts A_i. Hence,

$$\sum_{i=1}^{k} w(A_i) = 2w(A).$$

□

Multiway Cut

Theorem

The above algorithm achieves an approximation guarantee of $2 - \frac{2}{k}$.

Proof.

Let A be an optimal multiway cut in G. Since each edge of A is incident at two of these components, each edge will be in two of the cuts A_i. Hence,

$$\sum_{i=1}^{k} w(A_i) = 2w(A).$$

Clearly, A_i is an isolating cut for s_i. Since C_i is a minimum weight isolating cut for s_i, $w(C_i) \leq w(A_i)$. Notice that this already gives a factor 2 algorithm, by taking the union of all k cuts C_i. Finally, since C is obtained by discarding the heaviest of the cuts C_i,

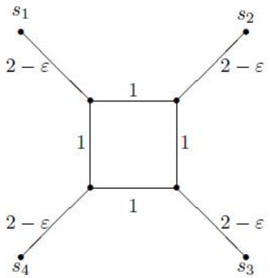
$$w(C) \leq \left(1 - \frac{1}{k}\right) \sum_{i=1}^{k} w(C_i) \leq \left(1 - \frac{1}{k}\right) \sum_{i=1}^{k} w(A_i) = 2\left(1 - \frac{1}{k}\right) w(A).$$

□

Multiway Cut

Example

A tight example for this algorithm is given by a graph on $2k$ vertices consisting of a k-cycle and a distinct terminal attached to each vertex of the cycle. The edges of the cycle have weight 1 and edges attaching terminals to the cycle have weight $2 - \epsilon$ for a small fraction $\epsilon > 0$.
For example, the graph corresponding to $k = 4$ is:



For each terminal s_i, the minimum weight isolating cuts for s_i is given by the edge incident to s_i. So, the cut C returned by the algorithm has weight $(k-1)(2-\epsilon)$. On the other hand, the optimal multiway cut is given by the cycle edges, and has weight k.

Minimum k-Cut

Problem (Minimum k-cut)

A set of edges whose removal leaves k connected components is called a k-cut. The k-cut problem asks for a minimum weight k-cut.

A natural algorithm for finding a k-cut is as follows.

Algorithm 0.2: GREEDY APPROACH FOR k-CUT(G)

repeat
  for $i = 1, \ldots, k$
  $\begin{cases} \text{compute a minimum cut in each connected component,} \\ \text{remove the lightest one.} \end{cases}$
until until there are k connected components.

This algorithm does achieve a guarantee of $2 - \frac{2}{k}$.

Remark

We will use the Gomory-Hu tree representation of minimum cuts to give a simpler algorithm achieving the same guarantee.

Definition (Gomory-Hu tree)

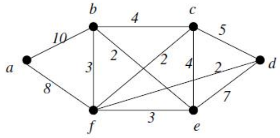
Let T be a tree on vertex set V; the edges of T need not be in E. Let e be an edge in T. Its removal from T creates two connected components. Let S and $\bar{S}$ be the vertex sets of these components. The cut defined in graph G by the partition $(S, \bar{S})$ is the *cut associated with e in G*. Define a weight function w' on the edges of T. Tree T will be said to be a Gomory-Hu tree for G if

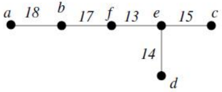
1. for each pair of vertices $u, v \in V$, the weight of a minimum u-v cut in G is the same as that in T.
2. for each edge $e \in T$, $w'(e)$ is the weight of the cut associated with e in G.

A Gomory-Hu tree encodes, in a succinct manner, a minimum u-v cut in G, for each pair of vertices $u, v \in V$ as follows. A minimum u-v cut in T is given by a minimum weight edge on the unique path from u to v in T, say e. By the properties stated above, the cut associated with e in G is a minimum u-v cut, and has weight $w'(e)$. So, for the $\binom{n}{2}$ pairs of vertices $u, v \in V$, we need only $n-1$ cuts, those encoded by the edges of a Gomory-Hu tree, to give minimum u-v cuts in G.

Minimum k-Cut

Example





Lemma

Let S be the union of cuts in G associated with l edges of T. Then, the removal of S from G leaves a graph with at least $l+1$ components.

Proof.

Removing the corresponding l edges from T leaves exactly $l+1$ connected components, say with vertex sets $V_1,\ V_2,\ \ldots,\ V_l+1$. Clearly, removing S from G will disconnect each pair V_i and V_j. Hence we must get at least $l+1$ connected components.

Minimum k-Cut

To construct a Gomory-Hu tree for an undirected graph, we use only $n - 1$ max-flow computations.

Algorithm 0.3: GOMORY-HU-TREE APPROACH FOR k-CUT(G)

Compute a Gomory-Hu tree T for G.
Output the union of the lightest $k - 1$ cuts
  of the $n - 1$ cuts associated with edges of T in G.
Let C be this union.

Theorem

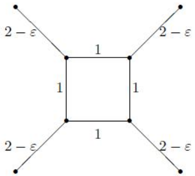
The above algorithm achieves an approximation ratio $2 - \frac{2}{k}$.

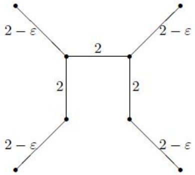
Proof.

Similar proof.. □

Example

The tight example given above for multiway cuts on $2k$ vertices also serves as a tight example for the k-cut algorithm (of course, there is no need to mark vertices as terminals). Below we give the example for $k = 4$, together with its Gomory-Hu tree.





The lightest $k-1$ cuts in the Gomory-Hu tree have weight $2-\epsilon$ each, corresponding to picking edges of weight $2-\epsilon$ of G. So, the k-cut returned by the algorithm has weight $(k-1)(2-\epsilon)$. On the other hand, the optimal k-cut picks all edges of weight 1, and has weight k.