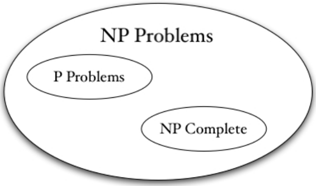
Formal Languages, Automata and Computation

NP-Completeness

Carnegie Mellon University in Qatar

SUMMARY

- Time complexity: Big-O notation, asympotic complexity
- Simulation of multi-tape TMs with a single tape deterministic TM can be done with a polynomial slow-down.
- Simulation of nondeterministic TMs with a deterministic TM is exponentially slower.
- The Class P: The class of languages for which membership can be *decided* quickly.
- The Class NP: The class of languages for which membership can be *verified* quickly.



- We do not yet know if P $=$ NP, or not.

NP PROBLEMS

- The best method known for solving languages in NP deterministically uses exponential time, that is

$$\text{NP} \subseteq \text{EXPTIME} = \bigcup_k \text{TIME}(2^{n^k})$$

- It is not known whether NP is contained in a smaller deterministic time complexity class.

NP-Complete Problems

- Cook and Levin in early 1970's showed that certain problems in NP were such that
  - If any of these problems had a deterministic polynomial-time algorithm, then
  - All problems in NP had deterministic polynomial-time algorithms.
- Such problems are called NP-complete problems.
- This is important for a number of reasons:
  1. If one is attempting to show that P$\neq$NP, s/he may focus on an NP-complete problem and try to show that it needs more than a polynomial amount of time.
  2. If one is attempting to show that P$=$NP, s/he may focus on an NP-complete problem and try to come up with a polynomial time algorithm for it.
  3. One may avoid wasting searching for a nonexistent polynomial time algorithm to solve a particular problem, if one can show it reduces to an NP-complete problem (as it is generally believed that P$\neq$ NP.)

DEFINITION – BOOLEAN VARIABLES

A boolean variable is a variable that can taken on values TRUE (1) and FALSE (0).

- We have Boolean operations of AND ($x \wedge y$), OR ($x \vee y$) and NOT ($\neg x$ or $\overline{x}$) on boolean variables.

AND	OR	NOT
$0 \wedge 0 = 0$	$0 \vee 0 = 0$	$\overline{0} = 1$
$0 \wedge 1 = 0$	$0 \vee 1 = 1$	$\overline{1} = 0$
$1 \wedge 0 = 0$	$1 \vee 0 = 1$	
$1 \wedge 1 = 1$	$1 \vee 1 = 1$	

The Satisfiability Problem

Definition – Boolean Formula

A Boolean formula is an expression involving Boolean variables and operations.
For example: $\phi = (\overline{x} \wedge y) \vee (x \wedge \overline{z}) \vee (y \wedge z)$ is a Boolean formula.

Definition – Satisfiability

A Boolean formula is satisfiable if some assignment of 0s and 1s to the variables makes the formula evaluate to 1.
We say the assignment satisfies ϕ.

- What possible assignments satisfy the formula above?

Definition – the Satisfiability Problem

The satisfiability problem checks if a Boolean formula is satisfiable.

$$SAT = \{\langle \phi \rangle \mid \phi \text{ is a satisfiable Boolean formula}\}$$

The Satisfiability Problem

Theorem 7.27 – The Cook-Levin Theorem

$SAT \in$ P iff P $=$ NP.

Proof

Coming slowly!

Polynomial Time Reducibility

Definition – Polynomial Time Computable Function

A function $f : \Sigma^* \longrightarrow \Sigma^*$ is a polynomial time computable function if some polynomial time TM M exists that halts with $f(w)$ on its tape, when started on any input w.

Definition – Polynomial Time Reducibility

Language A is polynomial time mapping reducible or polynomial time reducible, to language B, notated $A \leq_P B$, if a polynomial time computable function $f : \Sigma^* \longrightarrow \Sigma^*$ exists, where for every w,

$$w \in A \Leftrightarrow f(w) \in B$$

The function f is called the polynomial time reduction of A to B.

- To test whether $w \in A$ we use the reduction f to map w to $f(w)$ and test whether $f(w) \in B$.

POLYNOMIAL TIME REDUCIBILITY

THEOREM 7.31

If $A \leq_P B$ and $B \in \mathrm{P}$, then $A \in \mathrm{P}$.

PROOF

- It takes polynomial time to reduce A to B.
- It takes polynomial time to decide B.

- A literal is a Boolean variable or its negated version (x or $\overline{x}$).
- A clause is several literals connected with $\vee$ (OR), e.g., $(x_1 \vee \overline{x_2} \vee x_4)$.
- A Boolean formula is in conjuctive normal form (or is a cnf-formula) if it consists of several clauses connected with $\wedge$(AND), e.g.

$$(x_1 \vee \overline{x_2} \vee x_4 \vee x_5) \wedge (x_2 \vee \overline{x_3} \vee \overline{x_4}) \wedge (x_1 \vee x_2 \vee x_3 \vee \overline{x_5})$$

- A cnf-formula is a 3cnf-formula if all clauses have 3 literals, e.g.

$$(x_1 \vee \overline{x_2} \vee x_4) \wedge (x_2 \vee \overline{x_3} \vee \overline{x_4}) \wedge (x_1 \vee x_3 \vee \overline{x_5})$$

- $3SAT = \{\langle\phi\rangle \mid \phi$ is a satisfiable 3cnf-formula $\}$.
  - In a satisfiable cnf-formula, each clause must contain at least one literal that is assigned 1.

AN EXAMPLE REDUCTION: REDUCING *3SAT* TO *CLIQUE*

THEOREM 7.32

3*SAT* is polynomial time reducible to *CLIQUE*.

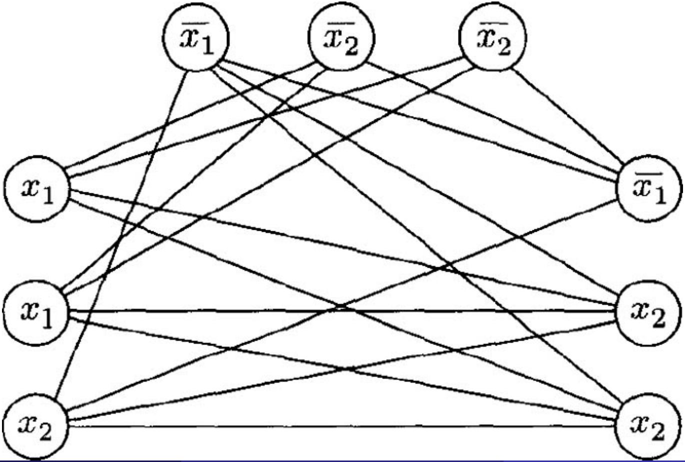
PROOF IDEA

Take any 3*SAT* formula and polynomial-time reduce it to a graph such that if the graph has a clique then the 3cnf-formula is satisfiable.

- Some details:
  - ϕ is a formula with k clauses each with 3 literals.
  - The k clauses in ϕ map to k groups of 3 nodes each called a triple.
  - Each node in the triple corresponds to one of the literals in the corresponding clause.
  - No edges between the nodes in a triple.
  - No edges between "conflicting" nodes (e.g., x and $\overline{x}$)

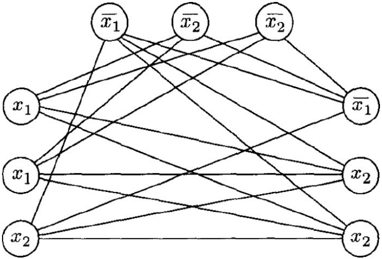
An Example Reduction: Reducing $3SAT$ to $CLIQUE$

$$\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$$



An Example Reduction: Reducing *3SAT* to *CLIQUE*

$$\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$$



- If ϕ has a satisfying assignment, then at least one literal in each clause needs to be 1.
- We select the corresponding nodes in the corresponding triples.
- These nodes should form a *k*-clique.
- If *G* has a *k*-clique, then selected nodes give a satisfying assignment to variables.

NP-Completeness

Definition – NP-Completeness

A language B is NP-complete if it satisfies two conditions:

1. B is in NP, and
2. Every A in NP is polynomial time reducible to B.

Theorem

If B is NP-complete and $B \in \text{P}$, then $\text{P} = \text{NP}$. (Obvious)

Theorem

If B is NP-complete and $B \leq_P C$ for C in NP, then C is NP-complete.

Proof

All $A \leq_P B$ and $B \leq_P C$ thus all $A \leq_P C$.

THE COOK-LEVIN THEOREM (AGAIN)

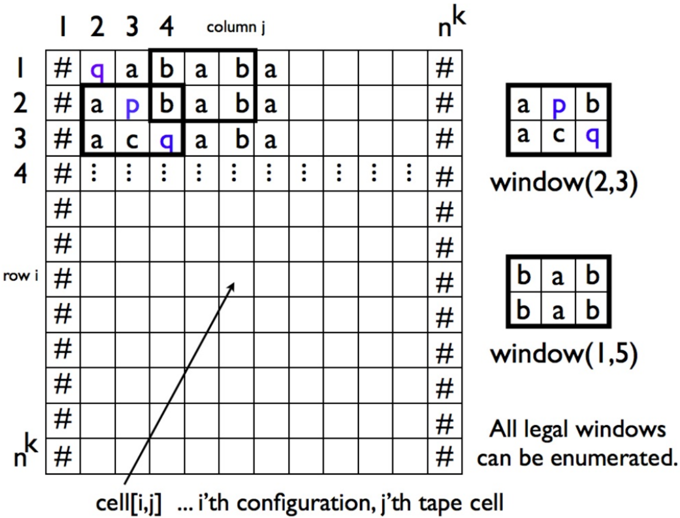
THEOREM

SAT is NP-Complete.

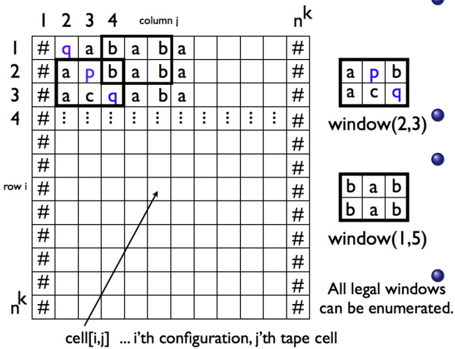
PROOF IDEA

- Showing *SAT* is in NP is easy.
  - Nondeterministically guess the assignments to variables and accept if the assignments satisfy ϕ
- We can encode the accepting computation history of a polynomial time NTM for every problem in NP as a *SAT* formula ϕ.
- Thus every language $A \in$ NP is polynomial-time reducible to *SAT*.
  - N is a NTM that can decide A in time $O(n^k)$
  - N accepts w if and only if ϕ is satisfiable.

BIRD'S EYE VIEW OF A POLYNOMIAL TIME COMPUTATION BRANCH



Bird's eye view of a polynomial time computation branch



- We represent the computation of a NTM N on w with a $n^k \times n^k$ table, called a tableau.
- Rows represent configurations
- First row is the start configuration (w + lots of blanks to fill the remaining of the n^k cells.)
- Each row follows from the previous one using N's transition function.

- A tableau is accepting if any row of the tableau is an accepting configuration.
- Every accepting tableau for N on w corresponds to an accepting computation branch of N on w.
- If N accepts w, then an accepting tableau exists!

Setting up formula ϕ

The variables

- Let $C = Q \cup \Gamma \cup \{\#\}$.
- For $1 \leq i, j \leq n^k$ and for each $s \in C$, we have a variable $x_{i,j,s}$.
- $x_{i,j,s} = 1$ if the *cell*$[i, j]$ contains the symbol s.
- Note that the number of variables is polynomial function of n.

The Formula ϕ

$$\phi = \phi_{cell} \wedge \phi_{start} \wedge \phi_{move} \wedge \phi_{accept}$$

- ϕ_{cell} makes sure that there is only one symbol in every cell!
- ϕ_{start} makes sure the start configuration is correct.
- ϕ_{accept} makes sure the accept state occurs somewhere.
- ϕ_{move} makes sure configurations follow each other legally.

ϕ_{cell}

- For all *i* and *j*, if *cell*$[i, j]$ contains symbol *s*, (that is $x_{i,j,s} = 1$), it can not contain another symbol (that is, no other variable with the same *i* and *j*, but a different symbol, is 1).

$$\phi_{cell} = \bigwedge_{1 \le i,j \le n^k} \left[\left(\bigvee_{s \in C} x_{i,j,s} \right) \wedge \left(\bigwedge_{\substack{s,t \in C \\ s \neq t}} (\overline{x_{i,j,s}} \vee \overline{x_{i,j,t}}) \right) \right]$$

ϕ_{cell}

- For all i and j, if $cell[i, j]$ contains symbol s, (that is $x_{i,j,s} = 1$), it can not contain another symbol (that is, no other variable with the same i and j, but a different symbol, is 1).

$$\phi_{cell} = \underbrace{\bigwedge_{1 \le i,j \le n^k}}_{\text{for all } i \text{and } j} \left[\left(\bigvee_{s \in C} x_{i,j,s} \right) \wedge \left(\bigwedge_{\substack{s,t \in C \\ s \ne t}} (\overline{x_{i,j,s}} \vee \overline{x_{i,j,t}}) \right) \right]$$

ϕ_{cell}

- For all *i* and *j*, if *cell*[*i*, *j*] contains symbol *s*, (that is $x_{i,j,s} = 1$), it can not contain another symbol (that is, no other variable with the same *i* and *j*, but a different symbol, is 1).

$$\phi_{cell} = \underbrace{\bigwedge_{1 \le i,j \le n^k}}_{\text{for all } i \text{ and } j} \left[\underbrace{\left(\bigvee_{s \in C} x_{i,j,s} \right)}_{\substack{\text{at least one symbol} \\ \text{is in a cell}}} \wedge \left(\bigwedge_{\substack{s,t \in C \\ s \neq t}} (\overline{x_{i,j,s}} \vee \overline{x_{i,j,t}}) \right) \right]$$

ϕ_{cell}

- For all *i* and *j*, if *cell*$[i, j]$ contains symbol *s*, (that is $x_{i,j,s} = 1$), it can not contain another symbol (that is, no other variable with the same *i* and *j*, but a different symbol, is 1).

$$\phi_{cell} = \underbrace{\bigwedge_{1 \le i,j \le n^k}}_{\text{for all } i \text{ and } j} \left[\underbrace{\left(\bigvee_{s \in C} x_{i,j,s} \right)}_{\substack{\text{at least one symbol} \\ \text{is in a cell}}} \wedge \left(\bigwedge_{\substack{s,t \in C \\ s \neq t}} \overbrace{(\overline{x_{i,j,s}} \vee \overline{x_{i,j,t}})}^{\text{only one symbol in a cell}} \right) \right]$$

- Note that ϕ_{cell} is in a conjuctive normal form.

ϕ_{start}

- ϕ_{start} sets up the first configuration.

$$\begin{aligned}\phi_{start} \quad = \quad & x_{1,1,\#} \wedge x_{1,2,q_0} \wedge x_{1,3,w_1} \wedge x_{1,4,w_2} \wedge \cdots x_{1,n+2,w_n} \wedge \\ & x_{1,n+3,\sqcup} \wedge \cdots x_{1,n^k-1,\sqcup} \wedge x_{1,n^k,\#}\end{aligned}$$

ϕ_{start}

- ϕ_{start} sets up the first configuration.

$$\phi_{start} = \overbrace{x_{1,1,\#} \wedge x_{1,2,q_0} \wedge x_{1,3,w_1} \wedge x_{1,4,w_2} \wedge \cdots x_{1,n+2,w_n} \wedge}^{q_0 \text{ and input symbols}}$$
$$\underbrace{x_{1,n+3,\sqcup} \wedge \cdots x_{1,n^k-1,\sqcup} \wedge x_{1,n^k,\#}}_{\text{all the blanks to the right}}$$

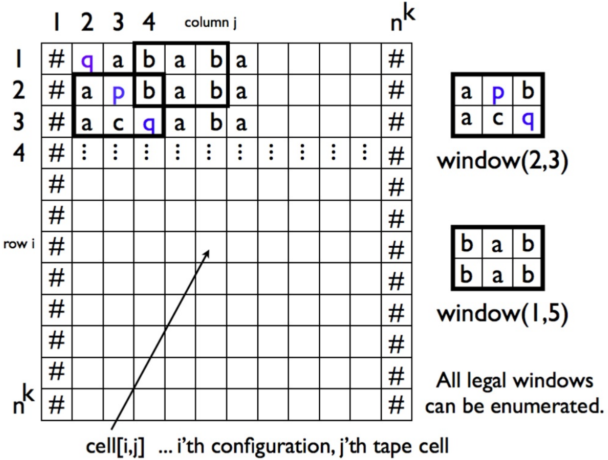
ϕ_{accept}

- ϕ_{accept} says q_{accept} occurs somewhere.

$$\phi_{accept} = \bigvee_{1 \le i,j \le n^k} x_{i,j,q_{accept}}$$

ϕ_{move}

- ϕ_{move} is the most interesting of the subformulas



- How many possible such windows are there?
- There are $|C|^6$ possible such windows.

Definition – Legal Window

A 2×3 window is legal if that window does not violate the actions specified by N's transition function.

- Suppose δ of N has the entries
  - $\delta(q_1, a) = \{(q_1, b, R)\}$
  - $\delta(q_1, b) = \{(q_2, c, L), (q_2, a, R)\}$
- The following windows are legal:

a	q_1	b
q_2	a	c

a	q_1	b
a	a	q_2

a	a	q_1
a	a	b

#	b	a
#	b	a

a	b	a
a	b	q_2

b	b	b
c	b	b

Definition – Legal Window

A 2 is legal if that window does not violate the actions specified by N's transition function.

- Suppose δ of N has the entries
  - $\delta(q_1, a) = \{(q_1, b, R)\}$
  - $\delta(q_1, b) = \{(q_2, c, L), (q_2, a, R)\}$
- The following windows are NOT legal:

a	b	a
a	a	a

a	q_1	b
q_1	a	a

b	q_1	b
q_2	b	q_2

Claim

If the top row of the table is the start configuration and every window in the tableau is legal, then every row of the table (after the first) is a configuration that follows the preceding one!

ϕ_{move}

Thus

$$\phi_{move} = \bigwedge_{1 \leq i < n^k, 1 < j < n^k} \text{(the } (i,j) \text{ window is legal)}$$

Where " (the (i,j) window is legal) " is actually the following formula

$$\bigvee_{\substack{a_1,a_2,a_3,a_4,a_5,a_6 \\ \text{is a legal window}}} (x_{i,j-1,a_1} \wedge x_{i,j,a_2} \wedge x_{i,j+1,a_3} \wedge x_{i+1,j-1,a_4} \wedge x_{i+1,j,a_5} \wedge x_{i+1,j+1,a_6})$$

- We have $O(n^{2k})$ variables $(= |C| \times n^k \times n^k)$
- The total formula size is $O(n^{2k})$, so it is polynomial time reduction.

3*SAT* IS NP-COMPLETE

COROLLARY

3*SAT* is NP-complete.

- Every formula in the construction of the NP-completeness proof of *SAT* can actually be written as a conjuctive normal form formula with 3 literals per clause.
  - If a clause has less that 3 literals, repeat one.
  - Disjunctive normal form clauses can be transformed into conjunctive normal form clauses, e.g.,

$$(a \wedge b) \vee (c \wedge d) = (a \vee c) \wedge (a \vee d) \wedge (b \vee c) \wedge (b \vee d)$$

  - Clauses longer than 3 clauses can be rewritten as clauses with 3 variable, e.g.,

$$(a \vee b \vee c \vee d) = (a \vee b \vee z) \wedge (\overline{z} \vee c \vee d)$$