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: Andrew Courtney
Collaborators: Felix Yeung
No. of late days used on previous psets: 6
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
# Solvable Cluster Instances	3	3	4
# Solvable Other Graphs	0	0	2

   (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.

Proof. (a)

[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.

(b) Running the code as described here, we can see that the SAT solver way outperforms the Exhaustive Search and ISET BFS algorithms. In particular, it destroyed the other solvers on the ring instances. This may be because in the exhaustive and ISET colorings, colorings which are "next" to each other do not get any better - the strategy of working down a list of potential colorings which are very similar naively could take a really long time. However, the ring pattern is pretty "predictable", so the SAT solver would work pretty quickly through the conditions and not try a bunch of different graphs. The cluster instances are more random, meaning that going through colorings in a list is plausibly more successful from a runtime point of view. In the hard instances, we see again that SAT is especially good - it only fails at $n = 30000$, while the other two algorithms fail even at $n = 5000$. It's clear at this point that ISET in some cases is not even much of an upgrade over exhaustive coloring, which is kind of hard to fathom, but reflects that in cases of large n, exponential time is still really slow.

$\square$

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

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

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

Solution. (a) As in the lecture 16 notes, we define $C_0 = (x_2 \vee \neg x_1), C_1 = (x_3 \vee x_1), C_2 = (x_0 \vee x_1), C_3 = (\neg x_3), C_4 = (\neg x_1 \vee \neg x_2)$. Now we continue:

$$C_5 = C_0 \diamond C_1 = (x_2 \vee x_3)$$

$$C_6 = C_0 \diamond C_2 = (x_0 \vee x_2)$$

$$C_7 = C_0 \diamond C_3 = 1$$

$$C_8 = C_0 \diamond C_4 = (\neg x_1)$$

$$C_1 \diamond C_2 = 1$$

$$C_9 = C_1 \diamond C_3 = (x_1)$$

$$C_{10} = C_1 \diamond C_4 = (x_3 \vee \neg x_2)$$

$$C_2 \diamond C_3 = 1$$

$$C_{11} = C_2 \diamond C_4 = (x_0 \vee \neg x_2)$$

$$C_3 \diamond C_4 = 1, C_5 \diamond C_6 = 1, C_5 \diamond C_7 = 1, C_5 \diamond C_8 = 1, C_5 \diamond C_9 = 1$$

$$C_{12} = C_5 \diamond C_{10} = (x_3)$$

$$C_{13} = C_5 \diamond C_{11} = (x_0 \vee x_3)$$

$$C_6 \diamond C_7 = 1, C_6 \diamond C_8 = 1, C_6 \diamond C_9 = 1, C_6 \diamond C_{10} = (x_0 \vee x_3) = C_{13}$$

$$C_{14} = C_6 \diamond C_{11} = (x_0)$$

$$C_7 \diamond C_8 = 1, C_7 \diamond C_9 = 1, C_7 \diamond C_{10} = 1, C_7 \diamond C_{11} = 1$$

$$C_8 \diamond C_9 = 0.$$

So this is unsatisfiable.

(b) We define $C_0 = (x_0), C_1 = (\neg x_0 \vee x_1), C_2 = (\neg x_1 \vee \neg x_2), C_3 = (x_2)$. Now we continue:

$$C_4 = C_0 \diamond C_1 = (x_1)$$

$$C_5 = C_0 \diamond C_2 = 1$$

$$C_0 \diamond C_3 = 1$$

$$C_6 = C_1 \diamond C_2 = (\neg x_0 \vee \neg x_2)$$

$$C_1 \diamond C_3 = 1,$$

$$C_7 = C_2 \diamond C_3 = (\neg x_1)$$

$$C_4 \diamond C_5 = 1, C_4 \diamond C_6 = 1$$

$$C_8 = C_4 \diamond C_7 = 0.$$

So this is also unsatisfiable.

(c) Finally, we define $C_0 = (x_0 \vee x_1 \vee \neg x_3), C_1 = (x_2), C_2 = (x_0 \vee \neg x_2), C_3 = (x_1 \vee x_2)$. Now we continue:

$$C_4 = C_0 \diamond C_1 = 1$$

$$C_0 \diamond C_2 = 1, C_0 \diamond C_3 = 1$$

$$C_5 = C_1 \diamond C_2 = (x_0)$$

$$C_1 \diamond C_2 = 1$$

$$C_6 = C_2 \diamond C_3 = (x_0 \vee x_1)$$

$$C_4 \diamond C_5 = 1, C_4 \diamond C_6 = 1, C_5 \diamond C_6 = 1.$$

This algorithm has terminated! This means we have a solution. To find it, we use the Extract Assignment algorithm. We have conditions $\mathcal{C} = \{C_0 = (x_0 \vee x_1 \vee \neg x_3), C_1 = (x_2), C_2 = (x_0 \vee \neg x_2), C_3 = (x_1 \vee x_2), C_4 = 1, C_5 = (x_0), C_6 = (x_0 \vee x_1)\}$. Then we do the algorithm as follows:

$$i = 0 : (x_0) \in \mathcal{C} \implies x_0 = 1.$$

Updating $\mathcal{C}$ gives us $\mathcal{C} = \{C_0 = 1, C_1 = (x_2), C_2 = 1, C_3 = (x_1 \vee x_2), C_4 = 1, C_5 = 1, C_6 = 1\}$. Next,

$$i = 1 : (x_1) \notin \mathcal{C} \implies x_1 = 0.$$

Updating $\mathcal{C}$ gives us $\mathcal{C} = \{C_0 = 1, C_1 = (x_2), C_2 = 1, C_3 = (x_2), C_4 = 1, C_5 = 1, C_6 = 1\}$. Next,

$$i = 2 : (x_2) \in \mathcal{C} \implies x_2 = 1.$$

Updating, $\mathcal{C} = \{C_0 = 1, C_1 = 1, C_2 = 1, C_3 = 1, C_4 = 1, C_5 = 1, C_6 = 1\}$. Finally,

$$i = 3 : (x_3) \in \mathcal{C} \implies x_3 = 0.$$

So we return $\alpha = (1, 0, 1, 0)$, which in fact satisfies the original conditions.

□

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 $L = \{\ell_0, \dots, \ell_{m-1}\}$ of programming languages; a finite set $S = \{s_0, \dots, s_{n-1}\}$ 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 $O(kn)$ 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.)

Proof. (a) Define kn variables as $v_{i,j}$, where $0 < i < n$ and $0 < j < k$. Consider each j as a "place" on the programming team; that is, student a occupies the 0^{th} place on the team if $v_{a,0} = 1$, student b occupies the 1^{st} place on the team if $v_{b,0} = 1$, and so on. Then the goal is to have k variables with value 1 and the rest with value zero at the end. We attach a few conditions:

- For each j such that $0 < j < k$, $(v_{0,j} \vee v_{1,j} \vee \cdots \vee v_{n-1,j})$. This means that at least one person is on each place on the team. We do this k times (once for each j), so there are k clauses.

- For each j such that $0 < j < k$, for all pairs $0 < a < b < n$, we need $(\neg v_{a,j} \vee \neg v_{b,j})$. This assures that no spot on the team is taken up by two people. We do this $kn(n-1)/2$ times - for each j, we have $n(n-1)/2$ pairs that we have to take care of.
- For each i such that $0 < i < n$ and all pairs $0 < c < d < k$, $(\neg v_{i,c} \vee \neg v_{i,d})$. This means that no student is assigned to two different spots on the team. We do this $k(k-1)n/2$ times - for each i, we have $k(k-1)/2$ pairs to take care of.
- For each p such that $0 < p < m$, find all q such that $\ell_p \in K(s_q)$. Let the list of all such q be $q_0, q_1, \dots, q_r$. Then we add a condition

  $$(v_{q_0,0} \vee v_{q_0,1} \vee \cdots \vee v_{q_0,k-1} \vee v_{q_1,0} \vee v_{q_1,1} \vee \cdots \vee v_{q_1,k-1} \vee \dots v_{q_r,0} \vee v_{q_r,1} \vee \cdots \vee v_{q_r,k-1}).$$

  This tells us that every single language is covered by at least one person on the team. We do this m times - once for every language.

So the number of clauses it takes to get to SAT are $k + kn(n-1)/2 + nk(k-1)/2 + m = m + O(kn^2)$ (because necessarily $n \geq k$). To get from the result of the SAT solver to the answer to our problem, we simply take all i with $0 < i < n$ for which there is some j for which $v_{i,j} = 1$, or return $\perp$ if no solution exists. Now it remains to prove correctness.

Suppose the reduction returned a solution which was incorrect. We must return k students - the first and second conditions together tell us that exactly one person is in each spot on the team, and the third condition tells us that a student can be in a maximum of one position; therefore, exactly k students are on the team (because students can have one position or no positions). Then if the reduction was incorrect, it would imply that our team does not know all the languages. But the fourth condition guarantees us that for every single language, at least one person on the team (situated in the correct spot) knows the language. So this is impossible - if the reduction returns a solution, it must be correct.

Now suppose that the algorithm returns $\perp$, but there exists some solution. Then the solution must contradict one of the conditions. Suppose that for some $0 < j < k$, we have $(v_{0,j} \vee v_{1,j} \vee \cdots \vee v_{n-1,j}) = 0$. But then this implies that there is some position on the team for which no student is listed. But we know that there are k students who satisfy this problem, so there is some position which has two students (by Pigeonhole); however, without problem, we can shift one student from a doubly-occupied slot to the unoccupied slot, and then we are okay.

Suppose that the first condition holds, but there is some j, a, b as defined in the second condition such that $(\neg v_{i,c} \vee \neg v_{i,d}) = 0$. Then some spot is taken up by two people; however, we know that there are k students which satisfy this solution, and the first condition tells us that every single position is occupied by at least one student; by Pigeonhole, we conclude that no position can hold two people.

Now suppose the second condition holds as well, and there are some i, c, d as defined in the third condition such that $(\neg v_{i,c} \vee \neg v_{i,d}) = 0$. We know that there are k students who satisfy the requirements. Also, every spot is occupied by exactly one student from the above conditions; therefore, this can't happen. So now we assume all these conditions are true and we let there be p, q as defined in the fourth condition such that

$$(v_{q_0,0} \vee v_{q_0,1} \vee \dots \vee v_{q_0,k-1} \vee v_{q_1,0} \vee v_{q_1,1} \vee \dots \vee v_{q_1,k-1} \vee \dots v_{q_r,0} \vee v_{q_r,1} \vee \dots \vee v_{q_r,k-1}) = 0.$$

But this cannot happen, as we need every single language to be covered. So we cannot contradict a condition and still return $\perp$ - therefore our solution is correct.

□