1 Vertex Cover above LP

So far we have seen algorithms and kernels for the Vertex Cover problem prameterized by the solution size k. One could argue that this is not a “practically relevant” parameterization. Indeed, a graph needs a very specific structure to have large n and small k. Now we will consider a parameter that can be useful more often.

New parameter. Recall the LP relaxation of Vertex Cover from the previous class. We will denote the value of its optimal solution simply by LP. Let us consider the parameter $\ell = k - LP$. It is always non-negative, because LP is a lower bound for the actual (integral) minimum vertex cover. More importantly, a graph can have a large optimal vertex cover k but still a small parameter ℓ (in other words, the LP-based lower bound can be almost tight). For such graphs, algorithms parameterized by ℓ are much more useful than those parameterized by k, even if their (exponential) running time dependence on the parameter is worse.

Algorithm. Now we will see an $\mathcal{O}^*(4^{k-LP})$ time algorithm for Vertex Cover. The algorithm is based on the bounded search tree technique. In each node of the search tree we begin by exhaustively applying a variant of the reduction rule we already know from the LP-based kernel: as long as there is a vertex v such that there exists an optimal LP solution assigning $x_v = 1$, we add v to the cover and decrease our budget k by one.

Exercise 1. Prove that the above reduction rule does not increase the parameter $k - LP$.

Now we branch on any edge $(u, v) \in E$, i.e. we recursively consider two cases: (1) u belongs to the cover, and (2) v belongs to the cover. In both cases k decreases by 1, but LP can also decrease – conveniently, it cannot decrease too much.

Exercise 2. Prove that if the only optimal half-integral LP solution assigns $1/2$ to all vertices, then removing any vertex from the graph decreases the value of the LP solution by at most $1/2$.

Hence, at each level of the search tree, we decrease the parameter $k - LP$ by at least $1/2$, thus the tree has depth at most $2 \cdot (k - LP)$, and the algorithm indeed runs in $\mathcal{O}^*(2^{2(k-LP)}) = \mathcal{O}^*(4^{k-LP})$ time.

Application. Now we will see an example in which the above parameterization is indeed useful. Let us consider a parameterized version of the Max-2-SAT problem (recall that Max-2-SAT is NP-hard, in contrast to 2-SAT, which is in P).

Given a 2-CNF formula and an integer k, the Almost-2-SAT problem asks to find an assignment that satisfy all but at most k clauses. The task can be reformulated as: remove at most k clauses to make the formula satisfiable.

We will begin by solving a variable deletion variant of the problem. Given a 2-CNF formula and an integer k, the Variable-Deletion-2-SAT problem asks to find a set of at most k variables such that after removing them (and all clauses that contain them) the formula is satisfiable. In other words, removing a variable is equivalent to setting it to both true and false at the same time.

Consider a 2-CNF formula with n variables, and construct the following graph. It has $2n$ vertices, corresponding to literals. For each variable v, literals v and $\neg v$ are connected with an edge. Moreover, for each clause, the two literals it contains are connected with an edge.

Exercise 3. What is the value of an optimal solution to the vertex cover LP relaxation for the above graph?

Exercise 4. How does the size of the minimum vertex cover in the above graph relate to the optimal Variable-Deletion-2-SAT solution for the initial formula?

Now it remains to show that the variable deletion and clause deletion variants are equivalent.

Exercise 5. Prove that the Almost-2-SAT and Variable-Deletion-2-SAT problems are equivalent in the sense that an $\mathcal{O}^*(f(k))$ time algorithm for one of them gives an $\mathcal{O}^*(f(k))$ time algorithm and vice versa. Note that the reductions have to preserve (the asymptotics of) the function f but can alter polynomial factors hidden in the $\mathcal{O}^*$ notation.

2 Iterative compression

In the last week's problem set we have seen an algorithm for Vertex Cover using the iterative compression technique. Now we will apply this technique to the Feedback Vertex Set in Tournaments problem: given a tournament and an integer k, remove a set of at most k vertices to make the tournament acyclic.

It would be enough to give an algorithm for the disjoint variant of the problem, but let us recall the whole framework again, this time from the top to the bottom.

First, we need to make sure that if the graph admits a solution of size k, then so does each of its (induced) subgraphs. This is the case, since the property of being acyclic is a hereditary property (i.e. if a graph has it, each subgraph has it as well).

We start with arbitrary $k + 2$ vertices of the input graph. The induced subgraph obviously has an FVS of size k. We will keep adding vertices one by one (in any order), and we will try to maintain an FVS of size k. When we add a vertex to the subgraph, we can also add it the FVS, and obtain an FVS of size $k + 1$ for the larger subgraph. Now, we need to *compress* that FVS so that it is again of size k.

In the compression procedure, we first guess which vertices of the old-too-large FVS will belong to the new-just-right FVS – there are $2^{k+1} - 1$ possible guesses, which we will have to check. For each particular guess, we can remove the guessed vertices from the graph (and decrease the budget k accordingly), and decide never to remove the remaining vertices of the old-too-large FVS. Note that these "protected" vertices constitute an FVS in the reduced graph. Here we arrive at the disjoint variant of the problem: given a tournament T and an FVS $F \subseteq T$, find the smallest size of an FVS in T disjoint from F.

Let us solve the disjoint variant. Note that the subtournament induced by F has to be acyclic – otherwise no solution exists – and hence it has a unique topological ordering. Moreover, $T \setminus F$ is also acyclic and has a unique topological ordering. Since T is a tournament, each vertex $v \in T \setminus F$ either (1) has a unique "slot" $s(v) \in [|F| + 1]$ where it can possibly fit F, or (2) forms a cycle with F and thus has to be included in every FVS disjoint from F. Let $v_1, v_2, \dots$ be the unique topological ordering of case (1) vertices of $T \setminus F$. Consider the sequence $s(v_1), s(v_2), \dots$.

Exercise 6. Show that any (weakly) increasing subsequence of the above sequence, together with the case (2) vertices, corresponds to an FVS in T disjoint from F and vice versa.

Exercise 7. Show how to solve the Longest (Weakly) Increasing Subsequence problem in quadratic time. Hint: design a dynamic program, or reduce to the Longest Common Subsequence problem for the initial sequence and its sorted copy. Remark: Longest Increasing Subsequence can be solved in $\mathcal{O}(n \log n)$ time with a slightly more complex algorithm.

To conclude, we solve the Disjoint FVS in Tournaments problem in polynomial time, and, as a consequence, the FVS in Tournament problem in $\mathcal{O}^*(2^k)$ time.

Disjoint variant in FPT time. So far we have seen examples of the iterative compression technique for problems which admit polynomial time algorithms for their disjoint versions. That is not always the case. For examples, for FVS in general undirected graphs, the best known algorithms for the disjoint variant of the problem run in time of the form $\mathcal{O}^*(c^k)$, for a constant c.

Exercise 8. Given access to an $\mathcal{O}^*(c^k)$ time algorithm for Disjoint FVS, solve FVS in $\mathcal{O}^*((c+1)^k)$ time.

3 Treewidth

Treewidth is a graph parameter often used in FPT algorithms. At first the parameter might seem unintuitive, so let's build some intuition.

Definition. For an undirected graph $G = (V, E)$, its *tree decomposition* is a tree T with the following properties. Each tree node $t \in T$ is associated with a subset of vertices $B_t \subseteq V$, which we call a *bag* of t. For each edge $(u, v) \in E$ there must exist a node $t \in T$ whose bag contains both both ends of the edge, i.e. $\{u, v\} \subseteq B_t$. Moreover, for each vertex $v \in V$, bags containing v must form a connected subgraph of T.

The *treewidth* of a graph is the smallest integer w such that the graph has a tree decomposition with all bags containing at most $w + 1$ vertices.

Properties.

Exercise 9. Prove that trees have treewidth 1.

Exercise 10. What are the treewidths of: a cycle C_n, a complete graph K_n, a complete bipartite graph $K_{n,n}$?

Exercise 11. Prove that adding a vertex to a graph can increase its treewidth by at most one, and subdividing an edge cannot increase the treewidth.

Exercise 12. Prove that if a graph has minimum degree d then it has treewidth at least d.

Exercise 13. Prove that every clique of a graph has to be contained in some bag of its tree decomposition.

A *coloring number* (also called *degeneracy number*) of a graph is the smallest integer δ such that the vertices of the graph can be ordered in such a way that each vertex has at most δ neighbors appearing before it in the order.

Exercise 14. Prove that graphs of treewidth k have coloring number at least k.

A graph is *outerplanar* if it can be embedded in the plane in such a way that all vertices are on the outer face.

Exercise 15. Prove that outerplanar graphs have treewidth at most 2.

An alternative definition. A graph is a k-*tree* if it can be obtained from the following process. We start with a clique on k vertices. Iteratively, we add a vertex to the graph and connect it with some k existing vertices which form a clique.

Exercise 16. Prove that a k-tree has treewidth exactly k.

Exercise 17. Prove that every graph of treewidth k is a subgraph of some k-tree.

Vertex Cover parameterized by treewidth. One of the reasons why treewidth is useful is that usually a dynamic programming algorithm working for trees can be generalized to work for graphs of bounded treewidth.

Exercise 18. Propose a $2^{\mathcal{O}(w)}n$ time algorithm for the Vertex Cover problem in graphs of treewidth at most w. Hint: for $t \in T$ and $X \subset B_t$ let $dp[t][A]$ denote the size of the smallest vertex cover of the subgraph induced by vertices appearing in bags below t such that the vertex cover contains A.