Topics in TCS: Randomized Algorithms September 12, 2025

Lecture 1: Introduction + Cuts in Graphs via Randomization

Lecturer: Ola Svensson *Scribes: Ola Svensson based on Anupam Gupta's handwritten notes*

1 Introduction

- Welcome
- Ola Svensson (*ola.svensson@epfl.ch*)
- *http://theory.epfl.ch/courses/topicstcs* (static). Most important resource will be the moodle page.
- Goal of course: cool results, nice powerful techniques that may be useful elsewhere.

1.1 Grading

- $4-5$ Homeworks plus oral or written exam on homeworks [75%].
- Project [25%].

1.2 Some Topics We can Cover

1. Introduction + Cuts in Graphs via Randomization
2. Probabilistic method, Balls and Bins. Concentration inequalities
3. Chernoff bounds, clever union bounds, applications
4. Lovasz Local Lemma
5. Martingales and concentration
6. Gaussian Random Variables (SDP rounding, discrepancy)
7. Random walks
8. Markov Chains
9. Decision Making Online: Prophets, Secretaries, Bandits

1.3 Why Randomized Algorithms?

Randomness is an amazing algorithmic resource, a "superpower". It often leads to:

- Simpler, better, or faster algorithms.
- Elegant algorithms and analysis.

For some problems, we know of randomized algorithms but no deterministic ones (yet). An important example is to generate a prime p within a given interval $[n, 2n]$. In some models of computation, like streaming, online, and distributed algorithms, randomness gives extra power. Working without randomness is like "tying one's hands behind one's back."

1.3.1 Several Algorithmic Reasons to Use Randomness

- **Random Sampling:** Useful when we know many good solutions exist but don't know how to find one because we don't understand the structure.
- **Random Walk:** A process to explore the structure of a sample space quickly (either exploring all of it or sampling uniformly from it, e.g., Markov Chain Monte Carlo).
- **Sparsification:** A random subset often preserves the essential structure of a larger object, allowing us to focus our work on this smaller subset.
- **Load Balancing / Symmetry Breaking:** For example, randomly throwing balls into bins results in an approximately equal load in each bin.
- **Making Algorithms Unpredictable:** In online models, an adversary that knows the algorithm's actions can devise a worst-case input. Adding randomness (against an "oblivious" adversary) gives us more power.
- **Powerful Language of Probability Theory:** Arguing about high-dimensional geometry, for example, is often difficult, but probabilistic arguments provide a convenient and powerful tool.
- **Existence Proofs:** The Probabilistic Method is a powerful tool for proving the existence of certain combinatorial objects.

1.3.2 Full Disclosure: Reasons to be Cautious

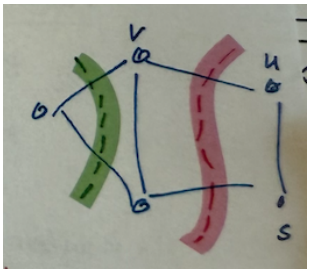
- Getting truly random bits can be challenging, so we may want to reduce an algorithm's dependence on randomness.
- There is significant work on using weak random sources to amplify randomness (under complexity assumptions). Often, it's enough to work with limited randomness. Check the terms "Extractors" and "Pseudorandom generators"
- Sometimes, generic "derandomization" theorems can be used.
- A fundamental question in computational complexity is whether randomness is truly necessary. Is $P = BPP$? This is what I would say a fantastic mystery in computer science.

2 Randomized Minimum Cut Algorithm

Given an undirected graph $G = (V, E)$, say unweighted here for simplicity, we want to find the (global) minimum cut. This is a partition of the vertices with the fewest edges crossing it.

Formally, for any given subset of vertices $S \subset V$ (where $S \neq \emptyset$ and $S \neq V$), the set of edges forming the cut is $\delta(S) = \{(u, v) \in E \mid u \in S, v \notin S\}$. We call $\delta(S)$ the "boundary of S" or the "cut of S". We want to find a set S that minimizes $|\delta(S)|$.

In the example graph below, both the red and green dashed lines represent minimum cuts. Each one has a size of 2.



2.1 Strawman Algorithms

- Enumerate over all cuts (not very clever ;)
- Since we can find minimum s-t cuts in polynomial time (using max flow), we could try all pairs of vertices (s, t) and compute the minimum s-t cut. This takes $O(n^2)$ minimum s-t cut computations.
- We can do better by observing that if $\delta(S)$ is a min-cut then so is $\delta(V \setminus S)$. Hence we can fix s and only test the $n-1$ possibilities for t, leading to $O(n)$ minimum s-t cut computations.
- We will see in the end of this lecture that recent techniques imply that we can find a min-cut by using $poly \log n$ many max-flow computations!!

3 Karger's Beautiful Contraction Algorithm

Discovered by Karger in 1993. It is a beautiful randomized algorithm for finding a min-cut that also leads to structural statements of graphs.

3.1 Algorithm Description

Assume $G = (V, E)$ is connected, otherwise the problem is trivial.

While there are more than two vertices in the graph:

1. Pick a uniformly random edge $\{u, v\}$
2. Contract the edge (u, v), merging vertices u and v into a single vertex.
3. Remove self-loops created by the contraction, but keep parallel edges

When the process ends (after $n-2$ iterations), graph has 2 nodes. Each corresponds to some subset of nodes contracted into it. Output one of these subsets.

3.2 Analysis

The algorithm succeeds if it never contracts an edge that is part of the minimum cut. (Note that there may be many min-cuts but for the analysis we fix an arbitrary one). Let's analyze the probability of this happening.

Probability of Success in a Single Run

Fix a particular minimum cut C, which corresponds to a partition $(S, V \setminus S)$. Let the size of this cut be $\lambda = |\delta(S)|$. The algorithm finds this cut if and only if it never contracts an edge in $\delta(S)$.

First, a simple but crucial observation: for any vertex v, its degree must be at least λ.

$$\text{degree}(v) \geq \lambda \quad \forall v \in V$$

If this were not the case, say $\text{degree}(v) < \lambda$, then the cut $(\{v\}, V \setminus \{v\})$ would have size $\text{degree}(v)$, contradicting that λ is the size of the minimum cut.

By the handshaking lemma, the sum of degrees is twice the number of edges, $|E|$: $\sum_{v \in V} \text{degree}(v) = 2|E|$. Using our observation, we get:

$$2|E| = \sum_{v \in V} \text{degree}(v) \geq \sum_{v \in V} \lambda = n\lambda \implies |E| \geq \frac{n\lambda}{2}$$

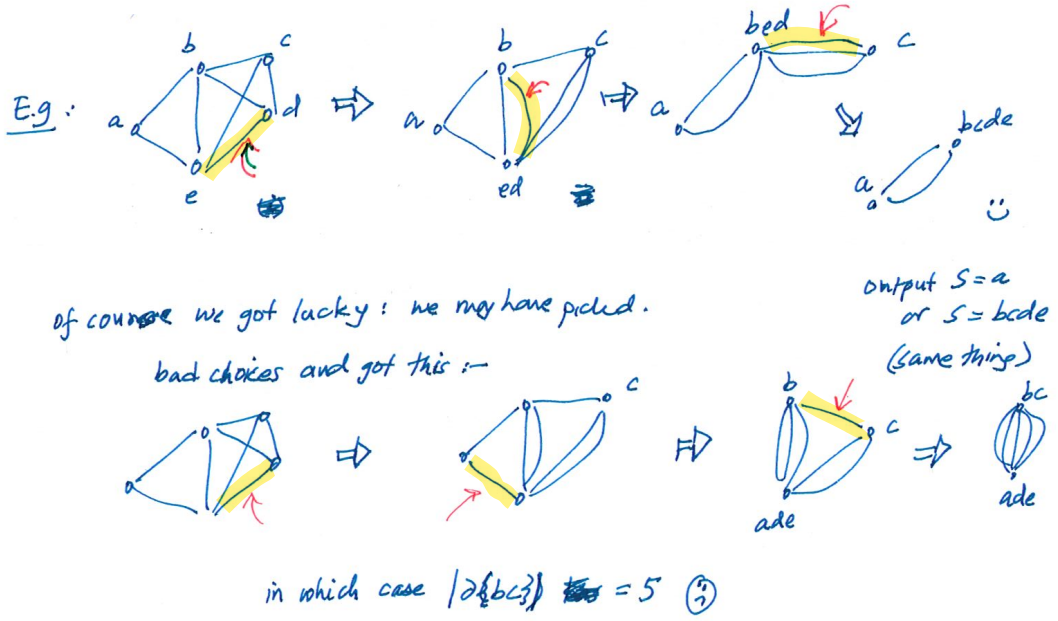


Figure 1: Example of Contraction Algorithm.

The probability of picking an edge from our specific min-cut C in the first step is:

$$\Pr(\text{contract an edge in } C) = \frac{|C|}{|E|} = \frac{\lambda}{|E|} \leq \frac{\lambda}{n\lambda/2} = \frac{2}{n}$$

So, the probability that the min-cut C survives the first step is at least $1 - \frac{2}{n}$.

Now, let's consider the entire process. Let G_i be the graph after i contractions, with $n - i$ vertices. If our min-cut C has survived the first i steps, it is still a cut in G_i of size λ. Crucially, it must also be a **minimum cut** in G_i, since any cut in G_i corresponds to a cut of the same size in the original graph G, and thus must have size at least λ.

The number of edges in G_i is at least $\frac{(n-i)\lambda}{2}$ by the same logic as before. The probability that we contract an edge from C at step $i+1$, given that it has survived so far, is at most $\frac{\lambda}{|E(G_i)|} \leq \frac{2}{n-i}$. The probability that C survives step $i+1$ is therefore at least $1 - \frac{2}{n-i}$.

The probability that the algorithm successfully finds C is the probability it survives all $n-2$ contraction steps:

$$\begin{aligned}\Pr(\text{min-cut survives}) &\geq \left(1 - \frac{2}{n}\right)\left(1 - \frac{2}{n-1}\right)\cdots\left(1 - \frac{2}{3}\right) \\ &= \frac{n-2}{n} \cdot \frac{n-3}{n-1} \cdot \frac{n-4}{n-2} \cdots \frac{2}{4} \cdot \frac{1}{3}\end{aligned}$$

This is a telescoping product. The terms $(n-2)$ through 3 in the numerators cancel with those in the

denominators, leaving:

$$\Pr(\text{min-cut survives}) \geq \frac{2 \cdot 1}{n(n-1)} = \frac{2}{n(n-1)} = \frac{1}{\binom{n}{2}}$$

This is good news and bad news. We have a non-zero probability of finding the min-cut, but this probability is small, e.g., for $n = 100$, it is about $1/5000$.

Boosting the Success Probability

To fix the low success probability, we repeat the experiment! This is a standard technique called probability amplification.

Let $p \geq \frac{1}{\binom{n}{2}}$ be the probability of success in a single run. If we run the algorithm T times, the probability of failing every single time is:

$$\Pr(\text{fail in all } T \text{ runs}) \leq (1-p)^T$$

Using the useful inequality $1 - x \leq e^{-x}$, we can bound this failure probability:

$$\Pr(\text{fail in all } T \text{ runs}) \leq (e^{-p})^T = e^{-pT}$$

We want this failure probability to be very small, say δ. We can achieve this by setting T appropriately. Let's set $e^{-pT} = \delta$. Solving for T gives $T = \frac{\ln(1/\delta)}{p}$.

Let's aim for a success probability that is "high," meaning it approaches 1 as n grows. We can choose $\delta = 1/n^{10}$. With our lower bound for p, the number of repetitions should be:

$$T = \binom{n}{2} \ln(n^{10}) = \binom{n}{2} \cdot 10 \ln n = O(n^2 \log n)$$

By running the Contraction Algorithm $T = O(n^2 \log n)$ times and returning the smallest cut found, we find the global minimum cut with probability at least $1 - \delta = 1 - 1/n^{10}$.

Runtime Analysis

Let's analyze the runtime of this "boosted" algorithm.

- A single run of the Contraction Algorithm performs $n - 2$ contractions. A naive implementation of edge contraction can take $O(m)$ time, but with an adjacency list representation, the total time for one run can be implemented in $O(n^2)$ time.
- We repeat this $T = O(n^2 \log n)$ times.

The total runtime is therefore:

$$\text{Runtime} = (\text{Number of runs}) \times (\text{Time per run}) = O(n^2 \log n) \cdot O(n^2) = O(n^4 \log n)$$

This is a polynomial-time algorithm, but not amazing. A more careful implementation and a clever idea by Karger and Stein can improve this runtime significantly. We cover Karger-Stein in Algorithms II and Anupam also sketches it in his notes. We skip it here.

4 The Probabilistic Method Strikes: Bounding The Number of Min-Cuts

The analysis of Karger's algorithm has a surprising and beautiful consequence. We used properties of graphs (e.g., the minimum degree is at least λ) to analyze a randomized algorithm. Now, we can use the existence of this randomized algorithm to prove a deterministic, structural fact about graphs. This is a classic example of the **Probabilistic Method**.

Let's recap. We fixed an arbitrary min-cut, say $\delta(S)$, and showed that the Contraction Algorithm returns it with a probability of at least $1/\binom{n}{2}$.

$$\Pr(\text{C.A. returns } \delta(S)) \geq \frac{1}{\binom{n}{2}}$$

Now, suppose there is another, distinct min-cut, $\delta(T)$. The same analysis holds for $\delta(T)$, so the probability that the algorithm returns $\delta(T)$ is also at least $1/\binom{n}{2}$.

$$\Pr(\text{C.A. returns } \delta(T)) \geq \frac{1}{\binom{n}{2}}$$

The key insight is that in a single run, the algorithm can only return one cut. Therefore, the event "the algorithm returns $\delta(S)$" and the event "the algorithm returns $\delta(T)$" are **disjoint** events.

This leads to the following theorem, which makes no mention of probability or algorithms.

Theorem. Any connected, undirected graph $G = (V, E)$ on n vertices has at most $\binom{n}{2}$ distinct minimum cuts.

Proof. Let $C_1, C_2, \ldots, C_k$ be the set of all distinct minimum cuts in the graph G. Let E_i be the event that the Contraction Algorithm outputs the cut C_i. From our analysis, we know that for any specific min-cut C_i, the probability of the algorithm returning it is:

$$\Pr(E_i) \geq \frac{1}{\binom{n}{2}}$$

Since the algorithm can only output one cut, the events $E_1, E_2, \ldots, E_k$ are all mutually disjoint. By the union bound (or the axiom of finite additivity for probabilities), the probability that one of them occurs is the sum of their individual probabilities. This total probability cannot exceed 1.

$$\sum_{i=1}^{k} \Pr(E_i) \leq 1$$

Substituting our lower bound for each probability:

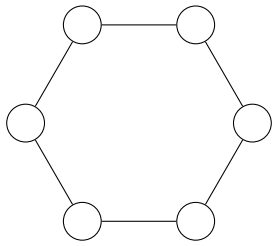
$$\sum_{i=1}^{k} \frac{1}{\binom{n}{2}} \leq 1$$

This simplifies to:

$$k \cdot \frac{1}{\binom{n}{2}} \leq 1 \implies k \leq \binom{n}{2}$$

Thus, the number of distinct minimum cuts, k, is at most $\binom{n}{2}$. ■

Tightness of the Bound. This bound is tight. Consider a simple cycle graph on n vertices, C_n. The minimum cut size is 2, achieved by removing any two edges. The number of ways to choose any two edges from the n edges of the cycle is precisely $\binom{n}{2}$. Each such pair of edges partitions the vertices into two disjoint paths, defining a distinct minimum cut of size 2. Therefore, an n-cycle has exactly $\binom{n}{2}$ minimum cuts.



A 6-cycle has $\binom{6}{2} = 15$ minimum cuts of size 2.

This is a perfect illustration of the probabilistic method: we proved a purely combinatorial statement about the structure of graphs by designing and analyzing a randomized algorithm. The algorithm itself is just a tool in the proof; the final theorem is a deterministic fact, independent of any notion of randomness.

5 Isolating Cuts - Min Cut Differently

This section introduces another, more recent, randomized approach to finding minimum cuts in graphs. This technique is powerful and potentially generalizable to other problems.

Basic Subroutine: Min Isolating Cuts. Given a graph $G = (V, E)$ and a set of "terminal" vertices $R \subseteq V$, the goal is to find, for each vertex $v \in R$, a cut $(S_v, V \setminus S_v)$ that is a minimum cut separating v from the rest of the terminals, $R \setminus \{v\}$. Formally, for each $v \in R$, we want to find:

$$S_v = \arg\min\{|\delta(S)| : v \in S \text{ and } S \cap R = \{v\}\}$$

Such a cut $\delta(S_v)$ is called an **isolating cut** for v with respect to R. This definition also works for graphs with non-negative edge weights, but we will focus on the unweighted case for simplicity.

Theorem 1 *(Li, Panigrahi 2021). Given a graph G and a terminal set R, it is possible to find all minimum isolating cuts for all vertices in R in time equivalent to solving $O(\log n)$ max-flow instances on a related graph.*

Since recent breakthroughs have shown that max-flow can be computed in near-linear time, this implies that finding all isolating cuts can also be done in near-linear time.

Note. The structure of isolating cuts depends heavily on the choice of the terminal set R.

- If $R = \{s, t\}$, the isolating cut for s (and for t) is precisely the global minimum $s - t$ cut.
- If $R = V$, then for any vertex v, the only set S satisfying the condition $v \in S$ and $S \cap R = \{v\}$ is the singleton set $S_v = \{v\}$. The isolating cut is simply the set of edges incident to v, and its size is the degree of v.
- For an intermediate choice of R, this subroutine can reveal other interesting cut structures within the graph.

This raises two natural questions:

Q1: How can one prove the theorem above? (This involves a clever deterministic reduction, which uses the submodularity of the cut function).

Q2: How can this subroutine for finding isolating cuts help us find the *global* minimum cut of the graph?

Regarding the second question, a naive approach would be to enumerate over all pairs of vertices $\{s, t\}$, set $R = \{s, t\}$ for each pair, and compute the isolating cut (i.e., the min $s - t$ cut). The smallest cut found over all $O(n^2)$ pairs would be the global min-cut. This is equivalent to the standard reduction to $O(n^2)$ max-flow computations. The key question is whether we can use this new primitive to do better. We start by answering the second question using randomness.

6 Fast Randomized Algorithm for Global Min Cut using Isolating Cuts (Q2)

The idea is to use the isolating cut subroutine to find the global minimum cut. Suppose $\delta(C)$ is a minimum cut, where C is the smaller side of the partition, so $|C| \leq n/2$. If we could sample a set of terminals R such that it contains exactly one vertex from C, say $R \cap C = \{x\}$, then $\delta(C)$ is a cut that isolates x from the other terminals in $R \setminus \{x\}$. Indeed, in that case, $\delta(C)$ would be a *minimum* isolating cut for x with respect to R. The idea is as follows. Suppose C has cardinality $k^* = |C|$. If we sample each vertex independently with probability $p = 1/k^*$,

$$\Pr[|R \cap C| = 1] = \binom{k^*}{1} p(1-p)^{k^*-1} = k^* \cdot \frac{1}{k^*} \cdot \left(1 - \frac{1}{k^*}\right)^{k^*-1} = \left(1 - \frac{1}{k^*}\right)^{k^*-1} \approx 1/e = \Omega(1).$$

So if we repeat this experiment $\Theta(\log n)$ times,

$$\Pr[\text{we fail each time}] \leq 1/n^{10}.$$

The problem is that we do not know the size of C, say $k^* = |C|$, to choose an appropriate sampling probability. The solution is to try all possible sizes, grouped into powers of two.

This leads to the following algorithm:

1. For each size class $k \in \{1, 2, 4, \ldots, 2^{\lfloor \log_2 n \rfloor}\}$:
   (a) Repeat $O(\log n)$ times:
      i. Sample a set of terminals R by including each vertex $v \in V$ independently with probability $1/k$.
      ii. Find all minimum isolating cuts for the pair (G, R).
2. Return the smallest cut found across all iterations.

Theorem 2 *The algorithm above finds a global minimum cut with probability* $1 - 1/\mathit{poly}(n)$.

Proof Let $\delta(C)$ be a fixed global minimum cut, and let $k^* = |C|$ be the number of vertices on the smaller side of the cut, so $k^* \leq n/2$.

The algorithm iterates through $k = 1, 2, 4, \ldots$. There must be one value of k in this sequence such that $k^* \leq k < 2k^*$. Let us fix this particular value of k and analyze the probability of success in a single trial within its corresponding block of iterations.

In such a trial, we sample a terminal set R where each vertex is included with probability $p = 1/k$. Let E be the event that exactly one vertex from C is chosen to be in R, i.e., $|R \cap C| = 1$. The probability of this event is:

$$\Pr[E] = \binom{k^*}{1} p(1-p)^{k^*-1} = k^* \cdot \frac{1}{k} \cdot \left(1 - \frac{1}{k}\right)^{k^*-1}$$

Using our bounds on k (i.e., $1/2 < k^*/k \leq 1$ and $k^* - 1 < k$), we can lower-bound this probability:

$$\Pr[E] > \frac{1}{2} \cdot \left(1 - \frac{1}{k}\right)^k$$

For $k \geq 2$, the term $(1 - 1/k)^k$ is bounded between $1/4$ (for $k = 2$) and $1/e$ (as $k \to \infty$). Thus, $\Pr[E]$ is lower-bounded by a constant, say $c = 1/8$. So, $\Pr[E] = \Omega(1)$.

Now, let's condition on the event E occurring. Suppose $R \cap C = \{x\}$. The cut $\delta(C)$ isolates vertex x from all other terminals in R (which are all in $V \setminus C$). Let $\delta(S_{min})$ be the minimum isolating cut for x with respect to R, which is what our subroutine finds. By definition of a minimum isolating cut, $|\delta(S_{min})| \leq |\delta(C)|$. However, $\delta(C)$ is a global minimum cut, so its size is the minimum possible for any cut in the graph. Therefore, $|\delta(C)| \leq |\delta(S_{min})|$. Combining these, we get $|\delta(S_{min})| = |\delta(C)|$. This means that if event E occurs, the isolating cut subroutine for vertex x will indeed find a cut of the global minimum size.

So, for the "correct" choice of k, the probability of finding the minimum cut in a single trial is at least $\Pr[E] = \Omega(1)$. Let's say this success probability is $p_{succ} \geq c$. The algorithm performs $T = O(\log n)$ repetitions for this k. Let $T = c' \log n$ for a sufficiently large constant c'. The probability of failing to find the min-cut in all T of these repetitions is:

$$\Pr[\text{fail all } T \text{ times}] \leq (1 - p_{succ})^T \leq (1 - c)^{c' \log n}$$

Using the inequality $1 - x \leq e^{-x}$, this is at most $e^{-c \cdot c' \log n} = n^{-c \cdot c'}$. By choosing c' large enough, we can make this failure probability $1/\text{poly}(n)$.

Therefore, with high probability, the block of iterations corresponding to the correct size class k will find a global minimum cut. Since the algorithm returns the overall minimum cut found, it will find the global minimum cut with probability $1 - 1/\text{poly}(n)$. ■

7 Finding Isolating Cuts via Max Flow (Q1)

We now address the first question (Q1): how can we efficiently find the minimum isolating cuts? The proof of the theorem by Li and Panigrahi relies on a very nice idea using the "submodularity" of the cut function.

7.1 Submodularity of the Cut Function

Definition 1 *A function $f : 2^V \to \mathbb{R}$ (which maps subsets of vertices to reals) is called **submodular** if for all $A, B \subseteq V$,*

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

An equivalent and often more informative definition captures the idea of "decreasing marginal returns": f is submodular if for all $A \subseteq B \subseteq V$ and $e \notin B$,

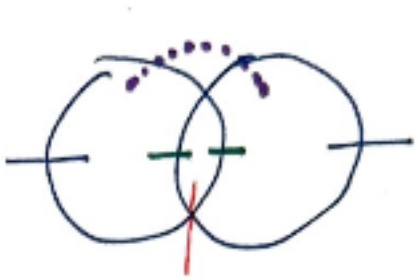
$$f(A \cup \{e\}) - f(A) \geq f(B \cup \{e\}) - f(B).$$

(That is, adding e adds more value to a smaller subset A than to a superset B.)

Notation. Let $d(S) = |\delta(S)|$ denote the size of the cut (number of edges crossing $(S, V \setminus S)$). This definition also works if we have non-negative weights w_e, where $d(S) = \sum_{e \in \delta(S)} w_e$.

Lemma 3 *The cut function d is submodular.*

Proof [Proof sketch by picture]



We can verify this by analyzing the contribution of individual edges to both sides of the inequality. ■

Lemma 4 *The cut function d also satisfies $d(A) + d(B) \geq d(A \setminus B) + d(B \setminus A)$ for all $A, B \subseteq V$.*

Proof [Proof sketch by same picture as above] ■

7.2 Structure of Isolating Cuts: Uncrossing

Submodularity allows us to "uncross" cuts, which leads to a crucial structural property of minimum isolating cuts.

Lemma 5 *We can choose a collection of minimum isolating cuts $\{S_v\}_{v \in R}$ such that the sets S_v are mutually disjoint.*

Proof Suppose we have a collection of minimum isolating cuts, and there exist $u, v \in R$ $(u \neq v)$ such that $S_u \cap S_v \neq \emptyset$. Define $S'_u = S_u \setminus S_v$ and $S'_v = S_v \setminus S_u$.

First, we verify that S'_u is an isolating cut for u. Since S_v isolates v, $S_v \cap R = \{v\}$. As $u \in R$ and $u \neq v$, we must have $u \notin S_v$. Thus $u \in S_u \setminus S_v = S'_u$. Furthermore, S'_u contains no other terminals since it is a subset of S_u. So S'_u isolates u. Similarly, S'_v isolates v.

By Lemma 4, we have:

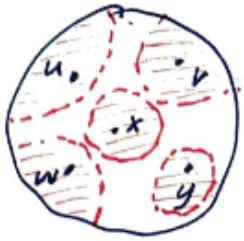
$$d(S_u) + d(S_v) \geq d(S'_u) + d(S'_v).$$

However, because S_u and S_v are *minimum* isolating cuts, we must have $d(S_u) \leq d(S'_u)$ and $d(S_v) \leq d(S'_v)$.

Combining these inequalities implies that they must all be equalities. Hence, $d(S_u) = d(S'_u)$ and $d(S_v) = d(S'_v)$. Therefore, S'_u and S'_v are also minimum isolating cuts.

We can replace S_u, S_v with S'_u, S'_v. This process (called "uncrossing") can be repeated until all sets are disjoint. ■

This shows that we can describe the minimum isolating cuts concisely as they can be chosen to be disjoint:



But can we find them fast? Yes, using a clever application of the above properties of the cut function $d(\cdot)$. We will focus on finding the inclusion-wise smallest minimum isolating cuts, denoted $\{S^*_x\}_{x \in R}$.

7.3 The Isolation Algorithm

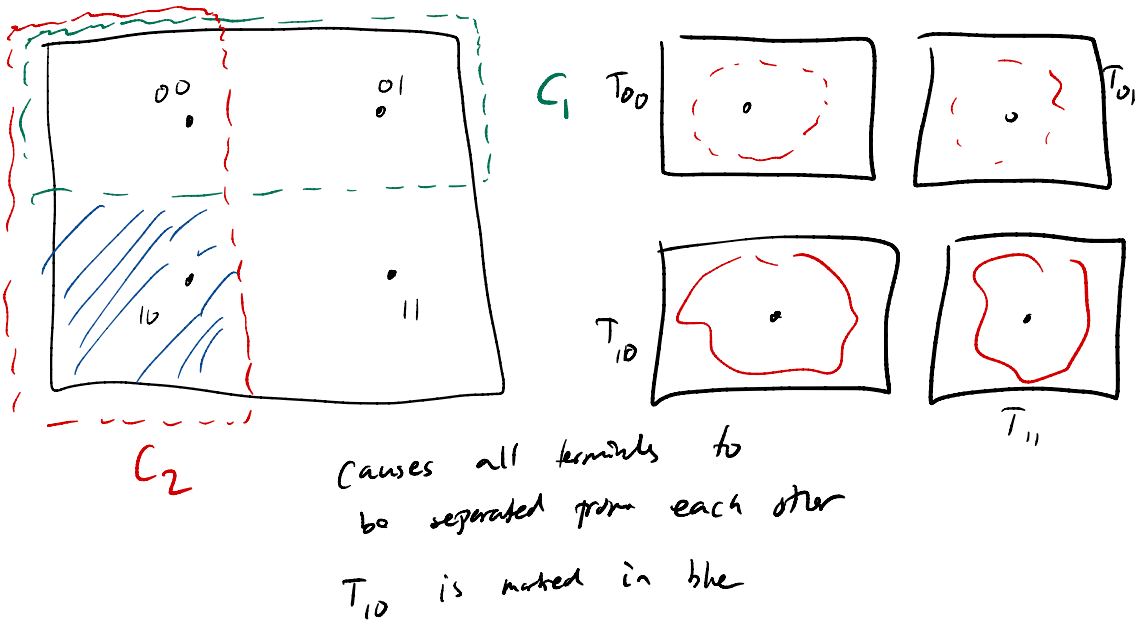


Figure 2: Illustration of the Isolation Algorithm.

The algorithm uses a deterministic divide-and-conquer approach based on separating the terminals R (see also Figure 2 for an illustration).

1. Label the vertices in R arbitrarily using $L = \lceil \log_2 |R| \rceil$ bits.
2. For each bit position $i = 1, \ldots, L$:
   (a) Define $R_i = \{v \in R \mid \text{the } i\text{-th bit of } v\text{'s label is 1}\}$ (and $R \setminus R_i$ are those with bit 0).
   (b) Find a minimum cut $\delta(C_i)$ that separates R_i from $R \setminus R_i$. (This can be done via one max-flow computation by contracting R_i to a source and $R \setminus R_i$ to a sink).

This procedure uses $O(\log |R|) = O(\log n)$ max-flow computations. These cuts separate all the terminals from each other, because any two distinct terminals $x, y \in R$ must differ in at least one bit position i, and the cut $\delta(C_i)$ separates them.

Let T_x be the "piece" containing $x \in R$ resulting from this partition. Formally, for each i, let C_i^x be the side of the cut $\delta(C_i)$ containing x. Then $T_x = \bigcap_i C_i^x$.

The crucial insight is that the smallest minimum isolating cut for x is contained within T_x.

Claim 6 *For all $x \in R$, the inclusion-wise smallest minimum isolating cut S_x^* satisfies $S_x^* \subseteq T_x$.*

Proof To show $S_x^* \subseteq T_x$, we must show $S_x^* \subseteq C_i^x$ for all i. Fix an index i. Without loss of generality (by flipping C_i and $V \setminus C_i$ if needed), assume $x \in C_i$, so $C_i^x = C_i$. We want to show $S_x^* \subseteq C_i$.

Suppose for the sake of contradiction that $S_x^* \not\subseteq C_i$.

Consider the set $S_x^* \cap C_i$. Since x is in both sets, x is in the intersection. Furthermore, S_x^* contains no other terminals. Thus, $S_x^* \cap C_i$ is an isolating cut for x.

Since we assumed $S_x^* \not\subseteq C_i$, the set $S_x^* \cap C_i$ is strictly smaller (inclusion-wise) than S_x^*. By the definition of S_x^* as the inclusion-wise smallest *minimum* isolating cut, it must be that $S_x^* \cap C_i$ has a strictly larger cut value:

$$d(S_x^* \cap C_i) > d(S_x^*).$$

(If the capacities were equal, $S_x^* \cap C_i$ would be a strictly smaller set achieving the minimum, contradicting the definition of S_x^*).

Now we use submodularity (Lemma 3) on S_x^* and C_i:

$$d(S_x^*) + d(C_i) \geq d(S_x^* \cap C_i) + d(S_x^* \cup C_i).$$

Since $d(S_x^* \cap C_i) > d(S_x^*)$, this implies:

$$d(C_i) > d(S_x^* \cup C_i).$$

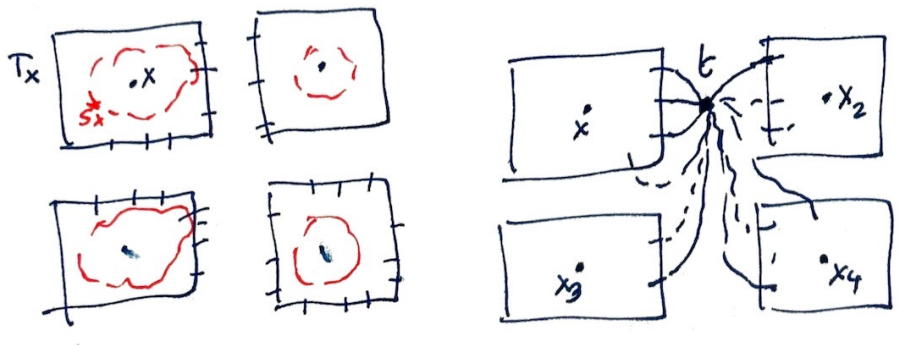
Now consider the terminals contained in $S_x^* \cup C_i$. The set of terminals in C_i is either R_i or $R \setminus R_i$. S_x^* only contains the terminal x. Since $x \in C_i$, the set of terminals in $S_x^* \cup C_i$ is the same as the set of terminals in C_i.

Thus, $\delta(S_x^* \cup C_i)$ is a cut separating the same terminals as $\delta(C_i)$, but with a strictly smaller size. This violates the fact that $\delta(C_i)$ is a minimum cut for that separation.

Therefore, the assumption $S_x^* \not\subseteq C_i$ must be false. This completes the proof. ■

7.4 Final Construction and Recap

The claim shows that we now just need to separate x from the boundary of T_x, for each $x \in R$. We can combine these tasks efficiently. We construct a single auxiliary graph instance (by introducing $O(|E|)$ extra edges that effectively model the boundaries, e.g., by redirecting edges leaving T_x to a common sink), and run a final max-flow/min-cut computation on this combined instance. The picture looks like:



Recap. To summarize the powerful results derived from this approach:

1. We can find the minimum isolating cuts **deterministically** in time equivalent to $O(\log n)$ max-flow computations. (This answers Q1 and proves the theorem by Li and Panigrahi).

2. We can use this isolating cuts routine $O(\log^2 n)$ times (as analyzed in Section 6: $O(\log n)$ size classes times $O(\log n)$ repetitions) to find the global minimum cut with high probability (**randomized**).

This yields a randomized algorithm for the global minimum cut using a total of $O(\log^3 n)$ max-flow computations!