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: Aika Aldayarova
Collaborators: Yana Hubyak, Talya Li, Liya Jin
No. of late days used on previous psets: 5
No. of late days used after including this pset: 7

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 1a 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	0	2	6 (all)
# Solvable Cluster Instances	4	4	6 (all)
# Solvable Other Graphs	0	0	1

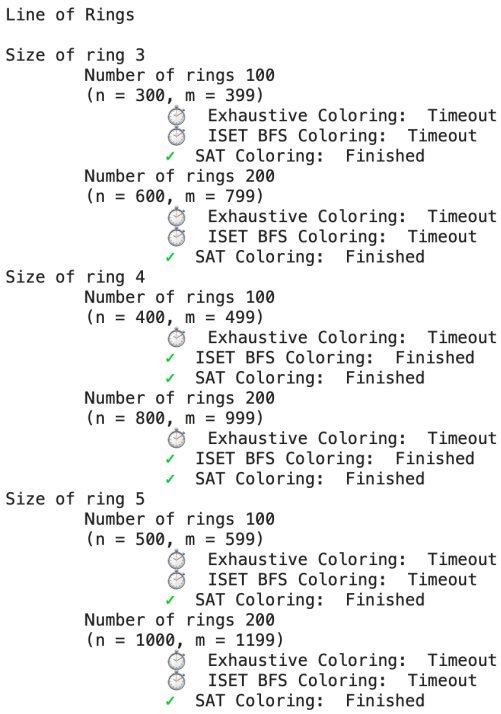

Figure 1: Output of the experiments file displaying the algorithms' performances on ring instances

```
Randomized Cluster Connections (Semi Independent Sets)

        Number of clusters 2
                Size of cluster 2
                (n = 4, m = 2)
                        ✓  Exhaustive Coloring:  Finished
                        ✓  ISET BFS Coloring:  Finished
                        ✓  SAT Coloring:  Finished
                Size of cluster 14
                (n = 28, m = 88)
                        ✓  Exhaustive Coloring:  Finished
                        ✓  ISET BFS Coloring:  Finished
                        ✓  SAT Coloring:  Finished
        Number of clusters 3
                Size of cluster 2
                (n = 6, m = 10)
                        ✓  Exhaustive Coloring:  Finished
                        ✓  ISET BFS Coloring:  Finished
                        ✓  SAT Coloring:  Finished
                Size of cluster 14
                (n = 42, m = 303)
                        ⏱  Exhaustive Coloring:  Timeout
                        ⏱  ISET BFS Coloring:  Timeout
                        ✓  SAT Coloring:  Finished
        Number of clusters 4
                Size of cluster 2
                (n = 8, m = 10)
                        ✓  Exhaustive Coloring:  Finished
                        ✓  ISET BFS Coloring:  Finished
                        ✓  SAT Coloring:  Finished
                Size of cluster 14
                (n = 56, m = 595)
                        ⏱  Exhaustive Coloring:  Timeout
                        ⏱  ISET BFS Coloring:  Timeout
                        ✓  SAT Coloring:  Finished
```

Figure 2: Output of the experiments file displaying the algorithms' performances on cluster instances

```
Hard instances

(n = 5000)
                ⏱  Exhaustive Coloring:  Timeout
                ⏱  ISET BFS Coloring:  Timeout
                ✓  SAT Coloring:  Finished
(n = 10000)
                ⏱  Exhaustive Coloring:  Timeout
                ⏱  ISET BFS Coloring:  Timeout
                ⏱  SAT Coloring:  Timeout
(n = 30000)
                ⏱  Exhaustive Coloring:  Timeout
                ⏱  ISET BFS Coloring:  Timeout
                ⏱  SAT Coloring:  Timeout
```

Figure 3: Output of the experiments file displaying the algorithms' performances on hard instances

Answer: For every type of graph, the 3-coloring to SAT reduction algorithm outperformed the other two algorithms (the exhaustive and ISET + BFS algorithms performed similarly, though ISET + BFS should theoretically be slightly faster than exhaustive). Let's examine the algorithms' runtimes to see if the theory supports practice. The exhaustive algorithm has a runtime of $O(n!)$ where n is the number of vertices (per lecture 10 notes). The ISET + BFS algorithm has a runtime of $O(1.89^n)$ where n is the number of vertices (per our SRE activity). Finally, the k-coloring reduction algorithm has a runtime of $O(n + km)$ where m is the number of edges and n is the number of vertices (per lecture 15 notes). By looking at the three algorithms' worst-case runtimes, one can see why the 3-coloring to SAT reduction algorithm performed the best – it has the

fastest runtime of all.

(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.)

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

   Answer: This formula is not satisfiable per the definition provided in lecture notes 14 and 15 because the output of resolving C_3 with C_{10} is 0. Thus, no assignment extraction is possible. The work for this problem is shown below.

[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.

② (a) $\varphi(x_0,x_1,x_2,x_3) = \underbrace{(x_2 \vee \neg x_1)}_{C_0} \wedge \underbrace{(x_3 \vee x_1)}_{C_1} \wedge \underbrace{(x_0 \vee x_1)}_{C_2} \wedge \underbrace{(\neg x_3)}_{C_3} \wedge \underbrace{(\neg x_1 \vee \neg x_2)}_{C_4}$

$C_0 \diamond C_1 = (x_2 \vee x_3) = C_5$
$C_0 \diamond C_2 = (x_2 \vee x_0) = C_6$
$C_0 \diamond C_3 = 1$
$C_0 \diamond C_4 = (\neg x_1) = C_7$

$C_1 \diamond C_2 = \not{1}$
$C_1 \diamond C_3 = (x_1) = C_8$
$C_1 \diamond C_4 = (x_3 \vee \neg x_2) = C_9$
$C_1 \diamond C_6 = \not{1}$
$C_1 \diamond C_7 = (x_3) = C_{10}$

$C_2 \diamond C_3 = \not{1}$
$C_2 \diamond C_4 = (x_0 \vee \neg x_2) = C_{11}$
$C_2 \diamond C_5 = \not{1}$
$C_2 \diamond C_7 = (x_0) = C_{12}$
$C_2 \diamond C_8 = \not{1}$
$C_2 \diamond C_9 = \not{1}$
$C_2 \diamond C_{10} = \not{1}$

$C_3 \diamond C_4 = \not{1}$
$C_3 \diamond C_5 = (x_2) = C_{13}$
$C_3 \diamond C_6 = \not{1}$
$C_3 \diamond C_7 = \not{1}$
$C_3 \diamond C_9 = (\neg x_2) = C_{14}$
$C_3 \diamond C_{10} = 0$

↳ unsatisfiable

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

Answer: This formula is also not satisfiable because the output of resolving C_3 with C_7 is 0. Thus, no assignment extraction is possible. The work for this problem is shown below.

(b) $\varphi(x_0, x_1, x_2) = \underbrace{(x_0)}_{c_0} \wedge \underbrace{(\neg x_0 \vee x_1)}_{c_1} \wedge \underbrace{(\neg x_1 \vee \neg x_2)}_{c_2} \wedge \underbrace{(x_2)}_{c_3}$

$c_0 \diamond c_1 = x_1 = c_4$

$c_0 \diamond c_2 = 1$

$c_0 \diamond c_3 = \cancel{1}$

$c_1 \diamond c_2 = (\neg x_0 \vee \neg x_2) = c_5$

$c_1 \diamond c_3 = \cancel{1}$

$c_2 \diamond c_3 = (\neg x_1) = c_6$

$c_2 \diamond c_4 = (\neg x_2) = c_7$

$c_3 \diamond c_4 = \cancel{1}$

$c_3 \diamond c_5 = (\neg x_0) = c_8$

$c_3 \diamond c_7 = 0$ $\Rightarrow$ unsatisfiable

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

Answer: This formula is satisfiable because all of the exhaustively-created clauses are either a valid clause or 1. Thus, assignment extraction after resolution is possible, per the definition provided in lecture notes 16. The work for this problem (the resolution and the assignment extraction) is shown below.

(c) $\varphi(x_0, x_1, x_2, x_3) = \overbrace{(x_0 \vee x_1 \vee \neg x_3)}^{C_0} \wedge \overbrace{(x_2)}^{C_1} \wedge \overbrace{(x_0 \vee \neg x_2)}^{C_2} \wedge \overbrace{(x_1 \vee x_2)}^{C_3}$

Resolution in Order :

$C_0 \diamond C_1 = 1$

$C_0 \diamond C_2 = \cancel{1}$

$C_0 \diamond C_3 = \cancel{1}$

$C_1 \diamond C_2 = (x_0) = C_4$

$C_1 \diamond C_3 = \cancel{1}$

$C_2 \diamond C_3 = (x_0 \vee x_1) = C_5$

↳ satisfiable : $\varphi_{fin} = (x_0 \vee x_1 \vee \neg x_3) \wedge (x_2) \wedge (x_0 \vee \neg x_2) \wedge (x_1 \vee x_2) \wedge (x_0) \wedge (x_0 \vee x_1)$

Assignment Extraction:

$x_0 = 1$
$x_1 = 0$
$x_2 = 1$
$x_3 = 0$

$\Rightarrow = (1 \vee 0 \vee 1) \wedge (1) \wedge (1 \vee 0) \wedge (0 \vee 1) \wedge (1) \wedge (1 \vee 0)$

$= 1 \wedge 1 \wedge 1 \wedge 1 \wedge 1 \wedge 1$

$= \boxed{1}$ ✓

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.

Proof. As suggested, let us prove this claim using a reduction. Let Π be a computational problem "ProgrammingTeam" and let Γ be another computational problem "Satisfiability" which will be used as a subroutine for Π. In order to show that $\Pi \leq \Gamma$, we follow through with the following steps:

(1) First we need to identify what pre- and post- processing of variables needs to be done for our reduction. We first establish that the initial input to our reduction will be the inputs to the computational problem Π, namely n concentrators, m languages, and k slots on the programming team that need to be accurately filled. We then establish that the final output of the reduction will be the output of the computational problem Π, which is an accurate team T of k students covering all m languages together. Because we plan to make an oracle call to the computational problem Γ (which takes as input a CNF formula and outputs a satisfying assignment if it exists), we deduce that we need to pre-process our input variables to fit the input requirements of the subroutine, and will need to post-process our subroutine output to fit the requirements of our original computational problem.

(2) **Pre-processing + Oracle Call**: In the description of the problem, it has already been given to us that we will be working with kn variables and $m + O(kn^2)$ clauses. So, instead of pre-processing our variables in this step, let us discuss how the staff came up with these values.

Once our oracle is called, our formula variables are going to be represented as $x_{i,j}$ where i represents the ith slot on the team and j represents the jth student. Each of the n possible students, can occupy any of the k spots on the team in theory. So, we must have kn such variables to represent all possible student-spot variations.

Using these variables, our formula's clauses will be constructed following the constraints outlined in the SAT SRE sender notes. Following constraint 1, since we want to have at least one student to know each of the m languages, a subset (specifically, m number of clauses) of our clauses will be checking for this condition.

- Next, following constraint 2, our clauses must ensure that no two students both have the same position on the team (specifically, for every k positions, we must check each of the n students against each other, which means we would have kn^2 clauses here), which means another subset of our clauses will be checking for this condition.

- Next, our clauses must encode the fact that each of the k positions must have someone assigned to it, meaning that for every k position, we need to check through all n students to see if they belong in that spot. In the worst case scenario, for a given position, the last checked student belongs to the spot (which means we have kn clauses ensuring this condition).

- Finally, following constraint 5, we must ensure that every student is assigned to at most one position on the team. This means that for every n students, we must check each of the k positions against each other, which means we would have nk^2 clauses here. The other constraints listed in the SAT SRE sender notes are not relevant for us to check.

Working under the assumption that $k \leq n$ (since we have n people total and only k open slots on the team), another way of encasing the above bullet points would be by $O(kn^2)$ since the quantity of clauses created under constraint 2 overshadows all other quantities. Therefore, putting everything together, we must have $m + O(kn^2)$ clauses for this reduction.

Next, let's feed these clauses to our SAT solver oracle. It will output a satisfying assignment (either 1 or 0) for each of our variables, if it exists. If it doesn't exist, then the solver will output $\perp$.

(3) **Post-processing**: If SAT returns a satisfying assignment, we iterate through each of the kn variables; else, our Π needs to return $\perp$. If SAT returns a satisfying assignment, then for every $x_{i,j} = 1$, where $j \in S$ and $0 \leq i \leq k-1$, we want to put student j in position i of the team. With this, our reduction is complete.
Q.E.D.

Proof of Runtime: To determine the runtime of the reduction, let's analyze the runtime of each of the pre-procesing, oracle call, and post-processing individually. **(1)** Pre-processing involved creating clauses such that all of the constraints were met. As was alluded to in the reduction above, the runtime of creating clauses satisfying constraint 2 overshadows others, making the runtime of the pre-processing step $O(kn^2)$. **(2)** The oracle call, by assumption, takes constant time, so the runtime of this step is $O(1)$. **(3)** Post-processing involved iterating through all of the kn variables to find those that equaled 1 (if the SAT solver outputted a satisfying assignment), so the runtime of the post-processing step is $O(kn)$. Putting it all together, the runtime is $O(kn^2) + O(kn) + O(1)$ or $O(kn^2)$.

Proof of Correctness: We need to prove two claims. The first claim is that if our reduction returns a solution, then ProgrammingTeam has a solution. The second claim is that if ProgrammingTeam has a solution, then our reduction returns a solution.

- To prove claim 1, if our reduction returns a solution, then by definition all clauses are satisfied and there is a satisfying assignment. If there is a satisfying assignment, then by construction the computational problem ProgrammingTeam (PT) has a solution such that there exist k students that can be enlisted onto the team and cover all m languages together.
- To prove claim 2, if PT has a solution such that there exist k students that can be enlisted onto the team and cover all m languages together, then by construction this must mean that the reduction outputted a satisfying assignment, which by definition

must mean that all clauses of our original formula are satisfied and our reduction returns a valid solution.

(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.)