1 Karger’s MIN-CUT algorithm

Let $G$ be an undirected and unweighted graph. A cut of $G$ is a set of edges whose removal disconnects the graph $G$. The size of a cut $C$ is the number of edges in $C$. A min-cut of $G$ is a cut of $G$ whose size is minimized over all cuts of $G$.

We will use the following problem to motivate the ideas of randomized algorithms:

**MIN-CUT.** Construct a min-cut for a given undirected graph $G$.

The problem can be solved using an algorithm for the MAX-FLOW problem, for which currently the best deterministic algorithm runs in time $O(n^3)$ [8]. A deterministic algorithm for MIN-CUT, which runs faster for sparse graphs and is not based on algorithms for MAX-FLOW, is due to Nagamochi and Ibaraki [15] and runs in time $O(nm + n^2 \log n) = O(n^3)$.

We present randomized algorithms for the MIN-CUT problem, which were originated by Karger [11] and later were improved by Karger and Stein [12]. Karger’s algorithm is extremely simple. A straightforward implementation of Karger’s algorithm runs in time $O(n^4)$. The refined algorithm by Karger and Stein runs in time $O(n^2 \log^2 n)$, which is better than the fastest deterministic algorithm for the problem.

We first consider the following algorithm, where $G/e$ denotes the graph $G$ with the edge $e$ contracted. A formal description of contracting an edge $e = [u, v]$ is given as follows: We first remove all edges between $u$ and $v$, then merge the vertices $u$ and $v$ into a single vertex $w$ (so that all edges of the form $[x, u]$ or $[x, u]$ in $G$, where $x \neq u, v$, now becomes an edge between $x$ and $w$.

**Algorithm 1 Contraction**

Input: An undirected graph $G$;
Output: A cut of $G$;

1. $G_0 = G$; $h = 0$;
2. while $G_h$ has more than 2 vertices do
   randomly pick an edge $e_h$ in $G_h$;
   $G_{h+1} = G_h/e_h$; $h = h + 1$;
3. return all edges in $G_{n-2}$.

We would like to add some explanations on how the algorithm **CONTRACTION** is implemented. Suppose that the input graph $G$ has $n$ vertices and $m$ edges. If $G$ is a simple graph, then $m < n^2$, and the graph $G$ can be represented by its adjacency matrix, which is an $n \times n$ matrix. However, if $G$ has multiple edges, as we allowed in the execution of the algorithm, then $m$ can be much larger than $n^2$. In this case, we should first go through the edges, and find for each pair of vertices the number of edges between them. With an $O(m)$-time preprocessing, we can construct the adjacency matrix $M_G$ for the graph $G$, in which $M_G[u, v]$ gives the number of edges between the vertices $u$ and $v$. Note that $M_G[u, u] = 0$ for all $u$ since we assume the graph $G$ has no self-loops. If the MIN-CUT problem can be solved in time $O(t(n))$ based on the adjacency matrix representation of the graph $G$, then the MIN-CUT problem in general form can be solved in time $O(m + t(n))$. Therefore, we will assume from now on that the input $G$ is given by an $n \times n$ adjacency matrix $M_G$.

Contracting an edge $[i, j]$ in the graph $G$ represented by its $n \times n$ adjacency matrix $M_G$ can be implemented as follows. Without loss of generality, assume $i < j$. We first add the $i$-th row to the $j$-th row in $M_G$. Note that this does not change the $i$-th column of the matrix since $M_G[i, i] = 0$. Then we add the $i$-th column to the $j$-th column. Again this does not change the $i$-th row of the matrix. Then we mark the $i$-th row and the $i$-th column of the matrix as “unusable”. The resulting matrix $M'_G$.
can be regarded as an \((n - 1) \times (n - 1)\) matrix, with the \(i\)-th row and the \(i\)-th column of the matrix “crossed”. Technically, \(M'_G\) represents the graph \(G'\) obtained from \(G\) by contracting the edge \([i, j]\), with all edges between \(i\) and \(j\) becoming self-loops on the vertex \(j\). Therefore, by setting \(M'_G[j, j] = 0\), we get the matrix \(M''_G\), which is the adjacency matrix for the graph \(G'[/i, j]\).

Since the \(i\)-th row and \(i\)-th column are unchanged during this process, we can easily get back the original matrix \(M_G\) from the matrix \(M''_G\), by subtracting the \(i\)-th row from the \(j\)-th row, and the \(i\)-th column from the \(j\)-th column (and setting \(M_G[j, j] = 0\)). In conclusion, this shows that the adjacency matrix for the graph \(G'[/i, j]\) can be constructed from the adjacency matrix for the graph \(G\) in time \(O(n)\).

This is one more issue we need to clarify: under the adjacency matrix representation \(M_G\), how do we “randomly pick an edge in the graph?” For this, we consider the upper-right triangle matrix \(M^\Delta_G\) of the matrix \(M_G\) (note that the graph \(G\) is undirected so the matrix \(M_G\) is symmetric along the diagonal). We prepare an array \(W[1..n]\) in which \(W[i]\) is the sum of the values in the \(i\)-th row in \(M^\Delta_G\). Thus, \(W[i]\) is the number of edges between \(i\) and \(j\) with \(i < j\), and \(\sum_{i=1}^n W[i]\) is total number of edges in the graph \(G\). Now to randomly pick an edge in \(G\), we can apply the following subroutine:

Algorithm 2 RandomPick\((M_G, W)\)

1. \(m = \sum_{i=1}^n W[i]\);
2. \(t = \text{rand()} \% m + 1;\)
3. \(i = 1; w = W[1];\)
4. \(\text{while } t > w \text{ do } \{ i = i + 1; \ w = w + W[i] \};\)
5. \(t = t - (w - W[i]); \ j = 0;\)
6. \(\text{while } t > 0 \text{ do } \{ j = j + 1; \ t = t - M_G[i, j] \};\)
7. \(\text{return}([i, j]).\)

Step 2 of the above algorithm, “\(t = \text{rand()} \% m + 1,\)” is supposed to give, with a uniform probability, an integer between 1 and \(m\). Here we have used the notation from C++, which generates such a “pseudo-random” integer. By the study in random number generations [14], the numbers generated this way are sufficiently good for the purpose of randomized algorithms. Also note that the array \(W[1..n]\) can be easily updated when an edge in the graph \(G\) is contracted if it is represented by its adjacency matrix. Finally, steps 3-6 of the algorithm RandomPick finds a pair of vertices \(i\) and \(j\) such that one of the edges between \(i\) and \(j\) is the \(t\)-th edge in the graph \(G\). The completes the description that a random edge can be picked in time \(O(n)\).

Summarizing the above discussion, we conclude that step 2 of the algorithm CONTRACTION, which randomly picks an edge then contracts it, takes time \(O(n)\), which proves the following lemma.

Lemma 1.1 For a graph \(G\) of \(n\) vertices, represented by its adjacency matrix, we can do the following in time \(O(n)\): randomly pick an edge then contract it. In particular, the algorithm CONTRACTION runs in time \(O(n^2)\).

The following facts can be easily observed:

Lemma 1.2 Let \(G\) be a graph and let \(e\) be an edge in \(G\). Then (1) Every cut for the graph \(G/e\) is also a cut for the graph \(G\); and (2) for a fixed min-cut \(C\) of \(G\) and for an edge \(e\) not in \(C\), \(C\) also makes a min-cut for \(G/e\).

Proof. Lemma 1.2(1) is simple: for a cut \(C'\) of \(G/e\), the vertex \(v\) of \(G/e\) resulted from the contraction of the edge \(e\) is in a connected component of \((G/e) \setminus C'\). Thus, expanding the vertex \(v\) back to the edge \(e\) would not make this connected component to connect with other components, i.e., \(G \setminus C'\) is still disconnected so \(C'\) is a cut for \(G\). To see Lemma 1.2(2), first observe that the cut \(C\) of \(G\) is obviously also a cut for \(G/e\). Thus, the size of \(C\) is not smaller than that of a min-cut for \(G/e\). Moreover, by Lemma 1.2(1), every min-cut of \(G/e\) is also a cut for \(G\). Thus, the size of a min-cut of \(G/e\) is not smaller than that of a min-cut of \(G\), which is equal to the size of \(C\). Combining these, we conclude that \(C\) is also a min-cut of \(G\). □
Fix a min-cut $C$ for $G$. By Lemma 1.2, if we can ensure that during the execution of the while-loop in step 1 of the algorithm CONTRACTION, each time the edge $e_h$ picked for the contraction is not in $C$, then $C$ remains as a min-cut for all the graphs $G_i$ constructed in step 1. In particular, since the graph $G_{n-2}$ in step 2 has only two vertices, it has a unique min-cut, which consists of all the edges between the two vertices. On the other hand, since $C$ remains as a min-cut for $G_{n-2}$, we conclude that the set returned in step 2, i.e., the output of the algorithm CONTRACTION, is just the min-cut $C$ for the original input graph $G = G_0$.

Thus, now the problem becomes: “what is the probability that none of the edges picked in step 1 of the algorithm is in the min-cut $C$?”

Let $E_i$ be the event that the $i$-th edge $e_i$ picked by the algorithm CONTRACTION is not in the min-cut $C$. Then, the probability that the min-cut $C$ remains in the graph $G_{n-2}$ when the algorithm reaches step 2, i.e., the probability that the algorithm CONTRACTION returns a correct min-cut of the input graph, is $\Pr[\bigcap_{i=1}^{n-2} E_i]$.

By the definition in Probability Theory, $\Pr[A|B] = \Pr[A \cap B]/\Pr[B]$, or $\Pr[A \cap B] = \Pr[A|B] \cdot \Pr[B]$. Therefore, we have

$$\Pr\left[\bigcap_{i=1}^{n-2} E_i\right] = \Pr\left[E_{n-2} \cap \left(\bigcap_{i=1}^{n-3} E_i\right)\right] = \Pr\left[E_{n-2} \cap \bigcap_{i=1}^{n-3} E_i\right] \cdot \Pr\left[\bigcap_{i=1}^{n-3} E_i\right]$$

We repeatedly apply this equality, and will get

$$\Pr\left[\bigcap_{i=1}^{n-2} E_i\right] = \Pr\left[E_{n-2} \cap \bigcap_{i=1}^{n-3} E_i\right] \cdot \Pr\left[\bigcap_{i=1}^{n-3} E_i\right]$$

$$= \Pr\left[E_{n-2} \cap \bigcap_{i=1}^{n-3} E_i\right] \cdot \Pr\left[E_{n-3} \cap \bigcap_{i=1}^{n-4} E_i\right] \cdot \Pr\left[\bigcap_{i=1}^{n-4} E_i\right]$$

$$= \cdots$$

$$= \Pr\left[E_{n-2} \cap \bigcap_{i=1}^{n-3} E_i\right] \cdot \Pr\left[E_{n-3} \cap \bigcap_{i=1}^{n-4} E_i\right] \cdots \Pr[E_2 \cap E_1] \cdot \Pr[E_1].$$

Now we consider the probability $\Pr\left[E_h \cap \bigcap_{i=1}^{h-1} E_i\right]$ for a general $h$. Under the condition $\bigcap_{i=1}^{h-1} E_i$, no edges in the min-cut $C$ were picked for contraction in the first $(h-1)$-st execution in step 1 of the algorithm. By Lemma 1.1, $C$ remains as a min-cut for the graph $G_{h-1}$ after the first $(h-1)$-st iterations of the while-loop in step 1. The number of vertices in the graph $G_{h-1}$ is $n_{h-1} = n - h + 1$. Let $k = |C|$, then each vertex $v$ in $G_{h-1}$ has degree at least $k$. Thus, the total number of edges in $G_{h-1}$ is at least $kn_{h-1}/2$. Thus, the probability that an edge in $C$ is picked for contraction in the $h$-th iteration is not larger than $k/(kn_{h-1}/2) = 2/n_{h-1}$. In conclusion, under the condition $\bigcap_{i=1}^{h-1} E_i$, the probability of the event $E_h$ is at least $1 - 2/n_{h-1}$:

$$\Pr\left[E_h \cap \bigcap_{i=1}^{h-1} E_i\right] \geq 1 - \frac{2}{n_{h-1}} = 1 - \frac{2}{n - h + 1}.$$
Bring this in Equation (1), we get

\[
\Pr \left[ \bigcap_{i=1}^{n-2} E_i \right] = \Pr \left[ E_{n-2} \bigg| \bigcap_{i=1}^{n-3} E_i \right] \cdot \Pr \left[ E_{n-3} \bigg| \bigcap_{i=1}^{n-4} E_i \right] \cdots \Pr[E_2 \mid E_1] \cdot \Pr[E_1]
\]

\[
\geq \left( 1 - \frac{2}{3} \right) \left( 1 - \frac{2}{4} \right) \cdots \left( 1 - \frac{2}{n-1} \right) \left( 1 - \frac{2}{n} \right)
\]

\[
= \left( \frac{1}{3} \right) \left( \frac{2}{4} \right) \left( \frac{3}{5} \right) \cdots \left( \frac{n-4}{n-2} \right) \left( \frac{n-3}{n-1} \right) \left( \frac{n-2}{n} \right)
\]

\[
= \frac{2}{n(n-1)}
\]

\[
\geq \frac{2}{n^2}.
\]

Therefore, the probability that the algorithm \textsc{Contraction} fails in returning a correct min-cut of the input graph \(G\) is bounded by \(1 - 2/n^2\). Now consider the following algorithm:

\begin{algorithm}[H]
\caption{Karger}
\begin{algorithmic}
\State Input: An undirected graph \(G\);
\State Output: A cut of \(G\);
\State 1. run the algorithm \textsc{Contraction} \(tn^2\) times;
\State 2. return the cut that is the smallest among those constructed in step 1.
\end{algorithmic}
\end{algorithm}

\section*{Theorem 1.3}
The algorithm \textsc{Karger} returns a min-cut of the graph \(G\) with a probability at least \(1 - 1/e^{2t}\), where \(e = 2.718 \cdots\) is the base of the natural logarithm.

\section*{Proof.}
The algorithm \textsc{Karger} does not return a min-cut of \(G\) if none of the calls on \textsc{Contraction} in step 1 returns a min-cut of \(G\). By the above discussion, the probability that \textsc{Contraction} does not return a min-cut is bounded \(1 - 2/n^2\). Therefore, the probability that none of the calls on \textsc{Contraction} in step 1 returns a min-cut of \(G\), i.e., the probability that the algorithm \textsc{Min-Cut} does not return a min-cut of \(G\) is bounded by

\[
\left( 1 - \frac{2}{n^2} \right)^{tn^2} = \left[ \left( 1 - \frac{2}{n^2} \right)^{n^2/2} \right]^{2t} \leq e^{-2t},
\]

where we have used the inequality \((1 - 1/x)^x \leq e^{-1}\) for \(x > 0\). The theorem then follows.

Thus, if we let \(t = 10\), then the probability that the algorithm \textsc{Karger} returns a min-cut of \(G\) is larger than 0.99999999.

Since the algorithm \textsc{Contraction} runs in time \(O(n^2)\), we conclude with the following theorem:

\begin{theorem}
For any constant \(\epsilon > 0\), the algorithm \textsc{Karger} can be implemented to run in time \(O(n^4)\), and returns a min-cut for the input graph \(G\) with a probability larger than \(1 - \epsilon\).
\end{theorem}

\section*{Appendix}
We provide a proof for an inequality we have used in (2), which will be one of the most used inequalities we need for analysis of randomized algorithms.
Lemma 1.5  For any real number $x \neq 0$, $1 + x < e^x$.

Proof. Consider the function $f(x) = 1 + x - e^x$. The derivative of $f(x)$ is

$$
(f(x))' = 1 - e^x \begin{cases} < 0 & \text{if } x > 0 \\ = 0 & \text{if } x = 0 \\ > 0 & \text{if } x < 0 \end{cases}
$$

Thus, $f(x)$ is strictly decreasing for $x > 0$, and strictly increasing for $x < 0$. Since $f(0) = 0$, we have

$$
(f(x)) = 1 + x - e^x \begin{cases} < 0 & \text{if } x > 0 \\ = 0 & \text{if } x = 0 \\ < 0 & \text{if } x < 0 \end{cases}
$$

That is, $f(x) < 0$ for $x \neq 0$, i.e., $1 + x < e^x$ for $x \neq 0$. \qed

Note that Lemma 1.5 holds true for any real number $x \neq 0$. In particular, if we let $x = y/t$, where $y$ and $t$ are non-zero real numbers, then we get $1 + t/y < e^{t/y}$. This gives the following theorem:

Theorem 1.6  Let $t$ and $y$ be both positive real numbers. Then $1 + t/y < e^t$, and $(1 - t/y)^y < e^{-t}$.

Proof. By Lemma 1.5, we have $1 + t/y < e^{t/y}$. Taking to the $y$-th power in both sides gives $(1 + t/y)^y < e^t$. To prove the second equality, we have, by Lemma 1.5, $1 - t/y = (1 + (-t/y)) < e^{-t/y}$. Again take to the $y$-th power in both sides, noting that $y > 0$, we have $(1 - t/y)^y < e^t$. \qed

Note that in equation (2) we have used Theorem 1.6.