CS120: Intro. to Algorithms and their Limitations Prof. Salil Vadhan

Problem Set 7

Harvard SEAS - Fall 2021 *Due: Wed Nov. 9, 2022 (11:59pm)*

Your name:
Collaborators:
No. of late days used on previous psets:
No. of late days used after including this pset:

The purpose of this problem set is to develop skills in implementing graph algorithms, appreciate the impact of different kinds of worst-case exponential algorithms in practice, and practice reducing problems to SAT.

1. (Another coloring algorithm) In the Github repository for PS7, we have given you basic data structures for graphs (in adjacency list representation) and colorings, an implementation of the coloring algorithm from ps5, and a variety of test cases (graphs) for coloring algorithms. For Windows users, use this Google Colab file to run your code.

   (a) Implement the reduction from 3-coloring to SAT given in class in the function `sat_3_coloring`, producing an input that can be fed into the SAT Solver Glucose, and verify its correctness by running `python3 -m ps7_tests 3`.

   (b) Compare the efficiency of Exhaustive-Search 3-coloring, the $O(1.89^n)$-time BFS-based algorithm for 3-coloring from problem set 5 (feel free to use the staff solution or your own implementations from problem set 5), and your implementation from Part **??** using `ps7_experiments`. In the experiments file, we've provided code to generate two types of graphs (lines of rings and clusters of independent sets) and some new specific graphs. For each of those types of graphs, how many of the given instances, if any, can each algorithm solve within 10 seconds (same time limit as problem set 5)? You should fill out the table and briefly discuss and try to explain your findings.

Algorithm	Exhaustive	ISET BFS	SAT Color
# Solvable Ring Instances # Solvable Cluster Instances # Solvable Other Graphs			

   (c) (optional[1]) Find a graph G such that Glucose takes more than 1 second to solve the SAT instance to which the 3-colorability of G was reduced in part a, and n is as small as you can make it. Describe your approach to finding such a G.

2. (Resolution) Use the algorithm `ResolutionInOrder` that we saw in Lecture 16 to decide the satisfiability of the following formulas, and use the algorithm `ExtractAssignment` to obtain a satisfying assignment for any that are satisfiable. (Please make sure to follow both algorithms *exactly*, including the order in which the clauses are processed. A correct final solution that does not show all of the intermediate steps of both algorithms will not receive full score.)

[1] This problem is meant to be done based on your enjoyment/interest and only if you have time. It won't make a difference between N, L, R-, and R grades, and course staff will deprioritize questions about this problem at office hours and on Ed.

(a) $\varphi(x_0, x_1, x_2, x_3) = (x_2 \vee \neg x_1) \wedge (x_3 \vee x_1) \wedge (x_0 \vee x_1) \wedge (\neg x_3) \wedge (\neg x_1 \vee \neg x_2)$.

(b) $\varphi(x_0, x_1, x_2) = (x_0) \wedge (\neg x_0 \vee x_1) \wedge (\neg x_1 \vee \neg x_2) \wedge (x_2)$.

3. (Reductions to SAT) Consider the following problem. From Harvard's n CS concentrators (e.g. $n = 400$), we want to form a team of exactly k students (e.g. $k = 30$) to represent Harvard in a new programming competition. The programming competition problems may require expertise in any of m different programming languages (e.g. $m = 100$). But each of the CS concentrators only knows a few different programming languages, with a different set per person. So we want to try to find k Harvard CS concentrators such that between them, they know all m languages. Formally, we want to solve the following computational problem:

Input	**:** A finite set $L = \{\ell_0, \ldots, \ell_{m-1}\}$ of programming languages; a finite set $S = \{s_0, \ldots, s_{n-1}\}$ of students; for each student $s \in S$, a set $K(s) \subseteq L$ of languages that student s knows; and a team size $k \in \mathbb{N}$
Output	**:** A team $T \subseteq S$ of size k that collectively knows all of the programming languages in L (i.e. $\bigcup_{s \in T} K(s) = L$), if one exists

Computational Problem ProgrammingTeam

(a) Show that ProgrammingTeam can be efficiently reduced to solving a SAT instance on kn variables and $m + O(kn^2)$ clauses. Prove the correctness of your reduction and analyze its runtime.

(b) (optional[1]) Come up with a more efficient reduction that produces a SAT instance with only n variables and $m+O(kn)$ clauses (or even $m+O(n \log k)$ clauses). (Hint: something like $\psi_{n,k}$ or τ_ℓ formulas from the Section 7 problem on IndependentSet$\leq$ SAT might be useful.)