Randomized Algorithms, Summer 2016 Lecture 2 (5 pages)

Min Cut, Fast Cut, Polynomial Identities

Instructor: Thomas Kesselheim and Kurt Mehlhorn

1 Min Cuts in Graphs

Throughout this section, $G = (V, E)$ is a multi-graph. A cut of G is a bipartition $(S, \bar{S})$ of the vertex set of G. The capacity of a cut is the number of edges having one endpoint on both sides of the cut. A min-cut is a cut of minimum capacity.

A minimum cut can be computed with the help of maxflow computations. For some vertex s and every other vertex t, one computes the minimum cut separating s from t, and then takes the smallest cut obtained in this way. There are better deterministic algorithms, see for example [SW97] and [MN99, Section 7.12]

Karger and Stein [KS96] found a simple and efficient randomized algorithm. We will introduce their algorithm in two steps. We first describe an extremely simple algorithms that finds the minimum cut with a nonzero, but small probability. In a second step, we will show how to increase the success probability. The resulting algorithm is a Monte Carlo algorithm. It finds the minimum cut with good probability, but is not guaranteed to find the minimum cut.

1.1 A First Algorithm

This section follows Section 1.1 in Motwani/Raghavan and Section 1.4 in Mitzenmacher/Upfal.

The algorithm contracts edges until a graph with only two vertices is obtained. It then returns the resulting cut.

Let $e = (u, v)$ be an edge of G. Contraction of e produces a graph with one less vertex, namely vertices u and v are removed and all edges connecting u and v are removed. A new vertex z is added and all edges incident to either u and v (except those connecting u and v) are made incident to z.

After a sequence of contractions, every vertex of the current graph represents a set of vertices of the original graph. It is easy to keep track of this set using linear lists.

```
define $S_v = \{v\}$ for all vertices $v$ of $G$;
while $|V| \geq 3$ do
    choose a random edge $e = (u, v)$ and contract it;
    let $z$ be the new vertex and define $S_z = S_u \cup S_v$;
end while
let $u$ and $v$ be the two remaining vertices; return $(S_u, S_v)$.
```

Figure 1: The contraction algorithm for finding min-cuts.

Theorem 2.1. *Let $(S, \bar{S})$ be a fixed min-cut in G. The algorithm above returns this cut with probability at least*

$$\frac{2}{n(n-1)}.$$

Proof. Let C be the set of edges of the min-cut, i.e., the set of edges having exactly one endpoint in S and one endpoint in $\bar{S}$, and let k be its capacity. The algorithm finds the cut if and only if it never contracts an edge in C. Let $G_n = G$, G_{n-1}, G_{n-2}, ..., G_3, G_2 be the sequence of graphs generated by the algorithm. The following Lemma is key for the analysis.

Lemma 2.2. *Let k be the capacity of the minimum cut of a graph G. Then G has at least $nk/2$ edges.*

Proof. For any vertex v, consider the cut $(v, V \setminus v)$. Its capacity is at least k. Thus every vertex has degree at least k and hence G has at least $nk/2$ edges. □

Let E_i be the event that no vertex of G_i (interpreted as a set of vertices of G) intersects S and $\bar{S}$. In other words, every vertex of G_i is either a subset of S or $\bar{S}$. In other words, the edges in C are edges of G_i. The event E_2 occurs if and only if the algorithm returns the cut $(S, \bar{S})$. We want to lower bound the probability of the event E_2. The probability of the event E_n is 1. Also,

$$\mathbf{Pr}[E_2] = \mathbf{Pr}[E_2 \mid E_3] \cdot \mathbf{Pr}[E_3] E_4 \cdots \mathbf{Pr}[E_{n-1}] E_n \mathbf{Pr}[E_n].$$

Assume E_i and let m_i be the number of edges of G_i. Then $m_i \geq ik/2$ since G_i has a i vertices and hence

$$\mathbf{Pr}[E_{i-1} \mid E_i] = \frac{m_i - k}{m_i} \geq \frac{ik/2 - k}{ik/2} = \frac{i-2}{i}.$$

Thus

$$\mathbf{Pr}[E_2] \geq \frac{1}{3} \cdot \frac{2}{4} \cdots \frac{n-2}{n} = \frac{2}{n(n-1)} = \frac{1}{\binom{n}{2}}.$$

□

Let $N = \binom{n}{2}$. The simple contraction algorithm returns any particular min-cut with probability at least $1/N$. This has a somewhat surprising consequence. There are at most $n(n-1)/2$ distinct min-cuts.

Is there any use in an algorithm that has a success guarantee as low as $\Theta(1/n^2)$? We could run it many times and return the best cut found. After an expected number of $\binom{n}{2}$ repetitions, we have seen a min-cut at least once. But now the running time is at least n^3 and this is slower than the best deterministic algorithm.

Let us do $\alpha N \ln n$ repetitions. Then the probability that we have not seen a min-cut is

$$(1 - \frac{1}{N})^{\alpha N \ln n} \leq \left(\frac{1}{e}\right)^{\alpha \ln n} = \frac{1}{n^\alpha}.$$

Exercise 1. *Show how to implement the simple contraction algorithm so that it runs in near linear time $O((n+m)\alpha(n+m))$. You may assume that you for any given L you can choose a random integer in $\{1, \ldots, L\}$ in time $O(1)$. Store the graph as an array of edges. Each edge knows its two endpoints in the original graph. Each vertex has a list of its incident edges. After a contraction, somehow reorder the edges. Use a union-find data structure to keep track of the set of vertices represented by a vertex of the current graph.*

1.2 FastCut

This section follows Section 10.2 in Motwani/Raghavan.

There is a more clever way to increase the success probability. Note that in the first iteration the probability that we do not destroy the cut $(S, \bar{S})$ is at least $n - 2/n$. This is close to one. The probability that the cut still exists in G_t is

$$\begin{aligned}\mathbf{Pr}[E_t] &= \mathbf{Pr}[E_t \mid E_{t+1}] \cdot \mathbf{Pr}[E_{t+1} \mid E_{t+2}] \cdots \mathbf{Pr}[E_{n-1} \mid E_n] \mathbf{Pr}[E_n] \\ &\geq \frac{t-1}{t+1} \cdots \frac{n-2}{n} \\ &= \frac{(t-1)t}{(n-1)n}.\end{aligned}$$

Lemma 2.3. *Let $(S, \bar{S})$ be a min-cut in G. The probability that the cut still exists in G_t is $\Omega(t^2/n^2)$. In particular, for $t = n/\sqrt{2}$, the probability is at least $1/2$.*

I am ignoring here the fact that t must be an integer. For the details, I refer to Motwani/Raghavan.

In order to obtain a reasonable success guarantee for the simple contraction algorithm, one needs to run many copies of it. Contraction makes mistakes near the end of the sequence of contractions. At the beginning the chance of an error is low. This suggests to run only a small number of copies of the simple algorithm in parallel as long as the graph is still large and to increase the number as the graph becomes smaller.

Algorithm FastCut realizes this idea as follows. It makes a copy of G and then runs the contraction algorithm on both copies until the number of vertices is reduced to $t = n/\sqrt{2}$. It then invokes the algorithm recursively on the graphs obtained, i.e., it makes two copies of each graph and then runs the contraction algorithm on all four graphs until the number of vertices is reduced to $t/\sqrt{2} = n/2$. Then it makes two copies of each of the resulting graphs and runs the contraction algorithm on each of the eight graphs until the number of vertices is reduced to $(n/2)/\sqrt{2}$. The recursion ends once the resulting graphs have only a constant number of vertices ($n = 6$ works). For such small graphs, the exact min-cut is computed using a deterministic algorithm.

What is the running time of this algorithm? What is its success probability? The recursion depth is $2\log n$ as the size of the graphs is reduced by a factor of $1/\sqrt{2}$ between recursive calls.

We will not analyze the time complexity, but only upper bound the total number of vertices in all the graphs constructed. Let $V(n)$ be the maximum number of vertices constructed. Then $V(n) \le 2n + 2V(n/\sqrt{2}) = O(n^2)$. We double the number of subproblems at each recursion step but reduce the size only by a factor of $\sqrt{2}$. Thus the total size of the problems grows in each recursion step by a factor of $\sqrt{2}$. Since the recursion depth is $2\log n$, the growth is by a factor of $\sqrt{2}^{2\log n} = n$. We start with a problem of size n and end up with total size n^2. The bottom of the recursion dominates. Formally,

$$\begin{aligned}
V(n) &\le 2n + 2V(n/\sqrt{2}) = 2(n + V(n/\sqrt{2}) = 2(n + 2(n/\sqrt{2} + V(n/\sqrt{2}^2)) \\
&\le 2(n + 2^1 \frac{n}{\sqrt{2}} + 2^2 \frac{n}{\sqrt{2}^2} + \ldots) \\
&= 2n \sum_{1\le i\le 2\log n} \frac{2^i}{\sqrt{2}^i} \\
&= 2n \sum_{1\le i\le 2\log n} \sqrt{2}^i \\
&= O(n\sqrt{2}^{2\log n}) \\
&= O(n^2).
\end{aligned}$$

In comparison, running the simple contraction algorithm n^2 times will construct $\Theta(n^3)$ vertices.

Let us now turn to the success probability. Let $P(n)$ be the probability that the algorithms returns a min-cut when applied to a graph with n vertices. Then $P(n_0) = 1$, when we stop the recursion at n_0 vertices. The algorithm returns a min-cut, if the min-cut survives one of the contractions from n to $t = n/\sqrt{2}$ vertices (probability $1/2$ for each of the contraction sequences) and the corresponding recursive call returns a min-cut (probability $P(n/\sqrt{2})$). Thus a branch fails with probability $1 - \frac{1}{2}P(n/\sqrt{2})$ and hence both branches fails with probability $(1 - \frac{1}{2}P(n/\sqrt{2}))^2$. Thus

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

How can one solve such a recurrence? The first step is always a change of variables. Instead of indexing by problem size, we index by the depth of the recursion. Define $p(k) = P(n)$, where $k = 2\log n - O(1)$ is the recursion depth. We also replace the inequality by an equality. Then

$$p(k) = 1 - (1 - \frac{1}{2}p(k-1))^2 = p(k-1) - \frac{p(k-1)^2}{4}.$$

This looks already much simpler. The p_k are decreasing and converge to zero. Also $p(0) = 1$. It is usually easier to work with large quantities and therefore we define $q(k) = 1/p(k)$. Then $q(0) = 1$ and (after a short calculation)

$$q(k) = q(k-1) + \frac{1}{4} + \frac{1}{16q(k-1) - 4}.$$

Thus $q(k) \ge 1 + k/4$ and $q(k) \le 1 + k/2$ as a simple induction shows.[1] So $q(k) = \Theta(k)$ and hence $p(k) = \Theta(1/k)$ and hence $P(n) = \Theta(\frac{1}{\log n})$.

Theorem 2.4. *Algorithm Fast-Cut returns a min-cut with probability $\Omega(1/\log n)$.*

Exercise 2. *How often do you have to run the algorithm in order to increase the success probability to 1/2 or to $1/n$, respectively.*

[1]Since $q(k) \ge 1$ for all k, we have $q(k) \ge q(k-1)+1/4$. Thus $q(k) \ge 1+k/4$. Since $q(k) \ge 1$, we have $1/(16q(k-1)-4) \le 1/4$ and hence $q(k) \le 1 + k/2$.

```
Require: S and T are multisets of size n in {0, ... m − 1}; p is a prime greater or equal to m.
  Construct the polynomials S(x) and T(x) as described in the text.
  Choose a random α ∈ Z_p and computer S(α) mod p and T(α) mod p.
  If S(α) ≠ T(α) mod p, declare S ≠ T. This answer is correct. If S(α) = T(α) mod p, return that
  the two sets are equal. This answer may be incorrect.
```

Figure 2: An algorithm for deciding whether two sets S and T are equal.

2 Set Equality and Polynomial Identities

This section summarizes Sections 7.1 and 7.2 in Motwani/Raghavan.

Let S and T be two equal-sized multi-sets of integers in the range 0 to $m-1$. Let $n = |S| = |T|$. We want to decide whether S and T are equal.

The standard deterministic solution is to sort S and T and then decide equality by a linear scan over the sequences. If bucket sort is used, this takes time $O(n+m)$, where $n = |S| = |T|$.

With randomization, we can do differently. We associate a polynomial of degree n with each set, namely

$$S(x) = \prod_{z \in S}(x - z) \quad T(x) = \prod_{z \in T}(x - z)$$

and consider their difference $D(x) = S(x) - T(x)$. If the sets S and T are equal, the polynomials $S(x)$ and $T(x)$ are identical and hence $D(x)$ is identically zero. If S and T are not equal, $D(x)$ is non-vanishing and hence has at most n roots.

Theorem 2.5. *Let p be a prime greater or equal to m.*

- *Let $D(x) = \sum_{0 \le i \le} a_i x^i$ with $a_i \in \mathbb{Z}_p$ for all i and $a_n \neq 0$ be a polynomial. Let $D(x) = \prod_{1 \le i \le n}(x - \alpha_i)$ be a factorization of $D(x)$ over $\mathbb{Z}_p$. Then $D(\alpha) \neq 0 \bmod p$ for $\alpha \notin \{\alpha_1, \ldots, \alpha_n\}$. In particular,*

$$\mathbf{Pr}\left[D(\alpha) \neq 0 \bmod p\right] \ge \frac{p-n}{p} = 1 - \frac{n}{p}.$$

  For this statement it is assumed that α is chosen uniformly from $\mathbb{Z}_p$.

- *Assume $S \neq T$. Then $D(x) = \sum_{0 \le i \le} a_i x^i$ with $a_i \neq 0 \bmod p$ for some i.*

Proof. $\mathbb{Z}_p$ is a field and in a field a product is zero if and only if one of the factors is zero.[2] Thus $D(\alpha) = 0 \bmod p$ implies $\alpha = \alpha_i$ for some i.

For the second part, let C be the intersection of S and T. For example, if $S = \{2,2,3\}$ and $T = \{2,3,3\}$, $C = \{2,3\}$. Write $S = C \cup S'$ and $T = C \cup T'$. Then S' and T' are disjoint and

$$D(x) = S(x) - T(x) = \prod_{z \in C}(x - z) \cdot \left(\prod_{z \in S'}(x - z) - \prod_{z \in T'}(x - z)\right) = C(x)D'(x).$$

By the first part, there are at most $|C|$ elements α for which $C(\alpha) = 0$. The polynomial $D'(x)$ has degree at most $n - |C|$. It is non-vanishing, since for any $z \in T'$, we have $D'(z) = S'(z) - T'(z) = S'(z) \neq 0 \bmod p$. Thus there are at most $n - |C|$ elements α with $D'(\alpha) = 0$. We conclude that there are at most n elements α for which $D(\alpha) = 0 \bmod p$. Thus D is non-vanishing. □

The theorem above is readily extended to multivariate polynomials.

Theorem 2.6 (Schwartz-Zippel)**.** *Let $Q(x_1, \ldots, x_n)$ be a non-vanishing multivariate polynomial with coefficients in the field F of total degree d, i.e., for any monomial $x_1^{e_1} \ldots x_n^{e_n}$ of Q, we have $\sum_i e_i \le n$. Let $S \subseteq F$ be a finite set. For r_1 to r_n chosen independently and uniformly at random from S, we have*

$$\mathbf{Pr}\left[Q(r_1, \ldots, r_n) = 0\right] \le \frac{d}{|S|}.$$

[2] If p is not a prime, $\mathbb{Z}_p$ is not a field and this is not true. Take $p = 100$. Then $10^2 = 100 = 0 \bmod 100$ and hence $x(x-20) = x^2 - 20x = x^2 - 20x + 100 = (x-10)(x-10) \bmod 100$. Thus the polynomial $D(x) = x^2 - 20x$ has several factorizations.

Proof. The proof is by induction on the number of variables. A non-vanishing univariate polynomial of degree d has at most d roots. This establishes the result for $n = 1$.

If $n > 1$, we write the polynomial as a polynomial in the variable x_n, say

$$Q(x_1, \ldots, x_n) = \sum_{0 \le i \le k} Q_i(x_1, \ldots, x_{n-1}) x_n^i,$$

where the degree of Q in x_n is equal to k. Then $Q_k(x_1, \ldots, x_{n-1})$ is non-vanishing and its total degree is at most $d - k$.

Consider now a random tuple $(r_1, \ldots, r_n)$. If $Q(r_1, \ldots, r_n) = 0$ then either $Q_k(r_1, \ldots, r_n) = 0$ or r_n is a zero of the non-vanishing polynomial $\sum_{0 \le i \le k} Q_i(r_1, \ldots, r_{n-1}) x_n^i$ of degree k. The former probability is at most $(d-k)/|S|$ and the later probability is at most $k/|S|$. Thus

$$\begin{aligned}
&\mathbf{Pr}\left[Q(r_1, \ldots, r_n) = 0\right] \\
&\quad \le \mathbf{Pr}\left[Q_k(r_1, \ldots, r_{n-1}) = 0\right] + \mathbf{Pr}\left[\sum_{0 \le i \le k} Q_i(r_1, \ldots, r_{n-1}) r_n^i = 0 \;\middle|\; Q_k(r_1, \ldots, r_{n-1}) = 0\right] \\
&\quad \le \frac{d-k}{|S|} + \frac{k}{|S|} = \frac{d}{|S|}.
\end{aligned}$$

□

A useful special case is as follows.

Theorem 2.7. *Let $u \in \mathbb{Z}_p^n$ be a nonzero vector. For a vector $r \in \mathbb{Z}_p^n$ of n independently chosen random components,*

$$\mathbf{Pr}\left[u^T r \ne 0\right] = 1/p.$$

Proof. We could appeal to Schwarz-Zippel with $S = \mathbb{Z}_p$ and $d = 1$. Let us give a direct proof.

Assume $u_i \ne 0$. Then $u^T r = 0$ iff $u_i r_i = -\sum_{j \ne i} u_j r_j$ iff $r_i = -\left(\sum_{j \ne i} u_j r_j\right)/u_i$. For fixed r_j, $j \ne i$, there is thus one exactly choice for r_i. Thus $\mathbf{Pr}\left[u^T r = 0\right] = 1/p$. □

Theorem 2.7 leads to an efficient randomized algorithm for checking matrix products.

```
**Require:** $A$, $B$, and $C$ are $n \times n$ matrices over $\mathbb{Z}_p$, where $p$ is a prime.
**Ensure:** decide if $A = B \cdot C$.
  choose a random vector $r \in \mathbb{Z}_p^n$ and compute $Ar$ and $B(Cr)$. The latter is two matrix-vector
  products. **return** equal if $Ar = B(Cr)$ and unequal otherwise.
```

Figure 3: An algorithm for verifying matrix products.

If $A = B \cdot C$, the algorithm always returns "equal". However, if $A \ne B \cdot C$, it may also return "equal". Thus the algorithm is a Monte Carlo algorithm (it may give the wrong answer) with one-sided error (the answer "unequal" is always correct, however, the answer "equal" may be incorrect.).

Theorem 2.8 (Freiwalds)**.** *The probability that the algorithm incorrectly returns "equal" is at most* $1/p$.

Proof. Let $D = A - B \cdot C$ and assume $D \ne 0$. Then D has at least one nonzero row, say u, and $\mathbf{Pr}\left[u^T r \ne 0\right] = 1/p$. □

References

[KS96] D.R. Karger and C. Stein. A new approach to the minimum cut problem. *Journal of the ACM*, 43(4):601–640, 1996.

[MN99] K. Mehlhorn and S. Näher. *The LEDA Platform for Combinatorial and Geometric Computing.* Cambridge University Press, 1999.

[SW97] M. Stoer and F. Wagner. A simple min-cut algorithm. *Journal of the ACM*, 44(4):585–591, July 1997.