PRINCETON UNIV. F'25 COS 521: ADVANCED ALGORITHM DESIGN

Lecture 16: Parameterized Algorithms

Lecturer: *Huacheng Yu*

In this lecture, we will talk about *parameterized algorithms*. In parameterized algorithms, we study problems that are typically algorithmically difficulty on general inputs. But we will focus on inputs where some parameters are small, and the goal is to exploit the bound on these parameters to design efficient algorithms.

1 FPT algorithms

As a motivating example, consider the vertex cover problem. A vertex cover of a graph G is a set of vertices that includes at least one endpoint of every edge. Computing the minimum vertex cover of a graph is NP-hard, and deciding if the min-VC is at most k is NP-complete. Consider the problem of deciding if a given graph has a vertex cover of size 5. This problem is clearly solvable in polynomial time, as one could simply enumerate over all sets of 5 vertices, and check if they are a vertex cover. It turns out that one can solve it in linear time. The idea is that for each edge that is not covered yet, we must add one of its endpoints to the vertex cover. However, as we are deciding if there is a VC of size 5, such decisions can be made at most 5 times. More formally, the problem can be solved using the following algorithm.

1. VC(S) // S is a set to be included in the VC
2. if all edges in G are covered by S, return YES
3. if $|S| = 5$, return NO
4. let $e = (u, v)$ be any uncovered edge
5. return VC($S \cup \{u\}$) OR VC($S \cup \{v\}$)

The recursion has depth at most 5, and each node has two branches. Thus, its running time is "$O(2^5 m)$". The algorithm generalizes to the problem of deciding if there is a VC of size at most k, and has running time $O(2^k \cdot m)$.

Definition 1. *A parameterized problem has input (x, k), for $|x| = n$ and a parameter $k \in \mathbb{N}$. It is called fixed-parameter tractable (FPT) if there is an algorithm (called a fixed-parameter algorithm) that solves the problem in time*

$$O(f(k) \cdot n^c),$$

for some computable function $f : \mathbb{N} \to \mathbb{N}$ and constant c. The above running time is often denoted by $O^*(f(k))$, which hides poly n *factors.*

Theorem 1. *k-vertex cover is FPT.*

2 Improved algorithm for k-VC

Next, we show that the above algorithm for k-VC can be improved. The main observations are: if a graph has max degree 1, then it is a disjoint union of edges, and the min-VC is

simply equal to the number of edges; for any vertex v, a VC must either include v or include all its neighbors.

This is still two decisions. However, if there is a vertex v with high degree, then in one of the choices (adding the neighbors of v), we will increase the current set of vertices to be added to the VC by a lot. Hence, the average depth of recursion is lower.

More concrete, consider the following algorithm.

1. VC(S) // S is a set to be included in the VC
2. if $|S| > k$, return NO
3. if G has max degree 1, solve the instance in linear time
4. let v be any vertex with degree at least 2
5. return VC($S \cup \{v\}$) OR VC($S \cup N(v)$)

We used $N(v)$ to denote the neighborhood of v. There are two branches:

- one with $k-1$ remaining vertices to be added
- one with $k - |N(v)| \leq k-2$ remaining vertices.

Thus, let $T(k)$ be the running time, we can write down the following recurrence: $T(k) \leq T(k-1) + T(k-2) + O^*(1)$. This solves to $T(k) \leq O^*(1.6181^k)$.

3 Color coding

Color coding is a very useful technique for designing FPT algorithms. Consider the k-path problem: given a graph G, decide if there is a (simple) path consisting of k vertices.

The idea is that we will give a random color to each vertex, and hope that there is some structure on the k vertices in the path. One solution is that we give a random color from $[k]$ to each vertex independently, then with probability $1/k^k$, the k vertices have colors $1, 2, \ldots, k$ respectively. Thus, the task reduces to finding such a path. This has a simple linear time algorithm: Partition the set of vertices into k sets according to the coloring, and only keep the edges from color i to $i+1$ for all i, and decide if there is a path from color 1 to color k by BFS. By repeating $O(k^k)$ times, we succeed with high probability. The running time is $O^*(k^k)$.

Next, we show that we can relax the requirement for a successful random coloring, while the remaining algorithmic problem given the coloring can still be solved.

Theorem 2. *k-path can be solved in $O^*(2^{O(k)})$ time by a randomized algorithm.*

Proof. We sample a random coloring $h : [n] \to [k]$. Suppose G has a k-path, then the probability that all vertices on the path have distinct colors is

$$k!/k^k \geq e^{-k},$$

where we used the fact that $k! \geq (k/e)^k$.

Assuming this happens, we will find a k-path with distinct colors using dynamic programming. For a set of colors $S \subseteq [k]$ and $u \in V$, let $g(S, u)$ denote if there is a path of length $|S|$ that ends at u and consisting of colors S. Then for the base case, we have

$$g(\{h(u)\}, u) = \text{True}.$$

In general, we have

$$g(S,u) = \begin{cases} \bigvee_{v:(v,u)\in E} g(S \setminus \{h(u)\}, v) & \text{if } h(u) \in S, \\ \text{False} & \text{if } h(u) \notin S. \end{cases}$$

By computing $g(S,u)$ according to $|S|$, we obtain all values in $O*(2^k)$ time. The existence of any k-path with distinct colors is simply

$$\bigvee_u g([k],u).$$

By repeating the coloring h $2^{O(k)}$ times, we find a good coloring with high probability. The total running time is $O^*(2^{O(k)})$. □

3.1 Derandomizing color coding

For the k-path problem, the idea for derandomizing the coloring function is to design a fixed set of $h : [n] \to [k]$ of $O^*(2^{O(k)})$ many, such that for any $P \subseteq [n]$ of size k, there exists some h in the set that maps P to distinct colors.

It turns out that this property is exactly achieved by perfect hashing. From the Hashing lecture, we showed that the FKS perfect hashing achieves the following: Given a set $P \subseteq [n]$ of size k, one can construct a two-level hash function h that maps $[n]$ to $O(k)$ buckets such that elements in P have no collisions. Moreover, h can be encoded in $O(k \log n)$ bits. It turns out that by a similar construction, one can improve the number of buckets to exactly k, and encode h in $O(k) + \log\log n$ bits.

The encoding length of h implies that there can be at most $2^{O(k)+\log\log n} = 2^{O(k)} \log n$ *different* functions h constructed in this way. Let this set of all possible functions h be the collection. Then, in particular, for any P, there exists one h in the collection that will map P to distinct buckets / colors. Then instead of sampling a random coloring, it suffices to try all h in this collection. If there is a k-path P, then some h will work for P.

Theorem 3. *k-path can be solved in $O^*(2^{O(k)})$ time deterministically.*

3.2 d-clustering

Finally, let us talk about another application of the color coding technique. A d-cluster is a graph with d connected components, and every component is a complete graph (a clique). Consider the following problem: Given a graph G, decide if we can add / remove a total of at most k edges to make G a d-cluster.

Suppose this is possible, let A be the set of modified edges (add or remove), then $|A| \leq k$. The algorithm is to sample a random coloring $h : [n] \to [2k]$, then with constant probability all edges in A have distinctly colored endpoints. Thus, assuming that this happens, if we divide the vertices into $2k$ sets according to their colors, then we cannot modify edges within in any set. In other words, if G can be modified to d-cluster and the coloring satisfies the above property, then

- each set consisting of vertices with the same color must be a disjoint union of at most d cliques.

There are a total of at most $2kd$ cliques that cannot be modified. Then it suffices to enumerate all possible ways to combine them into d cliques, and check if there is a way that modifies at most k edges. This can be done in $O^*(d^{2kd})$ time.

Without covering the details, it turns out that we can use only $O(\sqrt{k})$ colors, and show that with probability $2^{-O(\sqrt{k})}$, all edges in A have different colors in the two endpoints. This reduces the number of cliques in all sets to only $O(d\sqrt{k})$. Furthermore, deciding if one can combining them into d large cliques by modifying at most k edges can be solved using dynamic programming, in time $O^*(2^{O(d\sqrt{k})})$.