CSE 203A: Advanced Algorithms

Lecture Notes: Verifying Matrix Multiplication and Randomized Min-Cut Algorithm

Lecturer: Prof. Ramamohan Paturi
Scribe: Shrestha Malik

January 13, 2015

In earlier lectures, we discussed some basic concepts of probability theory (probability space, independent events, total and conditional probability, inclusion exclusion principle, Bayes Law) and randomized algorithm for polynomial testing. In this lecture, we consider two more simple applications of randomized algorithms and analyze their performance.

Verifying Matrix Multiplication

Given three $n \times n$ matrices A,B and C: $A, B, C \in R^{n \times n}$
We want to verify whether

$$AB = C$$

We assume integer module 2 operations. One way of doing this, is by multiplying A and B and then comparing each element of the product matrix AB with C. This is a deterministic algorithm and always produces a correct output. As the simple matrix multiplication operation takes $O(n^3)$ time (the more sophisticated Coppersmith- Winograd algorithm takes $O(n^{2.37})$ time) this method of verification takes $O(n^3)$ time.

We discuss a randomized algorithm, Freivalds algorithm, which takes $O(n^2)$ time and has a high probability of correctness.

Algorithm 1 Verify Matrix Multiplication

1: **Input:** Matrices $A, B, C \in R^{n \times n}$
2: $\bar{r} \leftarrow$ Draw a random vector $\bar{r} = (r_1, r_2, ...r_n) \in \{0, 1\}^n$
3: Compute matrix vector multiplications $B\bar{r}, A(B\bar{r}), C\bar{r}$
4: **if** $A(B\bar{r}) \neq C\bar{r}$ **then**
5: output $AB \neq C$
6: **else**
7: output $AB = C$
8: **end if**

Analysis

Complexity: Since there are three matrix vector multiplications, and each takes $O(n^2)$ time, the time complexity of this algorithm is $O(n^2)$.

Correctness: This randomized algorithm has a one-sided error property:
Case 1: When $AB = C$, it always outputs $AB = C$, which is the correct output
Case 2: When $AB \neq C$, it may output $AB = C$ for choices of $\bar{r}$ that set $AB\bar{r} = C\bar{r} \Rightarrow (AB - C)\bar{r} = 0$

We now claim that the probability of the algorithm giving an error, i.e., returning $AB = C$ when actually $AB \neq C$, is not more than $\frac{1}{2}$.

Theorem 1. *If $AB \neq C$, then for vector $\bar{r} \in \{0,1\}^n$ chosen uniformly at random $Pr(AB\bar{r} = C\bar{r}) \leq \frac{1}{2}$*

Proof. Let $D = AB - C \neq 0$ (as $AB \neq C$ given)
As $D \neq 0$, we can assume that there is a non-zero element in D. Let it be D_{11}.

Let $D\bar{r} = \bar{z}$.
Now, $AB\bar{r} = C\bar{r} \Rightarrow D\bar{r} = \bar{z} = 0 \Rightarrow z_i = 0, \forall 1 \leq i \leq n$.
So, the $Pr(\bar{z} = 0) \leq Pr(z_i = 0)$.

For z_1 :

$$z_1 = \sum_{j=1}^{n} D_{1j} r_j = 0$$

$$D_{11} r_1 + \sum_{j=2}^{n} D_{1j} r_j = 0$$

$$r_1 = -\frac{\sum_{j=2}^{n} D_{1j} r_j}{D_{11}} \tag{1}$$

Lemma 1. *Choosing a vector $\bar{r} \in \{0,1\}^n$ uniformly at random is equivalent to choosing each $r_i, \forall 1 \leq i \leq n$ independently and at random from $\{0,1\}$.*
Proof. *Each r_i can take two values 0 or 1 giving us 2^n possible vectors. And the probability of choosing any one vector is 2^{-n} which is the same as choosing one of the 2^n vectors.*

We argue that once we have selected $r_2, r_3, \ldots, r_n \in \{0,1\}$, there is only one value of r_1 for which (1) is satisfied. And since r_1 can take values either 0 or 1, the probability of choosing r_1 that satisfies (1) is at most $\frac{1}{2}$.

Hence, the conditional probability,

$$\Pr\left(\frac{r_1 = -\frac{\sum_{j=2}^{n} D_{1j} r_j}{D_{11}}}{(r_2, r_3, \ldots, r_n) = (x_2, x_3, \ldots, x_n)} \right) \leq \frac{1}{2} \tag{2}$$

Using the law of total probability,

$$
\begin{aligned}
&\Pr\left(AB\bar{r} = C\bar{r}\right) \\
&\quad = \sum_{(x_2,x_3,\ldots,x_n)\in\{0,1\}^{n-1}} \Pr\left((AB\bar{r} = C\bar{r}) \cap ((r_2, r_3, \ldots, r_n) = (x_2, x_3, \ldots, x_n))\right) \\
&\quad = \sum_{(x_2,x_3,\ldots,x_n)\in\{0,1\}^{n-1}} \Pr\left((\bar{z} = 0) \cap ((r_2, r_3, \ldots, r_n) = (x_2, x_3, \ldots, x_n))\right) \\
&\quad \leq \sum_{(x_2,x_3,\ldots,x_n)\in\{0,1\}^{n-1}} \Pr\left((z_i = 0) \cap ((r_2, r_3, \ldots, r_n) = (x_2, x_3, \ldots, x_n))\right) \\
&\quad = \sum_{(x_2,x_3,\ldots,x_n)\in\{0,1\}^{n-1}} \Pr\left((r_1 = -\frac{\sum_{j=2}^{n} D_{1j} r_j}{D_{11}}) \cap ((r_2, r_3, \ldots, r_n) = (x_2, x_3, \ldots, x_n))\right) \\
&\quad = \sum_{(x_2,x_3,\ldots,x_n)\in\{0,1\}^{n-1}} \Pr\left(\frac{r_1 = -\frac{\sum_{j=2}^{n} D_{1j} r_j}{D_{11}}}{(r_2, r_3, \ldots, r_n) = (x_2, x_3, \ldots, x_n)}\right) \Pr\left((r_2, r_3, \ldots, r_n) = (x_2, x_3, \ldots, x_n)\right) \\
&\quad \leq \sum_{(x_2,x_3,\ldots,x_n)\in\{0,1\}^{n-1}} \frac{1}{2} \Pr\left((r_2, r_3, \ldots, r_n) = (x_2, x_3, \ldots, x_n)\right) \\
&\quad = \frac{1}{2}
\end{aligned}
$$

Hence,

$$\Pr\left(AB\bar{r} = C\bar{r}\right) \leq \frac{1}{2}$$

Or, the probability of getting an error is not more than $\frac{1}{2}$.

To improve this probability we can make k independent iterations of this algorithm. If we come across a $\bar{r}$ such that $AB\bar{r} \neq C\bar{r}$, then the output $AB \neq C$ is correct. Else, if for all k iterations, $AB\bar{r} = C\bar{r}$, then we can say that the output $AB = C$, has a probability of error less than $(\frac{1}{2})^k$. The run time for k iterations, will be $O(kn^2)$.

Randomized Min-Cut Algorithm

Definitions:
A *cut* (A, B) of an undirected, unweighted graph $G = (V, E)$, is a partition of its nodes V into two non-empty sets $A, B \subset V$, such that $A \cup B = V$ and $A \cap B = \emptyset$.

A *cut-set* is the set of edges crossing the cut (A, B).

For $u, v \in V$ and $e_{u,v} \in E$,

$$cut - set(A, B) = \{e_{u,v} \mid u \in A \cap v \in B\}$$

The *min-cut* is the cut set with the minimum cardinality.

$$min - cut(G) = \arg\min_{cut(A,B)} \mid cut - set(A, B) \mid$$

Observation A graph with $|V|$ vertices has $2^{|V|-1} - 1$ different cuts.
Proof. This is because each vertex can either be in set A or B, giving us $2^{|V|}$ possibilities of partitioning them. A and B cannot be empty sets so we get $2^{|V|} - 2$ possibilities. Among these, as cut (A, B) is equivalent to cut (B, A), we get only $\frac{1}{2}$ of them as the total possible cuts.

Algorithm

The randomised algorithm for finding min-cut was given by Karger and is also called Karger's Algorithm. Given an undirected, unweighted graph $G = (V, E)$ with no self-loops, this randomised algorithm returns a min-cut of the graph with small probability of error. The main operation in the algorithm is that of edge contraction.

Edge Contraction: When contracting edge $e_{u,v} \in E$ we merge the two nodes u and v together eliminating all the edges between them, and retaining all the other edges. Hence, there are no self loops but there could be parallel edges. Each edge contraction results in one less vertex in the graph.

Algorithm 2 Randomized Min Cut Algorithm

1: **procedure** EDGE-CONTRACTION($G, e_{u,v}$)
2: Replace vertices u and v with a new supervertex $w = u, v$
3: Replace all edges $e_{x,u}$ and $e_{x,v}$ with $e_{x,w}$
4: Remove all self loops
5: **end procedure**

Karger's algorithm performs edge contraction on the graph till there are only two vertices left. So, there are a total of $|V| - 2$ iterations and in each iteration the algorithm chooses an edge from the existing edges uniformly at random and contracts it. After $|V| - 2$ iterations, the graph has two vertices which represent the cut and the edges represent the cut-set.

Algorithm 2 Randomized Min Cut Algorithm

1: **Input:** Graph $G = (V, E)$
2: **while** $|V| > 2$ **do**
3: $e_c \leftarrow$ Choose an edge $e \in E$ uniformly at random
4: EDGE-CONTRACTION(G, e_c)
5: **end while**
6: **Output:** G

Observation Any cut-set of the intermediate graph is also a cut-set in the original graph. However, a cut set in the original graph may not be a cut-set in the intermediate graph.
Proof In any intermediate iteration, we do not add any new edges but we do eliminate some exisiting edges of the graph. So, though some of the edges present in the cut-set of the original graph may not exist in the intermediate graph, all edges in the intermediate graph are present in the original graph also.

This algorithm does not always produce a min-cut, as can be seen in the Figure 1 and we will now demonstrate that the probability of it producing a min cut is at least $\frac{2}{|V|(|V|-1)}$.

Analysis

Theorem 2. *The algorithm outputs a min-cut with a probability at least* $\frac{2}{n(n-1)}$*, where* $n = |V|$.

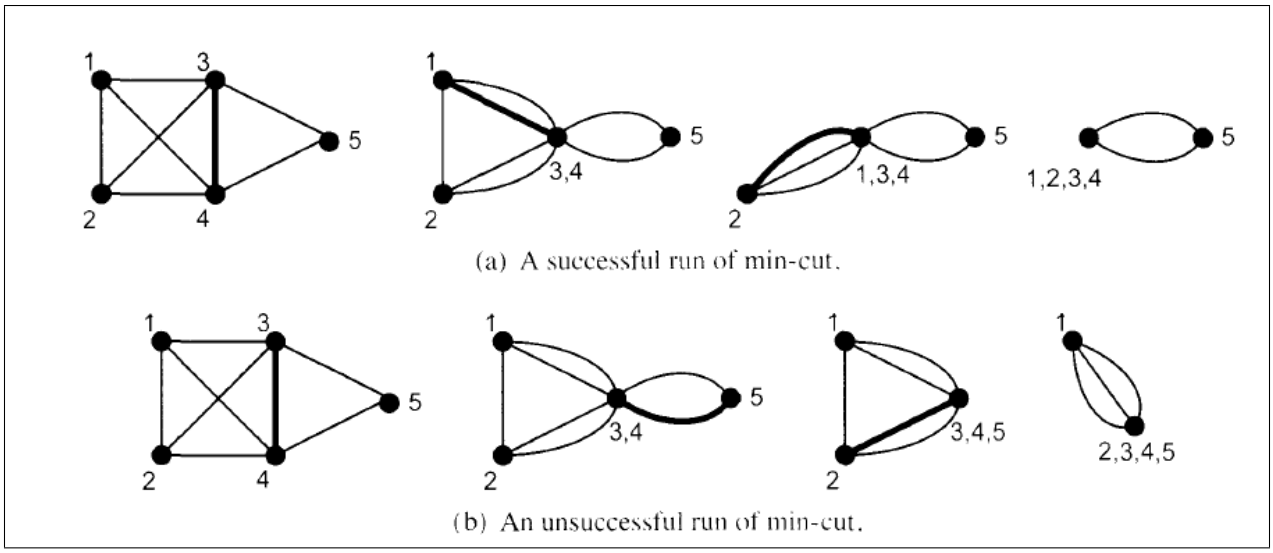

Figure 1: Two examples demonstrating Krager's algorithm

Proof. Let C be one of the min cut of the graph and k be the cardinality of the cut-set of C. Or, the size of the min-cut is k.

For the algorithm to output this min-cut C, it should not select any of the k cut edges during its $n-2$ iterations. Moreover, as k is the size of the min-cut of the graph, each vertex has a degree of at least k. For if any vertex had less than k degree, then it would form the min-cut and the size of the min-cut will be less than k.

So, $deg(v) \geq k, \forall v \in V$.

As each edge connects two vertices it contributes a degree of 2 to the total degree of all vertices.

$$2|E| = \sum_{v \in V} deg(v) \geq kn$$

$$|E| \geq \frac{nk}{2}$$

So, the probability that an edge of C is selected from $|E|$ edges is

$$\Pr \text{ (An edge of C is selected) } \leq \frac{k}{\frac{nk}{2}} = \frac{2}{n}$$

Let E_i be the event that the edge selected in the ith iteration is not in C. And let F_i be the event that none of the edges that get selected in the first i iterations are in C. So, $F_i = \cap_{j \leq i} E_i$.

Hence, for the first iteration, $i = 1$:

$$\Pr(E_1) = \Pr(F_1) \geq 1 - \frac{2}{n}$$

For the second iteration, $i = 2$, given that none of the min-cut edges are selected in the first iteration, we are left with $(n-1)$ vertices and the degree of each vertex is again at least k. By similar logic, the number of remaining edges is at least $\frac{k(n-1)}{2}$. Now,

$$\Pr\left(\frac{E_2}{F_1}\right) \geq 1 - \left(\frac{k}{\frac{k(n-1)}{2}}\right) = 1 - \left(\frac{2}{n-1}\right)$$

$$\Pr\left(F_2\right) = \Pr\left(E_2 \cap F_1\right) = \Pr\left(\frac{E_2}{F_1}\right).\Pr(F_1)$$

Similarly,

$$\Pr\left(\frac{E_i}{F_{i-1}}\right) \geq 1 - \left(\frac{2}{n-i+1}\right)$$

Now, for the algorithm to output the min cut C, it should not select any of the cut-set edges of C in each of the $n-2$ iterations.

Hence,

$$\Pr\left(\text{Algorithm outputs min -cut}\right) = \Pr(F_{n-2})$$

$$\begin{aligned}
\Pr(F_{n-2}) &= \Pr(E_{n-2} \cap F_{n-3}) = \Pr(\frac{E_{n-2}}{F_{n-3}}).\Pr(F_{n-3}) \\
&= \Pr(\frac{E_{n-2}}{F_{n-3}}).\Pr(\frac{E_{n-3}}{F_{n-4}}) \ldots \Pr(\frac{E_2}{F_1}).\Pr(F_1) \\
&\geq \prod_{i=1}^{n-2} \left(1 - (\frac{2}{n-i+1})\right) \\
&= \prod_{i=1}^{n-2} (\frac{n-i-1}{n-i+1}) \\
&= (\frac{n-2}{n})(\frac{n-3}{n-1})(\frac{n-4}{n-2}) \ldots (\frac{3}{5})(\frac{2}{4})(\frac{1}{3}) \\
&= \frac{2}{n(n-1)}
\end{aligned}$$

Hence,

$$\Pr\left(\text{Algorithm outputs min -cut }\right) = \Pr(F_{n-2}) \geq \frac{2}{n(n-1)}$$

To improve the probability of getting a min-cut we can make k independent iterations of this algorithm and return the output cut with the minimum number of edges as the final answer. The probability that the algorithm does not return a min cut is

$$\Pr\left(\text{Algorithm does not return min-cut}\right) \leq \left(1 - \frac{2}{n(n-1)}\right)$$

The probability that the algorithm does not return a min-cut in any of the k independent iterations is

$$\Pr\left(\text{Algorithm does not return min-cut}\right) \leq \left(1 - \frac{2}{n(n-1)}\right)^k$$

And, the probability that the algorithm returns a min-cut in any one of the k independent iterations is

$$\Pr\left(\text{Algorithm outputs a min-cut}\right) \geq \left(1 - \left(1 - \frac{2}{n(n-1)}\right)^k\right)$$

Using the identity, $1 - x \leq e^{-x}$

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

$$\left(1 - \frac{2}{n(n-1)}\right)^k \leq e^{-\left(\frac{2k}{n(n-1)}\right)}$$

So,

$$\Pr\left(\text{Algorithm does not return min-cut}\right) \leq e^{-\left(\frac{2k}{n(n-1)}\right)}$$

On choosing $k = n(n-1)\ln(n)$,

$$\Pr\left(\text{Algorithm does not return min-cut}\right) \leq \frac{1}{n^2}$$

And,

$$\Pr\left(\text{Algorithm outputs a min-cut}\right) \geq \left(1 - \frac{1}{n^2}\right)$$

Hence, by making $n(n-1)\ln(n)$ independent iterations of the algorithm, the probability of it returning a min-cut is at least $\left(1 - \frac{1}{n^2}\right)$. And in general by choosing $k = c\frac{n(n-1)\ln(n)}{2}$ where c is a constant, we get the probability of algorithm returning a min-cut is in order of $\left(1 - \frac{1}{n^c}\right)$.