1 Lecture Outline
In this talk we will show that it is possible to efficiently decide if a 2-CNF formula is satisfiable. We will also study the satisfiability threshold of random 2-CNF formulas.

2 Notation
Recall that we used the following terminology:

- Variables: \( x_1, x_2, \ldots, x_n \).
- Literals: \( x_1, \bar{x}_1, x_2, \bar{x}_2, \ldots, x_n, \bar{x}_n \).
- \( k \)-clause: a conjunction (OR) of \( k \) distinct literals.
- For example, a 2-clause: \( \ell_i \lor \ell_j \), a 1-clause (unit-clause): \( \ell_i \), and a 0-clause: \( \emptyset \) (contradiction).
- \( k \)-CNF formula \( \varphi = (C_1, \ldots, C_m) \) where \( C_i \) is \( k \)-clause.
- Assignment \( \sigma \in \{0, 1\}^n \).
- Partial assignment \( \sigma \in \{0, 1, *\}^n \).

3 Example
Consider the following 2-CNF formula:
\[
\bar{x}_1 \lor x_2, \quad x_1 \lor x_2, \quad \bar{x}_2 \lor x_3, \quad x_3 \lor \bar{x}_4, \quad x_1 \lor \bar{x}_2.
\]
Let’s try to set \( x_1 = 0 \). Then the formula simplifies to:
\[
T, \quad x_2, \quad \bar{x}_2 \lor x_3, \quad x_3 \lor \bar{x}_4, \quad \bar{x}_2.
\]
where \( T \) denotes the value “Truth”. We are now forced to assign \( x_2 = 1 \) (as there is a unit-clause), and the formula simplifies to
\[
T, \quad T, \quad x_3, \quad x_3 \lor \bar{x}_4, \quad \emptyset,
\]

2: The Simple Story of 2-SAT-1
where ∅ is the empty clause which denotes contradiction. So we have to backtrack to the last free step. Let’s try \( x_1 = 1 \):

\[
\begin{align*}
x_2, & \quad T, \quad \bar{x}_2 \lor x_3, \quad x_3 \lor \bar{x}_4, \quad T.
\end{align*}
\]

We are now forced to set \( x_2 = 1 \):

\[
\begin{align*}
T, & \quad T, \quad x_3, \quad x_3 \lor \bar{x}_4, \quad T.
\end{align*}
\]

We are now forced to set \( x_3 = 1 \):

\[
\begin{align*}
T, & \quad T, \quad T, \quad T, \quad T,
\end{align*}
\]

and any value for \( x_4 \) will satisfy the formula. Hence, we’ve found two satisfying assignments: \((1,1,1,0), (1,1,1,1)\).

### 4 An Efficient Algorithm based on Unit Clause Propogation

Abstracting the above example, we present an algorithm that attempts to satisfy a 2-CNF formula \( \varphi \) as follows.

**Algorithm(\( \varphi \))**

(0) Initialize empty assignment \( \sigma = ^n \).  

(1) If all variables are assigned return \( \sigma \).

(2) Choose an unassigned variable \( x_i \).

(a) (Try \( x_i = 1 \))

- Set \( \sigma_i = 1 \), \( \varphi' \leftarrow \text{Simplify}(\varphi, x_i) \).
- \( \varphi' \leftarrow \text{Unit Clause Propagation}(\varphi') \).
- If \( \varphi' \) does not contain ∅ goto (1).

(b) (Try \( x_i = 0 \))

- Unassign variables from step (a).
- Set \( \sigma_i = 0 \), \( \varphi' \leftarrow \text{Simplify}(\varphi, \bar{x}_i) \).
- \( \varphi' \leftarrow \text{Unit Clause Propagation}(\varphi') \).
- If \( \varphi' \) does not contain ∅ goto (1).

(3) Halt with "UNSAT".

**Simplify(\( \varphi, \ell_i \))**

-∀ clause \( C \in \varphi \):
  
  - If \( \ell_i \in C \), remove \( C \).
  
  - If \( \bar{\ell}_i \in C \), \( C \leftarrow C \setminus \bar{\ell}_i \).
- Otherwise, copy C as is.

- Output the modified formula.

Unit Clause Propagation(\(\varphi\))
- While \(\exists\) unit clause \(\ell_i\):
  - Update \(\sigma\): if \(\ell_i = x_i\) set \(\sigma_i = 1\), else (\(\ell_i = \bar{x}_i\)) set \(\sigma_i = 0\).
  - \(\varphi \leftarrow\) Simplify(\(\varphi, \ell_i\)).

**Complexity.** Let \(n\) denote the number of variables and let \(m\) denote the number of clauses. It is not hard to verify that there are at most \(n\) outer iterations and that each call to UCP takes at most \(O(m)\) time, therefore the running time of Algorithm is \(O(m \cdot n)\).

(Home assignment: Find an implementation in \(O(n + m)\) complexity.)

### 4.1 Correctness

**Lemma 1** If the algorithm outputs an assignment \(\sigma\), then \(\sigma\) satisfies \(\varphi\).

We will need the following definition: A partial assignment \(\sigma \in \{0, 1, *\}^n\) violate a clause \(C = \ell_i \lor \ell_j\) if: \(\sigma_i\) and \(\sigma_j\) are assigned (i.e., \(\sigma_i, \sigma_j \neq *\)) and \(\sigma_i\) doesn’t satisfy \(\ell_i\) and \(\sigma_j\) doesn’t satisfy \(\ell_j\).

The lemma follows from the following invariance.

**Invariance 2** At the beginning of each iteration, the current partial assignment \(\sigma^{(i)}\) does not violate any of the clauses of \(C\).

**Proof** of Invariance 2: By induction on \(i\). The basis is trivial as in the first iteration \(\sigma = *^n\) and so none of the clauses are violated.

Step: we’ll prove that none of the clauses \(C\) are violated by \(\sigma^{(i+1)}\). If both variables of \(C\) were assigned before the last iteration, then, by the induction hypothesis, \(\sigma^{(i)}\) doesn’t violate \(C\), and therefore, so is \(\sigma^{(i+1)}\). If both variables of \(C\) were assigned in the last iteration, then \(C\) must be satisfied by \(\sigma^{(i+1)}\), otherwise, the algorithm finds a contradiction.

**Lemma 3** If the algorithm outputs UNSAT, then \(\varphi\) is unsatisfiable.

**Proof** Let \(\varphi'\) be the formula at the beginning of the iteration in which A halts, and let \(x_i\) be the variable chosen at step (2) of this last iteration. Note that \(\varphi'\) is a 2-CNF formula and \(\varphi' \subseteq \varphi\) (i.e., all the clauses of \(\varphi'\) appear as clauses in \(\varphi\)). Hence, it suffices to show that \(\varphi'\) is unsatisfiable. Let \(\varphi_0 = \text{Simplify}(\varphi', x_i = 0)\) and \(\varphi_1 = \text{Simplify}(\varphi', x_i = 1)\). It suffices to show that both \(\varphi_0\) and \(\varphi_1\) are unsatisfiable. Recall that the formula UCP(\(\varphi_0\)) and the formula UCP(\(\varphi_1\)) contain a contradiction. The proof now follows by noting that if UCP(\(\psi\)) contains a contradiction, then \(\psi\) is UNSAT.

Therefore, we have an efficient algorithm for SAT of 2-CNF.

---

2: The Simple Story of 2-SAT-3
5 Graphical View

For a 2-CNF formula \( \varphi \), define the implication graph \( G = G_\varphi \) as follows:

- nodes \( x_1, \bar{x}_1, x_2, \bar{x}_2, \ldots, x_n, \bar{x}_n \)
- for a clause \( \ell_i \lor \ell_j \) define the edges:
  \[
  \bar{\ell}_i \to \ell_j \\
  \bar{\ell}_j \to \ell_i
  \]

Main property: Let \( \sigma \) be a satisfying assignment. If \( \sigma \) satisfies a node \( v \), then \( \sigma \) satisfies all nodes \( u \) achievable from \( v \).

The property can be proven by induction on the length of the path.

**Theorem 4** \( \varphi \) is satisfiable iff the graph \( G \) does not contain a “contradiction path” of the form:

\[
\ell_i \to \cdots \to \bar{\ell}_i \to \cdots \to \ell_i.
\]

**Proof**

1. (\( \exists \) contradiction path \( \Rightarrow \) \( \varphi \) is UNSAT):
   - Take a potential assignment \( \sigma \).
   - If \( \sigma \) satisfies \( \ell_i \), then by Property it must satisfy \( \bar{\ell}_i \). Contradiction.
   - If \( \sigma \) satisfies \( \bar{\ell}_i \), then by Property it must satisfy \( \ell_i \). Contradiction.

2. (\( \varphi \) is UNSAT \( \Rightarrow \) \( \exists \) contradiction path):
   If \( \varphi \) is UNSAT \( \Rightarrow \) Algorithm Halts.
   \( \Rightarrow \) for some \( x_i \) we have:
   (a) \( \ell_j \leftarrow \cdots \leftarrow x_i \to \cdots \to \bar{\ell}_j \)
   (b) \( \ell_k \leftarrow \cdots \leftarrow \bar{x}_i \to \cdots \to \bar{\ell}_k \)

In our graph, if \( \ell_i \to \ell_j \) is an edge, then \( \bar{\ell}_j \to \ell_i \) is also an edge.

By reversing edges and negating:
(a) \( \Rightarrow x_i \to \cdots \to \bar{\ell}_j \to \cdots \to \bar{x}_i \)
(b) \( \Rightarrow \bar{x}_i \to \cdots \to \bar{\ell}_k \to \cdots \to x_i \)

Therefore, there exists a contradiction path.

\[\blacksquare\]
6 The Satisfiability Threshold for Random 2-CNF

Reminder: $F_2(n, m)$ is the distribution over 2-CNF with $n$ variables, $m$ clauses and each clause is chosen uniformly at random from all possible 2-clauses.

**Theorem 5** Let $r$ be a positive constant. Then:

$$\lim_{n \to \infty} \Pr[F_2(n, r \cdot n) \text{ is Satisfiable}] = \begin{cases} 1 & \text{if } r < 1 \\ 0 & \text{if } r > 1 \end{cases}$$

**Proof** of the first case ($r < 1$):

A *bicycle* of length $s \geq 2$ is a sequence of clauses of the form:

$$(u, \ell_1), (\bar{\ell}_1, \ell_2), \ldots, (\bar{\ell}_s, v),$$

where $\ell_1, \ldots, \ell_s$ are distinct literals and $u, v \in \{\ell_1, \ldots, \ell_s, \bar{\ell}_1, \ldots, \bar{\ell}_s\}$.

By the previous theorem, if $\varphi$ is UNSAT, then $G_\varphi$ contains a bicycle. We’ll show that for $r < 1$,

$$\Pr[F_2(n, r \cdot n) \text{contains bicycle}] = o(1).$$

For a fixed s-bicycle $A$,

$$\Pr[A \in F_2(n, m)] = \left( \frac{m}{s + 1} \right) \cdot \left( \frac{1}{4} \right)^s \leq \left( \frac{m}{2n(n-1)} \right)^s$$

The number of all possible s-bicycles $\leq 2^s \cdot n^s \cdot (2s)^2 = 4(2n)^s s^2$. Overall, we get:

$$\Pr[\exists \text{ bicycle } \in F_2(n, m)] \leq \sum_{s=2}^{n} 4(2n)^s s^2 \left( \frac{m}{2n(n-1)} \right)^s = \frac{4m}{2n(n-1)} \sum_{s=2}^{n} s^2 \left( \frac{2n \cdot m}{2n(n-1)} \right)^s = \frac{2m}{n(n-1)} \sum_{s=2}^{n} s^2 \left( \frac{m}{n-1} \right)^s \leq \frac{2}{n-1} \left( \sum_{s=2}^{n} \frac{n^{0.1} \cdot \text{const}}{n^{0.2}} \cdot r^{n^{0.1}} + \sum_{s=n^{0.1}}^{n} \frac{n^{2} \cdot n^{0.1}}{n^{2} \cdot r^{n^{0.1}}} \right) \leq o(1)$$

($^*$) $m = r \cdot n$ and $r < 1.$

2: The Simple Story of 2-SAT-5