University of Michigan–Ann Arbor

Department of Electrical Engineering and Computer Science

EECS 498 004 **Advanced Graph Algorithms**, Fall 2021

Lecture 12: Local Min Cuts and Isolating Cuts

October 7, 2021

Instructor: Thatchaphol Saranurak

Scribe: Yaowei Long

1 Overview

In this lecture, we are going to introduce two new techniques for minimum cut problems. The first technique is called *local min cuts*, aiming at finding some small cut without reading the whole graph. The second technique is called *isolating cuts*. Given a set of terminals in a graph, the algorithm for isolating cuts will, for each terminal, find a cut separate it from the other terminals. These two techniques can lead to efficient algorithms for some graph problems. For example, global mincut in directed graphs can be found efficiently using local min cuts, and isolating cuts can be applied to minimum Steiner cuts problem in undirected graphs.

2 Local min cuts

Given a large directed graph $G = (V, E)$ and a vertex $x \in V$ called seed, with access to the adjacency list of each vertex, a local min cut algorithm can determine if there is a small cluster containing x in time roughly proportional to the size of this cluster. In this note, a cluster is a sparse cut, which is essentially a vertex set L s.t. there are few edges going from L to $V \setminus L$.

Formally speaking, for a seed x, let $k, \nu < m/k$ be given parameters about the cut. The local min cut algorithm $\mathsf{Local}(x, \nu, k)$ will either

- declare that no cut L s.t. $x \in L$, $\delta_{\text{out}}(L) < k$ and $\text{vol}(L) \leq \nu$, where $\delta_{\text{out}}(L)$ denotes the number of edges going out L and $\text{vol}(L)$ is the volume[1] of L, or
- return a cut L' s.t. $x \in L'$, $\delta_{\text{out}}(L') < k$ and $\text{vol}(L') \leq O(\nu k)$.

The running time of $\mathsf{Local}(x, \nu, k)$ is $O(\nu k^2)$, and it will give a correct result with constant probability[2].

[1] The volume of a vertex set L is the total degree (ignoring directions of edges) of vertices in L.

[2] We can amplify the correctness in $\log(\nu, k)$ time to make this algorithm correct with high probability w.r.t. ν and k.

2.1 Warm-up

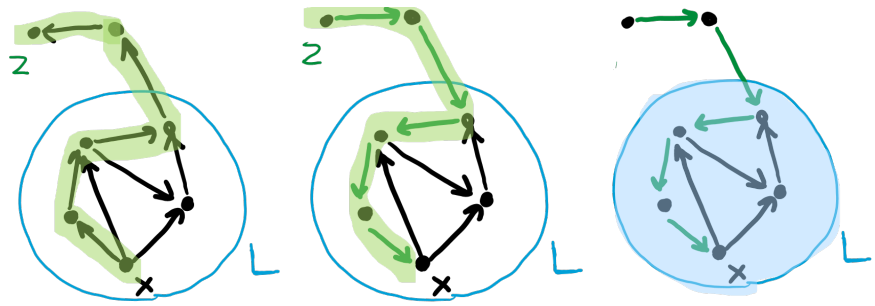
First, we look at a toy example. We assume that there is a cut L that includes x has $\mathrm{vol}(L) \leq \nu$, $\delta_{\mathrm{out}}(L) = 1$ and $\delta_{\mathrm{in}}(L) = 0$. In other words, there is only one edge going out from L to $V \setminus L$ and no edge going into L from $V \setminus L$.

Roughly speaking, the idea is as follows. See figure 1.

(1) We start a DFS from x. Suppose that we find a simple path P starting from x to z with length $\nu + 1$ in this DFS.

(2) Because $\delta_{\mathrm{in}}(L) = 0$ and the length of P is larger than $\mathrm{vol}(L)$, we know z is outside L. By reversing all edges in P, we have $\delta_{\mathrm{out}}(L) = 0$ in the new graph.

(3) Start another DFS from x in the new graph and let's say these explored vertices make up a cut L'. This DFS will not explored more than ν vertices and $\mathrm{vol}(L') \leq \nu$, because at the worst case we will get stuck after exploring the whole L. Because we only reverse edges on a simple path, we can also guarantee $\delta_{\mathrm{out}}(L') \leq 1$.



(a) Step (1) (b) Step (2) (c) Step (3)

Figure 1: The algorithm for the toy example

The intuition behind this example is that, for a cut L and a seed $x \in L$, if we can find a path P from x to some $z \notin L$, reversing edges in P will decrease $\delta_{\mathrm{out}}(L)$ by 1 but make $\delta_{\mathrm{in}}(L)$ unchanged. Once $\delta_{\mathrm{out}}(L)$ becomes 0, a DFS from x will find the cut L (or even a smaller one).

2.2 The algorithm

In this subsection we will introduce the whole algorithm. Roughly speaking, if we want to detect a cut L with $\delta_{\mathrm{out}}(L) < k$ and $\mathrm{vol}(L) \leq \nu$, we will repeat the above subroutine k times. Suppose that such a cut L exists. In each of the first $k - 1$ iterations, we will find a path from x to some $z \notin L$ with some good probability. Then in the kth round, we will have $\delta_{\mathrm{out}}(L) = 0$ with some constant probability and L can be detected by a DFS.

The formal description is as follows. The algorithm $\mathsf{Local}(x, \nu, k)$ will repeat the following k times. See figure 2

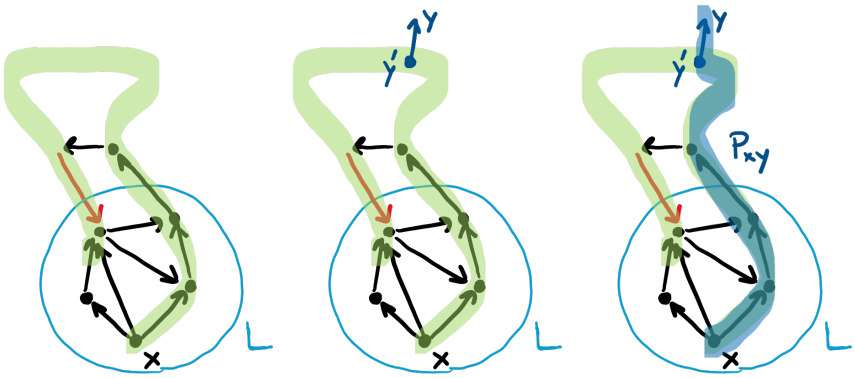
(1) Grow a DFS tree T from x by exploring exactly $k\nu + 1$ edges. Let L' be all explored vertices.

(2) If the DFS can only explore less then $kv + 1$ edges, we return L' and the whole algorithm terminates.

(3) Sample an edge (y', y) uniformly from the DFS tree.

(4) Reverse the path from x to y in T.

If the algorithm didn't terminate in all k iterations, it outputs there is no cut L s.t. $x \in L$, $\delta_{\text{out}}(L) < k$ and $\text{vol}(L) \leq v$.



(a) Step (1) (b) Step (3) (c) Step (4)

Figure 2: The local min cut algorithm

Lemma 2.1 (Time complexity)**.** *The running time of* $\textsf{Local}(x, v, k)$ *is* $O(vk^2)$.

Proof. Observe that in each iteration, the DFS takes $O(vk)$ time. Therefore, the total running time of k iterations is $O(vk^2)$. □

Lemma 2.2 (Soundness)**.** *If* $\textsf{Local}(x, v, k)$ *return a cut* L'*, then* $\delta_{\text{out}}(L') < k$ *and* $\text{vol}(L') = O(vk)$.

Proof. We can immediately get $\text{vol}(L') \leq 2vk = O(vk)$ from the algorithm. Also observe that $\delta_{\text{out}}(L') = 0$ at the moment when L' is returned. Since there are less then k paths are reversed before and each reversed path will decrease $\delta_{\text{out}}(L')$ by at most 1, we get $\delta_{\text{out}}(L') < k$ initially. □

Lemma 2.3 (Completeness)**.** *If there exists* L *s.t.* $\delta_{\text{out}}(L) < k$ *and* $\text{vol}(L) \leq v$*, then the algorithm will return a cut* L' *with probability at least* $1/10$.

Proof. Let say $\delta_{\text{out}}(L) = k' < k$ initially. In each of the first k' iterations, the edge (y', y) sampled in step (3) satisfies $y', y \notin L$ with probability $\frac{vk+1-v}{vk+1} \geq 1 - 1/k$, because there are $vk + 1$ edges in the DFS tree but only at most v of them will have endpoints in L. Therefore, the probability of $y \notin L$ for all the first k' iterations is at least

$$(1 - 1/k)^{k'} \geq (1 - 1/k)^k \geq 1/10.$$

since $k' < k$. In this case, $\delta_{\text{out}}(L) = 0$ at the $(k' + 1)$th iteration and the algorithm must return some L' since $\text{vol}(L) \leq v \leq vk + 1$. □

In summary, the algorithm introduced in this subsection can solve the local min cut problem efficiently. It is a randomized algorithm with errors only on one side, namely, it always guarantees the correctness of the returned cut L'. Therefore, the probability in lemma 2.3 can be amplified by repeating this algorithm. Essentially, the local min cut algorithm is a localized and randomized version of the Ford-Fulkerson algorithm.

2.3 Applications: Global Mincut in Directed Graphs

In the global mincut problem, given a directed graph $G = (V, E)$ with m edges and n vertices and a parameter k, we need to output a cut $(S, V \setminus S)$ where $|E(S, V \setminus S)| < k$ (if exists). Here $E(S, V \setminus S)$ denote edges from S to $V \setminus S$.

A following trivial algorithm can solve this problem. First we fix some source s and then compute an (s, t)-mincut in G and the reverse graph G^R for all $t \in V$. One of such (s, t)-mincuts should be a global mincut. This algorithm needs $\Theta(n)$ calls to max flows, which is $\Omega(mn)$ time. In what follows, we will introduce a efficient new algorithm for small k using local min cut technique. Precisely, the running time will be $\tilde{O}(mk^2)$.

We consider two main cases and solve them by different algorithms.

2.3.1 Balanced Case

In this case, we assume that $\text{vol}(S), \text{vol}(V \setminus S) \geq m/k$, i.e. the cut $(S, V \setminus S)$ is balanced.

We repeat the following $T = O(k \log n)$ times.

- Sample s and t with probability proportional to its degree (i.e. v is chosen with probability $\deg(v)/(2m)$).
- If (s, t)-mincut (A, B) has size less than k, return (A, B). Note that we only decide whether the size of (s, t)-mincut is less then k rather than the exact size of (s, t)-mincut. Therefore, a typical Ford-Fulkerson algorithm can handle this step in $O(mk)$ time.

The total time for this case is $O(mkT) = \tilde{O}(mk^2)$.

Lemma 2.4. *Suppose there exists a cut $(S, V \setminus S)$ of size less than k where $\text{vol}(S), \text{vol}(V \setminus S) \geq m/k$, then the above algorithm returns a cut (A, B) of size less than k with probability $1 - 1/n^{10}$.*

Proof. Without loss of generality, we assume S is the smaller side. We have

$$\Pr[s \in S \text{ and } t \in V \setminus S] = \Pr[s \in S] \cdot \Pr[t \in V \setminus S] = m/k/(2m) \cdot 1/2 \geq 1/(4k).$$

When $s \in S$ and $t \in V \setminus S$, the algorithm will return a cut of size less then k. After repeating $T = O(k \log n)$ times, a cut of size less than k will be returned with probability $1 - 1/n^{10}$. □

2.3.2 Unbalanced Cases

In this case, there exists a cut $(S, V \setminus S)$ of size less than k where $\text{vol}(S) < m/k$ or $\text{vol}(V \setminus S) < m/k$. Without loss of generality, we assume that S is the smaller side. We can run the same algorithm in the reverse graph G^R to handle the case that $V \setminus S$ is smaller. Let's say $\text{vol}(S) = v < m/k$.

We are going to detect S using the local min cut algorithm. Note that the local min cut algorithm works only when $v < m/k$. That's the reason why we consider these two cases.

Assume that we know v in advance. The algorithm will repeat the following $T = O(\frac{m}{v} \log n)$ times.

- Sample x probability proportional to its degree (i.e. x is chosen with probability $\deg(x)/(2m)$).
- If $\mathsf{Local}(x, v, k)$ return a cut (A, B) has size less than k, return (A, B). Note that here we hope that $\mathsf{Local}(x, v, k)$ is correct with high probability (such as $1 - 1/n^{100}$), so the running time of this step is $O(vk^2 \log n)$ by repeating the local min cut algorithm above $O(\log n)$ times.

The total running time is $O(vk^2 \log nT) = \tilde{O}(mk^2)$

Lemma 2.5. *Suppose there exists a cut $(S, V \setminus S)$ of size less than k where $\mathrm{vol}(S) = v < m/k$, then the above algorithm returns a cut (A, B) of size less than k with probability $1 - 1/n^{10}$.*

Proof. For each iteration, $\Pr[x \in S] = \mathrm{vol}(S)/\mathrm{vol}(V) = v/2m$. By repeating $T = O(\frac{m}{v} \log n)$ times, the probability that $x \in S$ in at least one iteration is $1 - 1/n^{100}$. The local min cut algorithm in this iteration will returns a cut of size less than k with probability $1 - 1/n^{100}$. Therefore, the probability of this algorithm being correct is $(1 - 1/n^{100})^2 > 1 - 1/n^{10}$. □

The last problem is that we don't know $v = \mathrm{vol}(S)$. However, we don't need to know the exact v, we can try $v = 2^i$ for all $1 \le i \le \log m$. As long as $v/2 \le \mathrm{vol}(S) \le v$, the above algorithm will still work by adjusting some constant.

2.3.3 Conclusions

In summary, let $(S, V \setminus S)$ be the cut with size less than k. If $\mathrm{vol}(S), \mathrm{vol}(V \setminus S) \ge m/k$, the algorithm for the balanced case will detect a cut with size less than k with high probability. If $\mathrm{vol}(S) < m/k$, then the algorithm for the unbalanced case will detect a desired cut with high probability. Therefore, if no sparse cut is detected, then with high probability there is no sparse cut in G.

3 Isolating Cuts

Consider an unweighted **undirected** graph $G = (V, E)$ with n vertices and m edges.

Definition 3.1. Let $A, B \subseteq V$ be some vertex sets. A vertex set S is an (A, B)**-cut** if $A \subseteq S$ and $B \subseteq V \setminus S$. A (A, B)**-mincut** is an (A, B)-cut with the cut size $\delta(S)$ minimized.

Given a vertex set $T \subseteq V$ (T is called a terminal set and vertices in T are called terminals), an isolating cuts algorithm will return, for each terminal $v \in T$, a cut S_v which is a $(v, T \setminus v)$-mincut. We also call S_v a **minimum** v**-isolating cut**. Any $(v, T \setminus v)$-cut is a v**-isolating cut**.

We can design a trivial algorithm for this problem by directly computing $(v, T \setminus v)$-mincut for each terminal v, which will invoke $|T|$ max flows. In what follows, we will introduce an algorithm only invoke $O(\log |T|)$ max flows. Strictly speaking, the following algorithm will run max flows on lots of small graphs and the total number of edges in these graph is $O(\log |T| m)$.

3.1 The algorithm

Initially we name vertices in T from $1, \ldots, |T|$ using binary strings of $\lceil \log |T| \rceil$.

The algorithm is as follows. First, for each $i = 1, \ldots, \lceil \log |T| \rceil$

- For each terminal v, color v **red** if the i-th bit of v is 0. Otherwise, color it **blue**.
- Let R_i and B_i denote the set of red and blue nodes.
- Compute the edge set $E_i \subseteq E$ crossing (R_i, B_i)-mincut C_i. See figure 3.

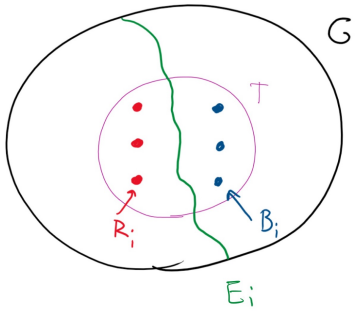


Figure 3: (R_i, B_i)-mincut

Note that deleting $\cup_i E_i$ "partitions" G into connected components where each component contains at most one terminal.

Indeed, for each $v \in T$, let $U_v := \cap_{i:\text{the } i\text{-th bit of } v \text{ is } 0} C_i \cap_{j:\text{the } j\text{-th bit of } v \text{ is } 1} V \setminus C_j$, then U_v is the set of vertices of the connected component in $G \setminus (\cup_i E_i)$ containing v, and formally,

Lemma 3.2. *$U_v \cap T = \{v\}$. That is, U_v isolates v from $T \setminus v$.*

Proof. By definition 3.1, $R_i \subseteq C_i$ and $B_i \subseteq V \setminus C_i$ for all i.

First, because $v \in R_i$ for all i s.t. the i-th bit of v is 0 and $v \in B_i$ for all i s.t. the i-th bit of v is 1, by the defition of U_v we immediately get $v \in U_v$.

Second, there is no another $v' \neq v$ in U_v. For each $v' \neq v$, v' and v will have a different bit. That is $\exists i \in [\lceil \log |T| \rceil]$,without loss of generality, assume $v \in R_i$ and $v' \in B_i$. We have $U_v \subseteq C_i$ and $v' \in B_i \subseteq V \setminus C_i$. Therefore, $v' \notin U_v$.

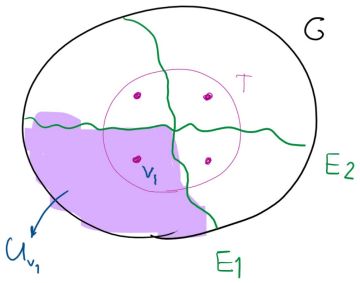


Figure 4: U_{v_1} for $v_1 \in T$, where T is a terminal set with 4 terminals

□

Before we continue introducing the algorithm, we first talk about the submodularity of cuts, which is useful to prove some properties of isolating mincuts.

Lemma 3.3 (Submodularity of cuts)**.** *Let $A, B \subseteq V$,*

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

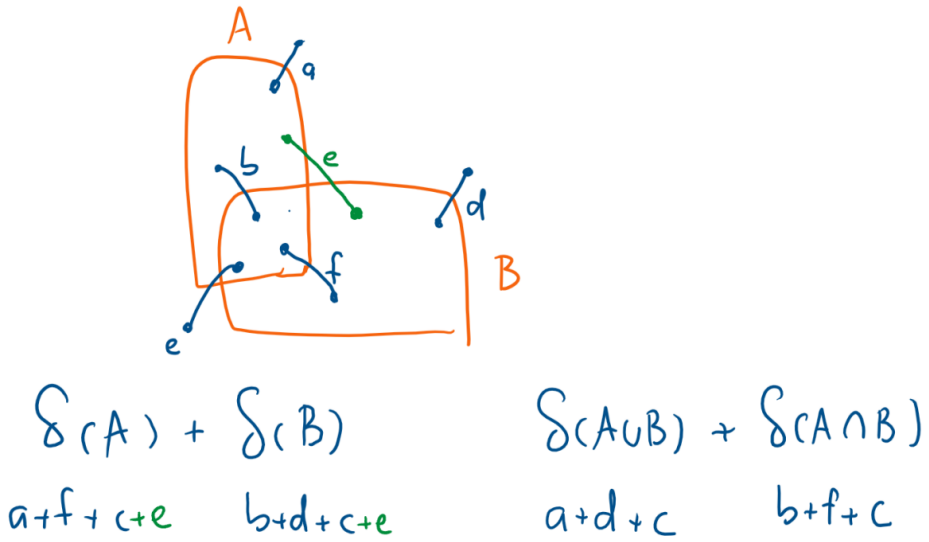


Figure 5: Proof of lemma 3.3

Proof. See figure 5. We can partition V into 4 disjoint regions $A \cap B$, $A \setminus B$, $B \setminus A$, $V \setminus (A \cup B)$, and partition edges into 6 types by considering regions of endpoints. Observe that only edges with two endpoints in $A \setminus B$ and $B \setminus A$ respectively will contribute 1 to $\delta(A)$ and $\delta(B)$ but nothing to $\delta(A \cup B)$ and $\delta(A \cap B)$. Other types of edges will contribute to two sides equally. □

Remark 3.4. If $E(A \setminus B, B \setminus A) = \emptyset$, $\delta(A \cap B) + \delta(A \cup B) = \delta(A) + \delta(B)$.

Corollary 3.5. *If S_1 and S_2 are (A, B)-mincuts, then $S_1 \cap S_2$ and $S_1 \cup S_2$ are also (A, B)-mincuts.*

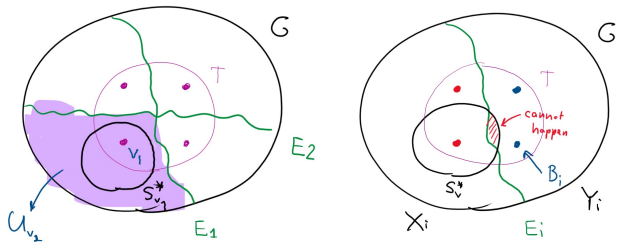
Proof. First observe that if S_1 and S_2 are (A, B)-cuts, then $S_1 \cap S_2$ and $S_1 \cup S_2$ are also (A, B)-cuts. Therefore, $\delta(S_1 \cap S_2), \delta(S_1 \cup S_2) \geq \delta(S_1) = \delta(S_2)$. By lemma 3.3, we immediately have $S_1 \cap S_2$ and $S_1 \cup S_2$ are (A, B)-mincuts. □

Definition 3.6. S_v^* is a inclusion-wise minimal $(v, T \setminus v)$-mincut if each $(v, T \setminus v)$-mincut S_v has $S_v^* \subseteq S_v$.

Lemma 3.7. *There exists a unique inclusion-wise minimal $(v, T \setminus v)$-mincut S_v^*.*

Proof. If there are two $(v, T \setminus v)$-mincut S_v' and S_v'' where $v \in S_v', S_v''$ and neither $S_v' \subseteq S_v''$ nor $S_v'' \subseteq S_v'$, then $S_v' \cap S_v''$ will be a $(v, T \setminus v)$-mincut by the submodularity of cuts (see lemma 3.3 and corollary 3.5). Therefore, let S_v^* be the intersection of all $(v, T \setminus v)$-mincuts and it will be the unique inclusion-wise minimal $(v, T \setminus v)$-mincut. □

Lemma 3.8. $S_v^* \subseteq U_v$.



(a) $S_v^* \subseteq U_v$ (b) $S_v^* \subseteq X$

Figure 6: Proof of lemma 3.8

Proof. See figure 6. $\forall i \in [\lceil \log |T| \rceil]$, let $X_i = C_i$ if $v \in R_i$ and let $X_i = V \setminus C_i$ if $v \in B_i$. Then $U_v = \bigcup_i X_i$. We are going to prove $S_v^* \subseteq X_i$ for all i, which immediately implies $S_v^* \subseteq U_v$.

Suppose for contradiction that $\exists i \in [\lceil \log |T| \rceil]$ s.t. S_v^* is not a subset of X_i. By the submodularity, we have

$$\delta(S_v^* \cap X_i) + \delta(S_v^* \cup X_i) \leq \delta(S_v^*) + \delta(X_i)$$

Since $S_v^* \cap X_i$ is a $(v, T \setminus v)$-cut and S_v^* is the inclusion-wise minimal $(v, T \setminus v)$-mincut, we have $\delta(S_v^* \cap X_i) > \delta(S_v^*)$.

Therefore, $\delta(S_v^* \cup X_i) < \delta(X_i)$. Because $S_v^* \cup X_i$ is a (R_i, B_i)-cut (or (B_i, R_i)-cut), this gives a contradiction.

□

By lemma 3.8, we only need to find a $(v, T \setminus v)$-mincut inside U_v. Formally speaking, for each terminal $v \in T$, we construct a graph G_v by contracting $V \setminus U_v$ into a single vertex t_v, and then compute a (v, t_v)-mincut in G_v as S_v.

Lemma 3.9. *Each S_v is a minimum v-isolating cut.*

Proof. S_v is a $(v, T \setminus V)$-cut because $S_v \subseteq U_v$. Since S_v^* is a (v, t_v)-mincut in G_v, so $\delta(S_v) \leq \delta(S_v^*)$ and S_v is a $(v, T \setminus v)$-mincut in G. □

Lemma 3.10. *The above algorithm of isolating cuts will run max flows on graphs with totally $O(m \log |T|)$ edges.*

Proof. In the first step, we need to compute $\log |T|$ red-blue mincuts in G, which means we run max flows on graphs with $O(m \log |T|)$ edges.

In the second step, we will compute a (v, t_v)-mincut for each $v \in T$. Since $\sum_{v \in T} |E(G_v)| \leq 2|E(G)| = O(m)$, this step we will run max flows on graphs with $O(m)$ edges. □

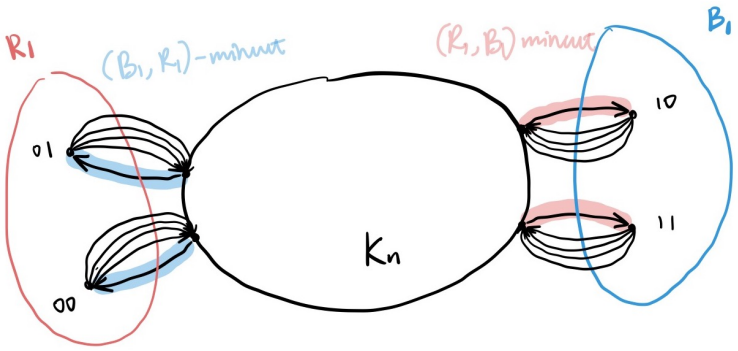
3.2 What goes wrong in directed graphs?

Let S_v^* be the inclusion-wise minimal $(v, T \setminus v)$-mincut, that is $\forall S_v$ where $v \in S_v$ and $\delta_{out}(S_v)$ is minimized, we have $S_v^* \subseteq S_v$. Then, by submodularity of δ_{out}, we can show the existence of S_v^*.

However, for each $i \in [\lceil \log |T| \rceil]$, given R_i and B_i, (R_i, B_i)-mincut and (B_i, R_i)-mincut might be different cuts even with different size and it's not clear which one we should take. Indeed,

- If we compute both of them and take for each v, U_v to be the weakly connected component containing v in G after deleting both cuts' edges, then it's possible that $\sum_{v\in T} U_v = \Omega(mn)$ whence even though the algorithm works ($\forall v \in T$, $S_v^* \subseteq U_v$) the speed of the algorithm is not guaranteed.

  Indeed, consider the graph as below where each terminal is connected to a vertex in a clique with $|T|$ outgoing edges and one ingoing edges, then deleting edges from $\forall i \in [\lceil \log |T| \rceil]$ (R_i, B_i)-mincut and (B_i, R_i)-mincut would be deleting all ingoing edges attach to each terminal, which means $\forall v \in T, \mathrm{vol}(U_v) = \mathrm{vol}(G[V(K_n) \cup \{v\}]) - 2 = \Omega(m)$. Below in the graph is an example when $|T| = 4$.



- If we only compute one of them, then it might be the case that U_v will contain more than one terminal for some v, which means the algorithm won't work at all.

  Indeed, wlog if we only delete (R_i, B_i)-mincut, still consider the example in the previous case, then $\forall i \in [\lceil \log |T| \rceil], v \in B_i$, U_v contains v and all vertices in R_i.

3.3 Application: Steiner Mincut in Undirected Weighted Graphs

In a Steiner mincut problem, we are given an undirected weighted graph $G = (V, E, \omega)$ and a terminal set $T \subseteq V$. The desired output is a T-Steiner mincut, which is a minimum cut S separating some parts of terminals (i.e. $S \cap T \neq \emptyset$ and $S \setminus T \neq \emptyset$).

This problem is a generalization of global mincut and (s,t)-mincut. When $T = V$, it turns to a global mincut problem. When $T = \{s,t\}$, it turns to an (s,t)-mincut problem. Trivially, an algorithm using $|T|-1$ max flow can solve this problem. We just need to fix an $s \in T$ and compute an (s,t)-mincut for each $t \in T$.

In what follows, we will introduce an algorithm using only $\mathrm{poly}\log(n)$ max flows.

3.3.1 The algorithm

Let's assume S^* is a T-Steiner mincut and $|S^* \cap T| \leq |T|/2$.

Lemma 3.11. *For $T' \subseteq T$ s.t. $T' \cap S^* = \{v\}$ and $|T' \setminus S^*| \geq 1$, S^* is a minimum v-isolating cut, namely, a $(v, T' \setminus v)$-mincut.*

Proof. First S^* is a $(v, T' \setminus v)$-cut since S^* separates v and $T' \setminus v$.

Second, S^* is a minimal one. Otherwise there is a smaller cut S' that separates T'. Thus S' also separates T and S^* cannot be a T-Steiner mincut. □

By lemma 3.11, we just need to construct a $T' \subseteq T$ s.t. $|T' \cap S^*| = 1$ and $|T' \setminus S^*| \geq 1$. Then we can find S^* by running an isolating cut algorithm on T'. We will construct T' by random sampling.

First we guess $|S^* \cap T|$. By iterating i from 0 to $\lfloor \log(|T|/2) \rfloor$, in one iteration we have $2^i \leq |S^* \cap T| \leq 2^{i+1}$. We construct T' by sampling each vertex in T independently with probability $p = 1/2^i$. Let $k = |S^* \cap T|$. We have $1/p \leq k \leq 2/p$ and $1/p \leq |T|/2$. Further,

$$\begin{aligned}
\Pr[|S^* \cap T'| = 1] &= kp(1-p)^{k-1} \\
&\quad \text{since exactly one vertex is chosen into } T' \text{ among } k \text{ vertices} \\
&\geq (1-p)^{2/p} \\
&\quad \text{by } k \leq 2/p \\
&= \Theta(1).
\end{aligned}$$

and

$$\begin{aligned}
\Pr[|T' \setminus S^*| \geq 1] &= \Pr[|T' \cap (T \setminus S^*)| \geq 1] \\
&\quad \text{since at least one vertex in } T \setminus S^* \text{ is chosen} \\
&= 1 - (1-p)^{|T \setminus S^*|} \\
&\geq 1 - (1-p)^{|T|/2} \\
&\quad \text{by } |T \setminus S^*| \geq |T|/2 \text{ since } |T \cap S^*| \leq |T|/2 \\
&\geq 1 - (1-p)^{1/p} \\
&\quad \text{by } 1/p \leq |T|/2 \\
&= \Theta(1).
\end{aligned}$$

Therefore,

$$\Pr[|S^* \cap T'| = 1 \text{ and } |T' \setminus S^*| \geq 1] = \Theta(1).$$

By running isolating cuts on $O(\log n)$ independent samples $T'_1, \ldots, T'_{O(\log n)}$, we can find S^* with high probability.

The above algorithm invokes isolating cuts algorithm $O(\log^2 n)$ times summing over $O(\log n)$ levels of i and $O(\log n)$ samples for each i, so it needs $O(\log^3 n)$ max flows.

4 Extensions and more applications

Both local min cuts and isolating cuts have versions about vertex-cuts.

For local vertex-mincut problem $\mathsf{Local}(x, \nu, k)$, the key idea is that we can convert a directed graph G into another directed graph G_{split} by

- splitting each vertex v into two vertex v_{in} and v_{out} (except x and we let $x_{\text{in}} = x_{\text{out}} = x$) and
- converting any edge $(u, v) \in E(G)$ into edge $(u_{\text{out}}, v_{\text{in}})$ and adding an edge $(v_{\text{in}}, v_{\text{out}})$ for each $v \neq x$.

Observe that a vertex-mincut S s.t. $x \in S$ in G is kind of one-to-one corresponding to an edge-mincut S' s.t. $x \in S'$ in G_{split}. We can solve the local vertex-mincut problem on G by running the local edge-mincut problem on G_{split}.

For isolating vertex-cuts, we assume that the given terminal T is an independent set in G. The key point is that the submodularity still holds on vertex cuts, so we can still get the v-isolating vertex-mincut by only considering the small graph U_v.

There are some more applications of these two techniques. For local min cut technique, we can use it to solve k-vertex connectivity[3] in $\tilde{O}(mk^2)$ time which is the fastest algorithm when k is small. Some other applications are property testing and vertex sparsifiers.

For isolating cuts, there are some applications on Gomory-Hu trees, general symmetric submodular function minimization and connectivity augmentation.

[3]See paper [FNSYY'19] *Computing and Testing Small Connectivity in Near-Linear Time and Queries via Fast Local Cut Algorithms* and video https://www.youtube.com/watch?v=V1kq1filhjk