Karlsruher Institut für Technologie

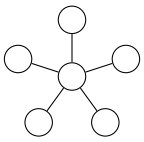
Solution to Exercise Sheet 5

Submission due by July 2, 2026

Problem 1: Stars

8 points

A *5-star* is the following graph:



Given an undirected graph $G = (V, E)$ and a parameter k, we want to decide whether G contains at least k vertex-disjoint induced 5-stars. Use *color-coding* to show that this problem is in FPT.

Solution 1:

We first consider the Colorful Stars problem. Here, in addition to a given graph we are also given a vertex coloring, and for the stars it must hold that all $6k$ star vertices have pairwise distinct colors. If we find an FPT algorithm for this problem, then using a $(n,6k)$-perfect family of hash functions we also obtain an FPT algorithm for the original problem.

Because in the Colorful Stars problem all star vertices must have different colors, each star uses 6 distinct colors and the colors of two different stars are also all distinct. Hence we try all possible assignments of colors to stars. Since there are only $6k$ colors and k stars, the number of combinations depends only on k. A rough estimate yields $O((6k)!)$. For each assignment we examine, for every one of the k stars with its associated color set, all 6-tuples of vertices ($O(n^6)$ many) and check whether the six vertices are correctly colored and induce a subgraph isomorphic to a 5-star. If for some assignment we find suitable vertex sets for all k stars, we output "yes", otherwise "no". Disjointness of the stars is guaranteed by the coloring.

Running time: A $(n,6k)$-perfect family of hash functions can be constructed in time $O(6.4^k \cdot n \cdot \log^2 n)$. Overall, we get an FPT algorithm with runtime

$$\mathcal{O}\big(6.4^k \cdot (6k)! \cdot n^7 \cdot \log^2 n\big).$$

Problem 2: Longest Cycle

14 points

In the problem Longest Cycle, we have given a graph $G = (V, E)$ and a parameter k and we have to decide whether G contains a cycle of length *at least* k. Provide an FPT algorithm for this problem.

Hint: Note that there are graphs that do not contain a cycle of length exactly k, but nevertheless contain a cycle of length at least k.

Solution 2:

In the following, we outline an FPT algorithm that, for a graph $G = (V,E)$ and a parameter k, decides whether G contains a cycle of length at least k.

We first develop an algorithm that decides whether there is a cycle of length exactly k. For this we examine every edge $\{u,v\} \in E$ and check whether there exists a path between u and v of length exactly k. This can be done with the color-coding based FPT algorithm for LongestPath from the lecture, by attaching leaf vertices u' and v' to u and v, respectively, and giving u' and v' unique colors so that any colorful path of length $k + 2$ must use both u' and v'.

We then proceed as follows: for each i between k and $2k$ we use the above algorithm to test whether G contains a cycle of length i. If there exists one, we answer "yes". Otherwise, we contract an arbitrary edge and continue the search on the resulting graph, repeating this process until the graph has only constantly many edges left. This is correct because contracting an edge can reduce the length of the longest cycle by at most a factor of two. Consequently, if G contains a cycle of length $\ell > 2k$, repeated edge contractions will eventually produce a graph in which the longest cycle length lies between k and $2k$ (or another sufficiently long cycle is discovered).

Problem 3: Colors and Kernels

4 + 14 = 18 points

This exercise builds on lecture 8 (June 22).

Show that the following problems do not admit polynomial kernels (when parameterized by the respective k), unless NP $\subseteq$ coNP/ poly.

Part (a) Colorful Graph Motif: Given a graph $G = (V,E)$ and a coloring of its vertices into k colors (not necessarily a proper coloring), does G have a colorful connected subgraph H of size exactly k?

You can use without proof that Colorful Graph Motif is NP-complete.

Solution 3a:

To rule out the existence of a polynomial kernel under the assumption that NP $\not\subseteq$ coNP/poly, we need to construct a cross-composition from an NP-hard problem to Colorful Graph Motif. Since the problem itself is NP-hard, we use it as the starting point of the composition. That is, we take t

"similar" input instances of Colorful Graph Motif $x_1 = (G_1,k), x_2 = (G_2,k), \ldots, x_t = (G_t,k)$. For the "similarity" of the input instances, we use the same number of colors k in all graphs G_i.

We transform them into one instance of Colorful Graph Motif $y = (G,k')$ by defining H as the disjoint union of the graphs G_i and setting k' to k. This can be done in time $\mathcal{O}(\max_{i\in[t]} |x_i| \cdot t)$. Also it simulates a logical OR of the input: if one G_i has a colorful connected subgraph of size k, then this subgraph is also contained in H. And since the parts of H are disjoint, if H contains a colorful connected subgraph of size k, it lies completely within one graph G_i. We can assume that the parameter k is at most n, so in particular polynomial in $\max_{i\in[t]} |x_i| + \log t$, as required.

Part (b) Colorful Left-Right Dominating Set: Given a bipartite graph G with sides of the bipartition L and R (called the *left* and *right* vertices, respectively), and a coloring of L into ℓ colors. Is there a colorful set $X \subseteq L$ of exactly ℓ left vertices that together dominate all right vertices, i.e., $N(X) = R$? The parameter we consider is $k = |R| + \ell$.

You can use without proof that Colorful Left-Right Dominating Set is NP-complete.

Hint: Construct an instance selector that consists of right vertices.

Solution 3b:

We construct a cross-composition from Colorful Left-Right Dominating Set to itself. To begin with, we let t instances $x_1, \ldots, x_t$ be given. We let the instances be "similar" by requiring the same number of colors ℓ and the same number of right vertices r. For an instance x_i, we denote its graph by G_i with sides of the bipartition L_i and R_i.

We construct a bipartite graph H with the same amount of colors as follows. First, we let the left side of H be the disjoint union of the left sides L_i of the input instances, i.e. $L = \dot{\bigcup}_{i=1}^{m} L_i$. For the right side of H, we identify all R_i into one set R of size r (note that we can do this since all R_i have the same size). We copy all edges from the input instances over to graph H, using instead of the right vertices in R_i the respective identified vertices in R. Now, the constructed graph is not yet the logical OR of the input: There could be a solution in H that combines vertices from different sets L_i, although no input instance has a solution on its own.

We fix this by constructing an instance selector gadget, which will consist of additional right vertices that have to be dominated. The idea is to ensure that once we picked one vertex from a set L_i, there has to exist a set of right vertices that can only be entirely dominated if we choose the other $\ell - 1$ vertices from the set L_i as well. For this, let $L_i^1, L_i^2, \ldots, L_i^\ell$ denote the color classes of vertex set L_i, i.e. L_i^c is the set of all vertices of color c in L_i. Furthermore, let s be the length of the binary representation of t. The instance selector is constructed as follows.

For each bit position $\alpha \in [s]$ in the binary representation of t, we create the two vertices d_α^0 and d_α^1, representing the possible bits 0 and 1 at position α. We denote the set of all these vertices as D. Then, we make $\ell - 1$ copies of D (one copy for every color except the first) and call them

$D_2, D_3, \ldots, D_\ell$ and denote the copies of d^0_α and d^1_α in the set D_c by $d^0_{\alpha,c}$ and $d^1_{\alpha,c}$. We add all sets D_i to the set R, i.e. we get the set $R' = R \cup (D_2 \cup D_3 \cup \ldots \cup D_\ell)$.

Now, the idea is to connect all left vertices from set L_i of the first color with the vertices in $D_2 \cup D_3 \cup \ldots \cup D_\ell$ which correspond to the binary representation of i; for every other color $c \neq 1$, we connect all left vertices of color c in L_i with all vertices in D_c that correspond to the complement of the binary representation of i. As we will argue below, this construction has the effect that the set D_c can only be dominated by exactly two vertices of two different colors if they lie in the same set L_i. The construction can be formalized as follows.

For an index $i \in [t]$, let its binary representation be denoted by $b^{(i)}_1 b^{(i)}_2 \ldots b^{(i)}_s$. We define the sets of corresponding right vertices $d_c(i) := \{d^{b^{(i)}_\alpha}_{\alpha,c} \mid \alpha \in [s]\} \subset D_c$ for every color $c \in \{2, \ldots, \ell\}$. For every i, we connect all left vertices of the first color, i.e. all vertices in L^1_i, to all (right) vertices in the set $d_c(i)$ for each color $c \in \{2, \ldots, \ell\}$. Then, for each color $c \neq 1$, we connect all vertices in L^c_i to all vertices in $D_c \setminus d_c(i)$.

We show that the constructed instance is equivalent to the logical OR of the input instances. Assume there is an index $i \in [t]$, such that the input instance x_i has a solution $X \subseteq L_i$. We take the same vertices in the graph H to obtain a solution there: the vertices in R are dominated since X is a solution for x_i. And for each color c, the set D_c is dominated by X because all vertices in X are in the same set L_i and, hence, the color-1 vertex in X dominates all vertices in $d_c(i)$ and the color-c-vertex in X dominates $D_c \setminus d_c(i)$.

For the other direction, let $X \subseteq L$ be a colorful set of size ℓ that dominates all vertices in R'. We claim that X is entirely contained in a left side L_i of an input instance x_i. To show this, let $c \in \{2, \ldots, \ell\}$ and assume there are two vertices v_1,$v_c \in L$ with colors 1 and c such that $v_1 \in L_i$ and $v_c \in L_j$. We need to show that then $i = j$. Consider the right vertices in the set D_c of the instance selector. By definition, v_1 dominates the set $d_c(i)$ and v_c dominates the set $D_c \setminus d_c(j)$. Since all remaining solution vertices in $X \setminus \{v_1, v_c\}$ are not adjacent to any vertex in D_c, it holds $d_c(i) \cup (D_c \setminus d_c(j)) = D_c$ since X dominates the entire set D_c. Since $|d_c(i)| = |d_c(j)| = |D_c|/2$ (every bit is either 0 or 1 in the bit representation), this implies that the sets $d_c(i)$ and $D_c \setminus d_c(j)$ are disjoint. This means that $d_c(i) = d_c(j)$ and hence, $i = j$. Therefore, X is contained in L_i. Since the set X in L_i also dominates the set R_i and is colorful, we get that it is a solution for instance x_i.

The reduction runs polynomial in $\sum_{i=1}^{t} |x_i|$ and our new parameter is $|R'| + \ell = |R| + \sum_{c=2}^{\ell} |D_c| + \ell = |R_1| + (\ell - 1) \cdot 2s + \ell$, which is polynomial in $\max_{i \in [t]} |x_i| + \log t$.

Problem 4: Closest String

10 bonus points

In this task, you again have to implement a program that solves the Closest String problem (in a simplified variant). We are given k strings of length n. The goal is to find the smallest D such that there exists a string s^* whose Hamming distance to each input string is at most D.

Solve the problem by means of an ILP as presented in lecture 6. You may introduce further optimizations. In the PDF submission, describe which modifications you made to the ILP formulation from the lecture and any additional insights that helped you solve the problem. Also list for each instance the smallest D for which you found a solution.

Additionally submit the source code and your obtained solutions (in the format described below) as a ZIP archive. You receive a point for every instance you solve.

Input file format: The first line contains k, the number of strings. The next k lines each contain a string. All strings have the same length and consist only of the lowercase letters `a-z`.

Output file format: One line with a string s^* that achieves the minimal Hamming distance to the given strings.

Hint: With the file `validator.py` you can compute the Hamming distance of your solution to the input strings; it reports the maximum over all distances.

Hint: This task requires an ILP solver. Many languages and solvers are available, but `Gurobi` is among the best. To use it, however, you need a license. You can obtain a free academic license with your university e-mail address.

Solution 4:

String Instance	**Minimum** D
Instance 0	72
Instance 1	731
Instance 2	871
Instance 3	8380
Instance 4	86835