# Résolution exacte de problèmes NP-difficiles
# Lecture 3: Randomized algorithms

4 February, 2019 Lecturer: Eunjung Kim

## 1 Simple randomized algorithms

### 1.1 $k$-Path

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. 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 one in $G$.

Let $\pi : V(G) \to [n]$ be a random permutation of $V(G)$. A DAG $\vec{G}_\pi$ can be defined from $\pi$: 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 $\pi$ 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 $\pi$ will allow $\vec{G}_\pi$ to contain a $k$-path.

The probability that a random permutation $\pi$ 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 $\pi$, 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 $O(k! \cdot |E|)$.

### 1.2 Feedback Vertex Set

We present a randomized algorithm for Feedback Vertex Set running in time $O^*(4^k)$. We first apply reduction rules first.

[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.

**Reduction Rule 0:** If $u$ has has a loop in $G$, then delete $v$ and decrease $k$ by one.
**Reduction Rule 1:** If $u$ has degree at most 1 in $G$ (and does not have a loop), then delete $v$.
**Reduction Rule 2:** If $u$ has degree 2 with neighbors $v, w$ in $G$ (possibly $v = w$), by delete $u$ and add an edge $(v, w)$.

Notice that application of Reduction Rule 2 may create parallel edges and loops - cycles of length two and one. Hence, $G$ is a graph with parallel edges in the remainder of this subsection. The degree of a vertex is the number of incident edges, not the number of neighbors. Provided Reduction Rules 0-2 have been applied exhaustively, we can assume that $G$ has minimum degree at least three.

The key observation behind the randomized $O^*(4^k)$-algorithm is sparsity of a forest. Let $S$ be a feedback vertex set of $G$. Then $G - S$ have at most $|V(G) \setminus S| - 1$ edges, which accounts for at most two among the minimum degree 3 of the vertices in $V(G) \setminus S$. Hence, more than $|V(G)| - |S|$ edges are lying between $S$ and $V(G) \setminus S$. This is formalized in the lemma below.

**Lemma 1.** *Let $G$ be a graph with minimum degree three. Then for any feedback vertex set $S$, at least half of $E$ are incident with $S$.*

**Proof:** We let $F := V(G) \setminus S$. Let us denote by $E(S)$ the set of edges whose both endpoints are in $S$ and by $E(S, F)$ the set of edges which has precisely one endpoint in each of $S$ and $F$. Let us count the sum $\sum_{v \in F} deg(v)$ in a different way. The only edges that contribute to this sum are $E(S, F) \cup E(F)$. Observe that an edge of $E(S, F)$ counts precisely once in this sum, and an edge of $E(F)$ is counted twice. Therefore, with the minimum degree condition on $V$ it holds that

$$\sum_{v \in F} deg(v) = |E(S, F)| + 2|E(S)| \geq 3|F|$$

It follows that

$$|E(S, F)| \geq 3|F| - 2|E(F)| \geq 3(|E(F)| + 1) - 2|E(F)| > |E(F)|,$$

where the second inequality is due to the fact that the number of edges in a tree $T$ is at most the number of vertices of $T$ minus one. Observe that the edge set incident with $S$ is $E(S) \cup E(F, S)$. From

$$|E(S)| + |E(S, F)| = \frac{1}{2}(2|E(S)| + 2|E(S, F)|) > \frac{1}{2}(|E(S)| + |E(S, F)| + |E(F)|) = \frac{1}{2}|E|,$$

we know that at least half of the edge set $E$ is incident with $S$. This complete the proof. □

Thanks to Lemma 1, a randomly chosen edge $e$ is incident with a (prescribed) feedback vertex set $S$ with probability at least $\frac{1}{2}$. By again randomly choosing one endpoint of $e$, we choose one of $S$ with probability at least $\frac{1}{4}$.

**Algorithm 1** Algorithm for FEEDBACK VERTEX SET

1: **procedure** FVS$(G, k)$
2: Apply Reduction Rules 1-2 exhaustively.
3: **if** $k = 0$ and $G$ has a cycle **then return** NO and terminate.
4: **else if** $k \geq 0$ and $G$ is acyclic **then return** $\emptyset$.
5: **else if** $G$ has a loop at some vertex $v$ **then return** FVS$(G - v, k - 1) \cup \{v\}$.
6: **else** ▷ $G$ has a cycle without loops and $k > 0$
7: Pick an edge $e$ uniformly at random.
8: Pick an endpoint of $e$ uniformly at random. Let $v$ be the chosen vertex.
9: **return** FVS$(G - v, k - 1) \cup \{v\}$.
10: **end if**
11: **end procedure**

**Lemma 2.** *On an input instance $(G, k)$ to* FEEDBACK VERTEX SET, *the procedure* FVS

*(i) runs in polynomial time,*

*(ii) outputs either* NO *or a feedback vertex set of $G$ of size at most $k$,*

*(iii) outputs a (feedback) vertex set of size at most $k$ with probability at least $\frac{1}{4^k}$ if $(G, k)$ is* YES.

**Proof:** The running time is straightforward. Before proceeding with the proof of (ii)-(iii), we point out that any input $(G', k')$ to the procedure FVS for the subsequent calls incurred by FVS$(G, k)$ is a legitimate instance of FEEDBACK VERTEX SET: that is, $k' \geq 0$. Indeed, we decrease the parameter $k$ by one every time we make a call to FVS, and when $k = 0$ an output is returned at Lines 3-4, which means no subsequent call is made.

We prove (ii) by induction on $k$. It suffices to prove that if FVS$(G, k)$ returns a vertex set $S$, then $S$ is a feedback vertex set of $G$ of size at most $k$. When $k = 0$, the fact that some vertex set $S$ is returned means that $(G, k)$ does not satisfy the condition of Line 3 and thus $G$ is acyclic. Now that $(G, k)$ meets the condition of Line 4, we know that $S = \emptyset$. It is clear that $S = \emptyset$ is a feedback vertex set of an acyclic graph $G$ of size at most $k = 0$. Consider $k > 0$ and notice that $S$ is returned at either Line 5 or 9. Especially, this means that the output of FVS$(G - v, k - 1)$ is $S \setminus v$. By induction hypothesis, $S \setminus v$ is a feedback vertex set of $G - v$ of size at most $k - 1$. Hence, $G - v - S \setminus v$ is acyclic and thus $S$ is a feedback vertex set of $G$. Clearly $|S| \leq k$. This proves (ii).

Now we prove (iii). Suppose that $(G, k)$ is a YES-instance. If $G$ is acyclic, then (iii) trivially holds with probability 1. In particular, $G$ is always acyclic when $k = 0$ due to the assumption that $(G, k)$ is a YES-instance. Therefore, we may assume that $G$ has a cycle and $k > 0$. We claim that the input $(G - v, k - 1)$ to a subsequent call at Line 5 or 9 is YES with probability at least $\frac{1}{4}$. If $G$ has a loop at $v$, then $v$ must be included in any solution, and $(G - v, k - 1)$ is again a YES-instance with probability 1. If $G$ does not have a loop, then Line 8 chooses a vertex $v$ contained in a solution $S$ of size at most $k$ with probability

$\frac{1}{4}$. Indeed, Lemma 1 implies that the edge $e$ chosen at Line 7 is in $E(S) \cup E(S, V \setminus S)$ with probability 0.5. In case $e \in E(S)$, the probability that a random endpoint $v$ of $e$ is in $S$ is 1. In case $e \in E(S, V \setminus S)$, the probability is 0.5. Therefore,

$$\begin{aligned}\mathsf{Pr}[v \in S] = \mathsf{Pr}[e \in E(S) \cup E(S, V \setminus S)] \times \mathsf{Pr}[v \in S | e \in E(S) \cup E(S, V \setminus S)] \\ \geq 0.5 \times 0.5\end{aligned}$$

Notice that when $v \in S$, then $S \setminus v$ is a feedback vertex set of size at most $k-1$ of $G - v$. That is, $(G - v, k - 1)$ is a YES-instance. Therefore, the created instance $(G - v, k - 1)$ at Line 9 is a YES-intance with probability at least $\frac{1}{4}$, as claimed.

To finalize the proof of (iii), we recall that by induction hypothesis, $\mathsf{FVS}(G - v, k - 1)$ returns a vertex set with probability at least $\frac{1}{4^{k-1}}$ when $(G - v, k - 1)$ is YES. Now[2],

$$\begin{aligned}&\mathsf{Pr}[\mathsf{FVS}(G, k) \text{ returns a vertex set at Line 9}] \\ &\quad = \mathsf{Pr}[(G - v, k - 1) \text{ is YES and } \mathsf{FVS}(G - v, k - 1) \text{ returns a vertex set at Line 9}] \\ &\quad \geq \mathsf{Pr}[(G - v, k - 1) \text{ is YES}] \\ &\quad \times \mathsf{Pr}[\mathsf{FVS}(G - v, k - 1) \text{ returns a vertex set at Line 9} | (G - v, k - 1) \text{ is YES}] \\ &\quad \geq \frac{1}{4} \times \frac{1}{4^{k-1}} = \frac{1}{4^k}.\end{aligned}$$

This completes the proof. □

By repeating $\mathsf{FVS}(G, k)$ $4^k$ times, we obtain an algorithm summarized in the lemma below.

**Lemma 3.** *There is an algorithm running in time $O(4^k \cdot poly(n))$ time which, given an input $(G, k)$ to* FEEDBACK VERTEX SET, *outputs*

*(i)* NO *if $(G, k)$ is a* NO*-instance, and*

*(ii) outputs a solution of size at most $k$ with probability $1 - e^{-1}$ if one exists.*

**Proof:** The algorithm $\mathcal{A}$ works as follows: on the input instance $(G, k)$, we run the procedure $\mathsf{FVS}$ $4^k$ times. If a run of FEEDBACK VERTEX SET returns a vertex set $S$, then return $S$ as an output of $\mathcal{A}$. If all $4^k$ executions of $\mathsf{FVS}$ on $(G, k)$ returns NO, then $\mathcal{A}$ returns NO. Clearly the algorithm $\mathcal{A}$ runs in the claimed running time because $\mathsf{FVS}$ runs in polynomial time and $\mathcal{A}$ invokes $\mathsf{FVS}$ as a subroutine $4^k$ times. That $\mathcal{A}$ satisfies (i) follows immediately from Lemma 2. To see (ii), observe that the probability that $\mathcal{A}$ returns NO when $(G, k)$ is YES equals

$$\mathsf{Pr}[\mathsf{FVS}(G, k) \text{ returns NO while } (G, k) \text{ is YES}]^{4^k} \leq (1 - \frac{1}{4^k})^{4^k} \approx e^{-1} (\approx 0.36),$$

where the equality holds because each run of $\mathsf{FVS}$ is independent, and the second inequality holds due to Lemma 2. The property (ii) follows. □

[2] In the inequality, all probabilities are conditional on that $(G, k)$ is YES. We assumed this at the beginning of the proof of (iii).

## 2 A randomized algorithm for $k$-Path based on color coding

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.

**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 $V_1, \dots, V_k$ 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 $C \subseteq \{1, \dots, k\}$ 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 $i = 1, \dots, k$, 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 $C \subseteq \{1, \dots, k\}$ 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 4.** *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 5.** *One can detect a simple $k$-path in $O((2e)^k \cdot |E|)$ expected running time, if one exists.*

**Lemma 6.** *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 4, the running time follows. □