# Algorithms and Probability (FS2025)
# Week 13

Rui Zhang

May 21, 2025

## Contents



## 1 Content

Oh boy, this week is gonna be funny.

### 1.1 Min-Cut (Initial Attempt)

Given a multigraph $G$, we want to find the minimum cut of this multigraph, more precisely $\mu(G)$, the minimum number of edges needed to disconnect a graph. Of course, we already have the means to solve this problem. By using a small detail I left out last week:

> **Theorem:** (Max-Flow Algorithm using Dynamic Trees) The maximum flow in a network can be found in $O(mn \log n) = O(n^3 \log n)$ time,

we can, in addition to the max-flow-min-cut theorem, find the cardinality of a minimum cut in $O(n^4 \log n)$ time, where we model the network by assigning integer weights according to the number of edges. Here, the extra $n$ factor comes from the fact that we don't want to find an $s$-$t$-cut, for concrete $s$ and $t$, but some arbitrary cut, which means that we will have to fix $s$ and iterate through all $n-1$ options of $t$.

### 1.2 Min-Cut (2nd Attempt)

Our goal now is to develop some sort of proabbilistic algorithm which has a better runtime.

To this end, develop the notion of a edge-contraction:

**Example 1.**

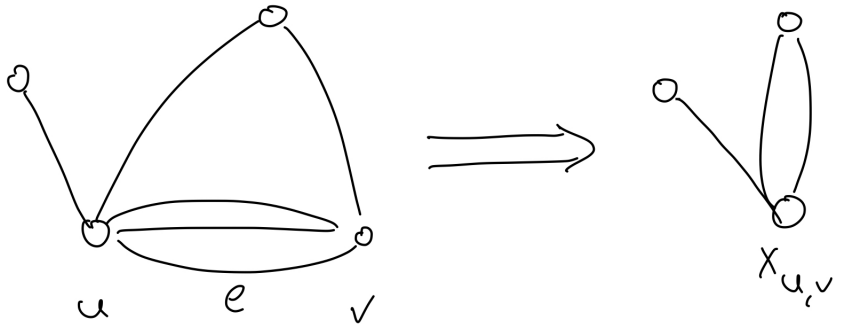


Figure 1: Example of a $\{u, v\}$-contraction or $e$-contraction

**Definition:** Let $\{u, v\} \in G$ be an edge in a multigraph. An edge-contraction on $\{u, v\}$ takes the two vertices $u$ and $v$ and merges them into one vertex $x_{u,v}$. We call the resulting graph $G/e$. Let $k$ be the number of edges $k$ between $u$ and $v$ before the contraction, then we have that:

$$\deg_{G/e}(x_{u,v}) = \deg_G(u) + \deg_G(v) - 2k$$

Notice that if we start with a graph $G$ whenever we contract by an edge, either that edge was in the minimum cut, in which case that cut has now disappeared. Thus we have $\mu(G) \leq \mu(G/e)$, since either all other cuts are larger or of equal size:

**Lemma:** Let $G$ be a graph and $e$ an edge in $G$. Then we have $\mu(G/e) \geq \mu(G)$ and if there exists a minimum cut $C$ in $G$ such that $e \notin C$, then we have $\mu(G/e) = \mu(G)$

Now, the idea goes as follows:

Cut(G):

1. choose some random ahh edge in $G$
2. construct $G/e$ and set it to be the new $G$.
3. repeat as long as we have more than 2 vertices.
4. Once we only have 2 vertices left, we return the cardinality of the unique cut.

Cool idea. Now analyse it: The thing with our algorithm is the following: if we contract edges in an unfortunate way, then we will end up outputting the wrong number at the end. For example if there is only one minimum cut in the graph and we contract one of the edges of that minimum cut then the only thing we can do is to cope heavily. Now, what is this type of probabilistic algorithm called? Las Vegas or Monte Carlo?

For each individual edge contraction, we have a certain probability of "having to cope heavily" (i.e. we contracted in an unfortunate fashion). We can calculate the probability of staying fortunate (and thus not having to cope heavily) as:

**Lemma:** Let $G = (V, E)$ be a multigraph with $n$ vertices. If we choose $e$ randomly and uniformly among all edges, then we have that

$$\Pr[\mu(G) = \mu(G/e)] \geq 1 - \frac{2}{n}$$

*Proof.* The probability of our minimum-cut-size $\mu(G)$ staying the same is

$$\Pr[\mu(G) = \mu(G/e)] \geq \Pr[e \notin C]$$

where the inequality follows from the fact that we could have multiple cuts $C$ of the same minimum sizes. Since

$$\Pr[e \notin C] = \frac{|E| - |C|}{|E|} = 1 - \frac{|C|}{|E|} \tag{1}$$

We also have that each vertex definitely has at least degree $|C|$, otherwise we could just cut around that vertex. This leaves us with

$$|E| = \frac{1}{2} \sum_{v \in V} \deg(v) \geq \frac{n|C|}{2}$$

which, when plugged into (1), gives us our desired property. □

Now, using this piece of information, we can calculate the probabilty that our algorithm $Cut(G)$ actually returns the desired result $\mu(G)$. We will immediately generalize to all possible graphs $G$ of $n$ vertices and get that

**Lemma:**

$$\hat{p}(n) \geq \left(1 - \frac{2}{n}\right) \hat{p}(n-1) = \left(\frac{n-2}{n}\right) \hat{p}(n-1)$$

which should be an intuitive result, since after each contraction, the probability of $\mu$ staying the same is $\left(1 - \frac{2}{n}\right)$ and we have one vertex less. The probability that our $\mu$ says the same until then end after this is another $\hat{p}(n-1)$. If we now write this out:

$$\hat{p}(n) \geq \left(\frac{n-2}{n}\right) \hat{p}(n-1) \geq \left(\frac{n-2}{n}\right) \left(\frac{n-3}{n-1}\right) \hat{p}(n-2) \geq \cdots$$
$$\hat{p}(n) \geq \left(\frac{n-2}{n}\right) \left(\frac{n-3}{n-1}\right) \left(\frac{n-4}{n-2}\right) \cdots \left(\frac{1}{3}\right) \hat{p}(2) = \frac{2}{n(n-1)}$$

giving us:

**Lemma:** For all graphs $G$ on $n$ vertices, the probability that one repetition of $Cut(G)$ returns $\mu(G)$ correctly is

$$\hat{p}(n) \geq \frac{2}{n(n-1)} = \frac{1}{\binom{n}{2}}$$

which holds for all $n \geq 2$

In that case, if we repeat the algorithm $\lambda\binom{n}{2}$ times and return the smallest cut size, the probability of returning a value larger than $\mu(G)$ is

$$(1 - \hat{p}(n))^{\lambda\binom{n}{2}} \leq \left(1 - \frac{1}{\binom{n}{2}}\right)^{\lambda\binom{n}{2}} \leq e^{-\lambda}$$

Now, what is the runtime? A single full repetition of the $Cut(G)$ algorithm takes $O(n^2)$ time. assuming that

- An edge contraction can happen in $O(n)$
- A random edge can be selected uniformly in $O(n)$

since we will do at most $n$ contractions. This gives us:

**Theorem:** The algorithm *RepeatedCut*$(G)$, which consists of repeating $Cut(G)$ $\lambda\binom{n}{2}$ times and returning the smallest cut size, returns $\mu(G)$ with probability at least $1 - e^{-\lambda}$ and runs in $O(n^4\lambda)$ time. Choosing $\lambda = \ln n$, you will notice that we spent the last 30 or so minutes developing an algorithm that is not even better than our initial attempt.

## 1.3 Min-Cut (3rd Attempt)

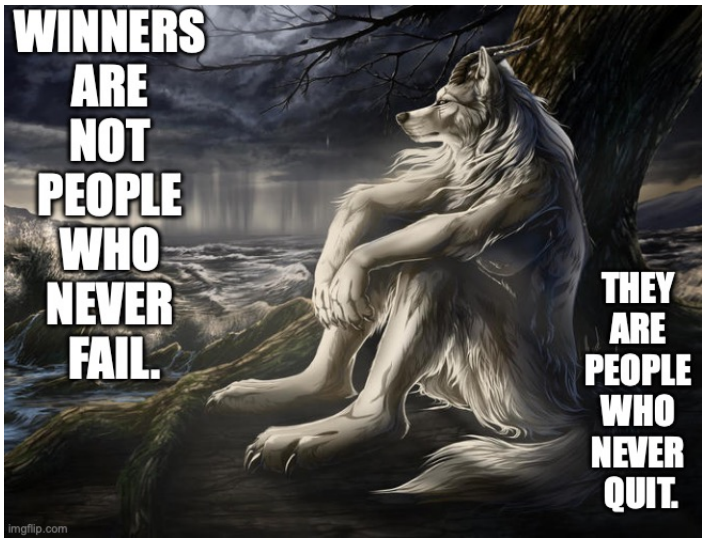


Figure 2: Although we might have failed in our 2nd attempt, we will not give up. Sigma Mindset.

Notice one thing with our previous algorithm: At the beginning, when we have not done that many contractions, the probability of us contracting wrong is low, since we have a bunch of edges to choose from. As we reach a smaller total edge count, the probability of us contracting and edge which belongs to the minimum cut (and thus risking increasing our $\mu$) increases significantly.

With this in mind, why don't we just stop contracting when we have $t$ vertices left, instead of 2? Afterwards, we can apply some other algorithm $A_{other}$ for these last $t$ vertices, giving us our result. Let us say we have that

- $A_{other}(G)$ runs in $z(t)$ time
- $A_{other}(G)$ returns the correct result $\mu(G)$ with probability $p^*(t)$

Then, we can apply almost the same analysis and get for the runtime $O((n(n-t)) + z(t))$ and for the probability of success:

$$p_{success} \geq \frac{n-2}{n}\frac{n-3}{n-1}\frac{n-4}{n-2}\cdots\frac{t+1}{t+3}\frac{t}{t+2}\frac{t-1}{t+1}p^*(t) = \frac{t(t-1)}{n(n-1)}p^*(t)$$

Now, we could do something like use a flow algorithm to process the final stretch (final $t$ vertices). However, as a wise man once said:

"Mathematiker sind Faul."

(literally every mathematics teacher ever)

($\star$) Therefore, we will ignore the fact that we study at ETH, let ourselves be "faule Mathematiker", and use $A_{other} = RepeatedCut$ with $\lambda = 1$. giving us $z(t) = O(t^4)$ and $p^*(t) = 1 - e^{-1}$ and a total runtime of $O(n(n-t) + t^4)$ and $p_{success} \geq \frac{t(t-1)}{n(n-1)}(1 - e^{-1}) = \frac{t(t-1)}{n(n-1)} \frac{e}{e-1}$. With the same logic as before, we would have to repeat this algorithm $\lambda \frac{n(n-1)}{t(t-1)} \frac{e}{e-1}$ times to get a $1 - e^{-\lambda}$ probability of success. Leaving us finally with:

$$\lambda \frac{n(n-1)}{t(t-1)} \cdot \frac{e}{e-1} \cdot O(n(n-t) + t^4) = O\left(\lambda \left(\frac{n^4}{t^2} + n^2 t^2\right)\right)$$

and if we choose $t = \sqrt{n}$, you will realize that by pure black magic, we got a better runtime of $O(\lambda n^3)$.

## 1.4 Bootstrapping

As such is the human spirit: We are greedy. One little success is not enough, and we constantly strive for better results. A particularly greedy individual at some point must have then noticed: But wait! Now that we have an algorithm which is $O(\lambda n^3)$ and the same probability of success $p_{success}$, we can use this algorithm with $\lambda = 1$ again and plug it in at the point ($\star$). This gives us an even faster algorithm and an even better choice of $t$!

This idea is what we call bootstrapping: repeatedly develop a better algorithm and plug it in again, getting even better results. Our situation converges to a final runtime of

$$O(n^2 \text{poly}(\log n))$$

## 1.5 Smallest Enclosing Circle - Initial Setup

Let $P$ be a set of $n$ points in a plane. We want to find the smallest circle $C(P)$ (smallest in terms of radius) which contains all of the points. For this, we have to define a few things:

**Definition:**

- $C$ is a circle. It contains only the points on the rim.
- $C^\bullet$ is the entire region of the circle, including the rim.
- $C(P)$ is the smallest circle which contains all of the points in $P$

There would be no point in finding this smallest enclosing circle if it is not unique. Luckily, we can prove that we indeed have a unique smallest enclosing circle:

**Lemma:** For every (finite) set of points $P$ in $\mathbb{R}^2$ there is a unique smallest enclosing circle $C(P)$

The proof follows visually:

**Lemma:** For every (finite) set of points $P$ in $\mathbb{R}^2$ with $|P| \geq 3$ there is a subset $Q \subseteq P$ such that $|Q| = 3$ and $C(Q) = C(P)$

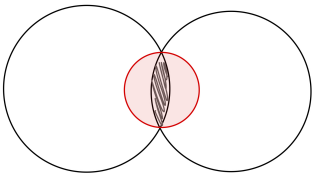

Figure 3: Assume we have two circles which are smallest possible and enclose all points $P$ (the left one and the right one). Then, $P$ must lie within the grey intersection area. Then, we can construct a smaller (red) circle.

We notice like Descartes after drinking a bunch of absinthe that for every set of points $P$, there are at least 3 points that determine the smallest enclosing circle. (i.e. at least two on the rim of the circle and another one on the rim in case the two other points are not enough). The proof is trivial, just like how Descartes trivially thought of the cartesian plane after drinking absinthe. (source: my philosophy teacher in the 11th grade, shoutout to Mr. Pollmann).

This lemma is helpful in the following way: For 3 points, we can determine the smallest enclosing circle quickly in $O(1)$ time as follows: Take two points and form the smallest enclosing circle. Then, "move" the circle towards the last point, while keep the two initial ones on the rim:

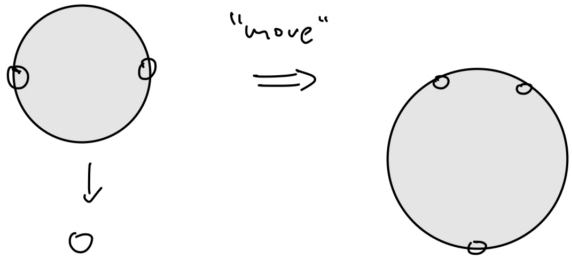


Figure 4: how to find the smallest enclosing circle for 3 points

Using this, we can now develop a way to find the smallest enclosing circle for arbitrary $n$ points, where we just enumerate through all possible $\binom{n}{3}$ combinations of 3 points. Which gives us a deterministic runtime

**Algorithm 1** CompleteEnumeration($P$)

```
for all Q ∈ (P choose 3) do
    compute C(Q)
    if P ⊆ C•(Q) then
        return C(Q)
    end if
end for
```

of $O(n^4)$.

## 1.6 Smallest Enclosing Circle - Randomized

Our goal is to develop a randomized algorithm of runtime $O(n \ln n)$. The game plan is to again turn on our AnP Ooga Booga brains that just throw a randomisation process onto some algorithm and repeat it a billion times to get a good final algorithm.

First, let us look at simply transforming the *CompleteEnumeration* algorithm into a randomized one, where the only difference is that we choose 3 points randomly from $P$ instead of enumerating through all possible combinations. This gives us a runtime of $n$ per iteration. Since exactly three points determine the smallest enclosing circle, the probabiliy of us choosing exactly these three points is $1/\binom{n}{3}$, giving us an expected runtime of $\binom{n}{3}$ (geometric distribution).

This is obviously not better. We will improve this algorithm using the two following ideas: First, we will choose a higher number of points, let us say, 11, to increase our odds in each iteration. Note that the constant runtime still stays, since we can go over all combinations of 3 points out of these 11 points to determine the smallest enclosing circle of all 11 (also $\binom{11}{3} = O(1)$). Second, we will double all points lying outside of the smallest enclosing circle we have calculated in each iteration. This will put a higher weight onto the points lying more towards the rim of the actual smallest enclosing circle, making them more probable of being selected. Intuitively, these ideas should yield a better algorithm. Let us analyze it again. With the help of

**Algorithm 2** Randomised_CleverVersion($P$)

```
repeatforever
    choose Q ⊆ P with |Q| = 11 randomly and uniformly
    determine C(Q)
    if P ⊆ C•(Q) then
        return C(Q)
    end if
    double all points of P outside of C(Q)
until
```

the following lemma

**Lemma:** Let $P$ be a set of $n$ (not necessarily distinct) points and for $r \in \mathbb{N}$, let $R$ be randomly uniformly selected from $\binom{P}{r}$. Then the expected number of points of $P$ that lie outside of $C(R)$ is at most $3\frac{n-r}{r+1} \leq 3\frac{n}{r+1}$.

we can prove that

**Theorem:** *Randomized_CleverVersion* runs in $O(n \ln n)$ expected time and finds the correct smallest enclosing circle. Las Vegas / Monte Carlo?

*Proof.* Let us define $T$ to be the random variable for the number of iterations the algorithm performs. We want to prove that

$$\mathbb{E}[T] = O(\ln n)$$

We will go a detour to do this: First define $X_k$ to be the random variable for the number of points in our setting after $k$ iterations. Of course, by definition $X_0 = n$ deterministically. Using the previous lemma, we can find an upper bound for the expectation of $X_k$:

$$\begin{aligned}
\mathbb{E}[X_k] &= \sum_{t=0}^{\infty} \mathbb{E}[X_k | X_{k-1} = t] \cdot \Pr[X_{k-1} = t] \\
&\overset{\text{Lemma}}{\leq} \sum_{t=0}^{\infty} \left(1 + \frac{3}{r+1}\right) t \cdot \Pr[X_{k-1} = t] \\
&= \left(1 + \frac{3}{r+1}\right) \cdot \sum_{t=0}^{\infty} t \cdot \Pr[X_{k-1} = t] \\
&= \left(1 + \frac{3}{r+1}\right) \cdot \mathbb{E}[X_{k-1}].
\end{aligned}$$

where the first inequality follows from the previous lemma and the fact that $\left(1 + \frac{3}{r+1}\right) t$ is just the original $t$ points in the previous round plus the ones that we have to double, which is, according to the lemma, upper bounded by $3\frac{t-r}{r+1} \leq 3\frac{t}{r+1}$. We can then chain the inequalities:

$$\mathbb{E}[X_k] \leq \left(1 + \frac{3}{r+1}\right) \mathbb{E}[X_{k-1}] \leq \cdots \leq \left(1 + \frac{3}{r+1}\right)^k \mathbb{E}[X_0] = \left(1 + \frac{3}{r+1}\right)^k n$$

Additionally, we also have a lower bound: Notice that after $k$ rounds, one of the three points that determine the smallest enclosing circle must have been chosen in less than 1/3rd of all rounds. Otherwise, the rounds in which the three points were chosen in would overlap and we would have found the smallest enclosing circle long ago. Therefore, this exact point must have at least $2^{k/3}$ copies.

$$\mathbb{E}[X_k] = \underbrace{\mathbb{E}[X_k | T \geq k]}_{\geq 2^{k/3}} \cdot \Pr[T \geq k] + \underbrace{\mathbb{E}[X_k | T < k]}_{\geq 0} \cdot \Pr[T < k] \geq 2^{k/3} \cdot \Pr[T \geq k].$$

which gives us in total:

$$\begin{aligned}
\left(1 + \frac{3}{r+1}\right)^k n &\geq \mathbb{E}[X_k] \geq 2^{k/3} \cdot \Pr[T \geq k] \\
\left(1 + \frac{3}{r+1}\right)^k n \cdot 2^{-k/3} &\geq \Pr[T \geq k]
\end{aligned}$$

since we are choosing $r = 11$ points in each round, we have that $\Pr[T \geq k] \leq \left(1 + \frac{3}{11+1}\right)^k \cdot 2^{-k/3} n = 0.995^k n$ and trivially also $\Pr[T \geq k] \leq 1$. giving us finally:

$$\begin{aligned}
\mathbb{E}[T] &= \sum_{k \geq 1} \Pr[T \geq k] \\
&\leq \sum_{k=1}^{k_0} 1 + \sum_{k > k_0} 0.995^k \mathfrak{n} \\
&\overset{k = k_0 + k'}{=} \sum_{k=1}^{k_0} 1 + \sum_{k' \geq 1} 0.995^{k'} \cdot \underbrace{0.995^{k_0} n}_{=1} \\
&= k_0 + O(1) \leq 200 \ln n + O(1).
\end{aligned}$$

where $k_0 = -\log_{0.995} n$ which is also the threshold where $0.995^k n$ starts being less than 1.

□