Lecture Outline 7
Topics in Graph Algorithms (CSCI5320-18S)

CAI Leizhen
Department of Computer Science and Engineering
The Chinese University of Hong Kong
lcai@cse.cuhk.edu.hk

March 8, 2018

Keywords: Parameterized complexity, FPT algorithms, k-VERTEX COVER, and bounded search tree.

1 Parameterized complexity

Parameterized complexity is a framework introduced by Downey and Fellows to deal with the complexity of an intractable problem (e.g., NP-complete problem k-VERTEX COVER that askes whether a graph contains a vertex cover of size at most k, where k is regarded as a parameter) with respect to both its input size $|I|$ and a chosen *parameter* k. A *parameterized problem* consists of a pair (I, k) where I is the input and k a parameter.

The key issue in parameterized complexity is to confine the exponential runtime of an algorithm for a parameterized problem to its parameter k, and therefore solve the problem efficiently when k is "small", which can be very useful in practice. An *FPT algorithm* runs in time $f(k)|I|^{O(1)}$ for some computable function $f(k)$, where FPT stands for fixed-parameter tractable. We also use FPT to denote the class of parameterized problems that admit FPT algorithms.

Fact: An algorithm runs in time $f(k)|I|^{O(1)}$ for some $f(k)$ iff it runs in time $g(k) + |I|^{O(1)}$ for some $g(k)$.

For instance, the k-VERTEX COVER problem can be solved in time $O(1.2738^k + kn)$ by an FPT algorithm of Chen, Kanj and Xia (2005), which makes the problem quickly solvable in practice for $n \leq 10^{15}$ and $k \leq 150$. We note that a straightforward exhaustive search algorithm takes $O(n^k k^2)$ time, which can hardly handle an instance with $n = 100$ and $k = 10$.

Downey and Fellows have also introduced a W-hierarchy

$$\mathrm{W}[1] \subseteq \mathrm{W}[2] \subseteq \mathrm{W}[3] \subseteq \cdots$$

to capture fixed-parameter intractability, where class W[1] contains class FPT and can be regarded as a parameterized version of the classical complexity class NP. A parameterized problem that is W[t]-complete (or W[t]-hard) for any W[t] in the hierarchy is unlikely to be fixed-parameter tractable and is thus *fixed-parameter intractable*. The relationship between FPT and W[1] is akin to that between P and NP.

NP-hard problems behave quite differently in the framework of parameterized complexity. For instance, k-VERTEX COVER is FPT, but k-CLIQUE is W[1]-complete and DOMINATING k-SET is W[2]-complete. *FPT algorithms are important and effective ways for solving NP-hard problems in practice.*

2 Forming parameterized problems

Parameter k tries to capture an aspect of a problem that most interests you, and FPT algorithms attempt to solve intractable problems such as NP-hard problems effectively in practice when parameter k is "small".

There are many different ways to introduce the parameter k to form parameterized problems. For instance, we can form the following parameterized problems related to VERTEX COVER.

(a) *Weighted case*: Find a vertex cover of weight at most k in a weight graph $G = (V, E; w)$ with $w : V \to Z$.

(b) *Parametric dual*: Find a vertex cover of size $n-k$. Equivalent to INDEPENDENT k-SET.

(c) *Fixed cardinality optimization*: Find k vertices to cover the maximum number of edges.

(d) *Combinatorial dual*: Find a minimum set of vertices to cover at least k edges.

(e) *Parameterized graphs*: VERTEX COVER on bipartite+kv graphs.

3 FPT algorithms for k-VERTEX COVER

FPT algorithms for k-VERTEX COVER: Diverse methods exist for designing FPT algorithms, and the following 6 different FPT algorithms for k-VERTEX COVER illustrate ideas in developing FPT algorithms. Note that if an n-vertex graph G admits a k-vertex cover, then it has at most kn edges.

1. **Algorithm 1:** Fellows and Langston (1986) $O(f(k)n^3)$ with astronomical $f(k)$.

   The algorithm is based on a celebrated theorem of Robertson and Seymour: Any family of graphs closed for minors is recognizable in $O(n^3)$ time. The family of graphs admitting vertex covers of size at most k is closed for minors. A graph H is a *minor* of graph G if a copy of H can be obtained from a subgraph of G by contracting edges in the subgraph.

2. **Algorithm 2:** Johnson (1987) $O(f(k)n^2)$ where $f(k) \approx 2^{2^{500k}}$.

   The algorithm uses the fact that if a graph admits a k-vertex cover then it has no $(2k+1)$-cycle as a minor. Then it combines the following three results:

   (a) For any planar graph H, it takes $O(n^2)$ time to tell whether H is a minor of G (Robertson and Seymour).

   (b) If G does not contain a planar graph H as a minor, then G is a partial t-tree for t a function of H.

   (c) VERTEX COVER on partial t-trees is solvable in $O(f(t)n)$ time by complicated dynamic programming (Courcelle, Seese).

3. **Algorithm 3:** Papadimitriou and Yannakakis (1993) $O(3^k kn)$

   The algorithm uses a maximal matching as a starting point.

   Step 1 Find a maximal matching M of G. No solution if $|M| > k$, and take all vertices $V(M)$ in M to form a solution if $2|M| \le k$.

   Step 2 For each of $3^{|M|}$ possible subsets V' of $V(M)$ do if $V' \cup \text{ext}(V')$ is a k-vertex cover then return "yes".

   $\text{ext}(V') = \{v : v \in V - V(M) \text{ and } \exists uv \in E \text{ with } u \notin V'\}$

4. **Algorithm 4:** Fellows (1988) (Also Mehlhorn) $O(2^k n)$ — branching out on an edge.

   The algorithm uses the *bounded search tree* method based on the simple fact that for every edge uv, any k-vertex cover must contain either u or v.

   We construct a binary tree B of height at most k as follows. Label the root of B by $(G, \emptyset)$. Choose an edge uv in the current graph G, branch out and label the two children of the root by $(G - u, u)$ and $(G - v, v)$ respectively. For each labeled node of B, label its two children in the same manner.

   Note that G has a k-vertex cover V' iff B has a leaf l with label $(\emptyset, x)$ for some vertex x, and V' can be obtained from the labels of the nodes on the path from l to the root.

5. **Algorithm 5:** $O(1.5^k n)$ — branching out on a vertex.

   For a vertex v with $d(v) \ge 3$, branch out for v and $N(v)$.

   Let S_k be the size of the search tree. Then $S_k \le S_{k-1} + S_{k-3} + 1$, implying $S_k \le 1.5^k$ and we obtain an $O(1.5^k n)$ algorithm.

6. **Algorithm 6:** Buss (1989) $O(kn + 2^k k^2)$

   The algorithm uses the *kernelization* method based on the simple fact that a vertex v with $d(v) > k$ must be in every k-vertex cover.

   Step 1 Find all vertices V' of degree $> k$, set $G' = G - V'$ and $k' = k - |V'|$. No solution if $|V'| > k$.

   Step 2 If G' has $> kk'$ edges then return "No solution" and stop.

   Step 3 Delete isolated vertices from G' to obtain G^*, and find a k' vertex cover V^* in G^* to form a k-vertex cover $V^* \cup V'$ of G.

   Note that after Step 1, k' vertices can cover at most kk' edges. Graph G^* has $\le k^2$ edges and $\le k + k^2$ vertices, and (G^*, k') is called a *kernel*.

4 Bounded search tree

A fundamental method for obtaining FPT algorithms is to bound the size of a search tree to a function of k only. Typically, to find a k-solution $(x_1, \ldots, x_k)$, we bound the number of choices for each x_i to a small number, which is often a constant c, and hence obtain an FPT algorithm with running time $c^k n^{O(1)}$.

1. TRIANGLE-FREE DELETION: $O(3^k n^\omega)$.

   Determine whether we can delete at most k vertices from a graph to obtain a triangle-free graph.

   For a triangle, there are three different ways to delete a vertex to destroy the triangle, and we branch out for each of the three possibilities.

2. SPLIT GRAPH DELETION: $O(5^k(m+n))$.

   Can we delete at most k vertices from a graph G to make it a split graph?

   A graph is a *split graph* if its vertices can be partitioned into a clique and independent set.

   Theorem. *A graph is a split graph iff it contains no induced subgraph isomorphic to C_4, $\overline{C}_4$, or C_5.*

   If G is not a split graph, we can find a forbidden induced subgraph F in G in $O(n^5)$ time, and branch out in 4 or 5 different ways depending on the number of vertices in F.

3. VERTEX RECOLORATION: $O(4^k(m+n))$.

   Can we transform a vertex 3-coloring of a graph G into a proper 3-coloring by recoloring $\leq k$ vertices?

   For a monochromatic edge uv, branch out by recoloring u or v whenever they are unmarked vertices and mark u or v accordingly.

   Question: Can you find a faster FPT algorithm?

4. VERTEX COVER on bipartite+kv graphs (Cai 2003): $O(2^k\sqrt{n}m)$.

   Given a graph G and at most k vertices V^* such that $G-V^*$ is bipartite, find a minimum vertex cover in G.

   For a vertex $v \in V^*$, a minimum vertex cover of G contains either v or $N(v)$.

   Furthermore, both $G-v$ and $G-N[v]$ are bipartite+$(k-1)$v graphs, where $N[v]$ denotes the closed neighborhood $\{v\} \cup N(v)$ of v. We branch out by considering VERTEX COVER for $G-v$ and $G-N[v]$.

   Note that one can use matching technique to solve VERTEX COVER for bipartite graphs in time $O(\sqrt{n}m)$.

5. DENSITY REDUCTION: $O(2^k n \log n)$.

   Remove k points from a set P of n points to maximize the minimum pairwise distance for the remaining points.

   For a closest pair (x, y) of points, we need to remove either points x or y, and we branch out for these two cases. Note that it takes $O(n \log n)$ to find a closest pair.

6. MULTICUT IN TREES: $O(2^k(l+n))$.

   Given an n-vertex tree T, l pairs $\{(u_i, v_i)\}$ of vertices and integer k, determine whether we can remove $\leq k$ edges E' from T to disconnect u_i and v_i for every (u_i, v_i).

Arbitrarily pick up a vertex r of T as the root. Let P_i denote the (u_i, v_i)-path in T, and w_i the least common ancestor of u_i and v_i. Let (u_{i^*}, v_{i^*}) be a pair that maximizes the distance from r to w_{i^*}, and e_{i^*} and e'_{i^*} the two edges of P_i incident with w'_{i^*}.

Lemma *There is a k-multicut that contains either e_{i^*} or e'_{i^*}.*

Find a required pair (u_{i^*}, v_{i^*}) and branch out by including either e_{i^*} or e'_{i^*} in a k-multicut.

7. Feedback Vertex Set: $O((2k)^k n^2)$.

   Can we remove at most k vertices from a graph G to obtain a forest?

   Theorem. *Let G be a graph with minimum degree at least 3. If G admits an FVS with k vertices, then G contains a cycle of length at most $2k$.*

   We preprocess G to obtain a graph G' with minimum degree at least 3 by repeatedly deleting degree-1 vertices and contracting an edge incident with a degree-2 vertex. For G', find a minimum-length cycle C, which contains at most $2k$ vertices if G' admits an FVS with k vertices, and branch out for each vertex in C.

8. Connected Vertex Cover

   Determine whether G contains a k-vertex cover V' with $G[V']$ connected.

   The $O(2^k kn)$ algorithm for k-Vertex Cover enumerates all minimal vertex cover of size $\leq k$. For each such vertex cover V', we determine whether V' can be extended to a connected subgraph on $\leq k$ vertices, which gives a Steiner tree problem.