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

Lecture 16: Resolution

Harvard SEAS - Fall 2022 *Oct. 27, 2022*

1 Announcements

- PS7 posted, due 11/9 (week after next)
- Sender-Receiver Exercise Tue 11/1: Senders and Receivers should all prepare.
- Adam & Salil OH after class 11:15-12:15, SEC 2.122
- For more individualized help in OH, come to the underloaded ones: the earliest ones on Mon, Tue, Thu, Sun.
- Participation & learning portfolio highlights 1 due Sat.

2 Recap

- A *literal* is
- A boolean formula is in *conjunctive normal form (CNF)* if
- It will be convenient to also allow 1 (true) to be a clause. 0 (false) is already a clause:

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)

Motivation for SAT (and logic problems in general): can encode many other problems of interest

- Graph Coloring (last time)
- Longest Path (SRE 4)
- Independent Set (section)
- Programming Team (ps7)
- Program Analysis (lec24)
- and much more (lec19)

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!

3 Resolution

Definition 3.1 (clause simplification)**.** Given a clause B, Simplify(B) returns 1 if B contains both a literal and its negation, and otherwise removes duplicates of literals from B and sorts the variables in B according to a fixed ordering on the variables (e.g. x_0, x_1, ...).

Examples:

Definition 3.2 (resolution rule)**.** For clauses C and D, define

$$C \diamond D = \begin{cases} \text{Simplify}((C - \ell) \vee (D - \neg\ell)) & \text{if } \ell \text{ is a literal s.t. } \ell \in C \text{ and } \neg\ell \in D \\ 1 & \text{if there is no such literal } \ell \end{cases}$$

Here $C - \ell$ means remove literal ℓ from clause C.

Examples:

$$(x_0 \vee \neg x_1 \vee x_3 \vee \neg x_5) \diamond (x_1 \vee \neg x_4 \vee \neg x_5) =$$

For singleton clauses:

$$(x_0) \diamond (\neg x_0) =$$

We could also have a clause that appears to be resolvable in two ways, either resolving on x_0 or x_4:

$$(x_0 \vee x_1 \vee \neg x_4) \diamond (\neg x_0 \vee x_2 \vee x_4) = \qquad \text{OR } =$$

The motivation for this definition is the following property:

Lemma 3.3. *If an assignment α satisfies clauses C and D, then α satisfies $C \diamond D$.*

Proof.

□

From now on, it will be useful to view a CNF formula as just a set $\mathcal{C}$ of clauses.

Definition 3.4 (Satisfiability)**.** Let $\mathcal{C}$ be a set of clauses over variables $x_0, \ldots, x_{n-1}$. We say that an assignment $\alpha \in \{0,1\}^n$ *satisfies* $\mathcal{C}$ if α satisfies all of the clauses in $\mathcal{C}$, or equivalently α satisfies the CNF formula

$$\varphi(x_0, \ldots, x_{n-1}) =$$

A corollary of Lemma 3.3 is the following:

Lemma 3.5. *Let $\mathcal{C}$ be a set of clauses and let $C, D \in \mathcal{C}$. Then $\mathcal{C}$ and $\mathcal{C} \cup \{C \diamond D\}$ have the same set of satisfying assignments. In particular, if $C \diamond D$ is the empty clause, then $\mathcal{C}$ is unsatisfiable.*

This gives rise to the *resolution meta-algorithm* for deciding satisfiability of a CNF formula φ. We keep adding resolvents until we find the empty clause (in which case we know φ is unsatisfiable by Lemma 3.5) or cannot generate any more new clauses.

There are many variants of resolution, based on different ways of choosing the order in which to resolve clauses. We give a particular version below, where starting with $\varphi = C_0 \wedge C_1 \wedge \cdots \wedge C_{m-1}$, we simplify all the clauses in φ and then:

1. Resolve C_0 with each of $C_1, \ldots, C_{m-1}$, adding any new clauses obtained from the resolution $C_m, C_{m+1}, \ldots$
2. Resolve C_1 with each of $C_2, \ldots, C_{m-1}$ as well as with
3. Resolve C_2 with each of $C_3, \ldots, C_{m-1}$ as well as with
4. etc.

Note that this process will resolve every pair of clauses, except for resolving C_i with resolvents of the form $C_i \diamond C_j$ for $j > i$. Omitting the latter is harmless by the following lemma:

Lemma 3.6. *For all clauses C and D, $C \diamond (C \diamond D) = 1$*

Proof. Exercise. □

Examples:

$$\phi(x_0, x_1, x_2) = (\neg x_0 \vee x_1) \wedge (\neg x_1 \vee x_2) \wedge (x_0 \vee x_1 \vee x_2) \wedge (\neg x_2)$$

For a second example:

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

In pseudocode:

```
1  ResolutionInOrder(φ)
   Input  : A CNF formula φ(x_0, ..., x_{n-1})
   Output : Whether φ is satisfiable or unsatisfiable
2  Let C_0, C_1, ..., C_{m-1} be the clauses in φ, after simplifying each clause;
3  i = 0 ;                  /* clause to resolve with others in current iteration */
4  f = m ;      /* start of 'frontier' - new resolvents from current iteration */
5  g = m ;                                                  /* end of frontier */
6  while f > i + 1 do
7  |  foreach j = i + 1 to f − 1 do
8  |  |  R = C_i ⋄ C_j;
9  |  |  if R = 0 then return unsatisfiable;
10 |  |  else if R ∉ {C_0, C_1, ..., C_{g-1}} then
11 |  |  |  C_g = R;
12 |  |  |  g = g + 1;
13 |  f = g;
14 |  i = i + 1
15 return satisfiable
```

Algorithm 15 raises two questions:

1. (Termination) Why does resolution always terminate? And what is its runtime?
2. (Correctness) Is Algorithm 15 correct? If it ever derives the empty clause $R = 0$, we know that φ is unsatisfiable (why?) but if never generates the empty clause, can we be sure that φ is satisfiable?

4 Termination and Efficiency

Q: Why does resolution terminate?

A:

Q: What is the runtime of resolution?

A:

However, in many cases, there is a *short* proof of unsatisfiability that resolution will find. One case is for the 2-SAT problem, defined as follows:

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

Q: What is the runtime of Resolution for 2-SAT?

A:

5 Correctness and Assignment Extraction

We will prove:

Theorem 5.1. *Algorithm 15 is a correct algorithm for deciding SAT, and when it outputs* `satisfiable`*, a satisfying assignment can be extracted from the final set $\mathcal{C}_{fin}$ of clauses it produces in time $O(n + k_{fin} \cdot |\mathcal{C}_{fin}|)$, where k_{fin} is the maximum size among the clauses in $\mathcal{C}_{fin}$.*

Corollary 5.2. *2-SAT can be solved in time*

To prove Theorem 5.1, we rely on the key property of the set of clauses generated by Resolution:

Definition 5.3. Let $\mathcal{C}$ be a set of clauses over variables $x_0, \ldots, x_{n-1}$. We say that $\mathcal{C}$ is *closed* if

Observe that if $\mathcal{C}$ is a closed, nonempty set of clauses, then $1 \in \mathcal{C}$, since $C \diamond C = 1$ for any $C \in \mathcal{C}$.

Lemma 5.4. *Let $\mathcal{C}_{fin}$ be the final set of clauses in any execution of Algorithm 15 that outputs* `satisfiable`*. Then $\mathcal{C}_{fin} \cup \{1\}$ is closed.*

Proof idea.

□

It turns out that closed sets of clauses that don't contain the empty clause are always satisfiable:

Lemma 5.5. *Let $\mathcal{C}$ be a closed set of clauses on n variables, each of width at most k, such that $0 \notin \mathcal{C}$. Then an assignment that satisfies $\mathcal{C}$ exists and can be found in time $O(n + k \cdot |\mathcal{C}|)$.*

Proof idea. We generate our satisfying assignment one variable v at a time:

1. If $\mathcal{C}$ contains a singleton clause (v), then
2. If it contains $(\neg v)$ then
3. If it contains neither (v) nor $(\neg v)$, then
4. $\mathcal{C}$ cannot contain both (v) and $(\neg v)$, because

Once we have assigned a variable to a value, we set that variable's value in every clause and simplify. Crucially, we argue that even after assigning the variable, the set of clauses (a) does not contain 0, and (b) remains closed. (a) holds because of how we set v. Intuitively, (b) holds because assigning v and then resolving two resulting clauses C' and D' is equivalent to first resolving the original clauses C and D and then assigning v. We know that $C \diamond D \in \mathcal{C}$ by closure of $\mathcal{C}$, so we have $C' \diamond D'$ after assigning v. □

Example: Consider applying this procedure to the set of clauses derived from the formula ψ above:

$$(\neg x_0 \vee x_3), (x_0 \vee \neg x_3), (\neg x_1 \vee x_2), (\neg x_2 \vee x_1), (\neg x_3), (\neg x_0)$$

Lemmas 5.4 and 5.5 imply Theorem 5.1.

6 SAT Solvers

Enormous effort has gone into designing SAT Solvers that perform well on many real-world satisfiability instances, often but not always avoiding the worst-case exponential complexity. These methods are very related to resolution. In some sense, they can be viewed as interleaving the assignment extraction procedure and resolution steps, in the hope of quickly finding either a satisfying assignment or a proof of unsatisfiability. For example, they start by assigning a variable (say x_0) to a value $\alpha_0 = 0$. Recursing, they may discover that setting $x_0 = 0$ makes the formula unsatisfiable, in which case they backtrack and try $x_0 = 1$. But in the process of discovering the unsatisfiability of $\mathcal{C}|_{x_0=\alpha_0}$, they may discover many new clauses (by resolution) and these can the translated to resolvents of $\mathcal{C}$ (in a manner similar to Lemma **??** below). These new "learned clauses" then can help improve the rest of the search. Many other heuristics are used, such as always setting a variable v as soon as a unit clause (v) or $(\neg v)$ is derived, and carefully selecting which variables and clauses to process next.

7 Formalizing Assignment Extraction

This section is optional reading, to give you more precision and proof details about the assignment extraction procedure.