CSC473 Assignment 1

Jeff Blair: 1002177057
jeffrey.blair@mail.utoronto.ca

Jaryd Hunter: 1002725893
jaryd.hunter@mail.utoronto.ca

February 2020

Question 1

a. Given in the question we know there are at most $r-1$ nodes with degree less than $|E(S,\bar{S})|$, then there are at least $n-r+1$ nodes such that the degree of the node is greater than or equal to $|E(S,\bar{S})|$. So we get the following inequality:

$$2|E| \geq (n-r+1)|E(S,\bar{S})|$$

Then we can use this inequality to bound the probability for e_1 in $E(S,\bar{S})$,

$$\begin{aligned} P(A_1) =& 1 - \frac{|E(S,\bar{S})|}{|E|} \\ \geq& 1 - \frac{2}{(n-r+1)} \\ =& \frac{n-r-1}{n-r+1} \end{aligned}$$

We can extend this probability bound to:

$$\begin{aligned} P(A_i|A_1,...,A_{i-1}) \geq& 1 - \frac{2}{n-r+1-i} \\ =& \frac{n-r-i}{n-r+2-i} \end{aligned}$$

Then we have:

$$\begin{aligned} P(A_1,A_2,...,A_{n-r-1}) =& P(A_1)P(A_2|A_1)P(A_3|A_1,A_2)...P(A_{n-r-1}|A_1,...,A_{n-r-2}) \\ \geq& \frac{n-r-1}{n-r+1} \cdot \frac{n-r-2}{n-r} \cdot ... \cdot \frac{n-r-(n-r-1)}{n-r+2-(n-r-1)} \\ =& \frac{(n-r-(n-r-2))(n-r-(n-r-1))}{(n-r+1)(n-r)} \\ =& \frac{2}{(n-r+1)(n-r)} \end{aligned}$$

b. Let A_i^j be the event that an edge in the j^{th} smallest cut was not contracted in the i^{th} iteration.

Recall from lecture if we run the base contraction algorithm we can find the min cut with constant probability in $O(n^4)$. Let the probability of failure be 0.1.

If we want to find the second smallest cut in G, we first need a bound on $|E|$ relative to $|E(S_2, \bar{S}_2)|$. Notice that there is at most one node of degree less than $|E(S_2, \bar{S}_2)|$, so $2|E| \geq (n-1)|E(S_2, \bar{S}_2)|$. So, then we have:

$$\begin{aligned}
P(A_1^2) =& 1 - \frac{|E(S_2, \bar{S}_2)|}{|E|} \\
\geq& 1 - \frac{2}{(n-1)} \\
=& \frac{n-3}{n-1}
\end{aligned}$$

Then,

$$\begin{aligned}
P(A_i^2 | A_1^2, ..., A_{i-1}^2) \geq& 1 - \frac{2}{n-i} \\
=& \frac{n-i-2}{n-i}
\end{aligned}$$

Then, we can change the contraction algorithm to stop when there are 3 vertices left, and check a constant number of cuts remaining comparing each to the smallest cut already found. This gives us the following probability of the second smallest cut surviving when three nodes remain:

$$\begin{aligned}
P(A_1^2, A_2^2, ..., A_{n-3}^2) =& P(A_1^2)P(A_2^2|A_1^2)P(A_3^2|A_1^2 \text{ and } A_2^2)...P(A_{n-3}^2|A_1^2, ..., A_{n-4}^2) \\
\geq& \frac{n-3}{n-1} \cdot \frac{n-4}{n-2} \cdot ... \cdot \frac{1}{3} \\
=& \frac{2}{(n-1)(n-2)}
\end{aligned}$$

Using this probability we can estimate the complexity to find the cut $(S_2, \bar{S}_2)$ with constant probability.

$$\begin{aligned}
P(\text{failure}) \leq& (1 - \frac{2}{(n-1)(n-2)})^T \\
\leq& e^{\frac{-2T}{(n-1)(n-2)}} \\
=& e^{-\ln(\frac{1}{\delta})} \\
=& \delta
\end{aligned}$$

$$\text{Let } T = \frac{(n-1)(n-2)}{2} \ln(\frac{1}{\delta})$$

Let $\delta = 0.1$, then $T = \frac{(n-1)(n-2)}{2} \ln(10) = O(n^2)$, so given we have the smallest cut, finding the second smallest cut with constant probability will run in $O(n^4)$.

Similarly, to find the third smallest cut in G, we need a bound on $|E|$ relative to $|E(S_3, \bar{S}_3)|$. Notice that there is at most two nodes of degree less than $|E(S_3, \bar{S}_3)|$, so $2|E| \geq (n-2)|E(S_3, \bar{S}_3)|$. So, then we have:

$$\begin{aligned}
P(A_1^3) =& 1 - \frac{|E(S_3, \bar{S}_3)|}{|E|} \\
\geq& 1 - \frac{2}{(n-2)} \\
=& \frac{n-4}{n-2}
\end{aligned}$$

Then,

$$\begin{aligned}P(A_i^3|A_1^3,...,A_{i-1}^3) \geq& 1-\frac{2}{n-i-1}\\ =& \frac{n-i-3}{n-i-1}\end{aligned}$$

Then, we can change the contraction algorithm to stop when there are 4 vertices left, and check each cut remaining comparing each to the smallest and second smallest cuts already found. Again there is a constant number of cuts left when 4 vertices remain. This gives us the following probability of the third smallest cut surviving when four nodes remain:

$$\begin{aligned}P(A_1^3,A_2^3,...,A_{n-4}^3) =& P(A_1^3)P(A_2^3|A_1^3)P(A_3^3|A_1^3 \text{ and } A_2^3)...P(A_{n-4}^3|A_1^3,...,A_{n-4}^3)\\ \geq& \frac{n-4}{n-2}\cdot\frac{n-5}{n-3}\cdot ... \cdot\frac{1}{3}\\ =& \frac{2}{(n-2)(n-3)}\end{aligned}$$

Using this probability we can estimate the complexity to find the cut $(S_3, \bar{S}_3)$ with constant probability.

$$\begin{aligned}P(\text{failure}) \leq& (1-\frac{2}{(n-2)(n-3)})^T\\ \leq& e^{\frac{-2T}{(n-2)(n-3)}}\\ =& e^{-\ln(\frac{1}{\delta})}\\ =& \delta\end{aligned}$$

$$\text{Let } T = \frac{(n-2)(n-3)}{2}\ln(\frac{1}{\delta})$$

Let $\delta = 0.1$, then $T = \frac{(n-2)(n-3)}{2}\ln(10) = O(n^2)$, so given we have the smallest cut, and second smallest cut finding the third smallest cut with constant probability will run in $O(n^4)$. Since each section runs in $O(n^4)$ then the entire process runs in $O(n^4)$.

Finally, we know:

$$\begin{aligned}P(\text{find } S_1 \text{ and find } S_2 \text{ and find } S_3) =& P(\text{find } S_1)P(\text{find } S_2|\text{find } S_1)P(\text{find } S_3|\text{find } S_1 \text{ and find } S_2)\\ =& (1-0.1)^3 = 0.729 > \frac{2}{3}\end{aligned}$$

Question 2

a.

```
def ColouredPath(G(V, E), K={1,...,k}):
    """
    @typedef PATH[K, v]:
        Hashtable (set, vertex) -> bool
        If PATH[K, v] returns 1 then there is a path that ends at vertex v
        and contains every colour in K exactly once
    """
    PATH <- 0 # All entries initiated to zero
    For v in V: # Base case
        PATH[c(v), v] = 1

    For k in Powerset(K):
        # Powerset(K) returns the set of all subsets of K in O(k2^k),
        # in increasing order. The set returned contains 2^k elements
        For v_1 in V:
            For each neighbour v_2 of v_1:
                if c(v_1) not in k: # O(k) by assumption
```

```
                k' = k.join({c(v_1)}) # join with 1 element is O(1)
                PATH[k', v_1] |= PATH[k, v_2] # where 'x |= y' equates to
                                              # 'x = x OR y'

    return ANY(PATH[K, v] for v in V) # O(n)
```

Correctness:

Claim 1:

It is possible to return (in increasing order) Powerset(K) in $O(2^k k)$

Proof:

Represent K={1,..,k} as a binary integer with k bits, where the i-th bit represents the i-th entry in the set K. Enumerate Powerset(K) by incrementing the integer by 1. There are a total of 2^k elements in Powerset(K), so enumerating every element takes $O(2^k)$. Then sort in $O(2^k \log 2^k) = O(2^k k)$

Claim 2:

There is a path P that contains every colour in K exactly once $\iff \exists v \in V : PATH[K, v]$

Proof:

Suppose $\exists v \in V: \; PATH[K, v] = 1$
$\implies$ a path P hits every colour in K exactly once by definition

Suppose a path exists that hits every colour in K exactly once and ends in vertex v. Then there must exist a path to a neighbour of v that hits every color in $K \setminus c(v)$ exactly once.

Base case: $\forall k \in K : \exists v \in P \mid PATH[k, v] = 1$ from the first loop
Inductive Step: $\forall k \in 2^K$:

If k is a colouring along the path P and k' is the set of colours when the next vertex along P is considered:

$$\exists (v, v') \in P \mid k = k' \setminus c(v') \text{ and } PATH[k, v] \implies PATH[k', v'].$$

Since the algorithm considers all pairs of neighbouring vertices and considers each subset of K in increasing order, we are guaranteed to find (v, v').

Claim 3:

The total running time of the algorithm is $O(2^k kn^2)$

Proof:

Preprocessing the Powerset of K: $O(2^k k)$ by Claim 1.
Initializing PATH is constant time assignments to n elements $\rightarrow O(n)$
Main loop is $O(2^k) \cdot O(n) \cdot O(n) \cdot O(k) = O(2^k kn^2)$
$\implies T(n) = O(2^k k) + O(n) + O(2^k kn^2) = O(2^k kn^2)$

b. Assume a path P of length k, and vertex colouring is performed uniformly at random. Then for vertices $v_1, ..., v_k \in P$:

$$\begin{aligned}
P(c(v_1) \neq ... \neq c(v_k)) &= 1 \cdot P(c(v_1) \neq c(v_2)) \cdot ... \cdot P(c(v_1) \neq ... \neq c(v_k) \mid c(v_1) \neq ... \neq c(v_{k-1})) \\
&= \frac{k}{k} \cdot \frac{k-1}{k} \cdot ... \cdot \frac{1}{k} \\
&= \frac{k!}{k^k}
\end{aligned}$$

```
def kPath(G, k):
    G' <- random uniform coloring of G with k colors
    return ColouredPath(G', k)
```

If there exists no Path of length k in G, kPath always returns NO. If there does exist a path, kPath returns YES with probability $\frac{k!}{k^k}$ in $O(2^k k n^2)$.

$$P(\text{Success}) = \frac{k!}{k^k} \geq e^{-k} = \frac{1}{e^k}$$

Running the algorithm $T = e^k \log(\frac{1}{\delta})$ many times gives us

$$\begin{aligned}
P(\text{Failure}) &\leq \left(1 - \frac{1}{e^k}\right)^T \\
&\leq e^{-\frac{1}{e^k} T} \\
&\leq e^{-\frac{1}{e^k} \cdot T} \\
&\leq e^{-\frac{1}{e^k} \cdot (-e^k \log(\delta))} \\
&\leq \delta \\
P(\text{Success}) &\geq 1 - \delta
\end{aligned}$$

This gives us a probability of returning YES of at least $\frac{2}{3}$ in
$T(n) = \log(3) \cdot e^k \cdot O(2^k k n^2) = O(e^k 2^k k n^2) = O((2e)^k k n^2)$