RWTH Aachen **SS 2013**
Lehrgebiet Theoretische Informatik Problem Set 1
Reidl—Rossmanith—Sánchez—Sikdar 23.04.2013

Parameterized Algorithms Tutorial

Tutorial Exercise T1

This exercise is concerned with the fixed parameter tractable algorithm for the Closest String problem that was discussed in class. Recall that an input to this problem consists of n strings $s_1, \ldots, s_n \in \Sigma^L$ of length L each and an integer parameter k. The question is whether there exists a string $s \in \Sigma^L$ such that $d(s, s_i) \leq k$, for all $1 \leq i \leq n$. Explain this algorithm and analyze its running time.

Solution

First observe that if there exists a string $s \in \Sigma^L$ such that $d(s, s_i) \leq k$ for all i, then for pairs of distinct strings s_i, s_j, we have

$$d(s_i, s_j) \leq 2k.$$

One can interpret this as follows. Consider the balls B_i and B_j of strings that are at Hamming distance at most k from s_i and s_j, respectively. A string s that is at distance at most k from both s_i and s_j must be in both these balls, meaning that these balls must intersect. This happens if and only if $d(s_i, s_j) \leq 2k$.

One could imagine trying to obtain a closest string s as follows. We first set $s = s_1$. Let s_i with $2 \leq i \leq n$ be the first string such that $d(s_i, s_i) > k$. Then by our observation, these two strings differ in at most $2k$ positions (if we assume that such a string exists in the first place). Given *any* set of $k + 1$ positions where s and s_i disagree, we have to make s and s_i agree in at least one of them. It is important to note that this fact holds irrespective of *which* set of $k + 1$ positions we choose. We therefore choose some set of $k + 1$ positions and create $k + 1$ instances of the problem by modifying s in the appropriate place in each of these instances. How does the parameter change? Every time we modify s, we increase the Hamming distance between s and s_1 by one. This is the crucial observation. Hence in each of these $k + 1$ recursive calls, we can modify at most $k - 1$ additional positions, since $d(s_1, s) = 1$. The size of the recursive tree is clear: each node has degree at most $k+1$ and the depth of the tree is at most k. The total size is $((k + 1)^{k+1} - 1)/k$. The branching vector of this recursion is $(k - 1, k - 1, \ldots, k - 1)$ ($k + 1$ times).

One should verify that there exists such a string s if and only if there exists at least one branch of the recursion tree which finds such a string.

Tutorial Exercise T2

Consider the Max Sat problem: given a formula φ in CNF on n variables and m clauses and an integer parameter k, decide whether there exists an assignment that satisfies at least k clauses. Show that this problem is fixed-parameter tractable.

Solution

The idea here is to branch on the two possible assignments to a variable. Since the parameter is the number of clauses to be satisfied, in order to ensure that the parameter decreases in each branch, we first take care of some special cases. If a variable occurs only positively then we set it to `TRUE`; if it occurs only negatively, we set it to `FALSE`. If this satisfies p clauses, we reset $k = k - p$. We remove all variables that are set and all clauses that are satisfied. Note that in the resulting formula φ, every variable occurs both positively and negatively. We start by selecting a variable x arbitrarily and consider the two instances: $(\varphi, x = \texttt{TRUE}, k - c_1)$ and $(\varphi, x = \texttt{FALSE}, k - c_2)$, by setting x to true in one case and false in the other. Here $c_i \geq 1$ is the number of clauses satisfied in each case. The branching factor is two and the depth of the recursion tree is at most k, the size of the recursion tree is at most $(2^{k+1} - 1)/2$. Another way of stating this is that the branching vector is $(1, 1)$ in this case. The running time is $O(2^k \cdot \text{poly}(n))$.

Tutorial Exercise T3

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 $3^k \cdot \text{poly}(|G|)$.

Homework H1

Consider the following problem: an instance of the problem consists of a finite set $U = \{1, \ldots, n\}$ and a set family $\mathcal{F} \subseteq 2^U$ such that for $S \in \mathcal{F}$, we have $|S| \leq d$, where d is a constant. You have to decide whether there exists a set $H \subseteq U$ of size at most k such that $H \cap S \neq \emptyset$ for all $S \in \mathcal{F}$. That is, the set H contains at least one element from each set of the family $\mathcal{F}$. Design a fixed-parameter algorithm for this problem with k as the parameter.

[10 points]

Solution

Since the set H intersects every set $S \in \mathcal{F}$ and since each set S has bounded size, the idea is to "branch on the elements of a set." We try to construct a set H by arbitrarily picking a set $S = \{s_1, \ldots, s_r\} \in \mathcal{F}$ and creating at most d instances:

$$(U = U - s_1, \mathcal{F} = \mathcal{F}_1, k - 1), \ldots, (U = U - s_r, \mathcal{F} = \mathcal{F}_r, k - 1),$$

where $\mathcal{F}_i$ denotes the family of all members of $\mathcal{F}$ that do not contain s_i. The interpretation of the instance $(U - s_i, \mathcal{F}_i, k - 1)$ associated ith branch is that we picked the element s_i in the solution set H. As such, we deleted this element from the universe, removed all members of $\mathcal{F}$ that were "hit" by this choice of s_i and decremented k by one. We now

recurse on each of these at most d instances. The size of the recursion tree is $(d^{k+1}-1)/d$. The algorithm reports "yes" if $k = 0$ and $\mathcal{F} = \emptyset$; otherwise it reports "no".

To show correctness, we need to show that if there exists a solution of size at most k the algorithm will find such a solution, and vice versa.

Suppose that there is indeed a solution H of size at most k, then this algorithm will find it. We induct on k. Suppose that the algorithm eliminates members in $\mathcal{F}$ in the order: $S_1, \ldots, S_m$ and let $k = 1$. Now H, being a valid solution, intersects S_1 in some element and this element is in each member of $\mathcal{F}$. The algorithm has one branch where this element is picked and therefore, for $k = 1$, the algorithm indeed obtains a solution if there exists one. Assume that the algorithm obtains a correct solution when the parameter is $k-1$. When the parameter is k, the algorithm chooses the correct element from S_1 in some branch: $(U = U - s, \mathcal{F} = \mathcal{F}_s, k-1)$. It follows that the instance $(U = U - s, \mathcal{F} = \mathcal{F}_s, k-1)$ is a yes-instance and therefore, by induction hypothesis, the algorithm obtains a correct solution H' of size at most $k-1$. Then $H' \cup s$ is a solution of size at most k for the original instance.

If the algorithm does find a solution, then this solution intersects each member of $\mathcal{F}$ and hence is indeed a solution.

Homework H2

Several problems in bioinformatics and databases can be modeled as follows. You are given a graph $G = (V, E)$ and an integer parameter k, as input. You are to delete existing edges or add new edges to the graph G so that each connected component of the resulting graph is a clique. However you can add or delete a total of at most k edges. Design a fixed-parameter algorithm for this problem with k as parameter.

[10 points]

Solution

The crucial observation here is that a graph that is a collection of disjoint cliques can be characterized by the absence of an induced path of lenth three. An induced path of length three is a path on three vertices u, v, w such that uv and vw are edges but uw is not. Therefore another way of viewing this problem is that of "destroying" induced paths oe length three. The operations that we are allowed to perform are to add edges or delete exiting edges. Given an induced path u–v–w on three vertices, we may either delete the edges uv or vw or add the new edge uw. One of these operations has to be performed and having performed one them we have a budget of at most $k-1$ operations that we may perform. This suggests a branching algorithm with branching vector $(1, 1, 1)$ with running time $O(3^k \cdot \text{poly}(|G|))$.