

CS 473 (Spring 2025)

Homework 9 (due Apr 24 Thu 10am)

Instructions: As in previous homeworks.

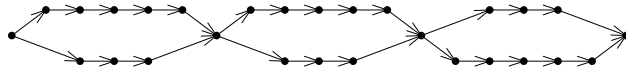
Problem 9.1: Consider the following problem COLORFUL-PATH:

Input: a directed acyclic graph (dag) $G = (V, E)$ with n vertices and m edges, and vertices $s, t \in V$, and a color $c(e) \in \{1, \dots, M\}$ for each edge $e \in E$, and a number ℓ .

Output: yes iff there exists a path from s to t that encounters at least ℓ *distinct* colors.

Prove that COLORFUL-PATH is NP-hard, by giving a polynomial-time reduction from 3-SAT to COLORFUL-PATH. Remember to describe precisely your construction of an input to COLORFUL-PATH from any given input to 3-SAT (a 3CNF formula – note that we are not given a satisfying assignment itself!), and carefully prove correctness of your reduction.

(Hint: for your construction of the dag G , try something like the picture below... Describe how to assign colors to the edges along these paths.)

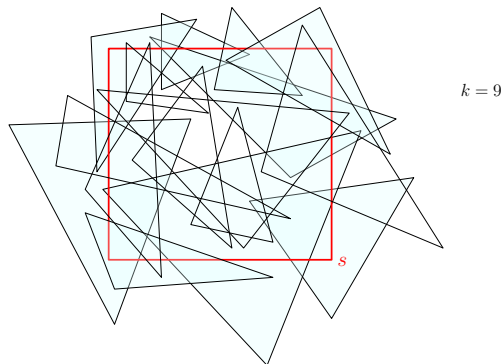


Problem 9.2: Consider the following problem COVERING-SQUARE-BY-TRIANGLES:

Input: a set T of n triangles in 2D, a square s , and an integer k .

Output: yes iff there exists a subset $Q \subseteq T$ of k triangles such that the union of Q covers the entire boundary of s .

(Note: we don't require the union of Q to cover the interior of s . Also, the variant of the problem of covering just one side of s , rather than the entire boundary of s , is easier and can actually be solved in polynomial time by a greedy algorithm.)



Prove that COVERING-SQUARE-BY-TRIANGLES is NP-hard.

You may use the fact that VERTEX-COVER is NP-hard even when restricted to graphs with maximum degree 3. You may also use the fact that any graph with maximum degree d is $(d + 1)$ -edge-colorable, i.e., we can assign a color from $\{1, \dots, d + 1\}$ to each edge, so that no two adjacent edges have the same color; such a coloring can be found in polynomial time.

As always, describe the construction of your reduction precisely, and carefully prove correctness of your reduction.

(Hint: map each edge of the input graph to a tiny interval along the boundary of $s \dots$)

Problem 9.3: As we all know, an *interval* is a set $[a, b]$ containing all real numbers between a and b . Define a *circular interval* to be either an interval $[a, b]$ (if $a \leq b$), or the union of two unbounded intervals $[a, \infty) \cup (-\infty, b]$ (if $a > b$). (In other words, we allow the real line to “wrap around”.)

Consider the following problem CIRCULAR-INTERVAL-SELECTION:

Input: N circular intervals I_1, I_2, \dots, I_N with positive integer weights w_1, \dots, w_N .

Output: a subset $S \subseteq \{1, \dots, N\}$ maximizing total weight $\sum_{i \in S} w_i$ such that for every $i, j \in S$ ($i \neq j$), the circular intervals I_i and I_j are disjoint.

It is possible to solve the CIRCULAR-INTERVAL-SELECTION problem in $O(N^2)$ time by dynamic programming (you don’t have to give such an algorithm). In contrast, without circularity, the problem can be solved in $O(N \log N)$ time. In this question, you will provide evidence on why the circular version of the problem is more difficult.

To this end, consider the following known graph problem called MIN-WEIGHT-TRIANGLE:

Input: a directed graph $G = (V, E)$ with n vertices, where each edge (u, v) has a positive integer weight $w(u, v)$.

Output: a cycle of length 3 in G with the minimum total weight, i.e., $v_i, v_j, v_k \in V$ with $(v_i, v_j), (v_j, v_k), (v_k, v_i) \in E$ minimizing $w(v_i, v_j) + w(v_j, v_k) + w(v_k, v_i)$.

It has been hypothesized that no algorithm can solve MIN-WEIGHT-TRIANGLE in $O(n^{2.999})$ time (in other words, the obvious cubic-time, brute-force algorithm is close to be the best possible).

Prove that under this hypothesis, no algorithm can solve CIRCULAR-INTERVAL-SELECTION in $O(N^{1.499})$ time.

(Hint: map each edge (v_i, v_j) of the input graph to three circular intervals $[i, M + j - 1]$, $[M + i, 2M + j - 1]$, and $[2M + i, \infty) \cup (-\infty, j - 1]$ for some M . But how to assign weights to these circular intervals?)