Last time we introduced FPT functions, algorithms and reductions. We also considered the k-Vertex Cover (k-VC) problem in which k is an integer, and one is given an undirected graph $G = (V, E)$ with $|V| = n, |E| = m$ and one is asked to return a set of vertices of size k such that every edge in E has at least one of its end points in the set; such a set is called a k-vertex cover. Last time we gave a simple $O(2^k n)$ time algorithm for k-VC.

Our plan for today is as follows:

- give improved FPT algorithms for k-VC,
- introduce kernelization,
- give kernels for k-VC

1 FPT Algorithms for k-VC

The $O(2^k n)$ algorithm $A(G, S)$ from last time proceeded by recursively building up a potential VC S, where S starts empty. While $|S| < k$, one picks an edge (u, v) and branches on $A(G \setminus \{u\}, S \cup \{u\})$ and $A(G \setminus \{v\}, S \cup \{v\})$. The leaves of the recursion tree are when $|S| = k$ and then if G is the empty graph, S is returned as a k-VC, and if no such leaf has an empty graph, then "NO" is returned.

Today we will see two improvements. We begin with the first.

Theorem 1.1. *k-VC is in $O^*(1.47^k)$ time.*

Proof. We'll modify the previous algorithm $A(G, S)$ to get a new algorithm $B(G, S)$.

We will add one more case before the crucial branching step:

- If the max degree of G is ≤ 2, then G is just a set of paths and cycles. Solve the VC problem optimally on each one, in polytime.
- Otherwise, G has a node v of degree ≥ 3.

  Now modify the branching step:

  - Accept iff ($B(G \setminus \{v\}, S \cup \{v\})$ accepts or $B(G \setminus N(v) \setminus \{v\}, S \cup N(v))$ accepts).
    (Either v is in the vertex cover, or it isn't and all of its neighbors must be!)
    As in $A(G, S)$, the leaves of the recursion tree are when $|S| = k$ and then if G is the empty graph, S is returned as a k-VC, and if no such leaf has an empty graph, then "NO" is returned.

The recurrence for this algorithm is

$$T(n, k) \leq T(n-1, k-1) + T(n-4, k-3) + \text{poly}(n),$$

and $T(n, 0) = O(1)$, which solves to ... what? How do we analyze this?

We know $T(n, k)$ must be of the form $x^k \text{poly}(n)$ (can't be worse than before!), and we want to determine x.

So our recurrence is $x^k \text{poly}(n) \leq x^{k-1}\text{poly}(n-1) + x^{k-3}\text{poly}(n-4) + \text{poly}(n)$.

Think of another recurrence, $T'(k) = T'(k-1) + T'(k-3)$, where we are ignoring the poly(n) factors above. We want to find x such that $T'(k) = x^k$. OK, let's solve $x^k = x^{k-1} + x^{k-3}$.

Exercise: Show that the solution to the recurrence above is $x = 1.46666\ldots$.

We get $T(n,k) \leq 1.4667^k \text{poly}(n)$. $\square$

Exercise: Generalize the above analysis to show that every recurrence of the form $T(n) = T(n-L_1) + T(n-L_2) + \cdots + T(n-L_i) + O^*(1)$ for constants $L_1, \ldots, L_i$ will have as a solution $T(n) = O^*(r(L_1, \ldots, L_i))$, where $r(L_1, \ldots, L_i)$ is the smallest positive root of the rational function $p(x) = 1 - \sum_{j=1}^{i} x^{-L_j}$.

For example, consider: $T(n) \leq T(n-1) + T(n-2) + O^*(1)$.

The respective expression to solve is $1 - 1/x - 1/x^2 = 0$, or equivalently, $x^2 - x - 1 = 0$. The solutions for x are $x = 1.618\ldots, -.6180333\ldots$, and we obtain $T(n) \leq O^*(1.619^n)$.

A second k-VC algorithm. So far we branched on nodes of degree ≥ 3 until there were no more such nodes, and then we solved in polytime.

You can get an even faster algorithm with more case analysis and thinking about the problem.

Suppose we keep branching on a node of degree ≥ 4. Then the running time recurrence becomes

$$T(k) \leq T(k-1) + T(k-4),$$

(either the node v is in S, or all its ≥ 4 neighbors are in S). If that were the only case, then our running time would solve to $T(k) \leq 1.39^k \text{poly}(n)$. But there are other cases. What if there are no nodes of degree ≥ 4? We need to do something different from before.

Suppose the maximum degree of G is 3. (Recall that if the max degree is ≤ 2, then we can solve k-VC in polynomial time.)

We can add more rules before the branching step:

- Suppose there is a v with $deg(v) = 1$. Then delete v and put its neighbor in S. (Why does this work?)
- Suppose there is a v with $deg(v) = 2$. Let its neighbors be u and w.
  - If $(u, w) \in E$, then (u, v, w) is a triangle. Put u, w in S and delete v. (Why?)
  - If $(u, w) \notin E$, then we "fold" u, v, and w into each other: merge them into one vertex (uvw) with neighborhood $(N(u) \cup N(w)) \setminus \{v\}$ and reduce k by 1. (Why does this work?)

Exercise: Convince yourself that the second "-" above makes sense. In particular, let G be the graph before "folding" u, v, w, and let G' be the graph after the folding. Show that G has a k-VC if and only if G' has a $(k-1)$-VC.

Now, suppose there are no vertices with degree 1 or 2, so every node has degree exactly 3. Consider branching on a degree 3 vertex v. When we put v in the cover and remove it from the graph, all neighbors of v now have degree 2, so we can fold at least one of them and reduce the VC by 1. So k drops by at least 2. When v is not in the cover, we put its 3 neighbors in the cover. This puts 3 vertices in the VC.

In this case we have the recurrence $T(k) \leq T(k-2) + T(k-3)$, which solves to $T(k) < 1.33^k$ and is in fact better than the solution to the recurrence when we branched on nodes of degree ≥ 4.

Thus, if we consider the algorithm that branches on vertices of degree ≥ 4 until there are no more and then runs the above degree 3 algorithm, we get that in total our running time recurrence is

$$T(k) \leq \max\{T(k-1) + T(k-4), T(k-2) + T(k-3)\} < 1.39^k.$$

The best known algorithm for k-VC is by [Chen, Kanj, Xia '06] and runs in $O^*(1.2738^k)$ time. It involves lots more case analysis.

Could we just keep improving with more and more case analysis and more branching rules, and get $O^*((1+\varepsilon)^k)$ time for every $\varepsilon > 0$? That would contradict ETH ...

2 A Theory of Problem Compression

There is a fundamentally different way of thinking about FPT problems, as a notion of "instance compression". When we can "compress" instances of a problem down to smaller, equivalent problems?

Note, this is not like data compression in the usual sense: we can potentially compress much better than data compression. We only have to preserve ONE bit in our compression: whether or not the problem is an "accept" or a "reject".

For example, if a problem is in P and we are allowed polynomial time to "compress" the problem, we can compress every instance down to a constant-sized instance:

[Just run a polynomial-time algorithm for solving the problem – if it outputs "accept" then output a constant-sized yes-instance, otherwise output a no-instance.]

Parametrization gives us a nice way of formalizing this.

Definition 2.1. *Let L be a parameteric problem. We say that L has a kernel (or, is "kernelizable") if there is a polynomial time reduction M from L to L such that for $(y, k') = M(x, k)$, we have $k' \leq k$ and $|y| \leq h(k)$ for some computable $h : \mathbb{N} \mapsto \mathbb{N}$.*

The function $h(k)$ is called the size of the kernel.

Intuitively, the kernel is the "hard part" of the instance. (Since it only took polynomial time to compress down to $h(k)$ size.) The polytime reduction is the "kernelization algorithm". Being kernelizable means we can compress arbitrary instances of L, so that the size of the entire problem instance depends only on the parameter k.

Concrete examples of this will come later. First order of business is to demonstrate an important equivalence:

Theorem 2.1. *Let L be parametric and decidable. L is in FPT if and only if L has a kernel.*

Proof. First, if L has a kernel, then we just run the poly-time reduction from L to L, in polynomial time. We get an output problem instance of size $h(k)$. Because L is decidable, we can solve the problem in $f(h(k))$ time for some computable f. Hence we get an FPT algorithm for L.

Suppose now that L is FPT. Then there's an algorithm A for L that runs in $f(k) + O(n^c)$ time for some function f.

Here's a kernelization algorithm for L:

Given (x, k):

1. if $|x|^c < f(k)$ then simply output (x, k). This is only $O(f(k)^{1/c})$ size because $|x| = n < f(k)^{1/c}$.
2. if $|x|^c > f(k)$ then $f(k) + O(n^c) = O(n^c)$, and in this case, the algorithm A is actually a polynomial time algorithm! We run A on (x, k) in polynomial time. If $A(x, k)$ accepts then output a constant-size yes-instance for L. If $A(x, k)$ rejects, then output a constant-size no-instance for L.

□

Kernelization becomes really interesting when we ask whether an FPT problem has *polynomial-size* kernels:

Definition 2.2. *A parametric L has "feasible" kernels if there is a poly-time reduction from L to L that outputs instances of poly(k) size.*

This is an extreme form of problem compression.

Asymmetry between feasible FPT algorithms and kernelizations.

Definition 2.3. *A "feasible" FPT problem has an algorithm running in $O^*(2^{poly(k)})$ time.*

Claim 1. *Let L be a parameterized problem with parameter k, where the non-parameterized version of L is in NP. A feasible kernelization for L implies a feasible FPT algorithm for L.*

Exercise: Prove the above claim.

The above claim says that if you have a feasible kernel then you also have a feasible FPT algorithm. However, a problem may be "feasible FPT" but *not* necessarily have a feasible kernelization.

Here is an example of this. Consider the k-Path problem: given a graph on n nodes and a parameter k, is there a simple path of length k in the graph?

We will see an $O^*(2^k)$ time algorithm for k-Path, however, there can be no kernel for k-Path of $poly(k)$ size (unless something unlikely happens in complexity theory, namely that the polynomial hierarchy collapeses to the 3rd level [Bodlaender, Downey, Fellows, Hermelin'07]).

Feasible kernels for k-VC.

Theorem 2.2. *k-Vertex Cover has kernels of $O(k^2)$ edges.*

[That is, there is a polytime reduction from k-VC to itself which outputs graphs having at most $O(k^2)$ edges (and vertices).]

Proof. Let $G = (V, E)$ be the given graph instance of k-VC. Let S be a set, initially empty.

Observe that if a node has degree $> k$, then it must be in the vertex cover! (Otherwise all its neighbors would be, but that would be a larger VC than k.)

We run the following procedure, where H is set to G initially, and k' is set to k:

- While there is a degree $> k'$ node x in the current graph H,
  - remove x from H and put x in the growing vertex cover S. Reduce k' by 1.
- // At this point the graph has max degree k'.
- If H has $> k'^2$ edges, return "NO".
- Output H and parameter k'.

Exercise: Show that the second-to-last step makes sense, i.e. that if a graph H has $> k'^2$ edges and max degree $\leq k'$, then it cannot have a k'-VC.

Assuming the exercise above, the output graph H has $O(k^2)$ edges.

Now if one solves k'-VC on H obtaining a vertex cover S', then $S' \cup S$ is a vertex cover of the original graph.

This is a polynomial-time reduction from k-VC to itself, such that the original graph has a k-VC iff the output graph has a k'-VC for some $k' \leq k$. □

It is known that if the polynomial time hierarchy doesn't collapse, there aren't kernels for k-VC of size $O(k^{2-\varepsilon})$ for $\varepsilon > 0$ [Dell and van Melkebeek'10].

Here's a tighter version of the kernelization result which can be useful.

Theorem 2.3. *k-Vertex Cover has kernels with $\leq 2k$ vertices (and hence $O(k^2)$ edges).*

Before we prove this theorem, let's take an aside to Linear Programming.

We begin with the Integer Programming formulation of Vertex Cover. Let $G = (V, E)$, k be given. We add a real-valued variable x_v for each $v \in V$.

$$\begin{gathered}\min \textstyle\sum_v x_v : \\ \text{For all } v \in V,\ x_v \in \{0, 1\}, \\ \text{For all } (u, v) \in E,\ x_u + x_v \geq 1.\end{gathered}$$

We will write an "LP relaxation" for k-VC by changing the Boolean requirement to a requirement that $x_v \in [0, 1]$. We write the following LP:

$$\begin{gathered}\min \textstyle\sum_v x_v : \\ \text{For all } v \in V,\ x_v \geq 0 \text{ and } x_v \leq 1, \\ \text{For all } (u, v) \in E,\ x_u + x_v \geq 1.\end{gathered}$$

We know (e.g. by [Kachiyan'79,Karmakar'84]) that LPs are solvable is in polynomial time. So we can solve this LP, get values $a_v \in [0, 1]$ for each vertex v, so that the settings $\{x_v = a_v\}_{v \in V}$ satisfy the above inequalities and minimize $\sum_v x_v$.

This LP can be used to get a 2-approximation to VC: output a vertex cover of size at most twice that of the optimal vertex cover by returning $S := \{v | a_v \geq 1/2\}$.

Claim 2. *$S = \{v | a_v \geq 1/2\}$ is a vertex cover.*

Proof. For every edge $(u, v) \in E$, we know that $a_u + a_v \geq 1$, so at least one of a_u and a_v is $\geq 1/2$, and hence at least one of u or v was put in S. □

Claim 3. *Let k be the minimum size of a vertex cover of G. Then $|S| \leq 2k$.*

Proof. Since the LP is a relaxation of the Integer Program for VC, we know that $\sum_v a_v \leq k$, as the value of the Integer Program is k and the value of the LP can only be smaller.

For every $v \in S$, we also know that $\sum_{v \in S} a_v \geq |S|/2$ by the definition of S. Hence, $|S| \leq 2 \sum_{v \in S} a_v \leq 2 \sum_v a_v \leq 2k$. □

Now let's prove k-VC has a kernel with $\leq 2k$ vertices, using the LP solution $\{x_v = a_v\}_{v \in V}$.

Partition the graph into sets $P = \{v \in V | a_v > 1/2\}$, $Z = \{v \in V | a_v = 1/2\}$, $N = \{v \in V | a_v < 1/2\}$.

Claim 4. *Suppose that G has a min vertex cover of size k. Then*

(a) $S := P \cup Z$ contains a minimum VC that contains P, and

(b) $|S| = |P| + |Z| \leq 2k$.

Proof. We already proved (b) in the previous claim, so let's consider (a).

(*) There are no edges between N and Z, or inside N. (Why? because for every (u, v) with $u \in N$ and $v \in Z \cup N$, we have $a_u + a_v < 1$, so (u, v) cannot be an edge.)

(**) Thus, all edges from N must go to P.

Now, let k be the size of the minimum VC. Let X be any minimum VC.

Form X' from X by removing $N \cap X$ and adding $P \setminus X$.

Clearly X' is contained in $P \cup Z$ and contains P completely.

We want to show that

1. X' is a vertex cover and
2. $|X'| \leq |X|$. (Hence X' is also optimal.)

(1) follows from (**) since the only edges that $(N \cap X)$ was responsible for have endpoints in P, and all of P is in X'.

For (2) it suffices to show that $|N \cap X| \geq |P \setminus X|$ (we remove at least as many as we add).

Consider any edge (u, v) with $u \in P \setminus X$ and $v \in (P \cup Z) \cap X$. Such an edge must have $a_u + a_v > 1$ by the definition of P and Z.

Let ε be the minimum over all $u \in (P \setminus X)$ of $(a_u - 1/2)$. Note that $\varepsilon > 0$.

Now, set

- $a'_u := a_u - \varepsilon$ for all $u \in (P \setminus X)$
- $a'_u := a_u + \varepsilon$ for all $u \in (N \cap X)$
- $a'_u := a_u$ otherwise.

Exercise: Show that a' defined above is still a feasible solution to the LP for k-VC.

Finally, consider the VALUE of the LP solution a'_v. It is:

$$\sum_u a'_u = \sum_u a_u - \varepsilon \cdot (|P \setminus X| - |N \cap X|).$$

(because we subtract ε for each node in $P \setminus X$ and add ε for each node in $N \cap X$)

Since a_u is an optimal solution to the LP already, a'_u can't achieve a better solution. So we must have that $\sum_u a'_u \geq \sum_u a_u$ and hence $|P \setminus X| - |N \cap X| \leq 0$, i.e. $|N \cap X| \geq |P \setminus X|$.

From this we get that X' is also a minimum vertex cover, and hence $P \cup Z$ contains a minimum vertex cover, proving the claim. □

Corollary 2.1. *k-VC has a $2k$ node kernel.*

Proof. Run the LP as above. If the LP solution is $> k$, return NO. Otherwise, partition into P, Z, N, then remove the nodes in $N \cup P$ from the graph, and place all of P inside the candidate vertex cover. The above claim showed that the graph induced by Z is a valid kernel on $\leq 2k$ nodes, as $P \cup Z$ contains a min vertex cover of the graph that contains P completely. The only edges not covered by P are those completely contained in Z, so solving k-VC reduces to finding a min VC in Z. Clearly, $|Z| \leq |P \cup Z| \leq 2k$. □

State of the art in kernelization for VC:

- Best known kernel for nodes is: $2k - c\log(k)$ for any constant c. [Lampis 2011]
- If there is a kernel for k-VC with $k^{2-\varepsilon}$ edges for some $\varepsilon > 0$, then coNP is contained in NP/poly. (cool stuff in complexity... unlikely stuff, though) [Dell and van Melkebeek 2010]