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

Lecture 16: Satisfiability and Solving 2-SAT

Harvard SEAS - Fall 2023 *Oct. 31, 2023*

1 Announcements

- PS7 out today, due 15 Nov.
- Sender-Receiver Exercise Thu Nov 2.
- Adam OH after class

2 Recap

- A *literal* is a variable (e.g. x_i) or its negation ($\neg x_i$).
- A boolean formula is in *conjunctive normal form (CNF)* if it is the AND of a sequence of *clauses*, each of which is the OR of a sequence of literals.
- It will be convenient to also allow 1 (true) to be a clause. 0 (false) is already a clause: the empty clause is always false.

Input	: A CNF formula φ on n variables
Output	: An $\alpha \in \{0,1\}^n$ such that $\varphi(\alpha) = 1$ (if one exists)

Computational Problem CNF-Satisfiability (SAT)

The problem introduced in Lecture 15 (in class) was a decision problem (decide if the given CNF formula is satisfiable or not); above is a search version (return the satisfying assignment).
Motivation for studying SAT (and logic problems in general): can encode (and be reduced to from) many other problems of interest:

- Graph Coloring (Lecture 15)
- Longest Path (SRE 4)
- Independent Set (section)
- Programming Team (ps7)
- and many more (Lecture 19)

Unfortunately, the fastest known algorithms for Satisfiability have worst-case runtime exponential in n. However, enormous effort has gone into designing heuristics that complete much more quickly on many real-world instances.

In particular, SAT Solvers—with many additional optimizations—were used to solve large-scale graph coloring problems arising in the 2016 US Federal Communications Commission (FCC) auction to reallocate wireless spectrum. Roughly, those instances had $k = 23$ colors (corresponding to UHF channels 14–36), n in the thousands (corresponding to television stations being reassigned to one of the k channels), m in the tens of thousands (corresponding to pairs of stations with overlapping broadcast areas). Over the course of the one-year auction, tens of thousands of coloring instances were produced, and roughly 99% of them were solved within a minute!

In this course, our focus is on algorithms that solve a computational problem in the worst case. The heuristic algorithms above lie outside this framework. Thus, we will focus on SAT solvers for a special case of CNF-SAT, where provably efficient algorithms are known.

3 k-SAT and efficient algorithms for 2-SAT

The computational problem k-SAT is obtained when we restrict the number of literals in each clause of SAT.

Input	: A CNF formula φ on n variables in which each clause has width at most k (i.e. contains at most k literals)
Output	: An $\alpha \in \{0,1\}^n$ such that $\varphi(\alpha) = 1$, or $\perp$ if no satisfying assignment exists

Computational Problem k-SAT

The case of $k = 2$ is already an interesting SAT problem, with applications in scheduling, graph drawing and evolutionary biology. This problem can be efficiently solved, as shown in the following theorem.

Theorem 3.1. *There is an algorithm that solves the computational problem 2-SAT in time $O(nm)$, where n is the number of variables and m is the number of clauses.*

At the heart of this theorem is the construction of a digraph for the CNF formula ϕ, which allows us to use some of the graph algorithms we have already seen. The edges in this digraph capture 'implications' in the CNF formula.

Implication: The clause $(\neg a \vee b)$ has the property that if a is TRUE (1), then b must be TRUE (1). This relation can be interpreted as "a implies b". This clause is false only when a is TRUE (1) and b is FALSE (0), which translates to the fact that a TRUE statement can never imply a FALSE statement (otherwise, mathematics would be full of inconsistencies).

Note that we can also interpret $(\neg a \vee b)$ as "$\neg b$ implies $\neg a$". This holds since $(\neg a \vee b) = (\neg(\neg b) \vee \neg a)$.

To illustrate, let a denote "Apple falls" and b denote "Gravity acts". Then the clause "Either Apple doesn't fall OR Gravity acts" is a true statement. It can be interpreted as "If Apple falls then Gravity acts" and also as "If Gravity does not act then Apple does not fall". One should be aware of a common logical fallacy - $(\neg a \vee b)$ should not be interpreted as "$\neg a$ implies $\neg b$". Indeed, if Apple doesn't fall, then we can't conclude that Gravity doesn't exist. Apple may not fall due to multiple reasons, such as a strong tree trunk.

With this discussion in mind, we can graphically represent $(\neg a \vee b)$ in two ways: as an edge from a vertex a to a vertex b, or as an edge from a vertex $\neg b$ to a vertex $\neg a$. These are both shown in Figure 1.

If we have several 2-CNF clauses, we can combine these graphical representations.

Example: Consider the CNF

$$\phi_1 = (\neg x_0 \vee x_1) \wedge (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \wedge (\neg x_3 \vee x_0).$$

$$\begin{tikzcd}
a \arrow[r] & b \\
\neg a & \neg b \arrow[l]
\end{tikzcd}$$

Figure 1: Two representations of $(\neg a \vee b)$.

ϕ_1 can be represented as shown in Figure 2.

$$\begin{tikzcd}
x_0 \arrow[r] & x_1 \arrow[r] & x_2 \arrow[r] & x_3 \arrow[lll, bend right] \\
\neg x_0 \arrow[rrr, bend left] & \neg x_1 \arrow[l] & \neg x_2 \arrow[l] & \neg x_3 \arrow[l]
\end{tikzcd}$$

Figure 2: Chain of implications

The CNF is satisfiable with the assignments $(1, 1, 1, 1)$ and $(0, 0, 0, 0)$.

Example: Consider another CNF

$$\phi_2 = (\neg x_0 \vee x_1) \wedge (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \wedge (\neg x_3 \vee \neg x_0).$$

This can be represented as shown in Figure 3.

$$\begin{tikzcd}
x_0 \arrow[r] \arrow[drrr] & x_1 \arrow[r] & x_2 \arrow[r] & x_3 \arrow[dlll] \\
\neg x_0 & \neg x_1 \arrow[l] & \neg x_2 \arrow[l] & \neg x_3 \arrow[l]
\end{tikzcd}$$

Figure 3: Another chain of implications

ϕ_2 is also satisfiable, by any of the assignments $(0, 0, 0, 1), (0, 0, 1, 1), (0, 1, 1, 1)$. To understand this, note that the path $(x_0, x_1, x_2, x_3, \neg x_0)$ of implications in the graph implies that "x_0 implies $\neg x_0$", which is the same as $(\neg x_0 \vee \neg x_0) = (\neg x_0)$. This evaluates to 1 only when $x_0 = 0$. Once we have set $x_0 = 0$, we can solve the smaller CNF, which has the above 3 solutions.

Example: As a final example, consider the CNF

$$\phi_3 = (\neg x_0 \vee x_1) \wedge (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \wedge (\neg x_3 \vee \neg x_0) \wedge (x_0).$$

Note that this is unsatisfiable, since its first four clauses are the same as those of ϕ_2 from the previous example and forced $x_1 = 0$. Graphically, if we write (x_0) as $(x_0 \vee x_0)$, we get the representation in Figure 4.

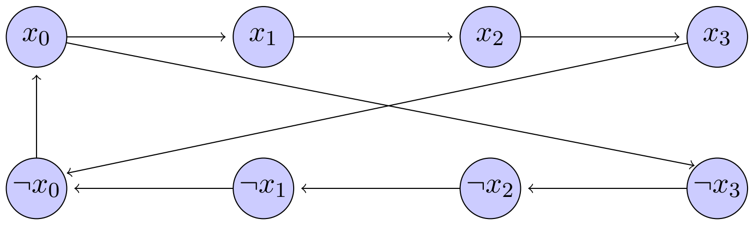

Figure 4: A bad cycle of implications

In Figure 4, the paths of implications tell us that x_0 implies $\neg x_0$ and also $\neg x_0$ implies x_0. Since one of these literals is TRUE and another is FALSE, we would be bound to conclude that TRUE implies FALSE, if all the clauses were satisfiable.

3.1 Warm up: Efficient algorithm for a decision version of 2-SAT

Before proving Theorem 3.1, let us consider a decision version of the problem k-SAT.

Input	**:** A CNF formula φ on n variables in which each clause has width at most k (i.e. contains at most k literals)
Output	**:** 1 if there exists an $\alpha \in \{0,1\}^n$ such that $\varphi(\alpha) = 1$, and 0 if no satisfying assignment exists

Computational Problem k-SAT-Decision

In general, k-SAT-Decision is known to be as hard as k-SAT and 2-SAT-Decision can be solved by an efficient algorithm.

Theorem 3.2. *There is an algorithm that solves the computational problem 2-SAT-Decision in time $O(nm)$, where n is the number of variables and m is the number of clauses.*

Implication digraph: In order to prove this theorem, we formally introduce the aforementioned digraph. Given a CNF formula ϕ on n variables $x_0, x_1, \ldots x_{n-1}$ such that each clause has width at most 2, implication digraph $G = (V, E)$ on $2n$ vertices is as follows. Each vertex is a literal x_i or $\neg x_i$ and for each clause $(\ell_i \vee \ell_j)$, we introduce two directed edges $(\neg \ell_i, \ell_j)$ and $(\neg \ell_j, \ell_i)$. The three examples in Figures 2, 3, 4 are implication digraphs. The following theorem captures the role of this digraph.

Theorem 3.3. *A CNF formula ϕ in which each clause has width 2 is satisfiable iff for any variable x_i, there is no path from x_i to $\neg x_i$ AND no path from $\neg x_i$ to x_i in the implication digraph G.*

In other words, as long as there are no bad cycles of the type shows in Figure 4, the CNF formula is satisfiable. Theorem 3.3 gives the following reduction from the problem 2SAT-Decision to the problem DistanceInGraph.

```
1 TwoSATDecSolve($\phi$)
  Input   : A CNF $\phi$ with width 2
  Output  : 1 if there is a satisfying assignment, else 0
2 Construct the implication graph $G = (V, E)$;
3 foreach $i = 0, \ldots, n-1$ do
4     Call the Oracle for DistanceInGraph on inputs $(G, x_i, \neg x_i)$; Output $\mathrm{dist}_{1,i}$
5     Call the Oracle for DistanceInGraph on inputs $(G, \neg x_i, x_i)$; Output $\mathrm{dist}_{2,i}$
6     if $\mathrm{dist}_{1,i} < \infty$ AND $\mathrm{dist}_{2,i} < \infty$ then return 0;
7 return 1;
```

This is a reduction

$$\text{2-SAT-Decision} \leq_{O(n),O(m)} \text{DistanceInGraph}.$$

Theorem 3.3 ensures the correctness of this reduction. The runtime follows from Line 3.

From here, we can prove Theorem 3.2 by noting that DistanceInGraph can be solved by BFS in time $O(n+m)$. Thus, the runtime of the algorithm for 2-SAT-Decision is $O(n(n+m))$, which is $O(nm)$ since $m = \Omega(n)$ (each variable occurs in at least one clause).

3.2 Proof of Theorem 3.1 and Theorem 3.3

We are close to proving Theorem 3.1, but not quite there yet. First, we haven't proved Theorem 3.3. Second, Theorem 3.2 cannot directly be used to prove Theorem 3.1, since the reduction from 2-SAT to 2-SAT-Decision incurs an additional time overhead of $O(n)$, leading to $O(n^2m)$ time algorithm for 2-SAT.

Here, we will solve both the issues together. We consider the following algorithm for 2-SAT, which extends upon `TwoSATDecSolve()`. In the process, Theorem 3.3 will be established via the proof of correctness of the algorithm.

```
1  TwoSATSolve($\phi$)
   Input   : A CNF $\phi$ with width 2
   Output  : A satisfying assignment, else $\perp$
2  Construct the implication graph $G = (V, E)$;
3  Set $(x_0, \ldots, x_{n-1})$ = (“unassigned”, …, “unassigned”);
4  foreach $i = 0, \ldots, n-1$ do
5      $\mathrm{dist}_{1,i}$ =BFS($G, x_i, \neg x_i$)
6      $\mathrm{dist}_{2,i}$ =BFS($G, \neg x_i, x_i$)
7      if $\mathrm{dist}_{1,i} < \infty$ AND $\mathrm{dist}_{2,i} < \infty$ then return $\perp$;
8  foreach $i = 0, \ldots, n-1$ do
9      if $x_i$ is assigned then $i = i + 1$ and GOTO line 8;
10     if $\mathrm{dist}_{1,i} = \infty$ AND $\mathrm{dist}_{2,i} < \infty$ then
11         Set $x_i = 1$
12         For all variables $x_j$ such that there is a path from $x_i$ to $x_j$, set $x_j = 1$.
13         For all variables $x_j$ such that there is a path from $x_i$ to $\neg x_j$, set $x_j = 0$.
14     if $\mathrm{dist}_{1,i} < \infty$ AND $\mathrm{dist}_{2,i} = \infty$ then
15         Set $x_i = 0$
16         For all variables $x_j$ such that there is a path from $\neg x_i$ to $x_j$, set $x_j = 1$.
17         For all variables $x_j$ such that there is a path from $\neg x_i$ to $\neg x_j$, set $x_j = 0$.
18     Delete all the edges incident to or from all assigned literals.
19 Assign the remaining variables such that the literals in each connected component of
     resulting graph $G$ get the same value.
20 return $(x_0, \ldots, x_{n-1})$;
```

Runtime: The loops on Lines 4 and 8 each run for at most n iterations. In each iteration, Lines 5 and 6 take $O(n + m)$ steps. Lines 12, 13, 15, and 16 take at most $O(n)$ steps since it suffices consider the vertices discovered by the BFS from x_i (Line 5) and $\neg x_i$ (Line 6). Line 18 takes $O(m)$ steps and Line 19 can be implemented in $O(n + m)$ steps (see section). Thus, the overall runtime is $O(n(n + m))$. Note that $m = \Omega(n)$, which simplifies that runtime to $O(nm)$.

Correctness: Lines 4-7 identify if there is variable such that “x_i implies $\neg x_i$” and “$\neg x_i$ implies x_i” (a *bad cycle* - see Figure 4). If so, the input formula is unsatisfiable, and we return so.

Suppose the algorithm does not abort (return $\perp$) at this stage. Then we claim that the algorithm finds a satisfying assignment. Lines 10 and 14 check whether any literal implies its negation: if $\neg x_i$ implies x_i, we set $x_i = 1$ (similar to Figure 3), and if x_i implies $\neg x_i$, we set $x_i = 0$. Since there were no *bad cycles* before, and we set variable values only according to implications, there are still no *bad cycles* and no false clauses. Lines 12, 13, 16, and 17 set the values of variables only according to implications, which don't create any bad cycles. Line 18 deletes only edges which are redundant (as ‘anything implies TRUE’ and ‘FALSE implies anything’). This step does not create any new bad cycles - if a digraph has no bad cycles, a subgraph cannot have a bad cycle. Further, this step removes any path from x_i to $\neg x_i$ or vice-versa (whichever existed).

After Line 18 has been executed, we have ensured that there are no paths between any two literals of the same variable, and there are still no bad cycles. Thus, the graph is divided into at least two connected components (similar to Figure 2), with only one variable for each literal in each connected component. Since 'FALSE implies FALSE' and 'TRUE implies TRUE', we can arbitrarily pick one of TRUE and FALSE to assign to each literal in a connected component: at the end of this, we've assigned a value to each variable and not violated any implications, that is, we've found a satisfying assignment. Note that this also proves Theorem 3.3.

Beyond 2-SAT: In the upcoming lectures, we will see that there is likely no polynomial time algorithm for k-SAT, for any $k \geq 3$. As remarked earlier, SAT solvers still do pretty good job at solving real world instances. In CS 124, an algorithm for 3-SAT is discussed that takes $O(1.34^n)$ time.