Disclaimer: *These notes have not been subjected to the usual scrutiny for formal publications. They are to be used only for the class.*
Outline: Improved Karger's algorithm for minimum cut

1 Improved Karger's algorithm for minimum cut

What is the running time of the min cut algorithm we covered in class? The probability of survival of a particular min cut after a single run of contraction is $1/\binom{n}{2} = O(\frac{1}{n^2})$. This means that we need to run contraction $O(n^2)$ times to get a reasonable error probability. Each run of contraction costs $O(n^2)$ because we shrink a graph of n vertices downto one that has only 2 vertices. By using adjacency matrix to keep track of the graph, each edge contraction costs $O(n)$ time and we'll have to contract $n-2$ times, hence the $O(n^2)$ cost. Therefore, overall complexity is $O(n^4)$.

Notice that the min cut is most likely to be lost during the later steps of contraction. In fact, the first edge contraction has a probability of $1-\frac{2}{n}$ to preserve the min cut, the second a bit less: $1-\frac{2}{n-1}$, till the last which is $1-\frac{2}{3}=\frac{1}{3}$, much worse than what we had at the start. Naturally, we'd like to somehow compensate for this loss.

1.1 Speeding up – first try

Without committing to anything, let's suppose we do edge contraction until a number of t vertices are left. The probability that a min cut survives is $\leq$

$$\prod_{i=1}^{n-t} \frac{n-i-1}{n-i+1} = \frac{t(t-1)}{n(n-1)} = O(\frac{t^2}{n^2})$$

Note when $t=2$, this gives us back the original algorithm. Suppose we stop at $t=\sqrt{n}$. And then run a deterministic algorithm on the remaining vertices to obtain a min cut. (One such algorithm is to repeatedly run s-t min cut. See appendix for details. The running time is $O(N^4)$). Since the probability of preserving the min cut is now improved to $O(t^2/n^2) = O(1/n)$, only a number of $O(n)$ repetition is needed to get a reasonable overall error probability.

For each run, the contraction process takes $O((n-t)^2) = O((n-\sqrt{n})^2) = O(n^2)$ as before. And the deterministic step takes $O(t^4) = O(n^2)$. Hence, the cost is $O(n^2)$ (this justifies the choice of $t=\sqrt{n}$, which balances the cost of contraction and the cost of the deterministic algorithm). The overall complexity is $O(n^3)$, a factor of n improvements from the naive implementation.

1.2 Speeding up – second try

In the first try, we contract until there are $t=\sqrt{n}$ vertices left to obtain G' and then run a deterministic algorithm on G'. Notice that G' is simply another instance of the min cut problem, only that it's smaller. This leads to the idea of recursion.

However, the effect of straightforward recursion is the same as running a sequential algorithm of contraction. A new and more subtle idea is to spawn more instances for smaller graphs, in hopes that the min cut survives in any of the spawned instances.

In doing so, we will be careful to balance the smaller probability of min-cut survival for small-size graphs and the larger number of spawned instances of those graphs. The decision boils down to the choice of the breakpoint of recursion, the number t.

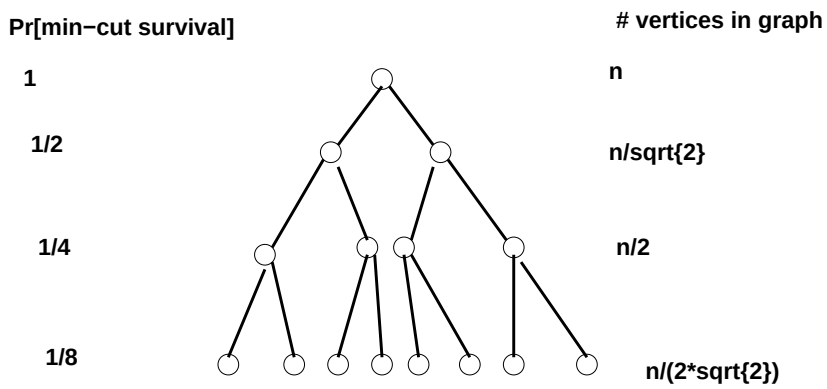

Figure 1: Recursion tree

If we spawn two smaller graphs at each level of recursion, the number of graphs at level i is 2^i. To balance the probability of min-cut survival, which is t^2/n^2, we would choose $t = n/\sqrt{2}$ so that $t^2/n^2 = 1/2$. This idea is illustrated in figure 1.

A listing of the algorithm can be found in section 10.2 of the textbook. There, one can prove that the probability of min cut survival $P(n)$ (where n denotes the number of vertices) satisfies the recurrence relation:

$$P(n) = 1 - (1 - \frac{1}{2}P(n/\sqrt{2}))^2$$

From which, one can prove that $P(n) = \Theta(1/\ln n)$ (once the asymptotic growth is conjectured, the proof is a simple matter of induction.) Also, the running time satisfies the recurrence relation:

$$T(n) = 2T(\frac{n}{\sqrt{2}}) + O(n^2)$$

By master's methods (cs170 material), it follows that $T(n) = O(n^2 \ln n)$. Therefore, the overall complexity is $T(n) \cdot \frac{1}{P(n)} = O(n^2 \ln^2 n)$. This is again an order of magnitude improvement.

Theorem (Karger '96): With high probability, one can find all min cuts in $O(n^2 \ln^3 n)$ time.

proof: We knew already that $P(n) \geq \frac{c}{\ln n}$ for large n, where c is a constant hidden in the Θ notation. By boosting, we have

$$\Pr[\text{miss a particular min-cut}] = (1 - P(n))^{3\ln^2 n/c} \leq (1 - \frac{c}{\ln n})^{3\ln^2 n/c} \leq e^{-3\ln n} = n^{-3}$$

Since there are at most $\binom{n}{2}$ min-cuts, by union bound,

$$\Pr[\text{miss any min-cut}] \leq \binom{n}{2} \cdot n^{-3} = O(\frac{1}{n})$$

Therefore, the failure probability is arbitrarily small. QED

So far, we've been trying to improve the complexity of the min cut algorithm. Is there a bound on the improvements we can make? Well, one observation is that we have to look at all the edges to decide a minimum cut, which means we'll have to spend at least $O(n^2)$ time. And the best algorithm so far runs in $O(n^2 \ln^{O(1)} n)$, which is close.

Appendix : Edmonds-Karp implementation of the augmentation path algorithm to solve the network flow problem takes $O(|E||V|) = O(n^3)$ time. By min-cut max-flow theorem, we get an algorithm for s-t min-cut as a by-product. A first attempt to solve the graph min-cut problem is to enumerate all possible (s,t) pairs. However, this is redundant because without loss of generality, the node s belongs to one of the partitions of the cut, therefore we only need to vary node t over all possible vertices. Hence, this gives a $O(n^4)$ algorithm for the graph cut problem.