1 Better branching for Vertex Cover

In the previous class we saw an $\mathcal{O}^*(2^k)$ time algorithm for Vertex Cover parameterized by solution size. The algorithm was based on the bounded search tree technique. Now we will see how to improve that running time by a more clever branching.

Our new recursive procedure is as follows. If all vertices in the graph have degree one or less, each connected component is either an isolated vertex or an edge, and the problem is trivial to solve in polynomial (even linear) time. Otherwise, we pick a vertex v of degree at least two. Either v belongs to a solution or not. In the latter case, however, the set of its neighbours $N(v)$ has to be contained in a solution. We branch on these two possibilities, either removing v or $N(v)$ from the graph, and decreasing k by 1 or $|N(v)|$, respectively.

Since $|N(v)| \geqslant 2$, the number of leafs[1] of the search tree can be bounded with the recursive formula

$$T(k) = T(k-1) + T(k-2).$$

Exercise 1. Prove that $T(k)$ grows as ϕ^k, where $\phi = \frac{1+\sqrt{5}}{2} \leqslant 1.6181$. How did we know that ϕ is the right answer?

This gives us an algorithm running in $\mathcal{O}^*(1.6181^k)$ time. In order to further improve this running time we would like to always be able to branch on higher degree vertices.

Exercise 2. Show an algorithm that solves Vertex Cover in polynomial time in graphs with maximum degree 2.

Now we can have a comfort of always branching on a vertex v with $|N(v)| \geqslant 3$, decreasing k by either 1 or at least 3, thus the tree size bound becomes

$$T(k) = T(k-1) + T(k-3).$$

Exercise 3. Solve the recursive equation above.

With more complex rules and a more intricate analysis one can further decrease the base of the exponential function, however we will stop here.

Exercise 4. Use `http://fpt.wikidot.com/` website to check what is the best known running time for Vertex Cover parameterized by solution size.

2 LP-based kernel for Vertex Cover

In the previous class we saw an $O(k^2)$ kernel for Vertex Cover. Now we will construct a linear one.

Consider the standard LP relaxation of Vertex Cover:

[1] Often it is more convenient to bound the number of leafs, rather than all tree nodes. Since in a branching tree each internal node has at least two children, the number of leafs bounds the number of all nodes up to a factor of 2.

$$\begin{array}{lll} \text{minimize} & \sum_{v \in V} x_v & \\ \text{subject to} & x_u + x_v \geqslant 1 & \text{for all } (u,v) \in E, \text{ and} \\ & x_u \geqslant 0 & \text{for all } v \in V. \end{array}$$

Exercise 5. Given an example of a graph for which the value of the above LP is stricly smaller than the minimum vertex cover.

Theorem 1 (Nemhauser-Trotter)**.** *There is always an optimal solution to the Vertex Cover LP that is* ***half-integral****, i.e.,* $\forall_{v \in V} x_v \in \{0, 1/2, 1\}$.

Proof sketch. Consider any optimal solution. Split vertices into three groups:

$$\begin{aligned} V_0 &= \{v \mid x_v < 1/2\}, \\ V_{1/2} &= \{v \mid x_v = 1/2\}, \\ V_1 &= \{v \mid x_v > 1/2\}. \end{aligned}$$

Note that there can be no edges between V_0 and $V_0 \cup V_{1/2}$.

First, consider the case that there is a matching between V_0 and V_1 that saturates V_1. For each edge of the matching we have $x_u + x_v \geqslant 1$, hence the total contribution of V_0 and V_1 to the solution value is at least $|V_1|$. Therefore, we can as well set all nodes in V_0 to 0, all nodes in V_1 to 1, and still get a feasible solution, which is no worse than the original.

In the remaining case, when there is no such matching, by Hall theorem there is a subset $X \subseteq V_1$ such that $|N(X) \cap V_0| < |X|$. Decrease all variables in X by ϵ, and increase all variables in $N(X) \cap V_0$ by ϵ. The resulting solution is still feasible, and smaller than the original one, contradiction. □

Exercise 6. Verify claims left without a proof in the above sketch.

Consider the following reduction rule: If there is an optimal LP solution with $x_v = 1$, remove v and decrease k by one.

Exercise 7. Show that the above reduction rule is safe. Hint: adapt the proof of Nemhauser-Trotter theorem.

After applying this reduction rule exhaustively, and removing isolated vertices, we are left with a graph such that an optimal VC LP solution assigns $1/2$ to every vertex. If the LP solution value is greater than k, we conclude this is a NO-instance, otherwise the number of vertices is at most $2k$. Hence, we obtained a $2k$-vertex kernel for Vertex Cover, which is current state of the art.

Note that all kernels we have seen so far have the property that a c-approximate solution to the kernel translates to a c-approximate solution to the initial instance. Obtaining a $1.99k$ kernel of that type would imply a breakthrough in approximation algorithms for Vertex Cover.

3 Longest Path via color coding

Now we will look at another useful tool for constructing FPT algorithm, the *color coding* technique, introduced by Alon, Yuster and Zwick. As an example application we will use the Longest Path problem.

In the Longest Path problem we are given a graph and an integer k, and we want to find a simple path of length k. The problem is known to be NP-hard in general graphs.

Exercise 8. Show that the variant of the problem where we lift the requirement that the path has to be simple is in polynomial time.

Exercise 9. Show that the problem restricted to DAGs is solvable in polynomial time.

Consider the following algorithm for Longest Path.

Algorithm 1 Simple color coding for Longest Path

1. Assign to each vertex v a *color* $c(v)$ selected independently at uniformly at random from $[k] = \{1, 2, \ldots, k\}$
2. Find a path $v_1, v_2, \ldots, v_k$ such that $\forall_i c(v_i) = i$.

Exercise 10. Show that Step 2 can be done in polynomial time.

If there is no path of length k, the algorithm won't find any. If there is such a path, the algorithm will succeed in finding it if it happens to be colored with consecutive integers, which happens with probability $1/k^k$. Repeating the algorithm k^k times rises the probability of success to $1 - (1 - \frac{1}{k^k})^{k^k} \geqslant 1 - 1/e$. Repeating it $\mathcal{O}(k^k \log n)$ times succeeds w.h.p. (with high probability, i.e., $1 - 1/n^c$ for arbitrarily high c).

This gives us a randomized $\mathcal{O}^*(k^k)$ time algorithm for Longest Path. A simple modification could improve it to $\mathcal{O}^*(k!)$ (do you see how?), but we will skip this step. Now, let us see how doing more work in Step 2 can save us a lot of iterations.

Algorithm 2 Improved color coding for Longest Path

1. Assign to each vertex v a *color* $c(v)$ selected independently at uniformly at random from $[k]$
2. Find a path $v_1, v_2, \ldots, v_k$ such that $\{c(v_1), c(v_2), \ldots, c(v_k)\} = [k]$ (or, equivalently, $\forall_{i \neq j} c(v_i) \neq c(v_j)$).

The probability of success grows to $\frac{k!}{k^k} \approx \frac{1}{e^k}$, and the required number of iterations drops accordingly.

The problem we deal with in Step 2 is no longer polynomial time solvable, but we can solve it in $\mathcal{O}^*(2^k)$ time, using dynamic programming. For $A \subseteq [k]$ and $v \in V$, let $dp[A][v]$ be a boolean value indicating whether there is path $v_1, \ldots, v_{|A|}$ such that $\{c(v_1), \ldots, c(v_{|A|})\} = A$ and $v_{|A|} = v$.

Exercise 11. Propose a recursive formula solving the above dynamic program.

Hence we obtain randomized $\mathcal{O}^*((2e)^k)$ time algorithm for Longest Path.

Most algorithms using color coding can be derandomized, using a pseudorandom object called *splitter*. The derandomization is very effective, e.g., for the above algorithm we lose only a factor $k^{\mathcal{O}(\log k)}$. We will not cover the topic in the course. If you are interested, Chapter 5.6 of *Parameterized Algorithms* by Cygan et al. contains further reading on derandomization.