CS120: Intro. to Algorithms and their Limitations Hesterberg & Anshu

Problem Set 7

Harvard SEAS - Fall 2023 *Due: Wed Nov. 15, 2023 (11:59pm)*

Your name: Cory Zimmerman
Collaborators: None
No. of late days used on previous psets: 0
No. of late days used after including this pset: 0

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`.

   Done.

   (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 hard graph instances. 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 / 18	2 / 18	6 / 18
# Solvable Cluster Instances	9 / 18	15 / 18	18 / 18
# Solvable Hard Graphs	0 / 6	0 / 6	4 / 6

   It makes sense that the exhaustive algorithm underperformed ISET BFS and SAT for all trials. I found it interesting how thoroughly SAT crushed ISET BFS and how well it performed on the hard graphs. Considering asympototic behavior, ISET BFS is expected to run in $O(1.89^n)$ time, while the new algorithm reduces to SAT in $O(n + km)$ time. While 3-SAT, which this algorithm effectively uses, also doesn't have a polynomial runtime bound, it seems reasonable that an optimized SAT solver like PySAT wrapping well-tuned C and C++ code is able to handily outperform ISET BFS.

(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. (2SAT) Consider the following Boolean statements.

   (a) A:= "Alice's grandparents got married."

   (b) B:= "Alice was born."

   (c) C:="Alice created a time machine in order to travel to the past."

   (d) D:="Alice landed the time machine at her grandparents' first date and ruined it."

   Consider the CNF formula

   $$(\neg A \lor B) \land (A \lor \neg B) \land (\neg B \lor C) \land (B \lor \neg C) \land (\neg C \lor D) \land (C \lor \neg D) \land (\neg D \lor \neg A) \land (D \lor A).$$

   If you interpret each clause above as an implication, are the implications reasonable? Explain briefly for 3 clauses of your choice. Draw the implication graph (for the whole formula, not just your chosen 3 clauses) and explain graphically why the CNF is unsatisfiable.

   A logical clause of the form $(\neg\alpha \lor \beta)$ can be stated naturally as an implication of the form "if α, then β". Expressing the terms of the CNF in this form makes them sound generally reasonable, assuming time travel is feasible and Alice has significant interest in her grandparents' first date:

   - $\neg A \lor B$: "If Alice's grandparents got married, then Alice was born": I find this more likely than if her grandparents did not get married.
   - $B \lor \neg C$: "If Alice was not born, then Alice did not create a time machine in order to travel to the past". If Alice does not exist, she can't create a time machine, so I find this reasonable.
   - $C \lor \neg D$: "If Alice did not create a time machine in order to travel to the past, then Alice did not land the time machine at her grandparents' first date and ruin it". I expect that Alice wouldn't have access to an available, pilotable time machine headed towards her grandparents' first date had she not created one, so, again, this makes sense to me.

[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.

$\neg A \vee B \equiv A \to B \equiv \neg B \to \neg A$

$A \vee \neg B \equiv \neg A \to \neg B \equiv B \to A$

$\neg B \vee C \equiv B \to C \equiv \neg C \to \neg B$

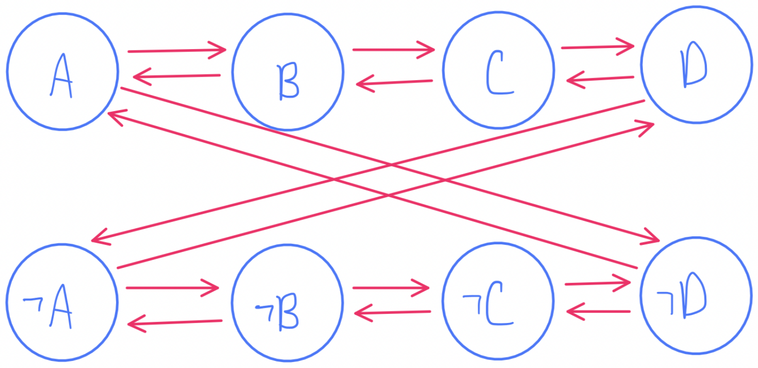
$B \vee \neg C \equiv \neg B \to \neg C \equiv C \to B$

$\neg C \vee D \equiv C \to D \equiv \neg D \to \neg C$

$C \vee \neg D \equiv \neg C \to \neg D \equiv D \to C$

$\neg D \vee \neg A \equiv D \to \neg A \equiv A \to \neg D$

$D \vee A \equiv \neg D \to A \equiv \neg A \to D$



By Theorem 3.3 from the lecture notes, this CNF of width two is unsatisfiable because there exists a path from A to $\neg D$ and from $\neg A$ to D in the implication digraph. Stated more visually, because there's a visible loop in this graph of a width 2 CNF, it cannot be satisfiable.

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.

I'll demonstrate this reduction by explaining how to algorithmically encode the inputs into a boolean formula that can be passed to SAT. First, however, recognize that, if k ever exceeds n, we can immediately return $\perp$. This follows from the observation that we can never create a team of k people from fewer than k people. So, for the nontrivial parts of this reduction, $k \leq n$. In my reduction, I check this case, perform the encodings below, and then call SAT. If

SAT returns $\bot$, my reduction also returns $\bot$. If SAT returns an assignment, my reduction extracts the student number from each assigned variable and returns those values in a set. With this in mind, consider how this reduction encodes inputs to SAT:

Let $S_{i,j}$ express that student i was selected for position j on a team where $0 \leq i < n$ and $0 \leq j < k$. By the rules of combinatorics, this allows kn variables in the boolean formula. Note: while the problem itself does not require ordered output, I'm introducing numerical order to the selected students in order to identifiy duplicates.

My reduction to SAT requires encoding four aspects in the boolean formula:

1. Every position on the team must be allocated.
2. No student may be allocated for more than one position.
3. No position may be allocated for more than one student.
4. The members of the chosen team must collectively know every language.

Encoding (1):
For every position, there must be at least one allocated student. This can be expressed in the following sequence:

$$(S_{0,0} \vee S_{1,0} \vee \cdots \vee S_{n-1,0}) \vee \cdots \vee (S_{0,k-1} \vee S_{1,k-1} \vee \cdots \vee S_{n-1,k-1})$$

In the sample above, the first clause verifies that at least one student is allocated to position 0, and the final clause verifies that at least one student is allocated to position $k-1$. With one clause for each position, there are k of these in total.

Encoding (2):
For every student, if they are assigned to position j, then they cannot be assigned to any position that is not j. Denote this as j^c. In implication form, this can be expressed as $S_{i,j} \implies \neg S_{i,j^c}$. Each implication can be expanded into $\neg S_{i,j} \vee \neg S_{i,j^c}$ to produce a formula like this:

$$(\neg S_{0,0} \vee \neg S_{0,1}) \wedge (\neg S_{0,0} \vee \neg S_{0,2}) \wedge \cdots \wedge (\neg S_{0,0} \vee \neg S_{0,k-1}) \wedge \cdots \wedge (\neg S_{0,k-2} \vee \neg S_{0,k-1})$$

$$\cdots$$

$$(\neg S_{n-1,0} \vee \neg S_{n-1,1}) \wedge (\neg S_{n-1,0} \vee \neg S_{n-1,2}) \wedge \cdots \wedge (\neg S_{n-1,0} \vee \neg S_{n-1,k-1}) \wedge \cdots \wedge (\neg S_{n-1,k-2} \vee \neg S_{0,k-1})$$

In the sample above, the first three clauses encode "if student 0 is in position 0, then student 0 must not be in any other position". The final term implies checking every possible position that student 0 may start in. The bottom row implies performing such verifications for every student. In total, this generates $O(k^2)$ clauses for n students. However, because of the trivial case addressed at the beginning, it's guaranteed that $k \leq n$ such that the number of clauses needed for this encoding is bounded by $O(k \cdot n^2)$.

Encoding (3):
For every position i, if student j is assigned to that position, then no other student i^c may be assigned there. In implication form, this is expressed as $S_{i,j} \implies S_{i^c,j}$ or equivalently as $\neg S_{i,j} \implies \neg S_{i^c,j}$. Encode this property like this:

$$(\neg S_{0,0} \vee \neg S_{1,0}) \wedge (\neg S_{0,0} \vee \neg S_{2,0}) \wedge \cdots \wedge (\neg S_{0,0} \vee \neg S_{n-1,0}) \wedge \cdots \wedge (\neg S_{n-2,0} \vee \neg S_{n-1,0})$$

$$\cdots$$

$$(\neg S_{0,k-1} \vee \neg S_{1,k-1}) \wedge (\neg S_{0,k-1} \vee \neg S_{2,k-1}) \wedge \cdots \wedge (\neg S_{0,k-1} \vee \neg S_{n-1,k-1}) \wedge \cdots \wedge (\neg S_{n-2,k-1} \vee \neg S_{n-1,k-1})$$

In the sample above, the first three clauses encode "if position 0 is allocated to student 0, then position 0 must not be allocated to any other student". The final term implies performing that check on every possible student who might be in position 0. The bottom row implies performing such verifications for every position. In total, this generates $O(n^2)$ clauses for k positions, yielding a clause count bounded by $O(kn^2)$.

Encoding (4):
For every language, someone on the team must know it. Thus, for each of M languages, add a single clause to verify that at least one person in any position on the team knows the language. In the sample below, students α and β are the only students who know language $m \in M$.

$$(S_{\alpha,0} \vee S_{\alpha,1} \vee \cdots \vee S_{\alpha,k-1} \vee S_{\beta,0} \vee S_{\beta,1} \vee \cdots \vee S_{\beta,k-1})$$

This produces one (potentially very long) clause verifying language inclusion in the selected group for every required language, so it adds m clauses in total.

Input requirements:
Because the input variable, $S_{n,k}$ is bounded by $n \cdot k$ total combinations, this satisfies the variable requirement. Encoding the variants adds clauses of size $O(k) + O(kn^2) + O(kn^2) + O(m) = O(m + kn^2)$. Because of this, both input requirements are satisfied.

Correctness:
The algorithm accepts inputs L, S, and K and uses these to encode a reduction to SAT. The boolean algorithm to SAT encodes the following properties: (1) the output team has k members, (2 and 3) each of those members is uniquely assigned to the team, and (4) the members of the team collectively know every required language. By properly encoding these characteristics and returning the properly decoded results of an oracle call to SAT, my program produces a valid output to ProgrammingTeam for every valid input, making it a correct algorithm.

Runtime:
The runtime of this reduction is dominated by performing the encodings. Consider each. Encoding (1) can be performed in a loop iterating "for each of k

positions, for each of n students" and has runtime $O(kn)$. Encoding (2) loops "for each of n students, for each of k positions, for each of k positions". As established earlier, $k \leq n$ such that this runtime can be bounded by $O(kn^2)$. Encoding (3) loops "for each of k positions, for each of n students, for each of n students", which completes in time $O(kn^2)$. And Encoding (4) loops "for each of m languages, for each of n students, for each of k positions", which runs in time $O(mnk)$. Finally, decoding the result of SAT into a set will take at most the number of results in a satisfying assignment, $O(k)$. Thus, the runtime of my reduction completes in time $O(kn + kn^2 + kn^2 + mnk + k)$, which can be simplified into $O(kn^2 + mnk)$.

(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.)

4. Please fill out the Problem Set 7 Feedback Survey here: https://forms.gle/bNfaHbXutBiXq97v6. We appreciate your feedback!