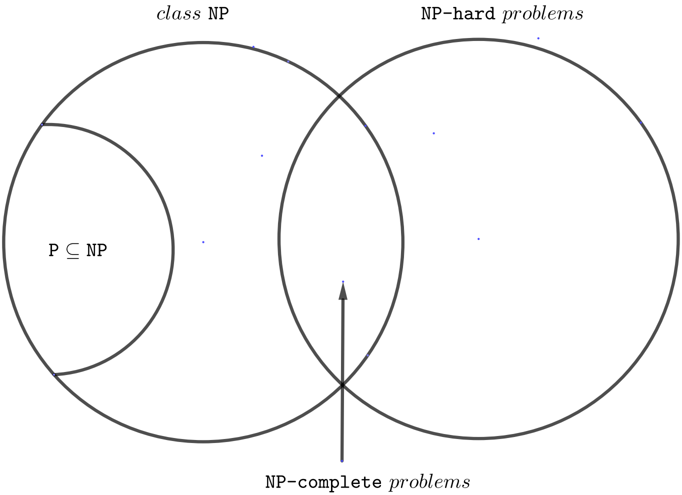
Algorithms Design.
Georgia Institute of Technology.

NP-complete problems: SAT.

Last time...

- Class NP: search problems verified in polynomial time.
- Class P: the subclass of those problems solvable in polynomial time.
- *Reductions*: $A \to B$: problem B is as hard as problem A.
- NP-hard: all problems in NP can be reduced to it.
- NP-complete: the intersection of NP and NP-hard.

Complexity classes.



SAT

Problem: (SAT)
`Input:` boolean formula in *conjunctive normal form**

$$f(x_1, x_2, \ldots, x_n) \rightarrow \{0, 1\}, \; x_i \in \{0, 1\}$$

`output:` An assignment of the variables x_i such that f returns 1, if such assignment exists.

SAT vocabulary

- *Variables*: $x_1, x_2, \ldots, x_n$.
- *Literals*: x_i and its negation $\bar{x}_i$.
- *Clauses*: disjunction of literals: $(x_1 \vee \bar{x}_3 \vee \bar{x}_4 \vee x_7)$.
- *Conjunctive normal form (CNF)*: f is the intersection ($\wedge$) of m clauses.

Input size: n variables and m clauses.

SAT is `NP-complete`.

Cook-Levin Theorem (1971)

SAT is `NP-complete`.

SAT variants

There are many applications and variants of the SAT problem:

- $k-$SAT: each clause has at most k literals.
- Exact $k-$SAT: each clause has exactly k literals.
- Max-SAT: find an assignment that maximizes the number of clauses that evaluates to 1.

All (almost) these variants are very hard: humans don't know yet how to solve this in polynomial time.

3-SAT is NP-complete.

3-SAT: The boolean formula f is in CNF and each clause has at most three literals.

$$f = (x_1 \vee x_4) \wedge (\bar{x}_2 \vee x_3 \vee x_5) \wedge (\bar{x}_1) \wedge (\bar{x}_2 \vee \bar{x}_3).$$

3-SAT is NP-complete.

Step 1: Show the problem is in NP (i.e.: candidate solutions can be verified in poly time).

For 3-SAT: same proof as for SAT.

3-SAT is NP-complete.

Step 2: Prove your problem is NP-hard (i.e.: reduce a known NP-hard problem to your problem).

We will show SAT $\rightarrow$ 3-SAT.

SAT $\rightarrow$ 3-SAT.

Building an instance of 3-SAT from an instance of SAT. Given a boolean function f we have two main cases:

- Clause has at most three literals.
- Clause has more than three literals.

SAT $\rightarrow$ 3-SAT.

Building an instance of 3-SAT from an instance of SAT. Given a boolean function f we have two main cases:

- Clause has at most three literals. Do nothing!
- Clause has more than three literals.

SAT $\rightarrow$ 3-SAT.

- Clause has more than three literals.

$$(x_1 \vee \bar{x}_2 \vee x_3 \vee x_5)$$

Idea: break these long clauses into pieces of size at most three.

SAT $\rightarrow$ 3-SAT.

Idea: break these long clauses into pieces of size at most three.

$$(x_1 \vee \bar{x}_2 \vee x_3) \wedge (x_5)$$

Forces $x_5 = 1$!

SAT $\rightarrow$ 3-SAT.

Idea: break these long clauses into pieces of size at most three. Create new variable $y \in \{0, 1\}$

$$(x_1 \vee \bar{x}_2 \vee x_3 \vee x_5) \quad \text{becomes} \quad C' = (x_1 \vee \bar{x}_2 \vee y) \wedge (\bar{y} \vee x_3 \vee x_5)$$

SAT $\rightarrow$ 3-SAT.

Claim: There exists an assignment of the variables such that the original clause evaluates to true if and only if there exists an assignment of the new variables such that, the pair of clauses C' also evaluates to true.

$$(x_1 \vee \bar{x}_2 \vee x_3 \vee x_5) \quad \text{becomes} \quad C' = (x_1 \vee \bar{x}_2 \vee y) \wedge (\bar{y} \vee x_3 \vee x_5)$$

SAT $\rightarrow$ 3-SAT.

Claim: There exists an assignment of the variables such that the original clause evaluates to true if and only if there exists an assignment of the new variables such that, the pair of clauses C' also evaluates to true.

$$(x_1 \vee \bar{x}_2 \vee x_3 \vee x_5) \text{ becomes } C' = (x_1 \vee \bar{x}_2 \vee y) \wedge (\bar{y} \vee x_3 \vee x_5)$$

($\Rightarrow$) Given a valid assignment for our clause, use the new variable y (if necessary) to make C' true.

SAT $\to$ 3-SAT.

Claim: There exists an assignment of the variables such that the original clause evaluates to true if and only if there exists an assignment of the new variables such that, the pair of clauses C' also evaluates to true.

$$(x_1 \vee \bar{x}_2 \vee x_3 \vee x_5) \text{ becomes } C' = (x_1 \vee \bar{x}_2 \vee y) \wedge (\bar{y} \vee x_3 \vee x_5)$$

($\Leftarrow$) Given a true assignment for C', we cannot use y to hold true both clauses.

SAT $\rightarrow$ 3-SAT.

Transforming clauses of length k: $(x_1 \vee x_2 \vee \cdots \vee x_k)$. Create $y_1, y_2, \ldots, y_{k-3}$ new variables. Build

$$C' = (x_1 \vee x_2 \vee y_1) \wedge (\bar{y}_1 \vee x_3 \vee y_2) \wedge (\bar{y}_2 \vee x_4 \vee y_3) \wedge \ldots (\bar{y}_{k-3} \vee x_{k-1} \vee x_k)$$

Check: original clause is true if and only if C' is also true.

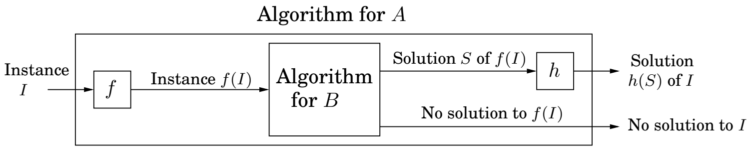
SAT $\rightarrow$ 3-SAT.

Transforming clauses of length k: $(x_1 \vee x_2 \vee \cdots \vee x_k)$. Create $y_1, y_2, \ldots, y_{k-3}$ new variables. Build

$$C' = (x_1 \vee x_2 \vee y_1) \wedge (\bar{y}_1 \vee x_3 \vee y_2) \wedge (\bar{y}_2 \vee x_4 \vee y_3) \wedge \ldots (\bar{y}_{k-3} \vee x_{k-1} \vee x_k)$$

Running time: Each clause creates $O(n)$ new literals and $O(n)$ new clauses so we have a total of $O(nm)$ literals and $O(nm)$ clauses on the input of 3-SAT.

Reductions



Algorithm for A
Instance I
f
Instance $f(I)$
Algorithm for B
Solution S of $f(I)$
h
Solution $h(S)$ of I
No solution to $f(I)$
No solution to I

SAT $\to$ 3-SAT.

Building a solution of SAT from a solution of 3-SAT:

$$(x_1 \vee \bar{x}_2 \vee x_3 \vee x_5) \text{ becomes } C' = (x_1 \vee \bar{x}_2 \vee y) \wedge (\bar{y} \vee x_3 \vee x_5)$$

Given a solution of 3-SAT, disregard the artificial variables y_j!!! By the claim, this is a solution of the original instance of SAT.

SAT $\rightarrow$ 3-SAT.

No solution for SAT implies no solution for 3-SAT: apply the claim directly.

$$(x_1 \vee \bar{x}_2 \vee x_3 \vee x_5) \text{ becomes } C' = (x_1 \vee \bar{x}_2 \vee y) \wedge (\bar{y} \vee x_3 \vee x_5)$$

The magic number three...

2-SAT: The boolean formula f is in CNF and each clause has at most two literals.

$$f = (x_1 \vee x_4) \wedge (\bar{x}_2) \wedge (x_3 \vee x_5) \wedge (\bar{x}_1) \wedge (\bar{x}_2 \vee \bar{x}_3).$$

The magic number three...

2-SAT: The boolean formula f is in CNF and each clause has at most two literals.

$$f = (x_1 \vee x_4) \wedge (\bar{x}_2) \wedge (x_3 \vee x_5) \wedge (\bar{x}_1) \wedge (\bar{x}_2 \vee \bar{x}_3).$$

Theorem

2-SAT is in the class P.

First reduction: `Exact-2-SAT`

All clauses with exactly one literal must hold true. For

$$(x_1) \wedge (x_2 \vee \bar{x_4}) \wedge (\bar{x_5}) \wedge (x_5 \vee x_6) \wedge (x_7 \vee x_8)$$

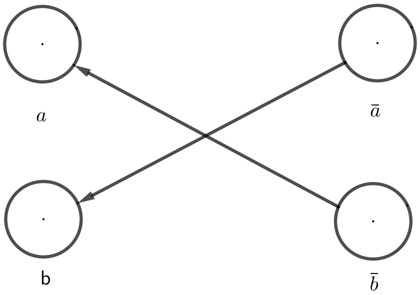
we set $x_1 = 1$ and $\bar{x_5} = 0$.
Simplify the input until you find a solution, a contradiction, or all clauses have exactly two literals.

Graph of implications

Given f in CNF with two literals on each clause, build a directed graph $G = (V, E)$.

V equals the set of **literals** $\{x_1, \bar{x_1}, x_2, \bar{x_2}, \ldots, x_n, \bar{x_n}\}$.

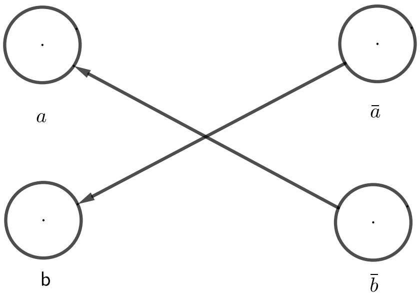
For each **clause:** $(a \vee b)$ create two edges: $\bar{a} \rightarrow b$ and $\bar{b} \rightarrow a$.



Edge set

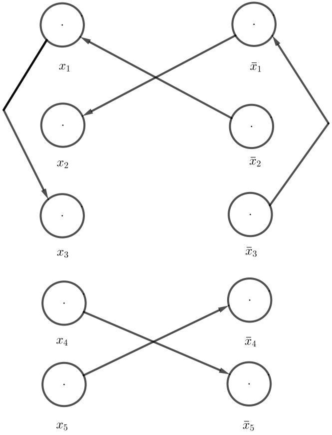
For each **clause:** $(a \vee b)$ create two edges: $\bar{a} \to b$ and $\bar{b} \to a$.

If $\bar{a}$ is true, then we must have b true in order for the corresponding clause to evaluate to 1 (analogous for $\bar{b} \to a$).



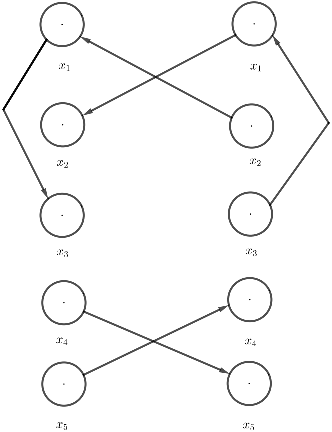
Example

Consider $f = (x_1 \vee x_2) \wedge (\bar{x_1} \vee x_3) \wedge (\bar{x_4} \vee \bar{x_5})$



The graph of implications

Fact: A path $\ell_1 \to \ell_2 \to \cdots \to \ell_t$ with ℓ_1 TRUE yields ℓ_i TRUE for all $2 \leq i \leq t$.



SCC in the graph of implications

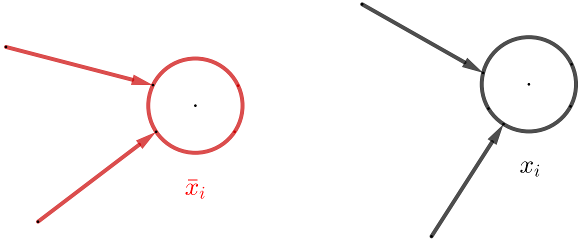
Lemma

If x_i and $\bar{x}_i$ are in the same SCC then the corresponding boolean function is not satisfiable.

Proof: there are paths $x_i \rightarrow \ell_1 \rightarrow \ell_2 \rightarrow \ldots \ell_t \rightarrow \bar{x}_i$ and $\bar{x}_i \rightarrow \ell_1 \rightarrow \ell_2 \rightarrow \ldots \ell_s \rightarrow x_i$. From the first one we conclude that $x_i = 1$ implies $x_i = 0$ and from the second we get $x_i = 0$ implies $x_i = 1$. $\square$

Finding a valid assignment

Setting $x_i = 1$ implies $\bar{x}_i$ is never true (i.e.: x_i cannot be equal to 0).



Problem! Incoming edges may force $\bar{x}_i$ to be true!

Structure of the graph of implications.

Claim 1 There is a path from literal a to literal b *if and only if* there is a path from literal $\bar{b}$ to literal $\bar{a}$.

Proof: If $a \to \ell_1 \to \ell_2 \to \ldots \ell_t \to b$ then, by construction, we have $\bar{b} \to \bar{\ell}_t \to \bar{\ell}_{t-1} \to \ldots \bar{\ell}_1 \to \bar{a}$. $\square$

Structure of the graph of implications.

Claim 2 If S is a strongly connected component of G, the set $\bar{S} = \{\bar{s},\ s \in S\}$ is also a strongly connected component.

Proof: Let $a, b \in S$. Then there exists a path from a to b and a path from b to a. By Claim 1, there is a path from $\bar{a}$ to $\bar{b}$ and from $\bar{b}$ to $\bar{a}$. $\square$

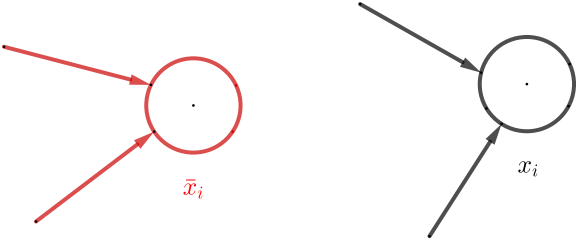
Structure of the graph of implications.

Claim 3 If S is a sink strongly connected component of G, $\bar{S}$ is a source strongly connected component.

Proof: Let $a \in S$. Then all edges $a \to b$ go to literals $b \in S$. Then, by Claim 1, all edges into $\bar{a}$ are coming from literals in $\bar{b} \in \bar{S}$. $\square$

Finding a valid assignment

<u>One fixing idea:</u> assign true value to literals in a sink component.



Problem! Incoming edges may force $\bar{x}_i$ to be true!

2−SAT algorithm via SCC

It suffices to have all pairs $x_i, \bar{x}_i$ are in different components!

<u>2−SAT</u>

- Build the graph of implications, $G = (V, E)$.
- Run the SCC algorithm on G. Let S be a sink component.
- FOR $a \in S$, set a to be true. Note that $\bar{a}$ cannot be true! Delete S and $\bar{S}$ from G.
- Repeat until all variables are assigned a value.