TECHNISCHE UNIVERSITÄT
KAISERSLAUTERN

Department of Computer Science
Sebastian Wild

Issue Date: 17.05.2017
Version: 2017-05-17 12:15

4th Exercise sheet for Advanced Algorithmics, Summer 17

Hand In: *Until Wednesday, 24.05.2017, 12:00 am, hand-in box in 48-4 or via email.*

Problem 7

20 + 25 points

We consider the Closest-String problem again.

a) The fixed-parameter algorithm `closestStringFpt` (from class) for the p-Closest-String problem solves the search variant of the problem: If a consensus string exists with at most k mismatches to any string, it finds one (and if not, it announces this instead).

   Assume we are given a Yes-instance, does `closestStringFpt` return an *optimal* solution, i.e., a string that minimizes the maximal number of mismatches to any of the input strings? Prove you claim.

b) We consider a restricted version of the Closest-String problem. We require k to be optimal, and furthermore that there are input words s_i and s_j with $d_H(s_i, s_j) = 2k$.

   Show that this problem can be solved by an algorithm with runtime in $\mathcal{O}(4^k \cdot mL)$.

Problem 8

30 points

In class, we have seen that the *p-variable*-3SAT problem, i.e., the 3CNF satisfiability problem parametrized by the number k of variables trivially satisfies *p-variable*-3SAT $\in \mathcal{FPT}$ since the $\mathcal{O}(2^k n)$ brute-force algorithm is an fpt-algorithm for this problem. But that algorithm actually solves the general SAT problem – indeed, even the satisfiability problem for boolean *circuits*.

Show that we can indeed exploit the special structure of 3SAT to improve upon this by designing an fpt-algorithm for *p-variable*-3SAT with search space $\mathcal{O}(\alpha^k)$ for some $\alpha < 2$.

Hint: Find a *disjoint* decomposition of the search space of all assignments that satisfy the first clause $l_1 \vee l_2 \vee l_3$.

Problem 9

30 + 20 points

Consider the following problem:

Input: A graph $G = (V, E)$ and $k \in \mathbb{N}$.

Question: Can we transform G, by deleting or adding at most k edges, into a graph that consists of a disjoint union of disconnected cliques (of arbitrary sizes)?

a) Consider the following algorithm for the problem:

Given $G = (V, E)$ and $k \in \mathbb{N}$, do the following:

1) If G is already a union of disjoint cliques, we are done: report "yes" and return.
2) Otherwise, if $k \leq 0$ we can not find a solution in this branch. Report "no" and return.
3) Otherwise, identify $u, v, w \in V$ with $\{u, v\} \in E$ and $\{u, w\} \in E$, but $\{v, w\} \notin E$. Call the algorithm on three instances $((V, E'), k')$ defined by
   (B1) $E' = E \setminus \{\{u, v\}\}$ and $k' = k - 1$,
   (B2) $E' = E \setminus \{\{u, w\}\}$ and $k' = k - 1$ and
   (B3) $E' = E \cup \{\{v, w\}\}$ and $k' = k - 1$,
   respectively. Report "yes" if at least one of the recursive calls reports such, and "no" otherwise.

Show that this algorithm solves the given problem and give a non-trivial upper bound on its runtime as $\mathcal{O}$-class.

b) In order to improve the algorithm given in a), we can distinguish three cases that can occur for any chosen *"conflict triple"* (u, v, w) as specified in step 3:

(C1) Vertices v and w do not share a common neighbour, that is

$$\forall x \in V \setminus \{u\} : \{v, x\} \notin E \vee \{w, x\} \notin E \,.$$

(C2) Vertices v and w have a common neighbour $x \neq u$ and $\{u, x\} \in E$.

(C3) Vertices v and w have a common neighbour $x \neq u$ and $\{u, x\} \notin E$.

It is possible to show that we can restrict ourselves to the following branching.

- In case (C1), we only have to take branches (B1) and (B2).
- In case (C2), we have to execute (B1) and refine the other two branches further so that each has one subbranch that deletes one, and another that deletes two additional edges.

- In case (C3), we have to execute (B1). Branch (B2) can be refined into one branch that deletes an additional edge, and one that adds and deletes one edge, respectively. Branch (B3) can be refined into one branch that deletes two additional edges, and one that adds one edge, respectively.

Give a non-trivial worst-case bound on the size of the search tree as explored by this improved algorithm!