Parameterized Algorithms WS 2021
Prof. Dr. P. Rossmanith
Dr. E. Burjons, M. Gehnen, H. Lotze, D. Mock

RWTH AACHEN UNIVERSITY

Date: December 13th, 2021

Exercise Sheet with solutions 07

Task T22

Consider the following variant of Vertex Cover:

Half Partial Vertex Cover

Input: A graph $G = (V, E)$, an integer k.
Parameter: The integer k.
Question: Can k vertices in G cover at least $|E|/2$ edges?

Show that Half Partial Vertex Cover is in $W[1]$ by reducing it to STMA.

Solution

The important point about this reduction is that one has to essentially compute how many edges a set of k candidate vertices covers. One cannot hope to do this in $f(k)$ time for any f, if the machine actually *writes* down the number of edges covered by each vertex and then adds these numbers. What one would do in such a case is observe that the size of the state space can be polynomial in the input graph size and use the state space to count the number of edges.

Let us assume that in addition to the adjacency matrix of the input graph, the machine has hardwired into it the number of edges covered by each vertex. If the machine guesses k candidate vertices as solution, then one possible way of counting the number of edges covered by such a set is as follows. The machine has a special set of $nm + 1$ states $\{q_0, q_1, \ldots, q_{nm}\}$ explicitly for counting. Initially the machine is at "count-state" q_0. If the machine is at count-state q_j on seeing vertex v_i from the candidate solution, the machine moves to state $q_{j+\deg(v_i)}$ and moves the head one cell to the right. The value of $\deg(v_i)$, the degree of vertex v_i, is already hardwired in the machine and this doesn't have to be computed. One can see that this allows computing the total number of edges covered by the candidate solution in $O(k)$ steps. However there's one snag. Edges whose both end-points are in the candidate solution are counted twice. The machine now constructs the subgraph induced by the candidate solution and actually counts the number of edges in it. This is fine as the size of this subgraph is at most $O(k^2)$. For every edge that is counted twice, the machine subtracts one from the count (in the state space). If the final count-state of the machine is q_i then the machine accepts iff $i \geq m/2$.

Task T23

Consider the following variant of Vertex Cover:

Partial Vertex Cover

Input: A graph $G = (V, E)$, an integer k, and an integer t.
Parameter: The integer k.
Question: Can k vertices in G cover at least t edges?

Show that Partial Vertex Cover is W[1]-hard.

Solution

Let (G, k) be an instance of Independent Set, where G is an r-regular graph. We claim that G has an independent set of size k if and only if (G, k, rk) is a yes-instance of Partial Vertex Cover. If G has an independent set S of size k, then every edge of G is covered by at most one vertex of S, hence together they cover exactly rk edges. Conversely, suppose that G has a set S of k vertices covering rk edges. As each vertex of G covers exactly r edges, this is only possible if no edge of G is covered twice by these k vertices, that is, if they form an independent set.

Task T24

In exercises T22 and T23 we saw that two different variants of the partial vertex cover problem are in W[1] or W[1]-hard respectively. Is one of the two variants W[1]-complete? Prove it!

Solution

We can reduce the Partial Vertex Cover to Half Partial Vertex Cover by padding. If $t > |E|/2$, we add $t - |E|/2$ independent edges to G with the same parameter k. The idea is raise the number of edges of the graph such that $|E'|/2 = t$ without changing the optimal solution. As independent edges need one vertex per edge to cover, those edges will not be chosen into an optimal set.

Otherwise, it holds that $t < |E|/2$. Then we add a star of some certain size s to G and increase the parameter by one. We set s such that $t + s = (|E| + s)/2$. The idea is to raise t to $|E|/2$. The edges of the star can be cover by exactly one vertex.

One can show with a detailed proof that this reduction is sound and a proper FPT-reduction.

Instead of this reduction, one could adapt T23 to work with Half Partial Vertex Cover. For Half Partial Vertex Cover one can appropriately choose the t in the Independent Set (for regular graphs) reduction.

Task H15 (7 credits)

Consider the following variant of Hitting Set:

Half 3-Hitting Set

Input:	A finite universe U, a family $\mathcal{F} \subseteq 2^U$ of sets of size exactly three, an integer k.
Parameter:	The integer k.
Question:	Can k elements of U hit at least $\lvert\mathcal{F}\rvert/2$ sets?

Show that Half 3-Hitting Set is in $W[1]$.

Solution

As in Exercise T22, we reduce to the STMA problem. Given an instance $(\mathcal{U}, \mathcal{F}, k)$ of the problem, we construct a non-deterministic Turing machine M which has hardwired into it a three-dimensional array $A[n \times n \times n]$ such that $A[p, q, r] = 1$ iff $\{p, q, r\} \in \mathcal{F}$ (assume that $\mathcal{U} = \{1, \ldots, n\}$). In addition, the machine also has hardwired into it, for each $i \in \mathcal{U}$, the value $m(i) = |\{S \in \mathcal{F} \mid i \in S\}|$. The machine has a special set of $\sum_i^n m(i) \leq n^4$ "count" states q_i which enables it to count the number of sets covered by a candidate solution.

The machine has as input alphabet $1, \ldots, n, \#$ and as tape alphabet the blank symbol and works as follows:

1. First it guesses a set of k numbers from the set $1, \ldots, n$ and writes these on to its tape. It then verfifes whether the numbers chosen are distinct. These represent the vertices of the candidate solution.

2. The machine moves to count state q_0 and starts reading the guessed vertices. If the machine is in count state q_i and sees vertex u, it then moves to count state $q_{i+m(u)}$ and moves its head once cell to the right. At the end of $k+1$ steps, when it has finished reading all guessed vertices, it moves to a count state that denotes the sum of the total number of times a set is covered by one of the candidate vertices.

3. Now for each triple T of distinct numbers from the candidate solution, if $T \in \mathcal{F}$, then the machine decreases it count state by two. The machine writes these sets on its tape.

4. At the end of this procedure, for each tuple T of the candidate solution, if T can be augmented to a set that has not already been written down in the last step, it decreases the count state by one.

5. The machine accepts if its final count state is q_i where $i \geq |\mathcal{F}|/2$.

Steps 1 and 2 take $O(k^2)$ time in total; Step 3 takes $O(k^3)$ time and Step 4 takes $O(k^2 \cdot k^3) = O(k^5)$ (as we might have to read the triples from Step 3) time.

Task H16 (8 credits)

The Partial Vertex Cover problem is defined as follows: given a graph G and integers k and t, decide whether there exists k vertices that cover at least t edges. The parameter is the integer t (when parameterized by k only, the problem is W[1]-complete). The point of this exercise is to use color-coding to obtain a randomized FPT-algorithm for this problem.

1. Show that if $t \leq k$ then the problem is polynomial-time solvable. What happens if the maximum degree of the input graph is at least t?

2. Now use the following idea for coloring the vertices of the graph with two colors, say, green and red. Assume that there exists $S \subseteq V(G)$ of size at most k such that S covers at least t edges. Color each vertex red or green with probability $1/2$. Show that the probability that vertices in S are colored green and all vertices in $\{u \in V(G) \setminus S \mid (u,v) \in E(G) \text{ for some } v \in S\}$ are colored red is a function of k and t. Call such a coloring a *proper coloring*.

3. Given a properly colored graph, we now need to identify a solution quickly. Note that the green vertices decompose the graph into connected components and that these contain the potential solution vertices. Show that in a properly colored graph, the solution is always the union of some green components, that is, the solution either includes all vertices of a green component or none. Hence any green component with k or more vertices that does not cover at least t edges can be discarded. Use this to design an algorithm that identifies a solution set in a proper two-colored graph.

4. Use all the above facts to design a randomized FPT-algorithm for the problem and analyze its time complexity.

Solution

1. If $t \leq k$, then pick k vertices each of degree at least one, in succession. Such a set will cover at least k edges. If such a set cannot be picked then the given instance is a no-instance. Also if the maximum degree is at least t, then picking a vertex of maximum degree suffices. Hence we may assume that $k < t$ and that the maximum degree is at most $t-1$.

2. Since each vertex has degree at most $t-1$, the size of the neighborhood set of S is at most $k(t-1)$. The probability that a coloring is proper is at least

$$\frac{1}{2^{k+k(t-1)}} = \frac{1}{2^{kt}}$$

3. By the very definition of a proper coloring, the solution is the union of some green components. Now a green component with k vertices or more can be discarded. For the remaining components, we maintain a table with indices $1 \leq i \leq k$. For index i, we store a component C_i with i vertices that covers the maximum number of edges among all components on i vertices. Define $\mathscr{C} = \{C_1, \ldots, C_k\}$. Try all possible subsets $\mathscr{C}' \subseteq \mathscr{C}$ such that $\sum_{C \in \mathscr{C}'} |V(C)| \leq k$ and check the number of edges covered by the components in $\mathscr{C}'$. If this number is at least t, then we output the components in $\mathscr{C}'$. The time taken is $O(2^k \cdot \text{poly}(k,t))$ (this can be achieved much faster, but this suffices for our purposes).

4. The randomized algorithm is now clear. It repeatedly does the following: color the vertices of the graphs using colors red or green and then test whether there exists at most k green vertices which cover at least t edges. If the instance is a yes-instance, the expected number of repetitions required is 2^{kt} and hence the expected running time is $O(2^{kt} \cdot 2^k \cdot \text{poly}(n)) = O(2^{t+t^2} \cdot \text{poly}(n))$.