1 Randomized FPT Algorithms

In the previous lecture, we talked about an FPT algorithm with kernalization for d-Hitting Set (Vertex Cover on d-hypergraph), which is based on Sunflower Lemma. We also introduced FPT reduction, and we showed that k-Clique $\leq_{\mathsf{FPT}}$ k-Hitting Set.

In today's lecture, we are going to discuss the randomized FPT algorithms for Longest Path and Feedback Vertex Set using methods of Color coding, Random sampling and Random separation.

First, we look at the following claim:

Claim 4.1. *Suppose a biased-coin lands head with probability p, and lands tail with probability $1-p$. Let X be the r.v. denoting the number of flips until the first head. Then we have:*

$$\begin{aligned}
&\mathbf{E}[X] = \sum_{i=0}^{\infty}(1-p)^i = \frac{1}{p} \\
\implies &\mathbf{Pr}[X > \frac{2}{p}] < \frac{\mathbf{E}[X]}{\frac{2}{p}} = \frac{1}{2} && \textit{(by Markov's)} \\
\implies &\mathbf{Pr}[X > \frac{2}{p}\log n] < \mathbf{Pr}[X > \frac{2}{p}]^{\log n} = 2^{-\log n} = \frac{1}{n}
\end{aligned}$$

A slightly stronger bound using the inequality $1 + x \leq e^x$ for all $x \in \mathbb{R}$ gives

$$\mathbf{Pr}[X > \frac{1}{p}\ln n] = (1-p)^{\frac{1}{p}\ln n} \leq (e^{-p})^{\frac{1}{p}\ln n} = e^{-\ln n} = \frac{1}{n}$$

We will now look at the problem Longest Path.

2 Longest Path

The Longest Path problem can be described as follows: Given a graph $G = (V, E)$ with n vertices and m undirected edges, and a parameter k. Does G have a simple path of length k?

Note that Longest Path $\in$ XP. Recall that XP is the class of parameterized problems with input size n and parameter k that can be solved in time $n^{f(k)}$ for some computable function f.

The question remains: Is Longest Path $\in$ FPT ? As it turns out, there exist randomized FPT algorithms for Longest Path, i.e., randomized algorithms that run in time $f(k)n^c$ that fails with a small probability ϵ.

2.1 Color-Coding Algorithm 1

Consider the following algorithm:

① Color the vertices of G independently and uniformly at random from $\{1, 2, \ldots, k\}$.

② Check if $\exists$ a path $P = (v_1, v_2, \ldots, v_k)$, s.t. v_i is assigned color i. If so, output P, otherwise call this attempt failed.

Repeat these two steps for $k^k \ln n$ times, if any attempt outputs a path P, output P. If all attempts failed, reject.

Lemma 4.2. *If $\exists P = (v_1, v_2, \ldots, v_k)$ in G, the probability that every v_i is assigned color i is $\frac{1}{k^k}$.*

By Claim 4.1, we get $\mathbf{Pr}[\text{fail in all } k^k \ln n \text{ attempts}] \leq \frac{1}{n}$ if $\exists P = (v_1, v_2, \ldots, v_k)$ in G. Thus, we get the following:

Theorem 4.3. *This algorithm has probability of success $\geq 1 - \frac{1}{n}$ if $\exists$ a path P of length k in G. If $\nexists$ such P, this algorithm always succeeds.*

Now we analyze the time complexity of this algorithm. Since checking if every v_i of P is assigned color i only takes $O(n)$ time, assuming random coloring n vertices takes $O(n)$ time, we get:

Theorem 4.4. *This algorithms runs in time $O(k^k \text{poly}(n))$. Thus it is a randomized* FPT *algorithm with $f(k) = k^k$.*

2.2 Color-Coding Algorithm 2

We now look at the Color-Coding algorithm by Alon, Yuster and Zwick [NRU95] in 1995, which is very similar to the algorithm introduced in section **2.1**, with a small modification:

① Color the vertices of G independently and uniformly at random from $\{1, 2, \ldots, k\}$.

② Check if $\exists$ a path $P = (v_1, v_2, \ldots, v_k)$, s.t. P is a *colorful* path. A path in G is *colorful* if each vertex on it is assigned a distinct color. If so, output P, otherwise call this attempt failed.

Repeat these two steps for $e^k \ln n$ times, if any attempt outputs a path P, output P. If all attempts failed, reject.

Lemma 4.5. *If $\exists P = (v_1, v_2, \ldots, v_k)$ in G, P is* colorful *with probability $\frac{k!}{k^k} \geq \frac{(\frac{k}{e})^k}{k^k} = \frac{1}{e^k}$ (using the convenient bound $k! \geq (\frac{k}{e})^k$).*

Similarly, by Claim 4.1, we get $\mathbf{Pr}[\text{fail in all } e^k \ln n \text{ attempts}] \leq \frac{1}{n}$ if $\exists P = (v_1, v_2, \ldots, v_k)$ in G. Thus, we get the following:

Theorem 4.6. *This algorithm has probability of success $\geq 1 - \frac{1}{n}$ if $\exists$ a path P of length k in G. If $\nexists$ such P, this algorithm always succeeds.*

When it comes to analyzing the time complexity of this algorithm, we would like to find out the time complexity of finding a *colorful* path P of length k.

Lemma 4.7. *$\exists$ an algorithm that can find a* colorful *path P of length k in time $O(2^k \text{poly}(n))$.*

Proof. We use Dynamic Programming.
Let $T(S, v)$ represent the state of whether or not there is a path ending at vertex v whose vertices are colored distinctly with colors in color set S. We could calculate the value of T over all $S \neq \emptyset$ and vertices v with the following equation:

$$T(S, v) = \begin{cases} \texttt{false} & \text{color}(v) \notin S \\ \texttt{true} & S = \{\text{color}(v)\} \\ \bigcap\limits_{u \in N(v)} T(S', u) & \text{color}(v) \in S, S \backslash \{\text{color}(v)\} = S' \neq \emptyset, N(v) \cap S' \neq \emptyset \end{cases}$$

Because there are $2^k n$ states and calculating each costs $O(n)$ time, calculating the table would cost $O(2^k \text{poly}(n))$ time. □

Thus, assuming random coloring n vertices takes $O(n)$ time, we get the following theorem:

Theorem 4.8. *This algorithms runs in time $O((2e)^k \text{poly}(n))$. Thus it is a randomized* FPT *algorithm with $f(k) = (2e)^k$.*

Remark 4.9. There exists a derandomization of algorithm 2, i.e., a deterministic version of algorithm 2 that runs in almost $O((2e)^k \text{poly}(n))$.
There also exists algorithms with time complexity $O(2^k n^c)$ that solves Longest Path. These use *algebraic* techniques, which we will see in a later lecture.

3 Subgraph Isomorphism

The Subgraph Isomorphism problem can be described as follows: Given a graph G with n vertices, and a connected graph H with k vertices. Does G have contain H as a induced subgraph? That is, does there exist a set $X \subseteq V(G)$ s.t. $|X| = k$ and $G[X] \cong H$? Here, $G[X]$ has vertex set X and contains all the edges in G between vertices in X.

Note that when $H = K_k$, the problem becomes k-Clique, which is W[1] hard.

3.1 A Special Case

Consider the problem when G has degree $\leq d$. We can solve it by brute force algorithm in time $O(nd^k f(k)\text{poly}(n))$.

An alternative way of solving the problem is as follows:

① Color the edges of G independently and uniformly at random with color $\{R, B\}$.

② Remove all edges with color R to acquire graph G'.

③ Among all maximal connected components in G' that have $\geq k$ vertices, check if $\exists$ a component on vertex set X s.t. $G[X] \cong H$. If so, output $G[X]$, otherwise call this attempt failed.

Repeat these three steps for $2^k d \ln n$ times, if any attempt gives a subgraph $G[X]$, output it and accept. If all attempts failed, reject.

Note that $|X| = k$ and the degree of each vertex is at most d. That means the number of edges with at least one end in $G[X]$ is at most dk. X is a maximal connected component in G' if the out-edges(i.e. the edges with one endpoint in X) are all colored R and the edges within X are all colored B. Therefore,

Lemma 4.10. *If $\exists$ a component on vertex set X s.t. $G[X] \cong H$, the probability that X is a maximal connected component in $G'$is $\geq (\frac{1}{2})^{dk}$.*

By Claim 4.1, we get $\mathbf{Pr}[\text{fail in all } 2^{dk} \ln n \text{ attempts}] \leq \frac{1}{n}$ if $\exists$ a component on vertex set X s.t. $G[X] \cong H$. Thus, we get the following:

Theorem 4.11. *This algorithm has probability of success $\geq 1 - \frac{1}{n}$ if $\exists$ a component on vertex set X s.t. $G[X] \cong H$. If $\nexists$ such X, this algorithm always succeeds.*

Remark 4.12. The time complexity for checking if $G[X] \cong H$ is $k!$, and there are at most $\frac{n}{k}$ such components. Assuming the time complexity for random coloring is $O(m) \in O(\text{poly}(n))$, the runtime for the above algorithm is $O(2^{dk} k! \text{poly}(n))$. Therefore, it is a randomized FPT Algorithm.

4 Feedback Vertex Set

The Feedback Vertex Set problem can be described as follows: Given a multigraph $G = (V, E)$ with n vertices, and a parameter k. Does there exist a vertex set X with k vertices s.t. $X \subseteq V(G)$ and $G \backslash X := G[V \backslash X]$ is acyclic, i.e., a forest.

4.1 A Recursive Algorithm

Consider the following rules for input (G, k):

(a) If $v \in G$ and degree$(v) \leq 1$, move to problem $(G \backslash \{v\}, k)$.

(b) If $\exists$ a self-loop on vertex v, add v to the solution and move to problem $(G \backslash \{v\}, k-1)$.

(c) If $\exists \geq 3$ edges between (u, v), drop all but 2 of these edges to get new graph G'. Move to problem (G', k).

(d) If $\exists v$ which has degree 2 and $N(v) = \{u, w\}$, drop v and add a new edge (u, w) to get new graph $G' = (V \backslash \{v\}, E \cup \{(u, w)\})$. Move to problem (G', k).

(e) If G is empty graph, output the solution.

(f) if $k < 0$, reject.

Note that the above rules either excludes the vertices that must not in the optimal solution, or picks the vertices that must be included in the solution. There's also a useful lemma, which we did not prove in class, but here is the proof (it's also Lemma 5.1 in the Cygan et al. textbook).

Lemma 4.13. *If none of the above rules can be applied, at least half of the edges have at least one endpoint in the optimal solution.*

Proof. Suppose X is the optimal Feedback Vertex set of G. Let $H = G \backslash X$.
Want to show: the number of edges incident on X $\geq$ the number of other edges, i.e., $|E(H)|$.

Let $J :=$ the set of edges between X and H. It suffices to show: $|J| \geq |E(H)|$.

Let $V_{\leq 1}, V_{=2}, V_{\geq 3}$ be the set of vertices with degree $\leq 1, = 2$ and ≥ 3 in H.
H is acyclic $\implies |E(H)| < |V(H)| \implies \mathbf{E}[\text{degree}(v)] = \frac{2|E(H)|}{|V(H)|} < 2 \implies |V_{\leq 1}| > |V_{\geq 3}|$.

Since degree$(v) \geq 3$ for all v, $|J| \geq 2|V_{\leq 1}| + |V_{=2}| \geq |V_{\leq 1}| + |V_{=2}| + |V_{\geq 3}| = |V(H)| > |E(H)|$. □

Thus, we can create a recursive algorithm based on the above rules:

① First, apply the above rules until none of which can be performed on the remaining graph G'.

② Second, pick a random edge $(u, u') \in G$, and define v to be a random endpoint of that edge (that is, either u or u', each with probability 1/2).

③ Recurse on $(G' \backslash \{v\}, k-1)$.

Fix an optimal feedback vertex S. From the lemma, there is at least 1/2 probability of selecting an edge incident to S, and conditioned on that, there is a 1/2 probability of selecting the correct endpoint. Therefore, each execution of (2) succeeds with probability at least 1/4, and the entire algorithm succeeds with probability at least $1/4^k$. We can therefore repeat the algorithm $4^k \ln n$ times, giving a running time of $O^\star(4^k)$.

References

[NRU95] A. Noga, Y Raphael, and Z. Uri. Color-Coding. *Journal of the Association of the Computing Machinery*, 42(4):844–856, 1995. 2.2