CS 0368-4246: Combinatorial Methods in Algorithms (Spring 2025) April 29, 2025

Lecture 6: Uses of Edge Contractions

Instructor: *Or Zamir* Scribes: *Omri Sde-Or*

1 Edge contraction

Definition 1. *Let G be an undirected graph, $e \in E$. We define G/e to be a graph in which we join u and v, where u, v are the vertices of e.*

Note that there may be some self-edges or double edges, most of the time we will ignore this but sometimes we will want to keep these.

2 Minimum Spanning Trees (MST)

Definition 2. *Let G be an undirected, weighted graph. A minimum spanning tree is a spanning tree with the minimum total edge weight.*

Note: we assume that all the weights are different.

Lemma 3. *Let $V = S \uplus S^C$ be a cut in the graph, then the minimal edge in the cut $\left(S, S^C\right)$ must be in an MST of G.*

Proof. We want to show an MST that uses e. Let T be an MST. If e is in T then we are finished. Otherwise, there is exactly one path between the end points of e. In this path there must be an edge that passes through the cut $\left(S, S^C\right)$. We replace this edge with e and we get an MST with better weight. (If the weights of the edges are different, we get a contradiction). □

Lemma 4. *Let G be a graph, and e an edge in an MST. Then every MST of G/e together with e is an MST of G.*

Proof. An edge that is minimal in a cut of G/e is also minimal in a corresponding cut of G. Therefore all the edges of the MST of G/e are also in an MST of G. Together with e we get a tree and therefore an MST. □

Algorithm 5. *Kruskal (50s) [1]:*

1. *Sort the edges*
2. *Go over the sorted edges $e_1, \ldots, e_m$*
3. *For each edge that we go over, if it isn't a self edge then contract it*

Proof.

Correctness: Every $e_i = (u, v)$ that we contract was minimal in some cut of the graph (in which we contracted it). For example, in the cut $\left(\{u\}, \{u\}^C\right)$

Running time:

1. Sort the edges - $m \log m = m \log n$
2. Store the contracted graphs + know if an edge is a self-edge (in algorithms: using Union Find)

Overall: $m \log n$ □

Algorithm 6. *Borůvka (20s) [4] - also runs in $m \log n$ (is easy to parallelize). While the graph has more than one vertex:*

1. *Every vertex chooses the edge with minimal weight that touches it*
2. *Contract all the edges that some vertex chose and add it to the MST*

Proof.

Correctness: Like in Kruskal.

Running Time: Every iteration is $O(m)$, and every iteration reduces the number of vertices by a multiplicative factor of at least 2. Therefore we finish in $\log n$ iterations. □

Algorithm 7. *Prim's algorithm [6] - runs in $m + \log n$.*

1. *Chose some arbitrary vertex v*
2. *For $n - 1$ steps, the vertex v chooses the minimal non-self edge that touches it and contracts it.*

Proof.

Correctness: Like the previous algorithms.

Running Time: The "edges" that touch the current v will be saved in a priority queue, such that out of all the edges that go to the same vertex, we remember only the minimal edge. So we have m Insert or DecreaseKey operations, and we have n FindMin and DeleteMin operations. So if we use, for example, a Fibonacci heap, we get Insert and DecreaseKey in $O(1)$ time and FindMin and DeleteMin in $O(\log n)$. □

Algorithm 8. *Algorithm Fredman-Tarjan (∼80) [3] - running time is $m\beta(m, n) \leq m \log^* n$.*

Idea: Instead of running Prim once, run it in parallel from a lot of vertices, and stop once there are too many elements in the priority queue.

In "simple terms": $\log^* n$ is the minimal k such that $\log^{(k)} n := \underbrace{\log \log \ldots \log n}_{k \text{ times}} \leq 2$, $\beta(m, n)$ is the minimal k such that $\log^{(k)} n \leq \frac{2(m+1)}{n}$

Notice: $\beta(m, n) \leq \beta(n, n) = \log^* n$, $\beta\left(n \log^{(k)} n, n\right) \leq k$

An iteration of FT (with parameter k): At the beginning, all the vertices aren't marked, and each vertex will create a Fibonacci heap with all its neighbors. As long as there is a vertex v in the graph which isn't marked, we start running Prim from it such that every vertex that is added to it is marked (together with its heap).

We stop in one of the following cases:

1. We saw at least k elements in the run of Prim's algorithm.
2. The vertex that we want to add is already marked. We still add it to the MST (and contract it), but we stop.

Proof. Running Time: Each iteration takes $O(m + n \log k)$, and we got $\leq \frac{n}{k}$ "components" (vertices of the contracted graph and maybe $\sim m$ edges).

Why? We "grew" some Prim components of sizes $n_1, n_2, \ldots, n_r$, such that $n_i \leq k$. If we look at the components including the contraction which caused us to finish (case 2), then all the components are of size $\geq k$. Therefore after all the contractions there $\leq \frac{n}{k}$ components. How much did it cost to "grow" each component? Every insertion and DecreaseKey are like earlier ($O(1)$). As for FindMin and DeleteMin, it costs us overall for the whole algorithm $\leq ((n \log k)$. Denote by m_i the number of edges that touch the i'th component. Then $\sum_i m_i \leq 2m$.

To summarize, it takes $m + n \log k$ time to go down from n vertices and m edges to a graph with $\frac{n}{k}$ vertices and $m \geq 1$ edges. □

Claim 9. *(By induction) There exists a constant C (not dependent on m, n, t) such that it is possible to solve MST in time $C \cdot \left(km + n \log^{(t)} n\right)$*

Proof. Base (t = 1): Prim

Before doing the general case, we will do the case $t = 2$: We run FT with $k = \log n$, and then the running time will be $\underbrace{(m + n \log t)}_{\text{FT}} + \underbrace{\left(m + \frac{n}{t} \log \frac{n}{t}\right)}_{\text{Prim}} = O(m + n \log \log n)$.

Step for general t: Run FT with $k = \log^{(t-1)} n$, and then use the algorithm for the $t-1$ case by induction. The FT takes $C\left(m + n \log\left(\log^{(t-1)} n\right)\right)$, and the recursion takes $C\left((t-1)\, m + \frac{n}{k} \cancel{\log^{(t-1)}\left(\frac{n}{k}\right)}\right)$ □

We saw an algorithm in time $tm + n \log^{(t)} n$. So if we take $t = \beta(m, n)$, we get $m\beta(m, n)$.

Deterministically: the best we know to do is an algorithm from Chazelle ('00) [2] in time $m\alpha(m, n)$ (where α is the inverse Ackermann function).

With randomness: (KKT '95 [5]): There is an algorithm with linear expected run time.

3 Minimum Cut in Graphs

Definition 10. *Let G be a non-directed, non-weighted graph. A minimum cut is a cut $V = S \uplus S^C$ with the minimum number of edges crossing it.*

Question: How fast can we find a minimal cut in a graph? (In the course algorithms this is done with flow graphs).

Algorithm 11. *Take the graph G and take a uniformly random edge e, contract it, and continue with the same procedure in G/e (with parallel edges). When we get to a graph with only 2 nodes, we return the cut that it defines.*

Intuition about why this works: We expect that most of the edges won't cross the cut, so when we take a random edge, we will probably take an edge that doesn't cross the cut, so contracting the edge won't affect the minimal cut.

We hope that the probability that the cut that we will be left with is minimal will be $\geq p$. Then we can run the algorithm $\frac{100}{p}$ times and return the smallest cut that we got.

Claim 12. *Let $\left(S, S^C\right)$ be a minimum cut in G, then the number of edges that edges that cross the cut is $\leq \frac{2}{n}m$.*

Proof. The sum of degrees in the graph is $2m$, therefore there is a vertex of degree $\leq \frac{2m}{n}$, so in particular we get that the cut that the vertex defines is at least as big as the minimal cut. □

What is the probability that a minimal cut $\left(S, S^C\right)$ survives the algorithm? Because we saved also parallel edges, each size of cut in the contracted graph is also the size of a cut in the original graph. Therefore the cut $\left(S, S^C\right)$ is always a minimum cut.

We have $n-2$ iterations on $n, n-1, n-2, \ldots, 3$ vertices. The probability that on i vertices we don't hit the cut $\left(S, S^C\right)$ is $\geq 1-\frac{2}{i}$. Therefore the probability that we didn't hit it throughout all the iterations is

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

So we run this $10n^2$ times, and return the minimum cut that we saw, which will be a minimum cut with probability $> 99\%$.

How much time does each iteration take? We can implement each iteration to take $\tilde{O}\left(m\right)$ time. We take a random order on the edges and "run Kruskal" until we remain with 2 components. (Takes $m\alpha\left(n\right)$ using Union-Find)

Overall the running time is $\tilde{O}\left(mn^2\right)$.

Corollary 13. *In every non-directed, non-weighted graph there are at most $\binom{n}{2}$ minimum cuts.*

Proof. Each minimum cut had probability $\geq \frac{1}{\binom{n}{2}}$ for it to be returned from the algorithm, and these probabilities can't sum up to more than 1. □

Algorithm 14. *Improved Algorithm:*

1. *Take a random edge and contract it $\underbrace{n-\left\lceil\frac{n}{\sqrt{2}}\right\rceil}_{\alpha\cdot n}$ times (until the graph is of size $\approx \frac{n}{\sqrt{2}}$).*
2. *Run on the remaining graph the same algorithm twice recursively, and return the better cut.*

Lemma 15. *The probability that in step 1 we didn't ruin the minimum cut $\left(S, S^C\right)$ is at least $\frac{1}{2}$.*

Proof.

$$\left(1-\frac{2}{n}\right)\cdots\left(1-\frac{2}{n-\frac{n}{\sqrt{2}}}\right) = \frac{\left(n-\left(n-\frac{n}{\sqrt{2}}\right)\right)\left(n-\left(n-\frac{n}{\sqrt{2}}\right)-1\right)}{n\left(n-1\right)} \approx \frac{\frac{n}{\sqrt{2}}\cdot\frac{n}{\sqrt{2}}}{n\cdot n} = \frac{1}{2}$$

□

What is the recursion depth? $\log_{\sqrt{2}} n = \Theta(\log n)$.

What is the running time?

$$T(n) = \tilde{O}\left(n^2\right) + 2T\left(\frac{n}{\sqrt{2}}\right) = C \cdot n^2 + 2 \cdot \left(C \cdot \left(\frac{n}{\sqrt{2}}\right)^2 + 2 \cdot \left(C \cdot \left(\frac{n}{2}\right)^2\right) + \dots\right) \leq C \cdot n^2 \cdot \log n$$

What is the probability of success of the algorithm? Denote the probability of success when when doing i iterations by p_i.

$$\begin{aligned} p_0 &= 1 \\ p_1 &= 1 - \left(\frac{1}{2}\right)^2 = \frac{3}{4} \\ p_2 &= 1 - (1-p_1)^2 = 1 - \left(\frac{1}{4}\right)^2 = \frac{15}{16} \end{aligned}$$

We will analyze this by induction.

Base: $p_0 = 1$.

Step: We can look at the tree of recursive calls. Each call has two children. Each child returns a correct answer with probability $\frac{1}{2}$, and if it returns a wrong answer then all its parents will also return a wrong answer. We want to know what the probability that a leaf returns a correct answer is. So we get the recursive equation $p_{i+1} = \frac{1}{2} \cdot \left(1 - (1-p_i)^2\right)$ - the $\frac{1}{2}$ is the probability that the current node returned a correct answer, and the $\left(1 - (1-p_i)^2\right)$ is the probability that at least one of its children returned a correct answer. Therefore we get

$$p_{i+1} = \frac{1}{2} \cdot \left(1 - (1-p_i)^2\right) = \frac{1}{2}\left(1 - \left(1^2 - 2p_i + p_i^2\right)\right) = \frac{1}{2}\left(2p_i - p_i^2\right) = p_i\left(1 - \frac{p_i}{2}\right)$$

So if we assume in the induction hypothesis that $p_i \geq \frac{1}{i}$, because $x\left(1 - \frac{x}{2}\right)$ is increasing in a neighborhood of 0^+, we get that

$$p_{i+1} = p_i\left(1 - \frac{p_i}{2}\right) \underset{\text{I.H.}}{\geq} \frac{1}{i}\left(1 - \frac{1}{2} \cdot \frac{1}{i}\right) = \frac{1}{i} \cdot \frac{2i-1}{2i} \geq \frac{1}{i} \cdot \frac{i}{i+1} = \frac{1}{i+1}$$

From this analysis we get that with probability $\frac{1}{\log n}$ we return a minimum cut. So we run this $\log n$ times. So overall $n^2 \log^3 n$. This is tight up to log in a dense graph.

(Today we know to do this in $m^{1+o(1)}$ in any graph, not only dense graphs.)

References

[1] *17 On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem (1956)*, pages 179–182. 2021.

[2] Bernard Chazelle. A minimum spanning tree algorithm with inverse-ackermann type complexity. *J. ACM*, 47(6):1028–1047, November 2000.

[3] M.L. Fredman and R.E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. In *25th Annual Symposium onFoundations of Computer Science, 1984.*, pages 338–346, 1984.

[4] V. Jarník. *O jistém problému minimálním: (Z dopisu panu O. Bor˚uskovi).* Práce Moravské přírodovědecké společnosti. Mor. přírodovědecká společnost, 1930.

[5] David R. Karger, Philip N. Klein, and Robert E. Tarjan. A randomized linear-time algorithm to find minimum spanning trees. *J. ACM*, 42(2):321–328, March 1995.

[6] R. C. Prim. Shortest connection networks and some generalizations. *The Bell System Technical Journal*, 36(6):1389–1401, 1957.