Lecture 4

Polynomial-time formula classes

Horn-SAT, 2-SAT, X-SAT, Walk-SAT

Dr Christoph Haase
University of Oxford
(with small changes by Javier Esparza)

Recap and some additional notation

- A **literal** is a propositional variable or the negation of a propositional variable:

$$x \text{ or } \neg x$$

- We call x a **positive literal** and $\neg x$ a **negative literal**
- A disjunction of literals is a **clause**
- A formula F is in **conjunctive normal form** (**CNF**) if it is a conjunction of disjunctions of literals $L_{i,j}$:

$$F = \bigwedge_{i=1}^{n} (\bigvee_{j=1}^{m_i} L_{i,j})$$

- Convention: *true* is CNF with no clauses, *false* is CNF with a single empty clause without literals

Agenda

1. **Polynomial-time fragments of propositional logic**

2. **Walk-SAT: A randomised algorithm for satisfiability**

The satisfiability problem

"**SAT is bad**": Only method so far to solve SAT is truth tables, which takes exponential time in worst case.

The satisfiability problem

"**SAT is bad**": Only method so far to solve SAT is truth tables, which takes exponential time in worst case.

But: can often do better for formulas of special form:

- **Horn formulas**: SAT can be decided in polynomial time
- **2-CNF formulas**: SAT can be decided in polynomial time
- **X-CNF formulas**: SAT can be decided in polynomial time

Horn formulas

Definition

A CNF formula is a **Horn formula** if each clause contains at most one positive literal.

Horn formulas

Definition

A CNF formula is a **Horn formula** if each clause contains at most one positive literal.

- An example of a Horn formula:

$$p_1 \wedge (\neg p_2 \vee \neg p_3) \wedge (\neg p_1 \vee \neg p_2 \vee p_4)$$

Horn formulas

Definition

A CNF formula is a **Horn formula** if each clause contains at most one positive literal.

- An example of a Horn formula:

$$p_1 \wedge (\neg p_2 \vee \neg p_3) \wedge (\neg p_1 \vee \neg p_2 \vee p_4)$$

- Horn formulas can be rewritten in a more intuitive way as conjunctions of implications, called **implication form**. E.g.:

$$(\mathit{true} \to p_1) \wedge (p_2 \wedge p_3 \to \mathit{false}) \wedge (p_1 \wedge p_2 \to p_4)$$

Horn formulas

Definition

A CNF formula is a **Horn formula** if each clause contains at most one positive literal.

- An example of a Horn formula:

$$p_1 \wedge (\neg p_2 \vee \neg p_3) \wedge (\neg p_1 \vee \neg p_2 \vee p_4)$$

- Horn formulas can be rewritten in a more intuitive way as conjunctions of implications, called **implication form**. E.g.:

$$(\mathit{true} \to p_1) \wedge (p_2 \wedge p_3 \to \mathit{false}) \wedge (p_1 \wedge p_2 \to p_4)$$

Horn formulas have many computer science applications: Programming languages Prolog and Datalog based on them.

Horn-SAT algorithm

Can decide **satisfiability** for Horn formulas in polynomial time!
Idea:

- maintain valuation $\mathcal{A}$ on propositional variables in formula F, starting with $p \mapsto 0$
- update $\mathcal{A}(p_i)$ from 0 to 1 until either F satisfied or contradiction reached

Horn-SAT algorithm

Can decide **satisfiability** for Horn formulas in polynomial time!
Idea:

- maintain valuation $\mathcal{A}$ on propositional variables in formula F, starting with $p \mapsto 0$
- update $\mathcal{A}(p_i)$ from 0 to 1 until either F satisfied or contradiction reached

INPUT: Horn formula F
$T := \emptyset$
while T does not satisfy F **do**
begin
  pick an unsatisfied clause $p_1 \wedge \cdots \wedge p_k \to G$
  if G is a variable **then** $T := T \cup \{G\}$
  if $G = \mathit{false}$ **then return** UNSAT
end
return T

Horn-SAT algorithm: correctness

- Encoding $T = \{p_i \mid \mathcal{A}(p_i) = 1\}$.
- Order valuations by $\mathcal{A} \leq \mathcal{B}$ when $\mathcal{A}(p_i) \leq \mathcal{B}(p_i)$ for each i
- Each iteration changes $\mathcal{A}(p_i)$ from 0 to 1
- There are at most n iterations, so overall polynomial time
- Any $\mathcal{A}$ returned must satisfy F by termination condition
- If UNSAT returned then F is unsatisfiable:
  - If $\mathcal{B}$ satisfies F, then $\mathcal{A} \leq \mathcal{B}$ is a **loop invariant**:
  - Consider implication $p_1 \wedge \cdots \wedge p_k \to G$ not satisfied by $\mathcal{A}$.
    Then $\mathcal{A}$ satisfies $p_1, \ldots, p_k$ but not G, so $\mathcal{B} \models p_1 \wedge \ldots \wedge p_k$.
    But then $\mathcal{B} \models G$, so $G \neq \mathit{false}$; contradiction.
    Moreover, $\mathcal{B}(G) = 1$ so $\mathcal{A}(G) := 1$ preserves invariant.

2-CNF formulas

Definition

A **2-CNF formula**, or **Krom formula** is a CNF formula F such that every clause has at most two literals.

2-CNF formulas

Definition

A **2-CNF formula**, or **Krom formula** is a CNF formula F such that every clause has at most two literals.

- For a literal L, define $\overline{L} := \begin{cases} p & \text{if } L = \neg p \\ \neg p & \text{otherwise} \end{cases}$

2-CNF formulas

Definition

A **2-CNF formula**, or **Krom formula** is a CNF formula F such that every clause has at most two literals.

- For a literal L, define $\overline{L} := \begin{cases} p & \text{if } L = \neg p \\ \neg p & \text{otherwise} \end{cases}$
- The **implication graph** of a 2-CNF formula F is a directed graph $\mathcal{G} = (V, E)$, where

$$V := \{p_1, p_2, \ldots, p_n\} \cup \{\neg p_1, \neg p_2, \ldots, \neg p_n\},$$

with $p_1, p_2, \ldots, p_n$ the propositional variables mentioned in F. For each pair of literals L and M, there is an edge (L, M) iff the clause $(\overline{L} \vee M)$ or $(M \vee \overline{L})$ appears in F.

2-CNF formulas

Definition

A **2-CNF formula**, or **Krom formula** is a CNF formula F such that every clause has at most two literals.

- For a literal L, define $\overline{L} := \begin{cases} p & \text{if } L = \neg p \\ \neg p & \text{otherwise} \end{cases}$
- The **implication graph** of a 2-CNF formula F is a directed graph $\mathcal{G} = (V, E)$, where

$$V := \{p_1, p_2, \ldots, p_n\} \cup \{\neg p_1, \neg p_2, \ldots, \neg p_n\},$$

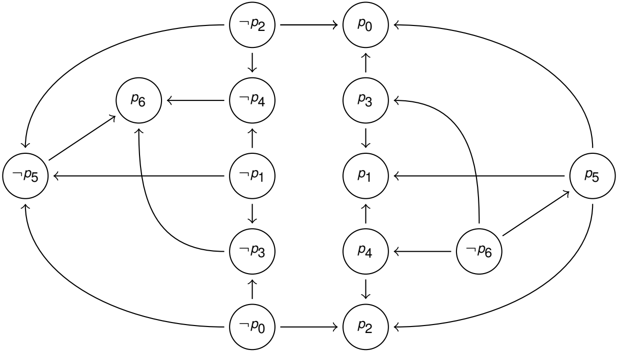
  with $p_1, p_2, \ldots, p_n$ the propositional variables mentioned in F. For each pair of literals L and M, there is an edge (L, M) iff the clause $(\overline{L} \vee M)$ or $(M \vee \overline{L})$ appears in F.
- Example: clause $x \vee y$ requires edge $(\neg y, x)$ in $\mathcal{G}$ (alternatively $(\neg x, y)$)

2-CNF formulas: example

$$(p_0 \vee p_2) \wedge (p_0 \vee \neg p_3) \wedge (p_1 \vee \neg p_3) \wedge (p_1 \vee \neg p_4) \wedge (p_2 \vee \neg p_4)$$
$$\wedge (p_0 \vee \neg p_5) \wedge (p_1 \vee \neg p_5) \wedge (p_2 \vee \neg p_5) \wedge (p_3 \vee p_6) \wedge (p_4 \vee p_6) \wedge (p_5 \vee p_6)$$

2-CNF formulas: example

$$(p_0 \lor p_2) \land (p_0 \lor \neg p_3) \land (p_1 \lor \neg p_3) \land (p_1 \lor \neg p_4) \land (p_2 \lor \neg p_4)$$
$$\land (p_0 \lor \neg p_5) \land (p_1 \lor \neg p_5) \land (p_2 \lor \neg p_5) \land (p_3 \lor p_6) \land (p_4 \lor p_6) \land (p_5 \lor p_6)$$



- Paths in $\mathcal{G}$ correspond to chains of implications.
- Edge $(\overline{M}, \overline{L})$ is *contrapositive* implication $\overline{M} \to \overline{L}$ corresponding to (L, M)

2-SAT

- Can reduce **satisfiability** for 2-CNF formulas to *reachability* problem of implication graph, which is solvable in *linear* time.
- Implication graph $\mathcal{G}$ is **consistent** if there is no propositional variable p with paths from p to $\neg p$ and from $\neg p$ to p.

- Can reduce **satisfiability** for 2-CNF formulas to *reachability* problem of implication graph, which is solvable in *linear* time.
- Implication graph $\mathcal{G}$ is **consistent** if there is no propositional variable p with paths from p to $\neg p$ and from $\neg p$ to p.

Theorem

A 2-CNF formula F is satisfiable iff its implication graph $\mathcal{G}$ is consistent.

Proof.

($\Rightarrow$) If $\mathcal{G}$ not consistent, there are paths $\neg p \rightarrow p$, $p \rightarrow \neg p$. So $\mathcal{A} \models F$ would imply $\mathcal{A}(p) \leq \mathcal{A}(\neg p) \leq \mathcal{A}(p)$.

($\Leftarrow$) Construct a satisfying assignment. $\square$

2-SAT Algorithm

INPUT: 2-CNF formula F
$\mathcal{A} :=$ empty valuation
while there is some unassigned variable **do**
  begin
  pick a literal L for which there is no path from L to $\overline{L}$, and
  set $\mathcal{A}(L) := 1$
  while there is an edge (M, N) with $\mathcal{A}(M) = 1$ and $\mathcal{A}(N)$ is undefined
    do $\mathcal{A}(N) := 1$
  end
return $\mathcal{A}$

2-SAT Algorithm: correctness

- Outer loop invariant: any node reachable from a true node is also true
  - If outer invariant holds and all variables assigned, we have a satisfying assigment.

2-SAT Algorithm: correctness

- Outer loop invariant: any node reachable from a true node is also true
  - If outer invariant holds and all variables assigned, we have a satisfying assigment.
- Inner loop invariant: no path from true node to false node

2-SAT Algorithm: correctness

- Outer loop invariant: any node reachable from a true node is also true
  - If outer invariant holds and all variables assigned, we have a satisfying assigment.
- Inner loop invariant: no path from true node to false node
- If outer invariant holds but not all variables assigned, there is unassigned literal L with no path $L \to \overline{L}$ (by consistency)

2-SAT Algorithm: correctness

- Outer loop invariant: any node reachable from a true node is also true
  - If outer invariant holds and all variables assigned, we have a satisfying assigment.
- Inner loop invariant: no path from true node to false node
- If outer invariant holds but not all variables assigned, there is unassigned literal L with no path $L \to \overline{L}$ (by consistency)
- After updating $\mathcal{A}(L) := 1$, inner invariant holds

2-SAT Algorithm: correctness

- Outer loop invariant: any node reachable from a true node is also true
  - If outer invariant holds and all variables assigned, we have a satisfying assigment.
- Inner loop invariant: no path from true node to false node
- If outer invariant holds but not all variables assigned, there is unassigned literal L with no path $L \to \overline{L}$ (by consistency)
- After updating $\mathcal{A}(L) := 1$, inner invariant holds
- Inner loop maintains invariant, so when it terminates every node reachable from a true node is true

3-CNF formulas

2-SAT solvable in polynomial time, 3-SAT not unless P=NP

- A **3-CNF** formula is a CNF one with $\leq$ 3 literals per clause
- Tseytin's transformation: for an arbitrary formula F, we can compute an equisatisfiable 3-CNF formula G in polynomial time.
- So a polynomial-time algorithm for 3-SAT would give us a polynomial algorithm for SAT.

XOR-CNF formulas

Can think of propositional logic as linear algebra over $\{0, 1\}$.

- **XOR-clause** is exclusive-or of literals.
  X-CNF formula is conjunction of XOR-clauses

$$F = (p_1 \oplus p_3) \wedge (\neg p_1 \oplus p_2) \wedge (p_1 \oplus p_2 \oplus \neg p_3)$$

XOR-CNF formulas

Can think of propositional logic as linear algebra over $\{0, 1\}$.

- **XOR-clause** is exclusive-or of literals.
  X-CNF formula is conjunction of XOR-clauses

$$F = (p_1 \oplus p_3) \wedge (\neg p_1 \oplus p_2) \wedge (p_1 \oplus p_2 \oplus \neg p_3)$$

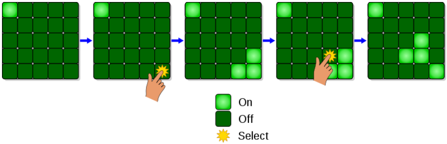
- Rewrite as system of equations over $\mathbb{Z}_2$ and solve

$$\begin{array}{ccccccc} p_1 & & & + & p_3 & = & 1 \\ p_1 & + & p_2 & & & = & 0 \\ p_1 & + & p_2 & + & p_3 & = & 0 \end{array}$$

- So X-SAT reduces to **Gaussian elimination**,
  which is solvable in *cubic* time

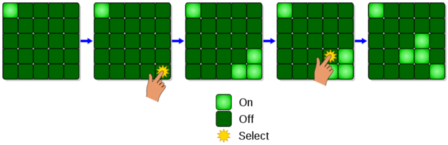
Lights out

Given: an $N \times N$ grid, each button coloured black or white.
Move: pressing a button inverts colours of its neighbours.
Goal: end up with all buttons black.
Question: translate to X-SAT.



Lights out

Given: an $N \times N$ grid, each button coloured black or white.
Move: pressing a button inverts colours of its neighbours.
Goal: end up with all buttons black.
Question: translate to X-SAT.



- Even number of same moves doesn't do anything
- Let variable $p_{i,j}$ denote whether button (i,j) is pressed
- Valuations of formula

$$\bigwedge_{1 \leq i,j \leq N} (p_{i,j} \oplus p_{i \oplus 1,j} \oplus p_{i \ominus 1,j} \oplus p_{i,j \oplus 1} \oplus p_{i,j \ominus 1})$$

correspond to solutions of the puzzle.

Randomised algorithms

Randomised algorithm decides satisfiability for CNF formulas.
Takes polynomial time on 2-CNF formulas.

- Guess assignment uniformly at random
- While there is unsatisfied clause F, pick literal and flip its truth value
- If no satisfying assignment after r steps, return UNSAT.

Idea: if formula unsatisfiable, algorithm will say so.

But algorithm could halt before finding satisfying assignment.

Want parameter r large enough so that this probability is small.

Walk-SAT

Input: CNF formula F with n variables, repetition parameter r
pick a random assignment
repeat r times
  Pick an unsatisfied clause
  Pick literal in the clause uniformly at random, and flip value
  if F is satisfied **then** return the current assignment
return UNSAT

Walk-SAT: analysis

- Let F be 2-CNF formula with satisfying assignment $\mathcal{A}$
  Will bound expected number of flips to find $\mathcal{A}$.

Walk-SAT: analysis

- Let F be 2-CNF formula with satisfying assignment $\mathcal{A}$
  Will bound expected number of flips to find $\mathcal{A}$.
- Distance between assignments := $\#$variables where differ

$$T_i := \max\{\mathbf{E}[\#\text{flippings } \mathcal{A} \to \mathcal{B}] \mid \text{distance}(\mathcal{A}, \mathcal{B}) = i\}$$

- Then T_n is time it takes to find satisfying assignment $\mathcal{A}$

Walk-SAT: analysis

- Let F be 2-CNF formula with satisfying assignment $\mathcal{A}$
  Will bound expected number of flips to find $\mathcal{A}$.
- Distance between assignments := $\#$variables where differ

$$T_i := \max\{\mathbf{E}[\#\text{flippings } \mathcal{A} \to \mathcal{B}] \mid \text{distance}(\mathcal{A}, \mathcal{B}) = i\}$$

- Then T_n is time it takes to find satisfying assignment $\mathcal{A}$

$$T_0 = 0$$
$$T_n = 1 + T_{n-1}$$
$$T_i \leq 1 + (T_{i+1} + (T_{i-1}))/2$$

Walk-SAT: analysis

- Replacing by equalities gives bound $T_i \leq H_i$:

$$H_0 = 0$$
$$H_n = 1 + H_{n-1}$$
$$H_i = 1 + (H_{i+1} + (H_{i-1}))/2$$

Walk-SAT: analysis

- Replacing by equalities gives bound $T_i \leq H_i$:

$$
\begin{aligned}
H_0 &= 0 \\
H_n &= 1 + H_{n-1} \\
H_i &= 1 + (H_{i+1} + (H_{i-1}))/2
\end{aligned}
$$

- $n + 1$ linearly independent equations in $n + 1$ unknowns.
  Unique solution: $H_i = 2in - i^2$
  So worst expected time to hit $\mathcal{A}$ is $H_n = n^2$

Walk-SAT: analysis

- Replacing by equalities gives bound $T_i \leq H_i$:

$$
\begin{aligned}
H_0 &= 0 \\
H_n &= 1 + H_{n-1} \\
H_i &= 1 + (H_{i+1} + (H_{i-1}))/2
\end{aligned}
$$

- $n+1$ linearly independent equations in $n+1$ unknowns.
  Unique solution: $H_i = 2in - i^2$
  So worst expected time to hit $\mathcal{A}$ is $H_n = n^2$

- **Markov's inequality**: If X is nonnegative random variable, then $\mathbf{P}[X \geq a] \leq \frac{1}{a}\mathbf{E}[X]$ for all $a > 0$.

- **Theorem**: Walk-SAT on n-variable satisfiable 2-CNF formula for $r = 2mn^2$ succeeds with probability $\geq 1 - 2^{-m}$.

  Proof: Divide $2mn^2$ iterations of main loop into m phases.
  Markov: not finding satisfying valuation in a phase has probability $\leq n^2/2n^2 = 1/2$.

What's bad about 3-SAT?

- Common feature of Horn-SAT and 2-SAT algorithms: build satisfying assignments incrementally, without backtracking. This is different for general CNF formulas.

What's bad about 3-SAT?

- Common feature of Horn-SAT and 2-SAT algorithms: build satisfying assignments incrementally, without backtracking. This is different for general CNF formulas.

- Walk-SAT: one-dimensional random walk on line $\{0, \ldots, n\}$ with absorbing barrier 0 and reflecting barrier n

  Similar trick for 3-CNF formulas with probability $2/3$ of going right and $1/3$ of going left

  However, then r needs to be exponential in n ...

Summary

SAT is bad, but we can do better in special cases:

- Horn-SAT, 2-SAT and X-SAT have a polynomial-time decidable satisfiability problem
- But 3-SAT is as “bad” as all of propositional logic