CS1200: Intro. to Algorithms and their Limitations Anshu & Vadhan

Lecture 15: Logic

Harvard SEAS - Fall 2024 *2024-10-24*

1 Announcements

- SRE next class (10/29). Remember to prepare and come on time!
- Salil's OH after class today (no OH next week due to conference travel), Anurag Zoom OH tomorrow 1:30-2:30.
- Remember to *acknowledge all sources* on your problem sets, including students not currently in the class and ChatGPT/AI.
- Problem Set 6 distributed, due 10/30.
- Reflection summaries
  1. PS3: most of you found RAM/WordRAM Models useful in concretizing constant-time operations and connecting to real-world computing, though some (10+) didn't see value in covering it.
  2. PS4: Top two topics that you all wanted to study more (pre-midterm) were RAM/Word-RAM and BSTs. Also want better understanding of what constitutes a rigorous proof, how to perform reductions, and intuition for randomness being useful.
  3. SRE3 (connected components): Found to be our hardest SRE so far. Students felt that examples and diagrams would have been helpful. We are hoping that you (senders) will try to come up with them!
  4. SRE4 (coloring): You all seemed to enjoy this one, and benefitted from past experience with SREs. Again visuals very helpful!

Recommended Reading (good for practice problems, too):

- Lewis–Zax Ch. 9–10.
- Roughgarden IV, Sec. 21.5, Ch. 24.

2 Propositional Logic

Motivation: Having completed our unit on graph algorithms, we are now going to study computational problems and algorithms related to logic. The motivation for doing so is twofold. First, logic is a fundamental building block of computation (e.g. the digital circuits that make up our computer hardware), and second, it is also a very expressive language for encoding computational problems we want to solve.

Definition 2.1 (boolean formulas, informal)**.** A *boolean formula* φ is a formula built up from a finite set of variables, say $x_0, \ldots, x_{n-1}$, using the logical operators $\wedge$ (AND), $\vee$ (OR), and $\neg$ (NOT), and parentheses.

Every boolean formula φ on n variables defines a boolean function, which we'll abuse notation and also denote by $\varphi : \{0,1\}^n \to \{0,1\}$, where we interpret 0 as false and 1 as true, and give $\wedge, \vee, \neg$ their usual semantics (meaning).

The Lewis–Zax text contains a formal, inductive definitions of boolean formulas and the corresponding boolean functions.

Examples:

$$\varphi_{maj}(x_0, x_1, x_2) = (x_0 \wedge x_1) \vee (x_1 \wedge x_2) \vee (x_2 \wedge x_0)$$

is a boolean formula. It evaluates to 1 if at least two of its inputs are 1. For instance, $\varphi_{maj}(1,1,0) = 1$ and $\varphi_{maj}(1,0,0) = 0$.

$$\varphi_{pal}(x_0, x_1, x_2, x_3) = ((x_0 \wedge x_3) \vee (\neg x_0 \wedge \neg x_3)) \wedge ((x_1 \wedge x_2) \vee (\neg x_1 \wedge \neg x_2))$$

is a boolean formula. It evaluates to 1 if the input $x_0x_1x_2x_3$ is a palindrome.

Definition 2.2 (DNF and CNF formulas)**.**

- A *literal* is a variable (e.g. x_i) or its negation ($\neg x_i$).
- A *term* is an AND of a sequence of literals.
- A *clause* is an OR of a sequence of literals.
- A boolean formula is in *disjunctive normal form (DNF)* if it is the OR of a sequence of terms.
- A boolean formula is in *conjunctive normal form (CNF)* if it is the AND of a sequence of clauses.

By convention, an empty term is always true and an empty clause is always false. (**Q:** Why is this a sensible convention?)

Q: For exach of the examples above, is it in DNF or CNF?

A: φ_{maj} is in DNF. φ_{pal} is neither.

Simplifying clauses. Note terms and clauses may contain duplicate literals, but if a term or clause contains multiple copies of a variable x, it's equivalent to a term or clause with just one copy (since $x \vee x = x$ and $x \wedge x = x$). We can also remove any clause or term with both a variable x and its negation $\neg x$, as that clause or term will be always true (in the case of a clause) or always false (in the case of a term). We define a function Simplify which takes a clause and performs those simplifications: that is, given a clause B, Simplify(B) removes duplicates of literals from clause B, and returns 1 if B contains both a literal and its negation. Also, if we have an order on variables (e.g. x_0, x_1, ...), Simplify(B) also sorts the literals in order of their variables.

One reason that DNF and CNF are used so commonly is that they can express all boolean functions:

Lemma 2.3. *For every boolean function $f : \{0,1\}^n \to \{0,1\}$, there are boolean formulas φ and ψ in DNF and CNF, respectively, such that $f \equiv \varphi$ and $f \equiv \psi$, where we use $\equiv$ to indicate equivalence as functions, i.e. $f \equiv g$ iff $\forall x : f(x) = g(x)$.*

Proof.
For a function $f : \{0,1\}^n \to \{0,1\}$, we can define the DNF

$$\varphi(x_0, \ldots, x_{n-1}) = \bigvee_{\alpha \in \{0,1\}^n : f(\alpha)=1} ((x_0 = \alpha_0) \wedge (x_1 = \alpha_1) \wedge \cdots \wedge (x_{n-1} = \alpha_{n-1})) .$$

Note that $x_i = \alpha_i$ can be rewritten as either x_i (with $\alpha_i = 1$) or $\neg x_i$ (with $\alpha_i = 0$). For example, applying to the palindrome function on 4 bits, we get

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

where the terms correspond to the satisfying assignments $(1,1,1,1)$, $(1,0,0,1)$, $(0,1,1,0)$, and $(0,0,0,0)$.

For the CNF, we define

$$\varphi(x_0, \ldots, x_{n-1}) = \bigwedge_{\alpha \in \{0,1\}^n : f(\alpha)=0} ((x_0 \neq \alpha_0) \vee (x_1 \neq \alpha_1) \vee \cdots \vee (x_{n-1} \neq \alpha_{n-1})) .$$

$\square$

Example: The majority function on 3 bits can be written in CNF as follows:

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

with the clauses corresponding to the 4 non-satisfying assignments $(0,0,0)$, $(1,0,0)$, $(0,1,0)$, $(0,0,1)$. This example shows that the DNF and CNF given by the general construction are not necessarily the smallest ones possible for a given function, as the majority function can also be expressed by the following simpler CNF formula:

$$(x_0 \vee x_1) \wedge (x_0 \vee x_2) \wedge (x_1 \vee x_2).$$

3 Computational Problems in Propositional Logic

Here are three natural computational problems about Boolean formulas:

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

Computational Problem Satisfiability

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

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

Computational Problem DNF-Satisfiability

Q: One of these problems is algorithmically very easy. Which one?

DNF satisfiability is easy - since we only need to satisfy a single term and then we are done, the only cases in which a DNF is unsatisfiable are when we have zero terms or every term has a contradiction (both a variable and its negation). On the other hand, it is known that Satisfiability can be reduced to CNF-satisfiability[1], so when we say "SAT," we refer to CNF-Satisfiability by default whenever it's convenient.

4 Modelling using Satisfiability

One of the reasons for the importance of Satisfiability is its richness in encoding other problems. Thus any effort gone into optimizing algorithms for (CNF-)Satisfiability (aka "SAT Solvers") can be easily be applied to other problems we want to solve.

Theorem 4.1. *Graph k-Coloring on graphs with n nodes and m edges can be reduced in time $O(n + km)$ to (CNF-)Satisfiability with kn variables and $n + km$ clauses.*

Proof. Given $G = (V, E)$ and $k \in \mathbb{N}$, we will construct a CNF ϕ_G that captures the graph k-coloring problem. In ϕ_G, we introduce *indicator variables* $x_{v,i}$ where $v \in V$ and $i \in [k]$, which intuitively are meant to correspond to vertex v being assigned to color i.

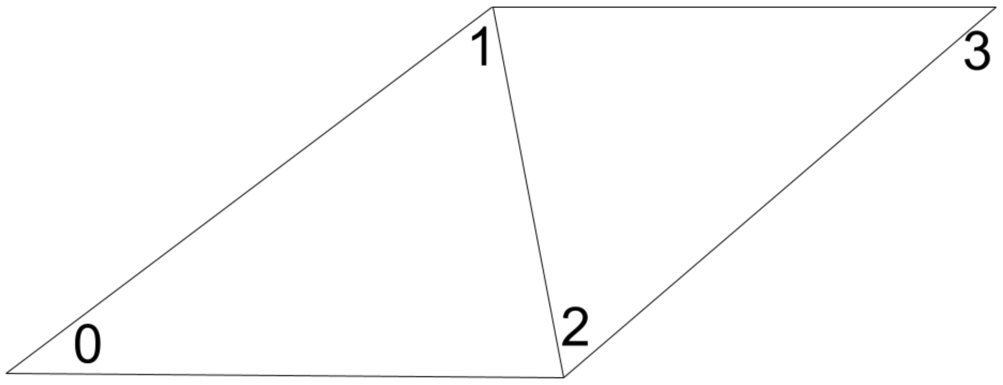
We then have a few types of clauses:

1. $(x_{v,0} \vee x_{v,1} \vee \cdots \vee x_{v,k-1})$ for all $v \in V$. (Each vertex must be assigned a color)

2. $(\neg x_{u,i} \vee \neg x_{v,i})$ for every edge $\{u, v\} \in E$ and every color $i \in [k]$. From De Morgan's Law, this is equivalent to the more intuitive $\neg(x_{u,i} \wedge x_{v,i})$. (The endpoints of an edge cannot be assigned the same color.)

3. In addition, we can require all vertices to be assigned at most one color. (The constraints above allow a satisfying assignment to assign multiple colors to the same vertex.) To avoid this, we can include clauses $(\neg x_{v,i} \vee \neg x_{v,j})$ for $i, j \in [k]$ where $i \neq j$. But for the case of coloring, we don't actually need to include these, and doing so would increase the number of clauses to nk^2.

For instance, if G is the graph below and $k = 3$,



[1] If you want to try to figure out the reduction, we recommend looking at the Lewis–Zax text's formal inductive definition of Boolean formulas.

then we make the following SAT instance ϕ_G:

$$(x_{0,0} \vee x_{0,1} \vee x_{0,2}) \wedge (x_{1,0} \vee x_{1,1} \vee x_{1,2}) \wedge (x_{2,0} \vee x_{2,1} \vee x_{2,2}) \wedge (x_{3,0} \vee x_{3,1} \vee x_{3,2}) \wedge$$
$$(\neg x_{0,0} \vee \neg x_{1,0}) \wedge (\neg x_{0,1} \vee \neg x_{1,1}) \wedge (\neg x_{0,2} \vee \neg x_{1,2}) \wedge$$
$$(\neg x_{0,0} \vee \neg x_{2,0}) \wedge (\neg x_{0,1} \vee \neg x_{2,1}) \wedge (\neg x_{0,2} \vee \neg x_{2,2}) \wedge$$
$$(\neg x_{2,0} \vee \neg x_{1,0}) \wedge (\neg x_{2,1} \vee \neg x_{1,1}) \wedge (\neg x_{2,2} \vee \neg x_{1,2}) \wedge$$
$$(\neg x_{3,0} \vee \neg x_{1,0}) \wedge (\neg x_{3,1} \vee \neg x_{1,1}) \wedge (\neg x_{3,2} \vee \neg x_{1,2}) \wedge$$
$$(\neg x_{2,0} \vee \neg x_{3,0}) \wedge (\neg x_{2,1} \vee \neg x_{3,1}) \wedge (\neg x_{2,2} \vee \neg x_{3,2})$$

We then call the SAT oracle on ϕ_G and get an assignment α. If $\alpha = \perp$, we say G is not k-colorable. Otherwise, we construct and output the coloring f_α given by:

$$f_\alpha(v) = \min\{i \in [k] : \alpha_{v,i} = 1\}.$$

(The minimum is just being used to pick out one of the colors i such that $\alpha_{v,i} = 1$ in case there are multiple, since we didn't include Clauses of Type 3.)

The runtime essentially follows from our description. If we start by removing isolated vertices (i.e. those with no adjacent edges), we have $n' \leq 2m$ actual vertices under consideration in our SAT clause, and so have a runtime of $O(n) + O(n'k) + O(mk) = O(n + km)$.

For correctness, we make two claims:

Claim 4.2. *If G has a valid k coloring, ϕ_G is satisfiable.*

$\implies$ don't incorrectly output $\perp$.

Claim 4.3. *If α satisfies ϕ_G, then f_α is a proper k-coloring of G.*

$\implies$ if we output a coloring, it will be proper.

Both of these claims are worth checking. Note that f_α is *well-defined* because α satisfies clauses of type 1 and is *proper* due to clauses of type 2. $\square$

We remark that introducing "indicator variables" like the $x_{v,i}$ is a common technique in reductions to SAT to represent k-way choices by boolean variables. When reducing other problems to SAT, it is sometimes important to add clauses enforcing that for each v, only one of the $x_{v,i}$'s are true, even though this wasn't necessary in the case of k-coloring.

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 $k = 23$ (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 — similarly to how you are viewing interval scheduling on ps6). 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!

Thus motivated, next time we will turn to algorithms for Satisfiability, to get a taste of some of the ideas that go into SAT Solvers.