CS 598CSC: Approximation Algorithms	Lecture date: February 11, 2009
Instructor: Chandra Chekuri	Scribe: Charles Blatti

In the study of approximation algorithms, there are many useful techniques that are demonstrated in the analysis of cut problems. Given a graph $G = (V, E)$ with an assignment of weights to edges $w : E \to \mathbb{R}^+$ a *cut* is a set $E' \subseteq E$, whose removal from the graph will partition the vertices into two disconnected sets. For two vertices, $s, t \in V$, an $s-t$ cut is the one that creates a partition with s and t in separate components. The minimum weight $s-t$ cut can be found in polynomial time. However, generalizations of the problem can be **NP**-Hard.

1 The Multiway Cut Problem

The Multiway Cut Problem is one generalization of minimum weight cuts that require approximation algorithms. This problem is also known as the Multiterminal Cut Problem. Given an undirected graph, $G = (V, E)$, edge weights, $w : E \to \mathbb{R}^+$, and a set of terminals $S = \{s_1, s_2, ..., s_k\} \subseteq V$, a *multiway cut* is a set of edges that leaves each of the terminals in a separate component. The goal of the Multiway Cut Problem is to find a minimum weight set of edges $E' \subseteq E$ such that removing E' from G separates all terminals. In other words, no connected component of $G(V, E - E')$ contains two terminals from S.

Facts:

1. The Multiway Cut Problem is precisely finding the minimum $s-t$ cut when there are only two terminals and thus can be computed efficiently.
2. In the case where there are three or more terminals in S, Multiway Cut is **NP**-Hard.
3. When $k \geq 3$ terminals, Multiway Cut is **APX**-Hard, meaning that there is a constant $\delta > 1$ such that it is **NP**-Hard to even approximate the solution to within a ratio of less than δ.
4. Multiway Cut can be solved exactly for fixed k in *planar graphs*.

1.1 Isolating Cut Heuristic

One greedy approach involves using the *isolating cuts* of the terminals [1]. For any terminal $s_i \in S$, we will define its *isolating cut* as the set of edges $E_i \subseteq E$, whose removal disconnects s_i from all other terminals. The minimum weight isolating cut for s_i can be found by connecting all other terminals to a shared new vertex t with edges of infinite weight and then running the maximum flow algorithm from s_i to t.

Isolating Cut Heuristic$(G(V, E), w : E \to \mathbb{R}^+, S \subseteq V)$:
For each $i = 1, ..., k$, compute a minimum weight isolating cut for t_i, E_i
Assume without loss of generality that $w(E_1) \leq w(E_2) \leq ... \leq w(E_k)$
Let A be the union of the $k-1$ lightest cuts, $A = E_1 \cup E_2 \cup ... \cup E_{k-1}$
Output A

Theorem 1 *The* Isolating Cut Heuristic *has an approximation ratio of* $2 - \frac{2}{k}$.

Proof: Let $E^* \subseteq E$ be an optimal multiway cut in G. By definition, the removal of E^* from E will create a graph of k connected components, $V_1, V_2, ..., V_k$, with each V_i uniquely containing the terminal t_i. Let E_i^* be the cut that separates the components V_i containing s_i from the rest of the graph. Then $E^* = \bigcup_{i=1}^{k} E_i^*$.

Next we define a function δ, where $\delta(V_i)$ is the set of all edges leaving the component containing terminal s_i.

Claim 2 $w(E_i) \leq w(\delta(V_i))$ *for all* $1 \leq i \leq k$. *The weight of a minimum weight isolating cut of a terminal must be less than or equal to the weight of all edges leaving its component.*

Proof: This is because $\delta(V_i)$ is an isolating cut of s_i. □

Claim 3 $2w(E^*) = w(\delta(V_1)) + w(\delta(V_2)) + ... + w(\delta(V_k))$

Proof: Every edge in $E*$ is incident to two of the connected components. Therefore, each edge will be in exactly two of the cuts $\delta(V_i)$. The sum of the weights of all cuts will be twice the weight of the optimal cut. □

The $2 - \frac{2}{k}$ ratio comes from the fact that the last isolating cut is discarded and not output. $w(A) \leq (1 - \frac{1}{k}) \sum_{i=1}^{k} w(E_i) \leq (1 - \frac{1}{k}) \sum_{i=1}^{k} w(\delta(V_i)) = 2(1 - \frac{1}{k}) w(E^*)$. □

A Tight Example: A tight example can be made with $2k$ vertices. k of the vertices form a cycle where each edge weight is equal to 1. Each other vertex is connected to exactly one vertex on the cycle with an edge weight of $2 - \epsilon$ for some $\epsilon > 0$. The isolating cuts chosen are always the $2 - \epsilon$ edges while the optimal cut is the central cycle. This is a $\frac{(2-\epsilon)(k-1)}{k}$ approximation, which shows our analysis is tight.

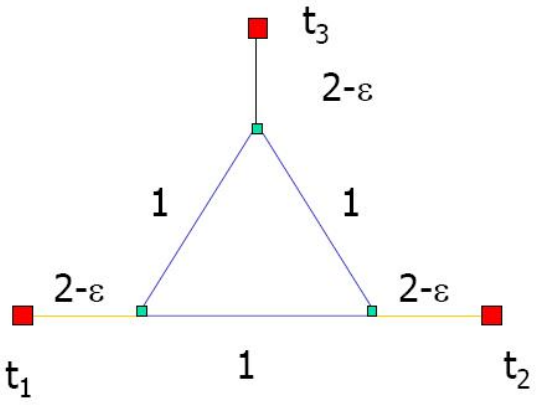

Figure 1: A tight example with the blue edges forming the optimal cut and the yellow edges forming the cut returned by the Isolating Cut Heuristic algorithm.

1.2 Greedy Splitting Algorithm

There is another efficient algorithm that can be used to solve the Multiway Cut Problem.

GREEDY SPLITTING ALGORITHM$(G(V,E), w : E \to \mathbb{R}^+, S \subseteq V)$:
Find the cheapest cut that splits G into 2 components such that each contains a terminal
Find the cheapest cut dividing the 2 components such that each of the 3 components contains a terminal
Find the cheapest cut dividing the 3 components such that each of the 4 components contains a terminal
...until each of the k components contains a terminal

Theorem 4 *The* GREEDY SPLITTING ALGORITHM *has an approximation ratio of* $2 - \frac{2}{k}$.

Proof: We will define some notation before giving the proof, due to [6]. Let $P_1, P_2, ..., P_{k-1}$ be the partitions of the the vertices generated at each iteration of the greedy algorithm. $P_0 = \{V\}$. (Partition P_i contains $i+1$ sets of vertices.) Notice that each P_i is a refinement of P_{i-1} with one of the components of P_{i-1} split into two components. We also define $w(P_i)$ as the cost of the edges cut in the partition P_i. A *valid* partition of the vertex set is one in which each set of vertices has at least one terminal from S.

Lemma 5 *For any valid partition* $\mathcal{V} = \{V_1, V_2, ...V_i\}$, $w(P_{i-1}) \leq \sum_{j=1}^{i-1} w(\delta(V_j))$.

Note that the index of summation in Lemma 5 runs from 1 to $i-1$; that is, we do not count $w(\delta(V_i))$. As the sets of a partition can be ordered arbitrarily, we can choose not to count edges leaving *any one* set in the partition.

Assuming that Lemma 5 is true, we complete the proof of Theorem 4. Consider an optimal solution for the problem, $\mathcal{A} = \{A_1, A_2, ...A_k\}$, where the ordering is such that the weight of edges leaving A_k is at least the weight of edges leaving any other set in the partition. $\sum_{j=1}^{k} w(\delta(A_j)) = 2\text{OPT}$, and so $\sum_{j=1}^{k-1} w(\delta(A_j)) \leq 2(1 - \frac{1}{k})\text{OPT}$. The output of the greedy splitting algorithm is the partition P_{k-1}, and from Lemma 5, $w(P_{k-1}) \leq \sum_{j=1}^{k-1} w(\delta(A_j)) \leq 2(1 - \frac{1}{k})\text{OPT}$. □

Proof of Lemma 5. The lemma can be proved by induction. The base case, $i = 1$, is trivial to check. The hypothesis is assumed to hold for $i-1$. Let $\mathcal{V} = \{V_1, V_2, ...V_i\}$ be an arbitrary valid partition into i sets. Consider P_{i-2}. Since $\mathcal{V}$ contains i sets and P_{i-2} contains $i-1$ sets, there must be some two terminals in the same set W of P_{i-2} but in different sets V_h, V_ℓ of $\mathcal{V}$, where $h < \ell$. The GREEDY SPLITTING ALGORITHM would consider splitting W into $W \cap V_h$ and $W - V_h$ when refining P_{i-2} into P_{i-1}. Since we greedily chose the cheapest split, $w(P_{i-1}) - w(P_{i-2})$ is at most the increase in cost of choosing to split W as described above.

Splitting W into $W \cap V_h$ and $W - V_h$ increases the partition's cost by at most $w(\delta(V_h))$. This is because any new edge e induced by the split of W is from $(V_h \cap W)$ to $(W - V_h)$, and so it must be in $\delta(V_h)$. Thus, $w(P_{i-1}) \leq w(P_{i-2}) + w(\delta(V_h))$.

Consider the partition $\mathcal{V}' = \{V_1, V_2, \ldots, V_{h-1}, V_{h+1}, \ldots, V_{i-1}, V_h \cup V_i\}$. (That is, remove V_h from $\mathcal{V}$, and merge it with V_i.) From the induction hypothesis, $w(P_{i-2}) \leq \left(\sum_{j=1}^{i-1} w(\delta(V_j))\right) - w(\delta(V_h))$. But $w(P_{i-1}) \leq w(P_{i-2}) + w(\delta(V_h)) \leq \sum_{j=1}^{i-1} w(\delta(V_j))$, proving the induction hypothesis for i. □

2 The k-Cut Problem

The K-CUT PROBLEM is a second generalization of minimum weight cuts that require approximation algorithms. Once again, we are given an undirected graph, $G = (V, E)$ and edge weights, $w : E \to$

$\mathbb{R}^+$, and an integer k. The goal of the k-Cut Problem is to find a minimum weight set of edges $E' \subseteq E$ such that removing E' from G leaves k connected components.

Facts:

1. The k-Cut Problem is precisely finding the global minimum $s-t$ cut when $k=2$ and is polynomial time solvable.
2. Given a fixed k, k-Cut Problem can be solved in polynomial time; a deterministic algorithm with running time $n^{O(k^2)}$) is given in [2]. The following randomized algorithm, due to [3], finds the correct cut with probability $\geq \frac{1}{n^{2k}}$ in a single iteration, which it repeats $O(n^{2k}\log n)$ times to return the correct cut with high probability. (A tighter analysis is given in [3].)

Global Minimum k-Cut$(G(V,E), w: E \to \mathbb{R}^+, k)$:
Pick an edge, $e \in E$ at random in proportion to its weight $w(e)$
Contract graph by merging the two endpoints of e
Repeat above two steps until there are k vertices remaining
The edges remaining define the cut of the original graph
Repeat the entire process many times

3. In the case where k is specified as part of the input, k-Cut Problem is **NP**-Hard.

2.1 Greedy Splitting Algorithm

The Greedy Splitting Algorithm described above can be used to solve the k-Cut Problem with the slight modification of ignoring the locations of the terminals.

Greedy Splitting Algorithm$(G(V,E), w: E \to \mathbb{R}^+, k)$:
Split G into two components
Split one of the two components
Split one of the three components
...until there are k components
At every step of the algorithm chooses the cheapest cut among all components.

Theorem 6 *The* Greedy Splitting Algorithm *has an approximation ratio of* $2-\frac{2}{k}$ *for the* k-Cut Problem.

We describe a simple proof below using Gomory-Hu Trees; the original proof is due to [4].

2.2 Gomory-Hu Tree Algorithm

There are several interesting structural properties in undirected graphs. Given a undirected graph $G=(V,E)$ with edge weights, $w: E \to \mathbb{R}^+$, define $c(a,b)$ as the weight of the minimum $a-b$ cut in G. There are $\frac{n(n-1)}{2}$ pair of vertices in G, so there are potentially $\frac{n(n-1)}{2}$ different minimum cut values. However, it turns out that there are only $n-1$ distinct minimum cut values in G. This fact introduces the idea of a Gomory-Hu tree representation of minimum cuts.

Definition: For any undirected graph $G = (V, E)$ with edge weights, $w : E \to \mathbb{R}^+$, there is a *Gomory-Hu Tree*, $T = (V, E_T)$ with an edge weight function $w' : E \to \mathbb{R}^+$, such that the following are true:

1. $\forall u, v$ pairs in V, the value of the minimum $u - v$ cut in T is the same as in G. This reinforces the fact that there are only $n - 1$ distinct minimum cut values in G.
2. $\forall e = (a, b) \in T$, the cut induced by e in T is a minimum cut in G between a and b. ($w'(ab) = c(ab)$).

The Gomory-Hu Tree encodes minimum cut information in a succinct manner. For every pair u, v in V, there is a unique path connecting the two in the Gomory-Hu Tree. To find the minimum cut between any two vertices we most only find the minimum weight edge on that unique path in T.

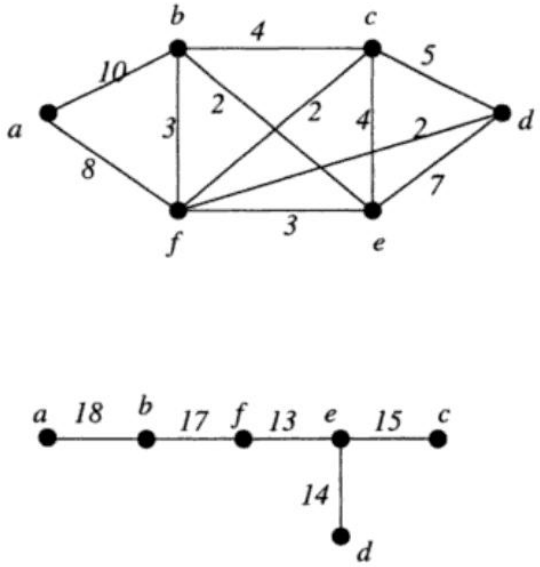

Figure 2: The top graph can be succinctly represented by the Gomory-Hu Tree.

The Gomory-Hu tree can be computed in $O(n)s - t$ cut computations. With each edge ab in E_T, we can also find an associated minimum $a - b$ cut, C_{ab}, of value $w'(ab)$. Removing the edges in C_{ab} from G will disconnect a and b.

This observation leads to the following simple algorithm.

k-Cut with Gomory-Hu Trees$(G(V, E), w : E \to \mathbb{R}^+, k)$:
Compute the Gomory-Hu Tree of G, T
Pick the $k - 1$ lightest edges in the T, call them $e_1, e_2, ..., e_{k-1}$
Return the union of the corresponding cuts,C, in G, $C = C_{e_1} \cup C_{e_2} \cup ...\cup, C_{e_{k-1}}$

Proof: Removing C from G results in at least k connected components. The proof states that removing $k-1$ edges from T, leaves k connected components in T, $V_1, V_2, ..., V_k$. Removing C from

G will disconnect every pair V_i and V_j. Therefore we must get at least k connected components in $G[V, E - C]$. If more than k connected components exist in $G[V, E - C]$, return some of the removed edges until there are exactly k. □

Theorem 7 *The* k-Cut with Gomory-Hu Trees Algorithm *has an approximation ratio of* $2 - \frac{2}{k}$.

Proof: Similar to above, Let $E^* \subseteq E$ be an optimal k-cut in G that defines a graph of k connected components, $V_1, V_2, ..., V_k$. Let E_i^* be the cut that separates the component V_i from the rest of the graph. Again, $E^* = \bigcup_{i=1}^{k} E_i^*$. We define a function δ, where $\delta(V_i)$ is the set of all edges leaving the component V_i. We can assume without loss of generality that $w(\delta(V_1)) \leq w(\delta(V_2)) \leq ... \leq w(\delta(V_k))$.

As before, every edge in $E*$ is incident to two of the connected components. Therefore, each edge will be in exactly two of the cuts $\delta(V_i)$ and $2w(E^*) = w(\delta(V_1)) + w(\delta(V_2)) + ... + w(\delta(V_k))$

Claim 8 $w'(e_i) \leq w(\delta(V_i))$.

Proof: To prove this claim we will show that for some distinct edges $f_1, f_2, ..., f_{k-1}$ of T, $w'(f_i) \leq w(\delta(V_i))$. This will prove the lemma because the algorithm will pick the lightest $k - 1$ of these edges.

Begin with a Gomory-Hu Tree, T. Shrink each V_i into a simple vertex. Make the shrunken graph into a tree T' by remove unnecessary, parallel edges arbitrarily. T' must be connected because T was connected to begin with. T' is a tree with k vertices,$t_1, t_2, ..., t_k$, with t_i corresponding to V_i and $k-1$ edges. Root the tree at V_k and orient all edges towards the root. Let f_i be the unique edge directed from t_i towards the root. f_i is a edge (a, b) where $a \in V_i$ and $b \notin V_i$. From the Gomory-Hu properties, $c(ab) = w'(f_i)$. Since $\delta(V_i)$ is a cut that separates a from b, $w(\delta(V_i)) \geq= c(ab) = w'(f_i)$. □

The $2 - \frac{2}{k}$ ratio comes from the fact that the last isolating cut is discarded and not output. $w(C) = \sum_{i=1}^{k-1} w'(e_i) \leq (1 - \frac{1}{k}) \sum_{i=1}^{k} w(\delta(V_i)) = 2(1 - \frac{1}{k})w(E^*)$. □

A Tight Example: The same tight example for the Multiway Cut Problem will also be a tight example for the k-Cut with Gomory-Hu Trees Algorithm with an approximation ration of $\frac{(2-\epsilon)(k-1)}{k}$.

References

[1] E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, P. D. Seymour, and M. Yannakakis. The Complexity of Multiterminal Cuts. SIAM J. Comput. 23: 864-894, 1994.

[2] Olivier Goldschmidt and Dorit S. Hochbaum. Polynomial Algorithm for the k-Cut Problem. *Proc. of IEEE FOCS*: 444-451, 1988.

[3] D. Karger. Global min-cuts in $\mathcal{RNC}$ and other ramifications of a simple mincut algorithm. *Proc. of the 4th ACM-SIAM SODA*, 21–30, 1993.

[4] Huzur Saran and Vijay V. Vazirani. Finding k Cuts within Twice the Optimal. SIAM J. Comput. 24: 101-108, 1995.

[5] Vazirani, Vijay. Multiway Cut and k-Cut. "Approximation Algorithms." New York: Springer, 2003.

[6] L. Zhao, H. Nagamochi, T. Ibaraki. A greedy splitting algorithm for approximating multiway partition problems. Mathematical Programming, vol. 102, no. 1, 2005, pp. 67-183.