CSC473 Assignment 1: Randomized k-Path

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

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

February 2020

Introduction:

A well-studied problem in graph theory is the Hamiltonian Path Problem (HAMPATH): Deciding whether there exists a simple path on a graph G that visits every node exactly once. This paper provides an efficient, randomized approach to solving the HAMPATH problem on a graph G.

Randomized k-PATH

We see that the HAMPATH problem on n vertices is a special case of the k-PATH problem where $k = n$. For our k-PATH problem, we will consider a k-Coloring of the graph G where each vertex is assigned a color $c(v) \in \{1, \ldots, k\}$ uniformly at random. Using this coloring we can define the following recurrence:

Let PATH(S, v) be a boolean function that returns 1 iff there is a path on G ending at vertex v that contains each color in the set S exactly once. Then

$$(v_1, v_2) \in G : S \subset [k] : c(v_1) \in S : c(v_2) \notin S : \text{PATH}(S \cup c(v_2), v_2) = \text{PATH}(S, v_1)$$

Using this recurrence, we can use dynamic programming to efficiently compute solutions to the k-PATH problem.

```
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 $O(2^k) \cdot O(n) \cdot O(n) \cdot O(k) = O(2^k kn^2)$
$\implies T(n) = O(2^k k) + O(n) + O(2^k kn^2) = O(2^k kn^2)$
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 $O(2^k kn^2)$.

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

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
$T(n) = \log(3) \cdot e^k \cdot O(2^k kn^2) = O(e^k 2^k kn^2) = O((2e)^k kn^2)$

Setting $k = n$, we see an algorithm that computes HAMPATH with time complexity $O((2e)^n n^3)$, which is an improvement over the brute force complexity $O(n!)$.