CS 580-Spring 2019
Instructor: Jeremiah Blocki
TAs: Hamidreza Amini Khorasgani, Akash Kumar
I collaborated with (...). I affirm that I wrote the solutions in my own words and that I understand the solutions I am submitting.
Name:

Homework 6

Due Date: April 23, 2019 at 11:59 PM on Gradescope.

Question 1

Given a graph G a triangle consists of a triple $\{u,v,w\}$ such that all three edges $\{u,v\},\{v,w\},\{u,w\} \in E(G)$ are in the graph. Let $T(G) = \{\{u,v,w\} ~:~ \{u,v\},\{v,w\},\{u,w\} \in E(G)\}$ denote the set of all triangles in G. We say that a subset $S \subseteq V(G)$ is a triangle cover if for each triangle $t = \{u,v,w\} \in T(G)$ we have $t \cap S \neq \emptyset$ i.e., S contains *at least* one of the three nodes u,v,w. In the weighted version of the triangle cover problem we have a weight $\mathtt{w}(v) \geq 0$ associated with each node v and we define $\mathtt{w}(S) \doteq \sum_{v \in S} \mathtt{w}(v)$. The goal is find a triangle cover with minimum weight.

1. Consider the unweighted version of the problem in which $\mathtt{w}(v) = 1$ for every node v. Find an algorithm that runs in time $T(n,k) = O\left(3^k \times n^{10}\right)$ where k is the size of the minimum triangle cover. (**Note:** You might find a solution that is faster than the given bound. The key requirement is that the dependence on n must be polynomial).
2. Find a 3-approximation algorithm for the weighted triangle cover problem. Your algorithm should run in polynomial time and should output a triangle cover S with the guarantee that $\mathtt{w}(S) \leq 3 \times \mathtt{w}(S^*)$ where S^* denotes the optimal triangle cover.

Question 2

The Geography on Graphs Game is defined as follows: Given a directed graph $G = (V,E)$ and a start node s, two players alternate turns by following, if possible, an edge out of the current node to an unvisited node. The player who loses is the first one who cannot move to a node that hasnt been visited earlier. In other words, a player loses if it is his turn, the game is currently at node v and for each directed edges of the form $(v,w) \in E$ the node w has already been visited. The Tripartite Geography on Graphs Game is defined in the same way except that the input graph G must satisfy the property that the nodes $V(G)$ can be partitioned into sets A,B,C and $E(G) \subseteq (A \times B) \cup (C \times A) \cup (B \times C)$ i.e. any edge starting at A ends in B, any edge starting in B ends in C and any edge starting in C ends in A. The decision version of the problem is to decide whether or not player 1 can force a win with optimal play.

1. Suppose that the graph G has no directed cycles. Show that the Geography on a Graph decision problem is solvable in polynomial time.
2. Prove that the Tripartite Geography on Graph decision problem is PSPACE-Complete. You may assume that the regular Geography on Graphs Game is PSPACE-Hard.

Question 3

It is NP-Hard to find a 3-coloring of a G (or determine that no 3-coloring exists). In this problem we will design an polynomial time algorithm that is *guaranteed* to either output a legal $4\sqrt{n}$-coloring of the graph G or determine that G is not 3-colorable.

1. (Warmup) Begin by showing that any graph with maximum degree d can be colored using $d+1$ colors. *(Hint: Go Greedy!)*

2. Let v denote a vertex in G with $deg(v) \geq \sqrt{n}$. Call v a heavy vertex. Show that it is either the case that (1) the subgraph $G[N(v)]$ induced on $N(v) = \{u : u \text{ is adjacent to } v\}$ is bipartite, or (2) the graph G is not 3-colorable.

3. Using the previous observation develop an algorithm which outputs a $4\sqrt{n}$-coloring (or determines that G is not 3-colorable). *(Hint: As a first step try to eliminate all heavy vertices.)*

Bonus (5 points) Extend the previous ideas to design an polynomial time algorithm which is *guaranteed* to either output a legal $O\left(n^{1-1/k}\right)$-coloring of the graph G or determine that G is not k-colorable for any constant $k = O(1)$.

Question 4

The difficulty in 3-SAT comes from the fact that there are 2^n possible assignments to the input variables $x_1, x_2, \ldots, x_n$, and there's no apparent way to search this space in polynomial time. This intuitive picture, however, might create the misleading impression that the fastest algorithms for 3-SAT actually require time 2^n. In fact, though it's somewhat counterintuitive when you first hear it, there are algorithms for 3-SAT that run in significantly less than 2^n time in the worst case; in other words, they determine whether there's a satisfying assignment in less time than it would take to enumerate all possible settings of the variables.

Here well develop one such algorithm, which solves instances of 3-SAT in $O(p(n) \cdot (\sqrt{3})^n)$ time for some polynomial $p(n)$. Note that the main term in this running time is $(\sqrt{3})^n$, which is bounded by 1.74^n.

(a) For a truth assignment Φ for the variables $x_1, x_2, \ldots, x_n$, we use $\Phi(x_i)$ to denote the value assigned by Φ to x_i. (This can be either 0 or 1.) If Φ and Φ' are each truth assignments, we define the distance between Φ and Φ' to be the number of variables x_i for which they assign different values, and we denote this distance by $d(\Phi, \Phi')$. In other words, $d(\Phi, \Phi') = |\{i : \Phi(x_i) \neq \Phi'(x_i)\}|$.

A basic building block for our algorithm will be the ability to answer the following kind of question: Given a truth assignment Φ and a distance d, we'd like to know whether there exists a satisfying assignment Φ' such that the distance from Φ to Φ' is at most d. Consider the following algorithm, $Explore(\Phi, d)$, that attempts to answer this question.

Prove that $Explore(\Phi, d)$ returns "yes" if and only if there exists a satisfying assignment Φ' such that the distance from Φ to Φ' is at most d. Also, give an analysis of the running time of $Explore(\Phi, d)$ as a function of n and d.

Algorithm 1 Explore(Φ, d)

```
if $\Phi$ is a satisfying assignment then
    return "yes"
else if if d = 0 then
    return "no"
else
    Let $C_i$ be a clause that is not satisfied by $\Phi$ (i.e., all three terms in $C_i$ evaluate to false)
    Let $\Phi_1$ denote the assignment obtained from $\Phi$ by taking the variable that occurs in the first term of clause $C_i$ and inverting its assigned value
    Define $\Phi_2$ and $\Phi_3$ analogously in terms of the second and third terms of the clause $C_i$
    Recursively invoke: $Explore(\Phi_1, d-1), Explore(\Phi_2, d-1), Explore(\Phi_3, d-1)$
    if any of these three calls return "yes" then
        return "yes"
    else
        return "no"
    end if
end if
```

(b) Clearly any two assignments Φ and Φ' have distance at most n from each other, so one way to solve the given instance of 3-SAT would be to pick an arbitrary starting assignment Φ and then run *Explore*(Φ, n). However, this will not give us the running time we want.

Instead, we will need to make several calls to *Explore*, from different starting points Φ, and search each time out to more limited distances. Describe how to do this in such a way that you can solve the instance of 3-SAT in a running time of only $O(p(n) \cdot (\sqrt{3})^n)$.

Bonus (10 points)

The Hitting Set problem is similar to the Set Cover problem. An instance is given by a list $S_1, \ldots, S_n \subseteq 1, , m = U$ of n subsets of a universe U of size $|U| = m$. In the Set Cover problem we want to find a minimum size subset of sets $S \subseteq 1, , n$ which covers the universe U i.e., $\bigcup_{i \in S} S_i = U$. In the Hitting Set problem we are given an absolute upper bound k on the number of sets in S and we are asked to *maximize* the number of elements covered by S i.e., $|\texttt{Covered}(S)|$ where $\texttt{Covered}(S) = \bigcup_{i \in S} S_i$. In the Weighted Hitting Set problem there is a weight $\texttt{w}(u)$ for each item $u \in U$ we are asked to find S with $|S| \le k$ maximizing $\texttt{score}(S) = \sum_{x \in \texttt{Covered}(S)} \texttt{w}(x)$. The search version of the Weighted Hitting Set problem is NP-Hard (you dont need to prove this).

1. Let $V^* = \texttt{Covered}(S^*)$ denote the items covered by the optimal solution the Weighted Hitting Set problem$|S^*| \le k$. Given another hitting set solution $P \subseteq \{1, \ldots, n\}$ we define $\textbf{Uncovered}(V^*, P) \doteq V^* \setminus \texttt{Covered}(P)$ to be the set of items covered by S^* but not by P. We let $uw_P = \sum_{x \in \textbf{Uncovered}(V^*, P)} \texttt{w}(x)$ denote the total weight of uncovered

elements in V^*. Show that for any subset $P \subsetneq \{1, \dots, n\}$ there exists some $j \notin P$ s.t. $\sum_{i \in S_j \cap \mathbf{Uncovered}(V^*, P)} \mathtt{w}(i) \geq \frac{uw_P}{k}$.

2. Develop a greedy algorithm that always returns a weighted hitting set solution S such that $\mathtt{score}(S) \geq (1 - 1/e) \times \mathtt{score}(S^*)$. (**Hint:** Go greedy! You may use without proof the fact that $(1 - 1/k)^k <= e^{-1}$ for all integers $k, x > 0$.)