COMPSCI 330: Design and Analysis of Algorithms 10/30/2014

Lecture #18

Lecturer: Debmalya Panigrahi *Scribe: Wenshun Liu*

1 Overview

In this lecture we introduced two kinds of randomised algorithms: Monte Carlo and Las Vegas algorithms. We discussed their difference in runtime and correctness, and introduced Contraction Algorithm as a randomised Monte Carlo algorithm to find the minimum cut of a graph.

2 General Properties of the Randomised Algorithms

2.1 Monte Carlo Algorithms

The *running time* of Monte Carlo algorithms is deterministic.
The *correctness* of the algorithms, however, is NOT deterministic. The output may be wrong with a *small probability*.

2.2 Las Vegas Algorithms

The *running time* of Las Vegas algorithms is NOT deterministic.
The *correctness* of the algorithm is deterministic. The algorithm will always output the correct answer.

3 Contraction Algorithm

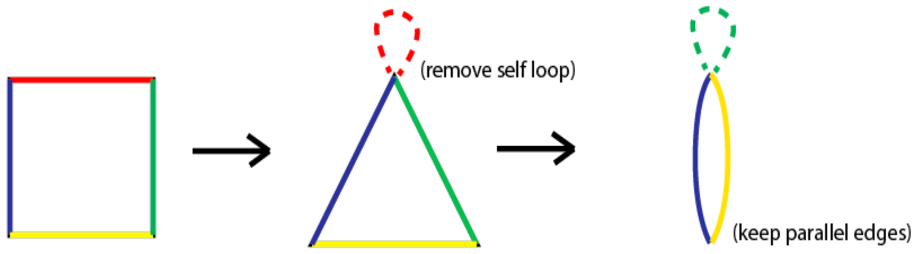
3.1 Overview

Contraction Algorithm is a Monte Carlo algorithm that computes the global minimum cut of a connected graph. The algorithm has deterministic runtime, but is not guaranteed to output the correct result.

3.2 Pseudocode

```
Contraction
        Repeat n-2 times
                contract an edge chosen uniformly at random
                remove all self-loops, and keep all parallel edges

        //we are down to two points by the end of the algorithm
        return the number of edges between the last two nodes as the mincut
```



3.3 Correctness Analysis - Probability that the Algorithm will Succeed

As a Monte Carlo algorithm, the mincut outputted by Contraction Algorithm is not guaranteed to be correct. Thus, we are interested in the *probability* that the algorithm will succeed (correctly output the mincut).

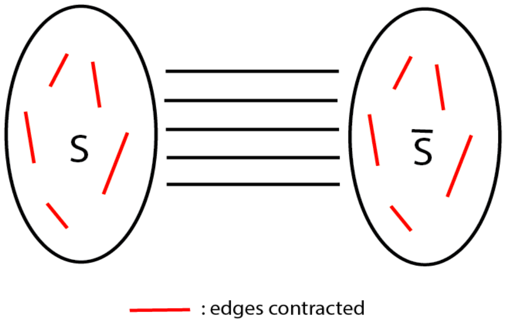
Remark 1. *Contraction Algorithm would output a different (and incorrect) cut when somewhere during the randomised process, an edge in the mincut is chosen and contracted.*

To find out this probability, we define

$(S, \bar{S})$ to be a mincut containing C edges.

X_i to represent the case where no edges in $(S, \bar{S})$ is contracted in the first i iteration.

(That is, the algorithm stays correct after the first i iterations)



G_i to be the graph after i iterations.

$Pr[Algorithm\ is\ sucessful]$ is essentially $Pr[X_{n-2}]$, which is when none of the mincut edges is contracted in all $n-2$ contractions. To find out this probability we take the following steps:

1. $Pr[X_{n-2}|x_{n-3}]$ represents the probability that none of the mincut edges is contracted in the $(n-2)^{th}$ iteration *given that* none of the mincut edges has been contracted in the previous $n-3$ steps.

   We have $Pr[X_{n-2}|x_{n-3}] = 1 - \frac{\#edges\ in\ (S,\bar{S})}{\#edges\ in\ G_{n-3}}$ $(which\ is:\ 1 - \frac{\#edges\ in\ mincut}{\#edges\ left\ in\ graph})$

   $\#edges\ in\ (S, \bar{S})$ is simply C, as no edges from the mincut has been contracted.

   Now we want to find out $\#edges\ in\ G_{n-3}$:

Observation 1. *#vertices in G_{n-3} is 3, as we lose exactly 1 vertex during each contraction.*

Claim 2. *Any contraction (and the removal of self loops) does not decrease the number of edges in a minimum cut.*

Proof. We proof the claim by contradiction.

We assume that after the contraction, we have found a cut of length λ in the graph smaller than all cuts before the contraction.

We know that the contracted edge (u, v) cannot lie across this new cut, or this cut would not exist anymore as the contraction would merge the partitions together.

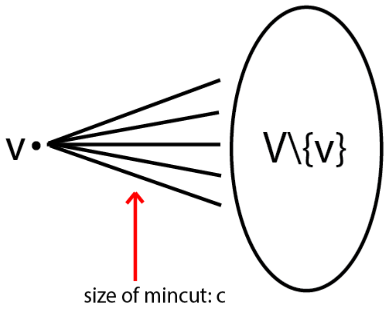
Thus, given that vertices u and v must lie on the same side of the cut, if we reverse the contraction, the number of edges of this cut in the original graph would indeed remain to be the same.

As this cut of length λ exists in the graph before the contraction, the mincut of the graph before the contraction must have at most λ edges.

Thus, we hereby proved that after the contraction, we cannot found a cut smaller than all cuts before the contraction. □

Claim 2 shows that the mincut in G_{n-3} has at least C edges. Thus, we know that:

$$\#edges\ incident\ on\ any\ vertex\ in\ G_{n-3}\ (=\ the\ degree\ of\ any\ vertex\ ) \geq C.$$



Thus, given that we have 3 vertices, and each has degree of at least C, $\#edges\ in\ G_{n-3} = \frac{3c}{2}$

2. We now generalise the case to any iteration. At the i^{th} contraction we have,

   $Pr[X_i|X_{i-1}] = 1 - \frac{c}{\#edges\ in\ G_{i-1}}$, where

   $\#edges\ in\ G_{i-1} \geq \frac{(n-i+1)c}{2}$,

   as G_{i-1} has $n-i+1$ vertices, and $\#edges\ incident\ on\ any\ vertex \geq c$.

   Thus, we have,

   $$Pr[X_i|X_{i-1}] = 1 - \frac{c}{\frac{(n-i+1)c}{2}} = \frac{n-i-1}{n-i+1}$$

3. Now we have,

   $$\begin{aligned} Pr[X_{n-2}] &= Pr[X_{n-2}|X_{n-3}] * Pr[X_{n-3}] \\ &= Pr[X_{n-2}|X_{n-3}] * Pr[X_{n-3}|X_{n-4}] \ldots Pr[X_2|X_1] * Pr[X_1] \\ &\geq \frac{1}{3} * \frac{2}{4} \cdots \frac{n-3}{n-1} * \frac{n-2}{n} \\ &= \frac{1*2}{(n-1)n} \\ &= \frac{1}{\binom{n}{2}} = \Theta(\frac{1}{n^2}) \end{aligned}$$

   We've found out the probability that the contraction algorithm would succeed.

3.4 Succeeds with High Probability

From the previous section we could observe that if the contraction algorithm is only run once, the chance that it would output the correct mincut relatively low. Thus, we wish to run the algorithm multiple times to increase the probability of success.

Suppose we run the algorithm k times, and report the smallest cut found in the k runs:

$Pr[failure\ in\ k\ runs] = Pr[failure\ in\ one\ run]^k \leq (1 - \frac{1}{\binom{n}{2}})^k$

We recall from the previous class that,

$\frac{1}{4} \leq (1 - \frac{1}{N})^N \leq \frac{1}{e}$ for $N \geq 2$

Thus, we know that $(1 - \frac{1}{\binom{n}{2}})^k = \Theta(1)$ if $k = \binom{n}{2}$

$... = \frac{1}{n}$ if $k = \binom{n}{2} logn = O(n^2 logn)$

$... = \frac{1}{n^4}$ if $k = 4\binom{n}{2} logn = O(n^2 logn)$

$... = \frac{1}{poly(n)}$ if $k = O(n^2 logn)$

Thus, if we pick $k = O(n^2 logn)$, we have,

$Pr[Algorithmwillsucceed] \geq 1 - \frac{1}{poly(n)} = 1 - o(1)$ (the "little" o)

We say an algorithm *succeeds with high probability* if the algorithm succeeds with probability $\geq 1 - o(1)$
.

3.5 Runtime Analysis

To *succeed with high probability*, we need to run the algorithm $O(n^2 logn)$ times. Each run would take $O(n)$ to finish with some advanced optimisations. Thus, the algorithm would run in $O(n^3 logn)$.

Remark 2. *The algorithm could be optimised to achieve a runtime of $O(n^2 logn)$.*