CS 3510 — Algorithms, Spring 2022

Lecture 17: Introduction to Complexity

April 5, 2022

Lecturer: Frederic Faulkner *Scribe: Aditya Diwakar*

In this part of the course, rather than solving problems, we will be finding *reductions* between problems such that some new problem given can be reduced to another problem (that we may already have an algorithm for).

1 CNF-SAT

This is the first type of problem that will be covered. It is a type of SAT (satisfiability) problem where we are given some boolean expression and we want to assign values to these boolean literals such that the expression is true. For example:

$$x_1 \vee \neg(x_3 \wedge x_4) \wedge (x_5 \vee x_1 \vee x_2 \vee_4)$$

In the above, how can we set these values such that the expression is true? In the above, we could set x_1 to true and x_3 to false such that this entire statement is true. This would make $\neg(x_3 \wedge x_4)$ true as well as $(x_5 \vee x_1 \vee x_2 \vee_4)$ meaning the entire expression would be true.

CNF means conjunctive normal form which is a specific format for the boolean expressions where we have clauses where individual literals are OR'd together and those clauses are AND'd together. For example, $(x_1 \vee x_3 \vee_4 \vee x_7)$ would be a clause. We can then take a bunch of clauses and AND them together:

$$(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_3 \vee \neg x_4) \wedge (x_2 \vee x_4 \vee \neg x_5) \wedge (\neg x_3 \vee x_5)$$

This is an example of a boolean expression in conjunctive normal form. Let's see if this can be satisfied (the SAT part of CNF-SAT): If we arbitrarily pick some assignments such as:

x_1	x_2	x_3	x_4	x_5
F	F	T	F	F

Then, we can evaluate each of the clauses (left to right) as c_1, c_2, c_3, c_4:

$$\begin{array}{c|l|l} c_1 & (x_1 \vee \neg x_2) & F \vee T = T \\ c_2 & (\neg x_1 \vee x_3 \vee \neg x_4) & T \vee T \vee T = T \\ c_3 & (x_2 \vee x_4 \vee \neg x_5) & F \vee F \vee T = T \\ c_4 & (\neg x_3 \vee x_5) & F \vee F = F \end{array}$$

Clearly, this assignment does not work because it does not satisfy c_4. We can try another assignment instead:

x_1	x_2	x_3	x_4	x_5
F	F	T	T	T

Then, we can evaluate each of the clauses (left to right) as c_1, c_2, c_3, c_4:

$$\begin{array}{c|l|l} c_1 & (x_1 \vee \neg x_2) & F \vee T = T \\ c_2 & (\neg x_1 \vee x_3 \vee \neg x_4) & T \vee T \vee F = T \\ c_3 & (x_2 \vee x_4 \vee \neg x_5) & F \vee T \vee F = T \\ c_4 & (\neg x_3 \vee x_5) & F \vee T = T \end{array}$$

Since this satisfies every clause and each clause is AND'd together, this means we have found an assignment satisfying the boolean expression.

Why do we care about CNF? It makes it significantly easier to reason about boolean expressions. It is also important to note that every boolean expression can be converted into CNF form (by using a truth table and going row by row making clauses).

1.1 Challenges

1. Is this satisfiable?

   $$(x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_3) \wedge (x_3 \vee \neg x_1)$$

   No, this boolean expression has no assignment such that it is satisfied.

2. Is this satisfiable?

$$(x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee x_3) \wedge (\neg x_3 \vee \neg x_2) \wedge (x_3 \vee x_4)$$

Yes, let us try the following assignment:

x_1	x_2	x_3	x_4
T	F	T	T

Then, we can evaluate each of the clauses (left to right) as c_1, c_2, c_3, c_4, c_5:

c_1	$(x_1 \vee \neg x_2)$	$T \vee T = T$
c_2	$(x_2 \vee \neg x_3 \vee x_4)$	$F \vee F \vee T = T$
c_3	$(\neg x_1 \vee x_3)$	$F \vee T = T$
c_4	$(\neg x_3 \vee \neg x_2)$	$F \vee T = T$
c_5	$(x_3 \vee x_4)$	$T \vee T = T$

Hence, we have found an assignment that satisfies this expression.

2 Polynomial Time

We call the set P the set of all problems that can be solved in polynomial time. For example, $O(n), O(n^2), O(n^{417}), O(n \log n)$ are all in polynomial time.

On the contrary, $O(2^n)$ and $O(n!)$ are not polynomial time. Why do we care about polynomial time? It is because it is reasonable for an algorithm to run in polynomial time, while exponential time is not.

Within the class P, P hides exponents. When we did Big-O, we said $O(n) = O(2n)$ as Big-O hid additive and multiplicative constants. In the same way, now $O(n^2)$ and $O(n^3)$ are in the same class P so we can say P hides/abstracts away exponents.

A lot of the algorithms we have studied in this class are in P.

Multiplying 2 n bit numbers: $O(n^{\log_2 3}) \in P$
Longest Increasing Subsequence: $O(n^2) \in P$
Kruskal's: linear $\in P$
Floyd Warshall: $O(n^3) \in P$

3 Non-Deterministic Polynomial

We call the set NP be the set of problems whose solution can be verified in polynomial time. This solution may not have been computed in polynomial time, but we can at least check it *quickly*.

In other words, P is easy to do, and NP is easy to check. We know that $P \subseteq NP$ since if a problem is easy to do, it would be easy to check. However, why is this $\subseteq$ rather than $\subset$? It is unknown if $P = NP$ (this is an open problem).

4 Back to CNF-SAT

Input: Boolean expression (list of clauses)
Output: Assignment of variables (list of true/false values)
Check Solution: Check a candidate solution in polynomial time with:

```
1 Function CheckSolution(problem, assignment):
2     for clause ∈ problem do
3         satisfied ← false
4         for variable ∈ clause do
5             if assignment[variable] = True then
6                 satisfied ← true
7         if satisfied = false then
8             return false
9     return true
```

We know that we can check a solution in polynomial time, but cannot find a solution in polynomial time. A simple solution of just checking every combination

has runtime of $O(2^n)$ and even a very optimized approach is roughly $O(1.307^n)$. Hence, this problem is not in P but it is in NP.

5 Graph Coloring

Given a graph, we want to assign a color to each vertex such that no two vertices which share an edge have the same color. There is no known way to quickly find n colorings. However, checking a candidate solution is quick because we can loop through the edges in the graph and check that the endpoints are of different colors. This takes $O(|E|)$ time which is polynomial time (linear in the number of edges).

Hence, graph coloring is in NP.

6 2-SAT and 3-SAT

These are variations of the CNF-SAT problem from before but has a restriction on the number of literals within each clause.

6.1 2-SAT

This is the question whether or not some boolean expression in CNF form where each clause has 2 literals. The actual algorithm used to do this is not important, but essentially uses path finding in a graph (converting the expression to a graph).

This *can* be solved in polynomial time. If we could find some way to convert every SAT problem to a 2-SAT problem in polynomial time, then we can solve every SAT problem in polynomial time.

6.2 3-SAT

In similar fashion to 2-SAT, this is a boolean expression in CNF where each clause has 3 literals. It is true that every CNF-SAT can be converted to 3-SAT in polynomial time . However, we do not have an algorithm to find satisfying assignments

for 3-SAT (otherwise, this would solve SAT).

How do we convert CNF-SAT $\rightarrow$ 3-SAT? Suppose we have some CNF-SAT expression which has a clause of length n:

$$(x_1 \vee x_2 \vee \cdots \vee x_{n-1} \vee x_n)$$

Since the clauses can only have 3 literals at most, then we claim that we can rewrite this as:

$$\underbrace{(x_1 \vee x_2 \vee z_1)}_{c_1} \wedge \underbrace{(\neg z_1 \vee x_3 \vee \cdots \vee x_{n-1} \vee x_n)}_{c_2}$$

The left clause is of length 3 and the right clause has size $n-1$ so if this process is valid, we should be able to reduce the n long clause into multiple clauses of length 3 each. However, is this process valid?

($\implies$) Suppose some assignment satisfies the original clause, then it must also satisfy c. If some x_i is true, then x_i is either in c_1 or c_2. If it is in c_1, then to satisfy c_2, we set z_1 to false. If x_i is in c_2, then we can simply set z_1 to true (remember: we control the value of z_1 since we are adding it).

($\impliedby$) Suppose some assignment satisfies F', then it satisfies c_1 and c_2. If z_1 is true, then c_1 is satisfied but to satisfy c_2, then one of $x_3, \ldots, x_n$ must be true. If that is the case, then it must have satisfied the original clause C.

If z_1 is false, then c_2 is already satisfied and then c_1 is satisfied by something other than z_1 (either x_1 or x_2). However, this implies that x_1 or x_2 was true in the original clause satisfying C.

Hence, we have shown that if we had a satisfying assignment before, we have one after altering the expression. Also, if we have a satisfying assignment now, we must have had a satisfying assignment before. Hence, this is a valid strategy to go from CNF-SAT $\rightarrow$ 3-SAT. $\square$