1 Karger’s Mincut Algorithm

The problem of Mincut can be formally defined as:

**Input:** A connected graph \( G = (V, E) \)

**Output:** A set \( E' \subseteq E \) of the smallest size such that \( G \setminus E' \) is disconnected

In this lecture we’ll discuss a simple Monte Carlo algorithm for Mincut.

1.1 Contract

This algorithm for Mincut heavily relies on the CONTRACT function which takes the graph \( G \) and an edge \{u, v\} as input and returns a graph after removing vertices \( u \) and \( v \) and updating the edges. Following is the pseudocode for this function:

```
function CONTRACT(G, \{u, v\})
    1) Replace \( u \) and \( v \) by a new vertex called \( uv \)
    2) Delete all edges \{u, v\}
    3) For every edge \{w, u\}, replace it by \{w, uv\}.
    4) Similarly for every edge \{w, v\}, replace it by \{w, uv\}.
end function
```

This allows parallel edges between vertices.

Figure 1: Example showing the application of Karger’s algorithm.
1.2 Karger’s Algorithm

Following is the pseudocode for the Karger’s mincut algorithm.

```plaintext
function KARGER(G)
   for i ← 1 to n − 2 do
      Pick an edge {u, v} in G uniformly at random
      G ← CONTRACT(G, {u, v})
   end for
   return Edges between two vertices that remain
end function
```

**Note:** This pseudocode can be modified to return information regarding which edges cross the cut.

**EXAMPLE:**
Figure ?? demonstrates the working of Karger’s algorithm.

**Observation:** This algorithm doesn’t always give the correct output. Figure ?? shows another way in which the graph in Figure ?? could be contracted by Karger’s algorithm. As can be seen in the figure, this doesn’t give the mincut.

We want to figure out the probability for Karger’s algorithm to compute the right value of Mincut. For this, let’s first take a look at the intuition behind when would Karger’s algorithm compute the correct mincut.
Intuition: Figure ?? shows a graph $G$ having two sets of vertices $A$ and $B$ such that the edges between these two sets is a mincut, represented by $C$.

In order to make sure that the mincut $C$ is not changed by the contractions, the algorithm should always contract edges within $A$ or $B$. Let’s compute the probability of Karger’s algorithm not picking an edge in $C$, to contract.

**Lemma 1.** Let $C$ be a set of edges representing mincut in the graph $G$ having $n$ number of vertices.

$$P[\text{Karger’s algorithm does not contract an edge in } C] \geq \frac{2}{n(n-1)}$$

**Proof.** Let $E_i$ (for $i = 1, 2, 3, \ldots n-2$) denote the event that the Karger’s algorithm does not contract an edge in $C$ in iteration $i$.

Let

$$F_i = \bigcap_{j=1}^{i} E_j \quad \text{for } i = 1, 2, 3, \ldots n-2.$$  

**Goal:** To show that

$$P[F_{n-2}] \geq \frac{2}{n(n-1)}.$$  

Let $G_i$ denote the graph after iteration $i$ and let $G_0 = G$. Then

$$P[F_1] = P[E_1]$$

$$P[E_1] = 1 - P[E_1]$$

$$P[E_1] = \frac{k}{e(G_0)}$$

where,

$E_1$ denotes the event that Karger’s algorithm did choose an edge in $C$ to contract, and $k = |C|$, i.e., number of edges in mincut $C$ and $e(G_0)$ number of edges in graph $G_0$. 
More generally $e(G_i)$ represents the number of edges in graph $G_i$. Since the mincut is of size $k$, every vertex in the graph $G$ should have degree at least $k$ (Reminder: $n$ represents the total number of vertices). Therefore,

$$e(G_0) \geq \frac{n \cdot k}{2}$$

$$P[E_1] \leq \frac{k}{\frac{n \cdot k}{2}} = \frac{2}{n}$$

$$P[E_1] \geq 1 - \frac{2}{n}.$$ 

Now we consider $F_2$.

$$P[F_2] = P[E_1 \cap E_2]$$

$$P[F_2] = P[E_1].P[E_2|E_1]$$

$$P[E_2|E_1] = 1 - P[E_2|E_1]$$

$$P[E_2|E_1] = \frac{k}{e(G_1)}$$

**Note:** Since $G_1$ has one less vertex than $G_0$, number of vertices in $G_1$ would be $n - 1$. And since we didn’t contract any edge in $C$, the size of $C$ remains the same in $G_1$ as well, i.e., $k$. Therefore

$$P[E_2|E_1] \leq \frac{k}{\frac{(n-1) \cdot k}{2}} = \frac{2}{n - 1}$$

$$P[E_2|E_1] \geq 1 - \frac{2}{n - 1}$$

$$P[F_2] \geq \left(1 - \frac{2}{n}\right) \cdot \left(1 - \frac{2}{n - 1}\right)$$

Now consider $F_{n-2}$.

$$P[F_{n-2}] = P[E_1].P[E_2|E_1].P[E_3|E_1 \cap E_2]...P[E_{n-2}|E_1 \cap E_2 \cap \ldots \cap E_{n-3}]$$

$$P[F_{n-2}] \geq \left(1 - \frac{2}{n}\right) \cdot \left(1 - \frac{2}{n - 1}\right) \cdot \left(1 - \frac{2}{n - 2}\right) \cdots \left(1 - \frac{2}{3}\right) = \frac{2}{n(n - 1)}$$
In order to improve this probability we can repeat the Karger’s algorithm $T$ times. Let’s compute the probability of this updated Karger’s algorithm giving the correct answer.

\[
P[ \text{algo returns correct answer } ] = 1 - P[ \text{algo returns wrong answer } ]
\]

\[
P[ \text{algo returns wrong answer } ] = P[ \text{all } T \text{ iterations return wrong answer }]
\]

Since the iterations are independent

\[
P[ \text{algo returns wrong answer } ] = \prod_{j=1}^{T} P[ j^{th} \text{ call to Karger’s algorithm returns wrong answer}]
\]

\[
P[ \text{algo returns wrong answer } ] \leq \left(1 - \frac{2}{n(n-1)}\right)^T
\]

Using, $1 + x \leq e^x$

\[
P[ \text{algo returns wrong answer } ] \leq e^{\frac{-2.T}{n(n-1)}}
\]

Plugging, $T = \frac{n(n-1)}{2} \cdot \ln(n)$

\[
P[ \text{algo returns wrong answer } ] \leq e^{-\ln(n)} = \frac{1}{n}
\]

So this means that if we repeat the Karger’s algorithm $T$ times where $T = \frac{n(n-1)}{2} \cdot \ln(n)$ we can increase the probability of finding the correct answer considerably, to $\geq 1 - \frac{1}{n}$.

2 Linearity of Expectation

The expectation of the sum of two random variables $X$ and $Y$ is just the expectation of $X$ added with the expectation of $Y$. This is called the linearity of expectation.

\[
\mathbb{E}(X + Y) = \mathbb{E}(X) + \mathbb{E}(Y)
\]

It doesn’t depend on any interaction between $X$ and $Y$ i.e., the linearity of expectation holds even when $X$ and $Y$ are correlated. We’ll now show the use of linearity of expectation in the analysis of randomized quick sort.

Randomized Quick Sort

Following is the pseudocode for randomized quick sort.

```
function RANDOMQS(L)
    if |L| \leq c(const) then
```
return nonRecursiveQS(L) // e.g., BubbleSort or InsertionSort
else
    pick an index \( i \in \{1, 2, \ldots, |L|\} \) uniformly at random
    Swap(L, i, 1)
    \( L_1 \leftarrow \emptyset \)
    \( L_2 \leftarrow \emptyset \)
    for \( i \leftarrow 2 \) to \(|L|\) do
        if \( L[i] \leq L[1] \) then
            append to \( L_1 \)
        else
            append to \( L_2 \)
        end if
    end for
    return RandomQS(L_1).p.RandomQS(L_2)
end if
end function

We will show in the next class that \( E(T) = O(n \log n) \) where, \( n = |L| \) and \( T \) = random variable denoting the running time of RandomQS(L).