CS 598CSC: Topics in Graph Algorithms	Lecture date: 09/05/2024
Instructor: Chandra Chekuri	Scribe: CC

1 Steiner Mincut

Given a graph $G = (V, E)$ with edge capacities $c : E \to \mathbb{R}_+$, we will be interested in two problems. The first is the global mincut problem that we already discussed. The second is a problem called the Steiner mincut problem. In this problem we are given G and a subset of vertices $T \subseteq V$ called the *terminals*. The goal is to find a cut $(S, V - S)$ of minimum capacity such that $S \cap T \neq \emptyset$ and $(V - S) \cap T \neq \emptyset$; in other words a minimum capacity cut that separates some pair of terminals. Note that if $T = \{s, t\}$, we have the familiar s-t mincut problem. When $T = V$ we have the global mincut problem. Thus Steiner mincut generalizes both. How can we compute Steiner mincut? It is easy to compute it via $|T| - 1$ s-t mincut computations as follows. Say $T = \{t_1, t_2, \ldots, t_k\}$. Pick t_1 and find t_1-t_j mincut for each $j > 1$ and pick the cheapest among them. In fact for even the global mincut problem this was the best known algorithm till the work of Nagamochi and Ibaraki. Quite recently Li and Panigrahy [LP20] developed a simple yet striking approach that computes the Steiner mincut with high probability using only $O(\log^3 n)$ s-t cut computations! The approach could have been discovered many years ago in terms of its simplicity. The core idea is based on computing *isolating cuts* (see also independent work of Abboud et al [AKT21]) and this has been very influential in the last few years for a number of problems.

1.1 Background on Submodularity

The isolating cut approach is essentially based on properties of symmetric submodular functions. Although one can prove various properties by appealing to only graph theoretic facts, it is useful to see that the proofs via submodularity. Here we give some background on submodularity and the cut function of graphs.

We will work with set functions. Given a finite ground set V a real-valued set function $f : 2^V \to \mathbb{R}$ assigns a number to each subset $S \subseteq V$. We are used to *modular* functions, that is functions that satisfy the property that $f(A) + f(B) = f(A \cap B) + f(A \cup B)$. We don't think of modular functions in this abstract fashion because a function is modular iff there is a weight function $w : V \to \mathbb{R}$ and a real valued shift parameter c such that $f(A) = c + \sum_{v \in A} w(v)$. It is worthwhile to prove this claim as an exercise. We will be interested in a more general class.

Definition 1. *A real-valued set function $f : 2^V \to \mathbb{R}$ is* submodular *iff $f(A \cup B) + f(A \cap B) \leq f(A) + f(B)$ for all $A, B \subseteq V$. A function f is* supermodular *iff $-f$ is submodular, equivalently, $f(A \cup B) + f(A \cap B) \geq f(A) + f(B)$ for all $A, B \subseteq V$.*

A different definition of submodularity is based on *diminishing marginal utility* property. The marginal value of an element v to a set A is defined as $f(A + v) - f(A)$ where we use the notation $A + v$ for $A \cup \{v\}$. A function f is submodular iff $f(A + v) - f(A) \geq f(B + v) - f(B)$ for all $A \subset B$.

Exercise 1. *Prove that the two definitions of submodularity are equivalent.*

A useful observation is that if f and g are submodular functions, the function $\alpha f + \beta g$ is submodular where $\alpha, \beta \geq 0$ are scalars. Note that negation can make a submodular function supermodular. Modular functions are both submodular and supermodular.

Why is submodularity important for graphs? Two aspects of graphs are closely related to submodular functions. Some properties of graphs are really about properties of submodular functions and it is useful to know this.

Cut function of graphs: The following claim is a simple exercise which shows that the cut function is submodular in undirected graphs. It naturally extends to non-negative capacities.

Claim 1. *Let $G = (V, E)$ be an undirected graph. For any two subsets $A, B \subseteq V$,*

$$|\delta(A)| + |\delta_G(B)| = |\delta(A \cap B)| + |\delta(A \cup B)| + 2|E(A - B, B - A)| \geq |\delta(A \cap B)| + |\delta(A \cup B)|.$$

Claim 2. *Let $G = (V, E)$ be a directed graph. Prove that $|\delta^+(\cdot)|$ and hence by symmetry $|\delta^-(\cdot)|$ are submodular.*

A set function is *symmetric* if $f(A) = f(V - A)$ for all $A \subseteq V$. Clearly the cut function of an undirected graph is symmetric while it is not necessarily the case for directed graphs. Symmetric submodular functions satisfy another important property.

Definition 2. *A real-valued set function $f : 2^V \to \mathbb{R}$ is* posi-modular *iff $f(A - B) + f(B - A) \leq f(A) + f(B)$ for all $A, B \subseteq V$.*

Exercise 2. *Prove that if f is symmetric and submodular then f is also posi-modular.*

Uncrossing properties of submodular functions: Uncrossing is a common and powerful technique that is frequently used in working with submodular functions. We will illustrate this in the context of minimum cuts.

Lemma 3. *Let $G = (V, E)$ be a graph and let A, B be two s-t mincuts. Then $A \cup B$ and $A \cap B$ are also s-t mincuts.*

Proof. We note that $s \in A, t \in V - A$ since A is an s-t cut. Similarly for B. We also see that $s \in A \cap B, t \in V - (A \cap B)$ and hence $A \cap B$ is also an s-t cut. Similarly $A \cup B$ is also an s-t cut. By submodularity of $|\delta(\cdot)|$ we have

$$|\delta(A)| + |\delta(B)| \geq |\delta(A \cap B)| + |\delta(A \cup B)|$$

and we also have

$$|\delta(A)| + |\delta(B)| \leq |\delta(A \cap B)| + |\delta(A \cup B)|$$

because A, B are s-t mincuts while $A \cap B$ and $A \cup B$ are s-t cuts so their value cannot be smaller than that of the mincuts. Thes two inequalities can be true only if $|\delta(A \cap B)| = |\delta(A \cup B)| = |\delta(A)| = |\delta(B)|$. ■

Corollary 4. *In a graph $G = (V, E)$ there is a unique minimal s-t cut and there is a unique maximal s-t cut.*

Proof. Suppose A, B are two s-t mincuts and both are minimal and distinct. This implies that $A - B \neq \emptyset$ and $B - A \neq \emptyset$ for otherwise one will be contained in the other contradicting the minimality. Then, by the preceding lemma $A \cap B$ is also an s-t mincut and $A \cap B$ is a strict subset of A and B, but again this is a contradiction of minimality of A, B. Similar argument shows the uniqueness of a maximal s-t mincut. ■

Exercise 3. *Show that one can find the unique minimal and unique maximal s-t-mincuts from an s-t maxflow in linear time.*

Note that the proof applies also to directed graphs. Since we only used submodularity, we can apply it to other settings on submodular cuts.

Graphic matroid: A second aspect of submodularity in graphs comes via matroids. We will not discuss it here but the rank function of a matroid is a special class of submodular functions; and in a formal sense matroid rank functions are building blocks for all submodular functions. Given an undirected graph $G = (V, E)$ there is a fundamental matroid associated with the edge set of G called the graphic matroid (there are other matroids that are also defined from graphs including the dual graphic matroid for instance). Several properties of trees and forests can be better understood in the context of the graphic matroid including the Tutte-Nash-Williams theorem we saw on tree packing. See notes and books on combinatorial optimization.

1.2 Isolating Cuts via Poly-log Maxflow Computations

Let $G = (V, E)$ be a capacitated graph and let $T \subseteq V$ be set of terminals. Let $T = \{t_1, t_2, \ldots, t_k\}$. We say that a cut $(S, V - S)$ is a t_i-isolating cut if $t_i \in S$ and $t_j \in V - S$ for all $j \neq i$. In other words the cut isolates t_i from the rest of the terminals. The minimum capacity t_i-isolating cut can be found by a single maxflow computation — we can shrink the terminals in $T - \{t_i\}$ into a single vertex s and compute a t_i-s cut. Thus, we can compute all isolating cuts via k maxflow computations. It turns out that we can compute them in $O(\log k)$ maxflow computations[1].

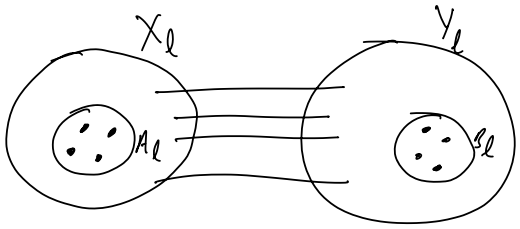
Via submodularity discussion earlier, we have the following.

Lemma 5. *There is a unique minimal t_i-isolating mincut $(S_i^*, V - S_i^*)$. That is, if $(S_i, V - S_i)$ is any t_i-isolating mincut then $S_i^* \subseteq S_i$.*

Now we describe the algorithm for computing the isolating mincuts. Let $h = \lceil \log k \rceil$. We consider h bipartitions of T: into $(A_1, B_1), (A_2, B_2), \ldots, (A_h, B_h)$ as follows. Consider the binary representation of k. It has h bits. We set $A_\ell = \{t_i \mid \text{binary representation of } i \text{ has 1 in } \ell\text{'th position}\}$. $B_\ell = T - A_\ell$.

1. For $\ell = 1$ to h compute a A_ℓ-B_ℓ mincut (X_ℓ, Y_ℓ)
2. For $i = 1$ to k define $U_i = (\cap_{\ell : t_i \in A_\ell} X_\ell) \cap (\cap_{\ell : t_i \in B_\ell} Y_\ell)$.

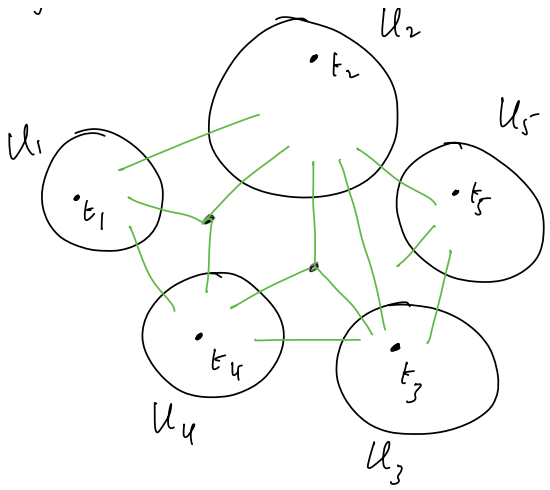
[1]The terminology of isolating cuts was already in use in the context of a simple approximation algorithm for the Multiway Cut problem. For instance see the chapter on Multiway Cut in Chandra's notes `https://courses.grainger.illinois.edu/cs583/fa2021/approx-algorithms-lecture-notes.pdf` for pointers and discussion.



Lemma 6. *For each $i \in [k]$, $(U_i, V - U_i)$ is a t_i-isolating cut. Further, $U_1, U_2, \ldots, U_k$ are pairwise disjoint sets.*

Proof. If $t_i \in A_\ell$, then $t_i \in X_\ell$. Similarly, if $t_i \in B_\ell$ then $t_i \in Y_\ell$. Thus $t_i \in U_i$. Consider t_j where $j \neq i$. Since $i \neq j$ there is some index ℓ in the binary representation of i and j differ in the bit position. Suppose i has 1 in the ℓ'th position and j has 0 (the other case is similar). Then $t_i \in A_\ell$ and $t_j \in B_\ell$. Thus $t_i \in X_\ell$ and $t_j \notin X_\ell$. Which implies that $t_j \notin U_i$.

The same reasoning as above shows that no vertex v can be in both U_i and U_j for $i \neq j$. There must be some ℓ such that $t_i \in A_\ell$ and $t_j \in B_\ell$ or vice-versa. Suppose it is first case; if $v \in X_\ell$ then v cannot be in U_j and if $v \in Y_\ell$ then v cannot be in U_i and hence v cannot be on both U_i and U_j. The other case is similar. ■



Exercise 4. *Show how to compute $U_1, \ldots, U_k$ in $O((m+n)\log k)$ time given (X_ℓ, Y_ℓ) for $\ell \in [h]$.*

The key lemma is the following. In other words, the unique minimal t_i-isolating cut is contained in U_i.

Lemma 7. *For each i, $S_i^* \subseteq U_i$.*

Proof. Consider the computation of an (A_ℓ, B_ℓ) mincut for some ℓ. Suppose $t_i \in A_\ell$. We claim that $S_i^* \in X_\ell$ and derive a contradiction by assuming that it is not true. We observe that $S_i^* \cap X_\ell$ is a strict subset of S_i^* and is also a t_i-isolating cut, and hence by minimality of S_i^*, $|\delta(S_i^* \cap X_\ell)| > |\delta(S_i^*)|$.

We observe that $S_i^* \cup X_\ell$ is an A_ℓ-B_ℓ cut (why?) and hence $|\delta(S_i^* \cup X_\ell)| \geq |\delta(X_\ell)|$ since X_ℓ is a A_ℓ-B_ℓ mincut. But these facts contradict the inequality that we have via submodularity:

$$|\delta(S_i^*)| + |\delta(X_\ell)| \geq |\delta(S_i^* \cap X_\ell)| + |\delta(S_i^* \cup X_\ell|.$$

A similar argument show that when $t_i \in B_\ell$, $S_i^* \subseteq Y_\ell$. Thus, $S_i^* \subseteq U_i$ based on the definition of U_i. ■

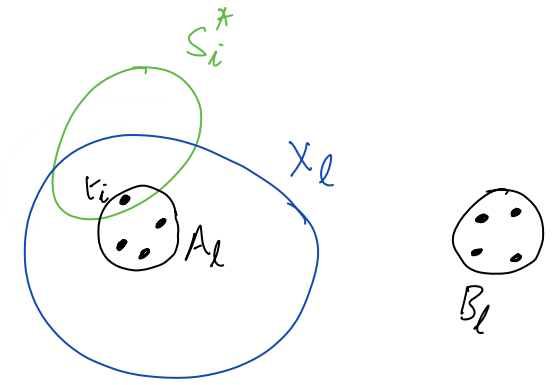


Figure 1: Figure for proof of Lemma 7.

The last step of the algorithm is the following. For each i we have an isolating cut $(U_i, V - U_i)$. The preceding lemma guarantees us that there is a minimum cost t_i-isolating cut $(S_i^*, V - S_i^*)$ where $S_i^* \subseteq U_i$. Thus, to compute S_i^* we can do the following. Construct a graph $H_i = (V_i, E_i)$ where we shrink $V - U_i$ to a single vertex s_i. We ask the reader to prove this for themselves.

Claim 8. *The min t_i-s_i cut in H_i is the minimum cost t_i-isolating cut in G.*

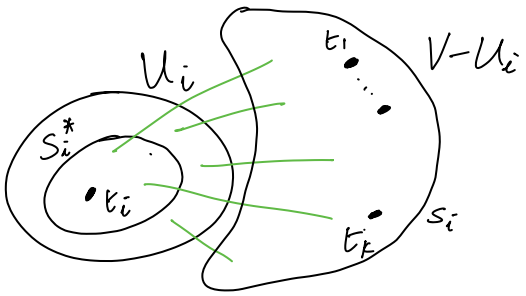


Figure 2: $V - U_i$ is shrunk to s_i to create graph H_i. U_i^* is found by computing t_i-s_i cut in H_i.

Thus the isolating cut algorithm becomes the following.

1. For $\ell = 1$ to h compute a A_ℓ-B_ℓ mincut (X_ℓ, Y_ℓ)
2. For $i = 1$ to k define $U_i = (\cap_{\ell : t_i \in A_\ell} X_\ell) \cap (\cap_{\ell : t_i \in B_\ell} Y_\ell)$.

3. For $i = 1$ to k, create H_i and compute a min t_i-s_i cut $(S_i^*, V_i - S_i^*)$.

4. For $i = 1$ to k output S_i^* as the min-cost t_i-isolating cut.

Analyzing the running time: It is easy to see that the first step can be done with $O(\log k)$-maxflow computations on G. In the third step we are actually doing k-maxflow computations so it seems that we did not meet our promise. However, let us understand the size of H_i. It is $n_i + 1$ vertices where $n_i = |U_i|$ and it has m_i edges where $m_i = |E[U_i]| + |\delta(U_i)|$. Thus, the running time of maxflow on H_i is $T(n_i + 1, m_i)$ where $T(a, b)$ is the running-time of maxflow on graph with a nodes and b edges. We observe that $\sum_i (n_i + 1) \leq 2n$ since the U_i are disjoint. We also note that $\sum_i m_i \leq 2m$. Why? Consider any edge $uv \in E$. If $uv \in E[U_i]$ for some i then it does not contribute to any other H_j. If $uv \in \delta(U_i)$ for some i then it can be in $\delta(U_j)$ for only one index $j \neq i$. Thus each uv can be part of only two of the graphs $H_i, i \in [k]$. Thus the total time to compute all the k maxflows is $\sum_i T(n_i, m_i) \leq T(2n, 2m))$ under reasonable assumption that $T(a, b)$ is super-additive. This may not be satisfactory explanation to claim that we have reduced the problem to $O(\log k)$-maxflow computations but we can make it more formal as follows. We can create a single H that includes each H_i as a copy in it and we can run a single maxflow on H to recover all the maxflow values in each H_i. H will have $O(n)$ vertices and $O(m)$ edges — we leave the construction of such a graph H as an exercise.

Theorem 9 ([LP20])**.** *There is a determininstic algorithm that given $G = (V, E)$ and terminal set $T \subseteq V$ with $|T| = k$ computes all the isolating cuts using $O(\log k)$-maxflow computations on graphs with $|V|$ vertices and $|E|$ edges each.*

Remark 10. *Another perspective on the bipartititions is that they are a way to derandomize a natural randomized algorithm that picks some $O(\log k)$ bipartitions of T at random and computes the cuts between them. With high probability every t_i and t_j, $i \neq j$ will be separated in at least one of the random bipartitions.*

1.3 Randomized algorithm for Steiner mincut via Isolating Cuts

Isolating cuts easily lead to a simple randomized algorithm for Steiner mincut. The basic idea is quite simple. Consider an optimum Steiner mincut $(S, V-S)$ and let $T_1 = S \cap T$ and $T_2 = (V-S) \cap T$ and let $k_1 = |T_1|$ and $k_2 = |T_2|$. We observe that $(S, V - S)$ is a min t_i-t_j cut for any $t_i \in T_1$ and $t_j \in T_2$. Without loss of generality $1 \leq k_1 \leq k_2$. Suppose we knew k_1. We sample each terminal in T independently with probability $1/k_1$ to obtain a susbet T'. Simple probabilistic calculations shows that with constant probability $|T' \cap T_1| = 1$ *and* $|T' \cap T_2| \geq 1$. Suppose T' satisfies these properties and let $T_1 = \{t_i\}$. Then one observes that $(S, V - S)$ is a minimum cost t_i-isolating cut with respect to the terminal set T'. Thus, computing the T'-isolating cuts and choosing the cheapest one identifies the minimum Steiner cut for T. But how do we know k_1? We do not so we can "guess" k_1 within a factor of 2; alterantively, we try $O(\log k)$ different sampling probabilities.

1. min-so-far $\leftarrow \infty$.

2. For $i = 0$ to $\lceil \log k \rceil$ do

   (a) Choose T' by sampling each terminal in T with probability $1/2^i$.

(b) Compute T' isolating cuts and update min-so-far with the cheapest of the cuts found

3. Output min-so-far and the corresponding cut

Lemma 11. *The algorithm finds the minum Steiner cut for T with probability at least $1/4$.*

Proof. We can assume that $k_1 > 1$ for if $k_1 = 1$ we obtain the correct solution deterministically when $i = 0$ which corresponds to $T' = T$. Following the high-level explanation we will consider when an index i such that $\frac{1}{2^{i+1}} < \frac{1}{k_1} \leq \frac{1}{2^i}$. Such an index exists since we try $i = 0$ to $\lceil \log k \rceil$ and $1 \leq k_1 \leq k/2$. Let $\ell = 2^i$ so we have $\ell \leq k_1 \leq 2\ell$. Let $\mathcal{E}_1$ be the event that $T_1 \cap T' = 1$, that is exactly one terminal from T_1 is chosen. We see that $\mathbf{P}[\mathcal{E}_1] = k_1 \frac{1}{\ell}(1-1/\ell)^{k_1-1} \geq (1-1/\ell)^{2\ell} \geq 1/e^2$. Let $\mathcal{E}_2$ be the event that $T_2 \cap T' \neq \emptyset$. We see that $\mathbf{P}[\mathcal{E}_2] \geq 1-(1-1/\ell)^{k_2} \geq (1-1/\ell)^{\ell} \geq 1-1/e$. Note that $\mathcal{E}_1$ and $\mathcal{E}_2$ are independent events since T_1 and T_2 are disjoint sets. Thus $\mathbf{P}[\mathcal{E}_1 \cap \mathcal{E}_2] \geq (1-1/e)/e^2$ which is a constant. ■

It is easy to see that the algorithm runs $O(\log k)$ iterations of the isolating cut heuristic and thus the total runtime is $O(\log^2 k)$ maxflow computations. This gets us success probability of a constant. To amplify this we can repeat the algorithm. If we want high probabably guarantee we repeate it $\Theta(\log n)$ times which corresponds to a total of $O(\log^2 k \log n)$-maxflow computations.

Deterministic algorithm: Li and Panigrahy [LP20, LP21] also developed deterministic mincut and Steiner mincut algorithms in graphs using additional ideas based on expander decompositions.

Submodularity: The core idea of isolating cuts relies only on submodularity and symmetry and thus applies in much more generality and to several other problems. This is explicitly discussed in [CQ21] though the ideas are implicit in [LP20].

Other applications: Isolating cuts have found many other uses including algorithms for vertex connectivity, Gomory-Hu trees, cactus computation, etc. See [LP21] (full version of [LP20]) for some pointers.

References

[AKT21] Amir Abboud, Robert Krauthgamer, and Ohad Trabelsi. Subcubic algorithms for gomory–hu tree in unweighted graphs. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 1725–1737, 2021.

[CQ21] Chandra Chekuri and Kent Quanrud. Isolating cuts, (bi-)submodularity, and faster algorithms for global connectivity problems. *CoRR*, abs/2103.12908, 2021.

[LP20] Jason Li and Debmalya Panigrahi. Deterministic min-cut in poly-logarithmic max-flows. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 85–92. IEEE, 2020.

[LP21] Jason Li and Debmalya Panigrahi. Deterministic min-cut in poly-logarithmic max-flows. *CoRR*, abs/2111.02008, 2021.