Lecture 4

Polynomial-time formula classes

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

Introduction to Logic for Computer Science

Prof Hongseok Yang
KAIST

These slides are minor variants of those made by Prof Worrell and Dr Haase for their logic course at Oxford.

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 $L_{i,j}$:

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

- 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**": The method for solving SAT that we studied so far is to use truth tables. It takes exponential time in the worst case.

The satisfiability problem

"**SAT is bad**": The method for solving SAT that we studied so far is to use truth tables. It takes exponential time in the worst case.

But, we 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.:

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

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.:

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

- Many applications in computer science. Prolog and Datalog are based on Horn formulas.

Horn-SAT

Can decide **satisfiability** for Horn formulas *F* in poly. time. Why?

Can decide **satisfiability** for Horn formulas F in poly. time. Why?

Without backtracking, we can search for a $\leq$**-least model** $\mathcal{A}$ of F.

$\mathcal{A} \leq \mathcal{B}$ iff $\mathcal{A}(p) \leq \mathcal{B}(p)$ for all propositional variables p.

Horn-SAT

Can decide **satisfiability** for Horn formulas F in poly. time. Why?

Without backtracking, we can search for a $\leq$**-least model** $\mathcal{A}$ of F.

$\mathcal{A} \leq \mathcal{B}$ iff $\mathcal{A}(p) \leq \mathcal{B}(p)$ for all propositional variables p.

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

Horn-SAT

Can decide **satisfiability** for Horn formulas F in poly. time. Why?

Without backtracking, we can search for a $\leq$**-least model** $\mathcal{A}$ of F.

$\mathcal{A} \leq \mathcal{B}$ iff $\mathcal{A}(p) \leq \mathcal{B}(p)$ for all propositional variables p.

Sat.: $(\mathit{true} \rightarrow p_1) \wedge (p_2 \wedge p_3 \wedge p_4 \rightarrow \mathit{false}) \wedge (p_1 \rightarrow p_2)$.

Unsat.:
$(\mathit{true} \rightarrow p_1) \wedge (p_2 \wedge p_3 \wedge p_4 \rightarrow \mathit{false}) \wedge (p_1 \rightarrow p_2) \wedge (p_1 \wedge p_2 \rightarrow \mathit{false})$.

Horn-SAT algorithm

Idea:

- Maintain an assignment $\mathcal{A}$ for propositional variables, starting with $p \mapsto 0$.
- Update $\mathcal{A}(p)$ from 0 to 1 if forced by F, until either F is satisfied or contradiction is reached.

Horn-SAT algorithm

Idea:

- Maintain an assignment $\mathcal{A}$ for propositional variables, starting with $p \mapsto 0$.
- Update $\mathcal{A}(p)$ from 0 to 1 if forced by F, until either F is satisfied or contradiction is 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

Idea:

- Maintain an assignment $\mathcal{A}$ for propositional variables, starting with $p \mapsto 0$.
- Update $\mathcal{A}(p)$ from 0 to 1 if forced by F, until either F is satisfied or contradiction is 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

Ex: Why correct?

Horn-SAT algorithm: correctness

- T represents $\mathcal{A}_T$ defined by: $\mathcal{A}(p) = 1$ iff $p \in T$.
- Order assignments by $\mathcal{A} \leq \mathcal{B}$ when $\mathcal{A}(p) \leq \mathcal{B}(p)$ for every p.

Horn-SAT algorithm: correctness

- T represents $\mathcal{A}_T$ defined by: $\mathcal{A}(p) = 1$ iff $p \in T$.
- Order assignments by $\mathcal{A} \leq \mathcal{B}$ when $\mathcal{A}(p) \leq \mathcal{B}(p)$ for every p.
- Each iteration changes $\mathcal{A}(p)$ from 0 to 1 for some var. p in F.
- There are at most n iterations, so overall polynomial time.

Horn-SAT algorithm: correctness

- T represents $\mathcal{A}_T$ defined by: $\mathcal{A}(p) = 1$ iff $p \in T$.
- Order assignments by $\mathcal{A} \leq \mathcal{B}$ when $\mathcal{A}(p) \leq \mathcal{B}(p)$ for every p.
- Each iteration changes $\mathcal{A}(p)$ from 0 to 1 for some var. p in F.
- There are at most n iterations, so overall polynomial time.
- Any $\mathcal{A}$ returned must satisfy F by termination condition.

Horn-SAT algorithm: correctness

- T represents $\mathcal{A}_T$ defined by: $\mathcal{A}(p) = 1$ iff $p \in T$.
- Order assignments by $\mathcal{A} \leq \mathcal{B}$ when $\mathcal{A}(p) \leq \mathcal{B}(p)$ for every p.
- Each iteration changes $\mathcal{A}(p)$ from 0 to 1 for some var. p in F.
- 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. Really?

Horn-SAT algorithm: correctness

- T represents $\mathcal{A}_T$ defined by: $\mathcal{A}(p) = 1$ iff $p \in T$.
- Order assignments by $\mathcal{A} \leq \mathcal{B}$ when $\mathcal{A}(p) \leq \mathcal{B}(p)$ for every p.
- Each iteration changes $\mathcal{A}(p)$ from 0 to 1 for some var. p in F.
- 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. Really?
- **Loop invariant**: If $\mathcal{B}$ satisfies F, then $\mathcal{A} \leq \mathcal{B}$.
- Ex1: Prove that this is a loop invariant.
- Ex2: Prove that this loop invariant gives the desired result.

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 $\overline{L} := \begin{cases} p & \text{if } L = \neg p, \\ \neg p & \text{if } L = p. \end{cases}$

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 $\overline{L} := \begin{cases} p & \text{if } L = \neg p, \\ \neg p & \text{if } L = p. \end{cases}$

- The **implication graph** of a 2-CNF formula F is a directed graph $\mathcal{G} = (V, E)$, where

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

  with $p_1, \ldots, p_n$ prop. variables mentioned in F.

- There is an edge (L, M) in $\mathcal{G}$ iff the clause $(\overline{L} \vee M)$ or $(M \vee \overline{L})$ appears in F. The edge represents the implication $L \to M$.

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 $\overline{L} := \begin{cases} p & \text{if } L = \neg p, \\ \neg p & \text{if } L = p. \end{cases}$

- The **implication graph** of a 2-CNF formula F is a directed graph $\mathcal{G} = (V, E)$, where

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

  with $p_1, \ldots, p_n$ prop. variables mentioned in F.

- There is an edge (L, M) in $\mathcal{G}$ iff the clause $(\overline{L} \vee M)$ or $(M \vee \overline{L})$ appears in F. The edge represents the implication $L \to M$.

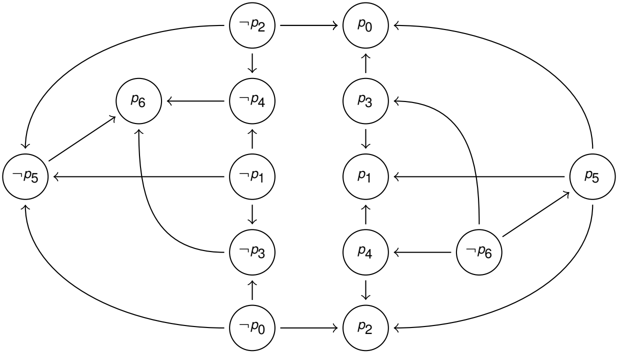
- Ex: If (L, M) is an edge, is $(\overline{M}, \overline{L})$ also an edge?

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

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



- Paths in $\mathcal{G}$ correspond to chains of implications.

2-SAT

- Can reduce **satisfiability** for 2-CNF formulas to *reachability* problem of implication graph, which is solvable in *poly.* 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.

2-SAT

- Can reduce **satisfiability** for 2-CNF formulas to *reachability* problem of implication graph, which is solvable in *poly.* 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.

2-SAT

- Can reduce **satisfiability** for 2-CNF formulas to *reachability* problem of implication graph, which is solvable in *poly.* 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.

($\Rightarrow$) Ex: Why?

2-SAT

- Can reduce **satisfiability** for 2-CNF formulas to *reachability* problem of implication graph, which is solvable in *poly.* 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.

($\Rightarrow$) Ex: Why?

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

2-SAT

- Can reduce **satisfiability** for 2-CNF formulas to *reachability* problem of implication graph, which is solvable in *poly.* 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.

($\Rightarrow$) Ex: Why?

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

($\Leftarrow$) Construct a satisfying assignment. Ex: How? $\square$

2-SAT Algorithm

INPUT: 2-CNF formula F
$\mathcal{A} :=$ empty (partial) assignment
while there is some unassigned variable **do**
  begin
  pick a literal L for which there is no path from L to $\overline{L}$
  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

INPUT: 2-CNF formula F
$\mathcal{A}$:= empty (partial) assignment
while there is some unassigned variable **do**
  begin
  pick a literal L for which there is no path from L to $\overline{L}$
  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 $\mathcal{A}$

Ex: Why correct? Invariants for the outer and inner loops?

2-SAT Algorithm: correctness

- Outer loop invariant: any node reachable from a true node is also true.
- Inner loop invariant: no path from a true node to a false node.

2-SAT Algorithm: correctness

- Outer loop invariant: any node reachable from a true node is also true.
- Inner loop invariant: no path from a true node to a false node.
- Ex: Why do they give the desired result? Why loop invariants?

2-SAT Algorithm: correctness

- Outer loop invariant: any node reachable from a true node is also true.
- Inner loop invariant: no path from a true node to a false node.
- Ex: Why do they give the desired result? Why loop invariants?
- If outer invariant holds and all variables in F assigned, we have a satisfying (partial) assignment $\mathcal{A}$.

2-SAT Algorithm: correctness

- Outer loop invariant: any node reachable from a true node is also true.
- Inner loop invariant: no path from a true node to a false node.
- Ex: Why do they give the desired result? Why loop invariants?
- If outer invariant holds and all variables in F assigned, we have a satisfying (partial) assignment $\mathcal{A}$.
- If outer invariant holds but not all variables assigned, there is an unassigned literal L with no path $L \to \overline{L}$ (by consistency).
- After updating $\mathcal{A}(L) := 1$, the inner invariant holds. Ex: Why?

2-SAT Algorithm: correctness

- Outer loop invariant: any node reachable from a true node is also true.
- Inner loop invariant: no path from a true node to a false node.
- Ex: Why do they give the desired result? Why loop invariants?
- If outer invariant holds and all variables in F assigned, we have a satisfying (partial) assignment $\mathcal{A}$.
- If outer invariant holds but not all variables assigned, there is an unassigned literal L with no path $L \to \overline{L}$ (by consistency).
- After updating $\mathcal{A}(L) := 1$, the inner invariant holds. Ex: Why?
- Inner loop maintains the invariant. So when it terminates, every node reachable from a true node is true.

3-CNF formulas

2-SAT solvable in poly. time, but 3-SAT captures all formulas.

- A **3-CNF** formula is a CNF one with ≤ 3 literals per clause.

3-CNF formulas

2-SAT solvable in poly. time, but 3-SAT captures all formulas.

- A **3-CNF** formula is a CNF one with ≤ 3 literals per clause.
- **Equisatisfiability**: F satisfiable if and only if G satisfiable.

Theorem

Given an arbitrary formula F, we can compute an equisatisfiable 3-CNF formula G in polynomial time.

3-CNF formulas

2-SAT solvable in poly. time, but 3-SAT captures all formulas.

- A **3-CNF** formula is a CNF one with ≤ 3 literals per clause.
- **Equisatisfiability**: F satisfiable if and only if G satisfiable.

Theorem

Given an arbitrary formula F, we can compute an equisatisfiable 3-CNF formula G in polynomial time.

Ex: How to construct such G for, say, $F = \neg((p \wedge q) \vee (r \vee \neg p))$?

3-CNF formulas

2-SAT solvable in poly. time, but 3-SAT captures all formulas.

- A **3-CNF** formula is a CNF one with ≤ 3 literals per clause.
- **Equisatisfiability**: F satisfiable if and only if G satisfiable.

Theorem

Given an arbitrary formula F, we can compute an equisatisfiable 3-CNF formula G in polynomial time.

Ex: How to construct such G for, say, $F = \neg((p \wedge q) \vee (r \vee \neg p))$?

Proof.

List all subformulas of F: $F_1 = p_1, \ldots, F_m = p_m, F_{m+1}, \ldots, F_n$.

Introduce new variables $p_{m+1}, \ldots, p_n$.

Associate formulas G_i asserting $p_i \leftrightarrow F_i$.

Take $G = G_{m+1} \wedge \cdots \wedge G_n \wedge p_n$. □

X-CNF formulas

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

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

- Ex: Can decide the sat. of X-CNF formulas in poly. time. Why?

X-CNF formulas

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

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

- Ex: Can decide the sat. of X-CNF formulas in poly. time. Why?
- Because we can use linear algebra over $\{0, 1\}$.

X-CNF formulas

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

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

- Ex: Can decide the sat. of X-CNF formulas in poly. time. Why?
- Because we can use linear algebra over $\{0, 1\}$.
- Rewrite a formula as a system of equations over $\mathbb{Z}_2$, and solve.

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

X-CNF formulas

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

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

- Ex: Can decide the sat. of X-CNF formulas in poly. time. Why?
- Because we can use linear algebra over $\{0, 1\}$.
- Rewrite a formula as a system of equations over $\mathbb{Z}_2$, and solve.

$$\begin{array}{ccccccccc} p_1 & & & + & p_3 & = & 1 \\ (1 + p_1) & + & p_2 & & & = & 1 \\ p_1 & + & p_2 & + & (1 + p_3) & = & 1 \end{array}$$

X-CNF formulas

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

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

- Ex: Can decide the sat. of X-CNF formulas in poly. time. Why?
- Because we can use linear algebra over $\{0, 1\}$.
- Rewrite a formula as a system of equations over $\mathbb{Z}_2$, and solve.

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

Can solve these equations using **Gaussian elimination**.

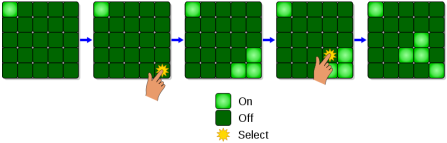
Lights out

Given: An $N \times N$ grid, each button coloured black or white.

Move: Pressing a button inverts colours of it and its neighbours.

Goal: The colours of all buttons are black.

Ex: Translate this to X-SAT.



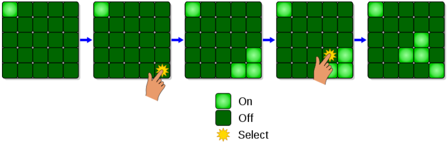
Lights out

Given: An $N \times N$ grid, each button coloured black or white.

Move: Pressing a button inverts colours of it and its neighbours.

Goal: The colours of all buttons are black.

Ex: Translate this to X-SAT.



Hint1: Even number of same moves doesn't do anything.

Hint2: Let $p_{i,j}$ denote whether the button (i, j) is pressed, and $c_{i,j}$ be the initial colour of the button (i, j), where $c_{i,j} = true$ means black.

1. Polynomial-time fragments of propositional logic

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

Walk-SAT: overview

Randomised algorithm for solving SAT for CNF formulas F.

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

Walk-SAT: overview

Randomised algorithm for solving SAT for CNF formulas F.

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

If F is unsat., the algorithm will say so.

But it may return UNSAT before finding a satisfying assignment.

Walk-SAT: overview

Randomised algorithm for solving SAT for CNF formulas F.

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

If F is unsat., the algorithm will say so.

But it may return UNSAT before finding a satisfying assignment.

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

Polynomially large for 2-CNF formulas.

Walk-SAT: overview

Randomised algorithm for solving SAT for CNF formulas F.

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

If F is unsat., the algorithm will say so.

But it may return UNSAT before finding a satisfying assignment.

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

Polynomially large for 2-CNF formulas.

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

Walk-SAT: algorithm precisely

Input: CNF formula F with n variables, repetition parameter r

pick an assignment (to the n variables) uniformly at random
if F is satisfied **then** return the current assignment
repeat r times
    pick an unsatisfied clause
    pick a literal in the clause uniformly at random, and flip value
    if F is satisfied **then** return the current assignment
return UNSAT

By assignments, we mean maps from the variables in F to $\{0, 1\}$.

Walk-SAT: analysis for a 2-CNF formula F

- Assume that $\mathcal{A}$ is a model of F.
  Will bound the expected number of flips to find $\mathcal{A}$.

Walk-SAT: analysis for a 2-CNF formula F

- Assume that $\mathcal{A}$ is a model of F.
  Will bound the expected number of flips to find $\mathcal{A}$.
- Distance between assignments := $\#$variables that differ.

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

- Then T_n is the worst expected time for the algorithm to find the satisfying assignment $\mathcal{A}$.

Walk-SAT: analysis for a 2-CNF formula F

- Assume that $\mathcal{A}$ is a model of F.
  Will bound the expected number of flips to find $\mathcal{A}$.
- Distance between assignments := $\#$variables that differ.

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

- Then T_n is the worst expected time for the algorithm to find the satisfying assignment $\mathcal{A}$.

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

Ex: Why do these relationships hold?

Walk-SAT: analysis

- Replacing inequalities by equalities gives bound $H_i \geq T_i$:

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

Ex: Why $H_i \geq T_i$? Prove it.

Walk-SAT: analysis

- Replacing inequalities by equalities gives bound $H_i \geq T_i$:

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

  Ex: Why $H_i \geq T_i$? Prove it.

- $n+1$ linearly independent eqs in $n+1$ unknowns.

Walk-SAT: analysis

- Replacing inequalities by equalities gives bound $H_i \geq T_i$:

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

  Ex: Why $H_i \geq T_i$? Prove it.

- $n + 1$ linearly independent eqs in $n + 1$ unknowns.
  Ex: Show that the unique solution is $H_i = (2i \cdot n) - i^2$.

Walk-SAT: analysis

- Replacing inequalities by equalities gives bound $H_i \geq T_i$:

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

  Ex: Why $H_i \geq T_i$? Prove it.

- $n + 1$ linearly independent eqs in $n + 1$ unknowns.
  Ex: Show that the unique solution is $H_i = (2i \cdot n) - i^2$.
  So the worst expected time to hit $\mathcal{A}$ is $H_n = n^2$.

Walk-SAT: analysis

- Replacing inequalities by equalities gives bound $H_i \geq T_i$:

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

  Ex: Why $H_i \geq T_i$? Prove it.

- $n + 1$ linearly independent eqs in $n + 1$ unknowns.
  Ex: Show that the unique solution is $H_i = (2i \cdot n) - i^2$.
  So the worst expected time to hit $\mathcal{A}$ is $H_n = n^2$.

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

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

  Ex: Prove the theorem.

Walk-SAT: analysis

- Replacing inequalities by equalities gives bound $H_i \geq T_i$:

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

  Ex: Why $H_i \geq T_i$? Prove it.

- $n+1$ linearly independent eqs in $n+1$ unknowns.
  Ex: Show that the unique solution is $H_i = (2i \cdot n) - i^2$.
  So the worst expected time to hit $\mathcal{A}$ is $H_n = n^2$.

- **Markov's inequality**: If X is a nonnegative random variable, then $\mathbb{P}[X \geq a] \leq \frac{1}{a}\mathbb{E}[X]$ 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 the main loop into m phases.
  Markov: not finding a satisfying assignment 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 can be solved by polynomial-time algorithms.
- But 3-SAT is as “bad” as the satisfiability of the entire propositional logic.