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

RWTH AACHEN UNIVERSITY

Date: October 18th, 2021

Exercise Sheet with solutions 01

Task T1

The Independent Set problem is defined as follows. Given a graph $G = (V, E)$ and an integer k, is ther a set S of size k such that for all $u, v \in S$ where $u \neq v$ it holds $uv \notin E(G)$? Is Independent Set restricted to graphs of maximal degree d, where d is a constant, fixed paramater tractable parameterized by the size of the solution k?

Solution

The problem is in FPT. One can design an FPT algorithm as follows. Take any node v. It will have degree d or lower. We know that either v or one of its neighbors must be in a maximal independent set. We branch over every possibility by taking a node of $N[v]$ into the independent set, deleting it and all of its neighbors and recursively solving the remaining graph. This gives us an algorithm with running time $O((d+1)^k \cdot n)$.

Task T2

The Triangle Vertex Deletion problem is defined as follows. Given a graph $G = (V, E)$ and an integer parameter k, are there k vertices whose deletion results in a graph with no cycles of length three? Show that this problem is fixed-parameter tractable. What is the running time of your algorithm? Is there some easy way to improve the running time?

Solution

The idea now is to branch on the vertices of a triangle. Let (u, v, w, v) be a three cycle in G. We recurse on the instances $(G - u, k - 1)$, $(G - v, k - 1)$ and $(G - w, k - 1)$. The branching vector is $(1, 1, 1)$ and the running time is $3^k \cdot \text{poly}(|G|)$.
To improve this, one can take into account that if a triangle is independent no branching is needed, deleting one of their vertices is enough, but if a triangle with vertices x, y, z intersects with a triangle with vertices x, a, b, one has five branching possibilities, deleting x, which decreases the size by 1, or deleting y or z and a or b, which are 4 possibilities that decrease the size by 2. Thus, the branching vector becomes $(1, 2, 2, 2, 2)$, with a branching number less than 2.5616.

Task T3

Given a boolean formula φ in CNF with n variables and m clauses and an integer k, you have to decide whether there exists an assignment to the variables that satisfies at least k clauses. Assume that the literals appearing in a clause are all distinct and that no clause contains a literal and its negation. We first consider several special cases.

1. If φ contains only unit clauses (clauses with one literal), then how can you find the optimum assignment?
2. If there are k clauses in φ that each contain k literals, then show that one can find an assignment satisfying all these clauses in $|\varphi|$ time.
3. Show that one can always find an assignment that satisfies at least $m/2$ clauses of φ.

Use these facts to design an FPT-algorithm with k as parameter.

Solution

1. Set a literal to `true` or `false` depending on which satisfies more clauses.

2. Let the clauses be denoted $C_1, \ldots, C_k$. Pick an arbitrary literal from C_1 and set it such that C_1 is satisfied. Inductively, proceed to C_i and pick a literal l_i that has not been set in the previous $i-1$ rounds. We are guaranteed that such a literal exists since there are at least k distinct literals in the k clauses that we started out with. Set l_i such that C_i is satisfied. The total time taken is linear in the sizes of all the clauses $C_1, \ldots, C_k$.

3. Start will the all-`false` assignment. If this does not satisfy at least half the clauses, then it falsifies half the clauses. But then the all-`true` assignment satisfies all those clauses falsified by the all-`false` assignment. In either case, you have an assignment that satisfies at least half the clauses.

Our algorithm works as follows:

1. If $k > m$ output "No" and halt.

2. If $k \leq m/2$ output "Yes" and halt.

3. Separate the clauses of φ into short and long clauses: short clauses have fewer than k literals and long clauses have at least k literals. Let φ_l and φ_s be the conjunction of the long and short clauses respectively. Let there be b long clauses. If $b \geq k$ then output "Yes" and halt.

4. Construct a binary tree of the following type: the root is labeled with the pair $(\varphi_s, k-b)$. In general, each node of the tree is labeled by a pair (ψ, j), where ψ is a boolean formula in CNF and j is a non-negative integer. If the label of a node satisfies one of these three categories it is a *leaf* node:

   (a) If j exceeds the number of clauses in ψ, then (ψ, j) is a leaf-node labeled "No."

   (b) If $j = 0$, then (ψ, j) is a leaf-node labeled "Yes."

   (c) If no literal in ψ occurs positively *and* negatively then (ψ, j) is a leaf node labeled "Yes."

   Pick a literal v that occurs both positively and negatively in ψ. Let the number of clauses that contain this literal in the positive form be l_{pos} and the number of clauses that contain it negatively be l_{neg}. Let ψ_v and $\psi_{\bar{v}}$ be the formulas obtained by setting v to `true` and `false`, respectively. Then (ψ, j) has two children labeled $(\psi_v, j - l_{\text{pos}})$ and $(\psi_{\bar{v}}, j - l_{\text{neg}})$. If the tree has a leaf node labeled "Yes", output "Yes" and halt. Since l_{pos} and l_{neg} are both at least 1 and j is at most k, we get a branching algorithm of 2^k.

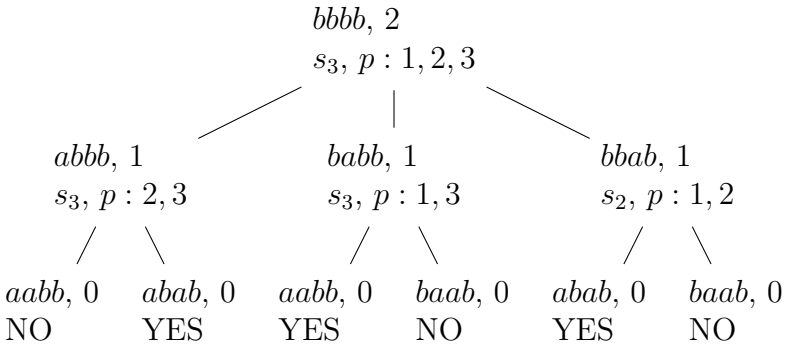
Task H1 (5 credits)
Use the algorithm presented in the lecture to solve the following instance of Center String. Recall that the parameter is the hamming distance which we fix to be $d = 2$.

$$\begin{aligned} s_1 &= bbbb \\ s_2 &= aabb \\ s_3 &= aaaa \\ s_4 &= abaa \end{aligned}$$

Solution

Here is the full computation tree of the algorithm. Note that the tree is not unique.

The first line are the inputs for center string. The second line gives the chosen string whose Hamming distance is to high and the selected positions where it differs. Our always chose the first string and the first positions that do not match. The result for this input is YES with *abab*.



Task H2 (10 credits)

a) Invent a reduction rule for Vertex Cover that removes all pendant vertices (vertices of degree one). It should generate an equivalent instance in polynomial time.

b) Design a bounded search tree algorithm for Vertex Cover in graphs of minimum degree two. Analyze its running time. It should be faster than $1.7^k n^{O(1)}$.

c) Use the results above to get an algorithm that solves Vertex Cover on general graphs with the running time of b).

Solution

a) There is always an optimal solution that does not contain a given pendant vertex. This is because the neighbor can be taken instead and the solution will not be bigger.

So one can delete a pendant vertex and its neighbor and decrease k by one. This yields an equivalent instance.

b) Given any vertex u. It has degree at least two. If we branch on this vertex we delete u in one branch and the neighbors, at least two, in the other branch. This gives us the following recurrence for the size of the search tree:

$$t_k \leq t_{k-1} + t_{k-2}$$

The corresponding reflected characteristic polynomial is $1-z-z^2$ with the root $1/\phi$ in the unit interval, where $\phi \approx 1.618033989$ is the golden ratio. The running time is therefore $\phi^k n^{O(1)}$, which is clearly good enough.

c) Just use the branching algorithm of b) and guarantee that the graph has minimum degree two before each branch by exhaustively applying the reduction rule from a).