KAIST, School of Computing, Spring 2024 Scribed By: Eunjung KIM
Algorithms for NP-hard problems (CS492) Lecture #9
Lecture: Eunjung KIM 26 March 2024

Contents

We consider a parameterized version of the LONGEST PATH problem. The problem k-PATH is, give a graph G and an integer k, to find a simple path on k vertices, if one exists.

> k-PATH
> **Instance:** a graph $G = (V, E)$, a positive integer k.
> **Question:** Does G have a path on k vertices?

This problem is NP-complete. We give a randomized FPT-algorithm for k-PATH. The underlying idea is to transform (in a randomized way) the given graph into a graph in which detecting a k-path becomes a simpler task. It is known that on a directed acyclic graph(DAG), finding a longest (directed) path can be solved in time $O(|E|)$ using dynamic programming. So, we shall transform G into a DAG $\vec{G}$ so that a (directed) k-path in $\vec{G}$ corresponds to a k-path in G. A k-path in G does not necessarily yield a k-path in $\vec{G}$. The hope is that if we perform the transformation sufficiently many times, but not too many times to stay within the running time bound of FPT-algorithm, we will hit on $\vec{G}$ which contains a k-path corresponding to a k-path in G.

Let $\pi : V(G) \to [n]$ be a random permutation of $V(G)$. A DAG $\vec{G}_\pi$ can be defined from π: it has $V(G)$ as the vertex set, and

(u, v) is an arc of $\vec{G}_\pi$ if and only if $\pi(u) < \pi(v)$.

Notice that if G contains a k-path P, and π happens to order the vertices of P in an orderly manner (two possible ways), then P can be detected by a longest path algorithm on $\vec{G}_\pi$. If G does not contain a k-path, then no permutation π will allow $\vec{G}_\pi$ to contain a k-path.

The probability that a random permutation π turns a k-path into a directed k-path in $\vec{G}$ is $\frac{2}{k!}$. Therefore, the expected number of random permutations to hit a sucessful $\vec{G}$ is $\frac{k!}{2}$. For each random permutation π, we test[1] whether $\vec{G}_\pi$ contains a k-path in time $O(|E|)$. Therefore, the expected running time of to detect k-path in G, if G contains one, is $O(k! \cdot |E|)$.

1 Color-coding technique: k-PATH

We can improve the running time of k-PATH from $O^*(k!)$ to $2^{O(k)}$ time using color coding introduced by Alon, Yuster and Zwick. Color coding is a technique to transform a problem of detecting an object in a graph into a problem of colored object in a colored graphs, which is hopefully an easier task. In color coding for the problem k-PATH, we randomly color the vertices of G with k colors and the hope is that in the colored graph, a k-path becomes *colorful*. We say that a path in a colored graph is *colorful* if all vertices have distinct colors.

One pass of our color coding algorithm consists of two steps:

A. Color the vertices of G with $\{1, \dots, k\}$ uniformly at random. Let $c : V(G) \to [k]$ be the coloring.

B. Find a colorful k-path in G, if one exists. Otherwise, report that none was found.

[1] The problem LONGEST PATH is in P on acyclic digraphs. First, we obtain a topological order of the vertex set of $\vec{G}_\pi$, then solve compute the length of a longest path to each vertex via dynamic programming over this ordering.

Step A. The probability that step A. make a k-path P colorful is

$$\frac{\#\text{of colorings in which } P \text{ becomes colorful}}{\#\text{ of all possible colorings}} = \frac{k!}{k^k} \approx \frac{1}{e^k}$$

So, the expected number of runs of A. before a k-path P becomes colorful is e^k. Notice that any colorful k-path is also a k-path in G. Below, we provide an algorithm for B. running in time $O(2^k \cdot |E|)$.

Step B: Detecting a colorful k-path. Now we present an algorithm for detecting a colorful k-path given a vertex partition $V_1, \ldots, V_k$ of $V(G)$, where each V_i are the vertices colored in i. We aim to set the values of indicator variables $P[C, u]$ for every color subset $C \subseteq \{1, \ldots, k\}$ and for every vertex $u \in V(G)$, so that

$P[C, u] = 1$ if there is a colorful path exactly consisting of colors in C and ending in u.
$P[C, u] = 0$ otherwise.

At each i-th iteration over $i = 1, \ldots, k$, for all $u \in V(G)$ we set the value of $P[C, u]$ for $C \subseteq [k]$ with $|C| = i$ using dynamic programming. At $i = 1$, $P[C, u] = 1$ if and only if $C = \{c(u)\}$. At $i + 1$-th iteration, for each $u \in V(G)$ and $C \subseteq \{1, \ldots, k\}$ of size $i + 1$, we compute $P[C, u]$ as:

- $P[C, u] := 1$ if $c(u) \in C$ and there is $v \in N(u)$ such that $P[C \setminus c(u), v] = 1$.
- $P[C, u] := 0$ otherwise.

This recurrence computes $P[C, u]$ correctly indeed: if there is a colorful $i + 1$-path Q using colors in C and ending at u, then for a neighbor v which is a neighbor of u in Q, $Q - u$ is a colorful i-path using colors in $C \setminus \{c(u)\}$. Conversely, if for some neighbor v of u there is a colorful i-path using colors in $C \setminus \{c(u)\}$, such a path can be extended to a colorful $i + 1$-path by adding u. The new path uses colors in C and ends at u. As the base case when $i = 1$ trivially holds, the correctness of the above recurrence follows.

After finishing k-th iteration, there is a vertex u such that $P[\{1, \ldots, k\}, u] = 1$ if and only if there is a colorful k-path. This dynamic programming algorithm runs in time

$$O(\sum_{i=1}^{k} \binom{k}{i} \cdot |E|) = O(2^k \cdot |E|).$$

Lemma 1. *One can detect a colorful k-path in time $O(2^k \cdot |E|)$, if one exists.*

The following lemma summarizes the above analysis of Step A. and B.

Lemma 2. *One can detect a simple k-path in $O((2e)^k \cdot |E|)$ expected running time, if one exists.*

Lemma 3. *One can detect a simple k-path with probability at least e^{-1} in time $O((2e)^k \cdot |E|)$, if one exists.*

Proof: The probability that a coloring fails to turn a k-path P colorful is at most $1 - e^{-k}$. Therefore, the probability that all e^k colorings (each, independent at random) reports no colorful k-path is at most

$$(1 - \frac{1}{e^k})^{e^k} \approx e^{-1}.$$

Together with Lemma 1, the running time follows. □

2 Random Separation: SUBGRAPH ISOMORPHISM

Another useful technique for designing a randomized algorithm is a *random separation* technique. Like color-coding, it is useful to design an algorithm to detect a small-sized substructure in a graph.

We exemplify this technique with the problem SUBGRAPH ISOMORPHISM: given an input graph G and a pattern graph H on k vertices, the task is to find a copy of H in G or correctly decide that G does not have H as a subgraph. We take the parameter $k+d$, where d is the maximum degree of G and present a randomized algorithm that runs in time $2^{dk+O(k\log k)} \cdot n^{O(1)}$ which detect H as a subgraph in G with high probability, if exists one[2].

The intuition behind is to color the edges of G in blue or red so that the edges of H are 'isolate' in this coloring, and thus this isolated copy of H is easy to detect. Fix a subgraph $\tilde{H}$ of G which is isomorphic to H. A coloring $c : V(G) \to \{\text{red, blue}\}$ is *successful* if the next two conditions are satisfied:

1. all edges of $\tilde{H}$ is colored blue, and
2. all edges of $E(G) \setminus E(\tilde{H})$ incident with a vertex of $V(\tilde{H})$ is colored blue.

On how many edges does a successful coloring requests a specific color? All edges incident with a vertex of $\tilde{H}$ are requested to be either blue or red, depending on whether it belongs to $E(\tilde{H})$ or not. As there are at most $d \cdot V(\tilde{H}) = dk$ such edges, a random coloring c is successful with probability at least 2^{-dk}.

Now assume that the current coloring c is successful. Consider the subgraph G' of G consisting of the blue edges. Let us call a connected component in G' a blue component, and let $\mathcal{B}$ be the set of all blue components. Now we have narrow down which part of G we need to match H. To begin with, any blue component having more than k vertices has no chance of being $\tilde{H}$ (under a successful coloring).

Let $H_1, \dots, H_p$ be the connected components of H (possibly $p = 1$). We make a bipartite graph W in which one part of the vertex bipartition is $\mathcal{H} = \{H_1, \dots, H_p\}$ and the other part is $\mathcal{B}$, the set of all blue components. The bipartite graph W has an edge between $H \in \mathcal{H}$ and $B \in \mathcal{B}$ if H is isomorphic to B. As the isomorphism between H and B (on at most k vertices each) can be tested in time $k! \cdot k^{O(1)} = 2^{O(k\log k)}$, the construction of W can be done in time $2^{O(k\log k)} \cdot n$. It remains to observe that if $\tilde{H}$ exists and the current color is successful for $\tilde{H}$, this copy must be a disjoint union of p blue components each of which is isomorphic to $H_1, \dots, H_p$ respectively. We can decide whether such p blue components exist by examining whether a maximum matching on W exists saturating all vertices in $\mathcal{H}$. The latter problem is polynomial-time solvable.

To summarize, if G contains a copy of H (say $\tilde{H}$), then with probability at least 2^{-dk} a random coloring is successful for $\tilde{H}$. Given a successful coloring, $\tilde{H}$ can be correctly retrieved from G in time $2^{O(k\log k)} \cdot n^{O(1)}$. Let us call this procedure $\mathcal{A}$. The probability[3] that no copy of H is detected after 2^{dk} repetitions of $\mathcal{A}$ while G contains a copy of H is at most

$$(1-2^{-dk})^{2^{dk}} = (1-2^{-dk})^{(-2^{dk})\cdot(-1)} \approx e^{-1}$$

[2]Mind that if you parameterize by k only, then we cannot expect to have an fpt-algorithm: when H is a clique on k vertices, it is known that deciding if G contains H as a subgraph or not is known to be W[1]-hard (you will learn this notion in the next class) parameterized by k. This means that under a widely-accepted complexity assumption that $W[1] \neq FPT$, SUBGRAPH ISOMORPHISM is unlikely to be fixed-parameter tractable with respect to k only.

[3]We use the fact the natural log base e equals $\lim_{x\to 0}(1+x)^{\frac{1}{x}}$.

that is, with constant probability a copy of H is detected after 2^{dk} runs of $\mathcal{A}$. This constant success probability can be boosted to an arbitrarily high constant probability by repetitions.

3 Derandomization

The randomization technique of color-coding and random separation can be derandomized. For derandomizing color-coding, we use a family of functions called an $(n.k)$-*perfect hash family*. A family $\mathcal{F}$ of functions $f : [n] \to [k]$ is an $(n.k)$-*perfect hash family* if for every subset $S \subseteq [n]$ of size k, there exists $f \in \mathcal{F}$ such that f assigns pairwise distinct values to the elements of S. Note that if S is the fixed object that we want to find, then such f will make S colorful. A repetition of random colorings in color-coding technique can be replaced by an (n, k)-perfect hash family with almost negligible computational overhead, due to the following theorem.

Theorem 1 (Naor, Schulman, Srinivasan 1995)**.** *For every $n, k \geq 1$, an (n, k)-perfect hash family of size $e^{k+o(k)} \cdot \log n$ can be constructed in time $e^{k+o(k)} \cdot n \log n$.*

For random separation, we use a different method. An (n, k)-*universal set* $\mathcal{U}$ is a family of subsets of $[n]$ such that for any $S \subseteq [n]$ of size k, all possible subsets of S appear in the projection of $\mathcal{U}$ on S, that is, $\{S \cap A : A \in \mathcal{U}\} = 2^S$. In our application to SUBGRAPH ISOMORPHISM on graphs with maximum degree at most d, S will correspond to the set of edges incident with a vertex of $V(\tilde{H})$, whose size is at most 2^{dk}, and with a successful coloring we are looking for a partition of these edges into edges in $\tilde{H}$ and the rest. Now repeated random colorings can be replaced by trying the colorings in $\mathcal{U}$, interpreting a set $A \in \mathcal{U}$ as blue edges. This strategy works with little overhead because of the following theorem

Theorem 2 (Naor, Schulman, Srinivasan 1995)**.** *For every $n, k \geq 1$, an (n, k)-universal set of size $2^{k+o(k)} \cdot \log n$ can be constructed in time $2^{k+o(k)} \cdot n \log n$.*