Seminar Notes for CMSC5719

An Effective Way for Solving Intractable Problems – FPT algorithms

CAI Leizhen

Department of Computer Science and Engineering
The Chinese University of Hong Kong
Shatin, Hong Kong, China
E-mail:lcai@cse.cuhk.edu.hk

Nov. 17, 2014

Abstract

Many problems from applications are NP-hard and therefore very unlikely to admit polynomial-time algorithms. On the other hand, it is often the case that an NP-hard problem we need to solve in practice has the property that certain part of the problem, called parameter, is "small" relative to the input size. For such intractable problems, fixed-parameter tractable algorithms (FPT algorithms in short) provide a very effective tool to solve them efficiently by confining the exponential running time to the parameter only. For instance, the current fastest FPT algorithm for the classical NP-hard Vertex Cover problem can easily solve the problem for graphs with billions of vertices when their vertex covers have size, which is a parameter, at most 150.

In this talk, we will discuss some tools for designing FPT algorithms — from the basic techniques of bounded search tree and kernelization to novel ideas of iterative compression of Reed, Smith and Vetta, colour coding of Alon, Yuster and Zwick, and random separation of Cai, Chan and Chan.

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 $f(k)|I|^c$ for some function $f(k)$ and constant c, where $|I|$ is input length. 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: $O(2^k n)$ time for k-Vertex cover

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 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.

The above algorithm runs in $O(2^k n)$ time as B has at most $O(2^k)$ nodes and each node of B takes $O(n)$ time.

Algorithm 2: $O(2^k n)$ time for Multicut on trees.

Given tree T, l pairs $\{(u_i, v_i)\}$ of vertices, and integer k, we want to determine whether there are $\leq k$ edges E' in T whose removal disconnects u_i and v_i for all pairs (u_i, v_i). The problem is NP-complete if k is not a parameter.

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 be the vertex on P_i closest to r, i.e., w_i is the least common ancestor of u_i and v_i. Let (u_{i^*}, v_{i^*}) be a pair that maximizes $d(r, w_{i^*})$, and e_{i^*} and e'_{i^*} the two edges of P_i incident with w'_{i^*}.

Lemma 2.1 *There is a k-multicut that contains either e_{i^*} or e'_{i^*}.*

We can use Lemma 2.1 to construct a binary tree B of height k as follows. Label the root of B by $(T, \emptyset)$. Choose a pair (u_i, v_i) that maximizes $d(r, w_i)$, branch out and label the two children of the root by $(T - e_i, e_i)$ and $(T - e'_i, e'_i)$. For each labeled node of B, label its two children in a similar manner. The algorithm runs in $O(2^k n)$ time.

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.*

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.

4 Iterative compression

B. Reed, Bruce, K. Smith, and A. Vetta. Finding odd cycle transversals. Operations Research Letters 32(4): 299-301, 2004.

The main idea is to start with a large solution and repeatedly reduce it to a smaller solution. A key component of the method is a compression subroutine to construct a k-solution from a $(k+1)$-solution in FPT time.

Algorithm 4 (Cai 2005): k-Vertex Cover

For a vertex cover V' and a subset $X \subseteq V'$, let $N'(X)$ denote the set of vertices in $V - V'$ that are adjacent to some vertices in X. The following lemma enables us to determine in FPT-time whether a $(k+1)$-vertex cover of G is a minimum vertex cover, and obtain a smaller one when it is not.

Lemma 4.1 (Cai 2005) *A k-vertex cover V' is a minimum vertex cover iff for every independent set X in V', $|N'(X)| \geq |X|$.*

Proof. If some independent set $X \subseteq V'$ satisfying $|N'(X)| < |X|$, then $(V' - X) \cup N'(X)$ is a smaller vertex cover of G. Conversely, suppose that G has a smaller vertex cover V^*. Let $X = V' - V^*$. Then X is an independent set in V', and $N'(X) \subseteq V^* - V'$. Since $|V^*| < |V'|$, we have $|V^* - V'| < |X|$ and thus $|N'(X)| < |X|$. ∎

We use the above lemma to obtain the following compression routine: Consider each independent set X in a $(k+1)$-vertex cover, and check if $|N'(X)| < |X|$. If so we get a smaller vertex cover, otherwise G has no k-vertex cover. The time for this compression routine is $O(2^k kn)$.

There are at least three different ways to obtain a $(k+1)$-vertex cover in the first place.

1. *Recursively*: Arbitrarily choose a vertex v in G and recursively solve the problem for $G - v$. If $G - v$ has no solution then neither has G. Otherwise, we obtain a k-vertex cover S, which yields a $(k+1)$-vertex cover $S \cup \{v\}$ for G. We need to compress $O(n)$ times, which results in an $O(2^k kn^2)$-time algorithm.

2. *Iteratively*: Order vertices as $v_1, \ldots, v_n$ and let $G_i = G[\{v_1, \ldots, v_i\}]$. Then a k-vertex cover V' of G_i yields a $(k+1)$-vertex cover $V' \cup \{v_{i+1}\}$ of G_{i+1}, which is then compressed into a k-vertex cover. Again, we get an $O(2^k kn^2)$-time algorithm.

3. *Approximation*: Use a 2-approximation algorithm for vertex covers to find a k'-vertex cover of G first. If $k' > 2k$ then the answer is no, otherwise we compress solutions at most k times, which gives us an $O(4^k k^2 n)$-time algorithm.

5 Color coding

N. Alon, R. Yuster, and U. Zwick, Color-coding, J. ACM 42(4):844-856, 1995.

The basic idea of this novel method is to use k colors to color elements randomly and then try to find a colorful k-solution, i.e., a k-solution whose elements are in distinct colors. If we can find a colorful k-solution in FPT time, then we can find a k-solution in FPT time with probability $k!/k^k > e^{-k}$. The algorithm can be derandomized by a family of perfect hash functions.

Color coding is useful for the following types of parameterized problems: (a) k-solutions with good structures (linear, cyclic, tree), and (b) partition into small structures.

Algorithm 5 (Alon, Yuster and Zwick 1995): $2^{O(k)} m \log n$ time for k-PATH: Does graph G contain a path with k vertices?

Randomly color vertices with colors $\{1, \ldots, k\}$, and call a path *colorful* if colors of its vertices are distinct. Given a coloring $c : V \to \{1, \ldots, k\}$, we can use dynamic programming to determine whether there is a colorful k-path.

Construct G' from G by adding a new vertex v_0 of color 0 and connect it with every vertex in G. Now the problem is to determine whether there is a colorful $(k+1)$-path in G' that starts at v_0.

For each vertex v, define $C^i(v) = \{c(P) : P \text{ is a colorful } (v_0, v)\text{-path of length } i\}$, where $c(P)$ is the set of colors of vertices in P. Then G has a colorful k-path iff G' has a vertex v with $C^k(v) \neq \emptyset$. Note $|C^i(v)| \leq \binom{k}{i}$.

Initially, $C^1(v) = \{c(v)\}$ for each vertex v. For $2 \leq i < k$,

$$C^{i+1}(v) = \{C \cup \{c(v)\} : u \in N(v), C \in C^i(u) \text{ and } c(v) \notin C\}.$$

Note that we keep track of color sets but not paths. Time: $O(\sum_{i=1}^{k} i\binom{k}{i}m) = O(k2^k m)$.

We can derandomize the algorithm using an (n, k)-family of perfect hash functions to obtain an FPT algorithm with running time $2^{O(k)}m \log n$. An *(n, k)-family of perfect hash functions* is a family $\mathcal{F}$ of functions mapping a domain of size n into a range of size k such that for every k-subset S from the domain, there is a function in $\mathcal{F}$ that is 1-to-1 on S. Based on a construction of Schmidt and Siegel (1990), Naor (1995) gave a construction of an (n, k)-family of perfect hash functions of size $2^{O(k)} \log n$ in time $2^{O(k)}n \log n$.

6 Random separation

L. Cai, S.M. Chan and S.O. Chan, Random separation: a new method for solving fixed-cardinality optimization problems, LNCS 4169 (pp.239-250), 2006.

The basic idea of this innovative method is to use a random partition of the vertex set V of a graph $G = (V, E)$ to separate a solution from the rest of G into connected components and then select appropriate components to form a solution. Algorithms obtained from this method can be derandomized by a family of universal sets.

Random separation is very effective for a large variety of parameterized problems on graphs with bounded degree or bounded degeneracy, and also useful for some parameterized problems on general graphs.

Algorithm 6 (Cai, Chan and Chan 2006): Dense k-Vertex Subgraph (a.k.a. Maximum k-Vertex Subgraph) for degree-bounded graphs: Find k vertices V' in a graph $G = (V, E)$ of maximum degree d, where d is a constant, to maximize the number of edges in $G[V']$.

First we randomly colour each vertex of G by either green or red each with probability 1/2 to form a random partition (V_g, V_r) of V. Green vertices V_g induce the *green subgraph* $G_g = G[V_g]$, and the connected components of G_g are *green components*.

Let G' be a maximum k-vertex induced subgraph of G. A random partition of V is a "good partition" for G' if all vertices in G' are green and all vertices in its neighbourhood $N_G(G')$ are red. Note that $N_G(G')$ has at most dk vertices as $d_G(v) \leq d$ for each vertex v. Therefore the probability that a random partition is a good partition for G' is at least $2^{-(d+1)k}$ and thus, with at least this probability, G' is the union of some green components.

To find a maximum k-vertex induced subgraph for a good partition of G', we need only find a collection $\mathcal{H}'$ of green components such that the total number of vertices in $\mathcal{H}'$ is k and the total number of edges in $\mathcal{H}'$ is maximized. For this purpose, we

first compute in $O(dn)$ time the number n_i of vertices and the number m_i of edges inside each green component H_i. Then we find a collection $\mathcal{H}'$ of green components that maximizes

$$\sum_{H_i \in \mathcal{H}'} m_i$$

subject to $\sum_{H_i \in \mathcal{H}'} n_i = k$.

Since for any two green components H_i and H_j, the number of vertices (resp. the number of edges) in $H_i \cup H_j$ equals $n_i + n_j$ (resp. $m_i + m_j$), we can solve the problem in $O(kn)$ time by the standard dynamic programming algorithm for the 0-1 KNAPSACK problem. Therefore, with probability at least $2^{-(d+1)k}$, we can find a maximum k-vertex induced subgraph of G in $O((d+k)n)$ time.

To derandomize the algorithm, we use a family of partitions with the property that for every partition Π of any $(d+1)k$ vertices into k vertices and dk vertices, there is a partition in the family that is consistent with Π.

A collection of binary vectors of length n is (n,t)-*universal* if for every subset of size t of the indices, all 2^t configurations appear. Naor, Schulman and Srinivasan have a construction for (n,t)-universal sets of size $2^t t^{O(\log t)} \log n$ that can be listed in time $2^t t^{O(\log t)} n \log n$. We can interpret a binary vector of length n as a red-green coloring of G: vertex v_i is colored green if the i-th position of the vector is 1 and red if 0. Then a family of $(n,(d+1)k)$-universal sets can be used as the required family of partitions. Therefore we obtain an FPT-algorithm that runs in $O(f(k,d)n \log n)$ time where

$$f(k,d) = 2^{(d+1)k}(dk+k)^{O(\log(dk+k))}(d+k).$$