Lecture Outline (CSCI3160-17F, 13th week)

CAI Leizhen
CSE-CUHK-HK-CHN

December 1, 2017

Keywords: FPT algorithm, parameterized complexity, bounded search tree, and kernelization.

1 FPT algorithms

Parameterized complexity is a framework introduced by Downey and Fellows to deal with the complexity of a problem with respect to both its input size $|I|$ and a chosen *parameter* k. For instance, we can take the size k of a vertex cover in the classical Vertex Cover problem as a parameter to form the parameterized problem k-Vertex Cover.

> k-Vertex Cover
> Instance: Graph $G = (V, E)$, positive integer k as *parameter*.
> Question: Does G have a vertex cover of size at most k?

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. 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 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 $k = 10$.

1.1 Parameterized problems

A *parameterized problem* (a.k.a. *fixed-parameter problem*) consists of a problem Π and a parameter k, where the choice of the parameter depends on the aspect of the problem that interest you. There are many different ways to introduce a parameter k to form parameterized problems, and the following examples show various parameterized problems related to the classical Vertex Cover problem.

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

2. *Parametric dual*: Find a vertex cover of size $n - k$ in an n-vertex graph, which is equivalent to the INDEPENDENT k-SET problem.

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

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

5. *Parameterized graphs*: VERTEX COVER on graphs that can be made into bipartite graphs by deleting at most k vertices.

1.2 Fixed-parameter tractability

An algorithm for a parameterized problem (I, k) is an *FPT algorithm* if it runs in time $O(f(k)|I|^c)$ for some function $f(k)$ and constant c, where $|I|$ is input length. We also use $O^*(f(k))$ to denotes $O(f(k)|I|^c)$ when we emphasize $f(k)$ part. A parameterized problem (I, k) is *fixed-parameter tractable* if it admits an FPT algorithm, and the class of fixed-parameter tractable problems is denoted by FPT.

Downey and Fellows have also introduced a W-hierarchy

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

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.*

1.3 Tools for designing FPT algorithms

In addition to tools for designing polynomial algorithms, we have several useful tools for designing FPT algorithms. Two basic tools are *bounded search tree* and *kernelization* (a.k.a. *reduction to kernel*). Other important and novel tools include *iterative compression* of Reed, Smith and Vetta (2004), *colour coding* of Alon, Yuster and Zwick (1995), and *random separation* of Cai, Chan and Chan (2006).

2 Bounded search tree

Suppose that we wish to find a k-solution, i.e., a solution with k elements $\{x_1, \cdots, x_k\}$. The main idea of the bounded search tree method is to restrict the choices of each x_i to c possibilities and therefore restrict the size of the search space to $O(c^k)$. Typically, c is a constant, but can be $g(k)$ for some function g.

Algorithm 1. k-Vertex cover: $O(2^k kn)$.

A classical example of the bounded search tree method is the following algorithm for solving k-Vertex Cover based on the fact that for every edge uv, any k-vertex cover must contain either u or v.

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

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

Note that G has at most kn edges when it admits a k-vertex cover. The above algorithm runs in $O(2^k kn)$ time as B has at most $O(2^k)$ nodes and each node of B takes $O(kn)$ time.

Algorithm 2. Minimum vertex cover in k-almost-forests: $O(2^k m)$.

A graph G is a *k-almost-forest* if G contains at most k vertices X such that $G-X$ is acyclic, i.e., a forest. Note that every graph is a k-almost-forest for some k, and that a forest is a 0-almost-forest. Let $\beta(G)$ denote the size of a minimum vertex cover of G.

Task: Find a minimum vertex cover of a k-almost-forest G, given k vertices X such that $G - X$ is a forest.

For an arbitrary vertex $x \in X$, we observe the following facts:

1. For any vertex $x \in X$, either x or $N(x)$ is contained in an optimal solution.
2. $\beta(G) = \min\{\beta(G-x)+1, \beta(G-N[x]) + |N(x)|\}$.
3. Both $G-x$ and $G-N[x]$ are $(k-1)$-almost-forests.

Idea: For each vertex $x \in X$, consider two possibilities: either x or $N(x)$ is contained in an optimal solution, which reduces the problem from k to $\leq k-1$.

Use a search tree T and label the root by (G, X). For any node (G, X) in T, branch out by choosing a vertex x from X, making two nodes $(G-x, X-x)$ and $(G-N[x], X-N[x])$ as children of (G, X).

For a node (G, X) in T, G is a forest whenever $|X| = 0$, and we can compute a minimum vertex cover of G directly in linear time. The height of T is at most k, and hence the total time is $O(2^k m)$.

3 Kernelization

Kernelization (a.k.a. *reduction to kernel*) is a preprocessing procedure to reduce a problem instance to a much smaller one. The main idea is to reduce an instance (I, k) of a parameterized problem in polynomial time to an equivalent instance (I', k') with $|I'| \leq g(k)$ and $k' \leq k$ for some functions $g(k)$, where (I', k') is called a *kernel*. We then solve (I', k') directly by exhaustive search or other methods, and use a solution of (I', k') to obtain a solution of (I, k). *Most kernelization algorithms use reduction rules to reduce instances to smaller ones.*

Theorem 3.1 *A parameterized problem is FPT iff it is decidable and admits a kernel.*

Example: INDEPENDENT k-SET: $4k$ kernel for planar graphs

By the 4-color theorem, an n-vertex planar graph G always contains an independent set of size $\geq k$ when $n \geq 4k$. Otherwise (G, k) is a kernel with less than $4k$ vertices.

Problem: Design an FPT algorithm to find an independent k-set with maximum weight for weighted planar graphs $G = (V, E; w)$ with $w : V \to Z^+$.

Algorithm 3 (Buss 1989): $O(k^2)$ kernel for solving k-VERTEX COVER in $O(kn + 2^k k^2)$ time.

The following algorithm of Buss solves k-VERTEX COVER by kernelization based on the simple fact that a vertex v with degree $d(v) > k$ must be in every k-vertex cover.

Step 1 Find all vertices V' of degree $> k$. If V' has more than k vertices then return "No solution" and stop. Otherwise put V' into S, construct $G - V'$ and delete all isolated vertices to form graph G', and set $k' = k - |V'|$.

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

Step 3 If G' contains a k'-vertex cover S' then return $S' \cup S$ else return "No solution".

Note that after Step 1, k' vertices can cover at most kk' edges. Therefore the kernel (G', k') has at most k^2 edges and $2k^2$ vertices.