CS 133 : Automata Theory and Computability

Lecture Slides

Time Complexity

Francis George C. Cabarle
fccabarle@up.edu.ph

Algorithms and Complexity Laboratory
Department of Computer Science
College of Engineering
University of the Philippines Diliman

Day 26

NP-completeness

More on NP-completeness

NP-completeness

More on **NP**-completeness

Class NP

or problems solvable in polynomial time with a 1-tape NTM. More formally

Definition

Language L is in class **NP** if there is a NTM M such that $L = L(M)$ and M has polynomial run time $t(n)$.

In some sources (e.g. book by M. Sipser) class **NP** is defined as class of problems with poly time **verifiers**:

Class NP

or problems solvable in polynomial time with a 1-tape NTM. More formally

Definition

Language L is in class **NP** if there is a NTM M such that $L = L(M)$ and M has polynomial run time $t(n)$.

In some sources (e.g. book by M. Sipser) class **NP** is defined as class of problems with poly time **verifiers**: a verifier is an algo with poly run time and verifies if a given *witness* or *certificate* is a solution to a problem in **NP**.

Examples of problems in **NP**:
$HAMPATH$, $CLIQUE$, $SUBSETSUM$

Class NP

or problems solvable in polynomial time with a 1-tape NTM. More formally

Definition

Language L is in class **NP** if there is a NTM M such that $L = L(M)$ and M has polynomial run time $t(n)$.

In some sources (e.g. book by M. Sipser) class **NP** is defined as class of problems with poly time **verifiers**: a verifier is an algo with poly run time and verifies if a given *witness* or *certificate* is a solution to a problem in **NP**.

Examples of problems in **NP**:
$HAMPATH$, $CLIQUE$, $SUBSETSUM$
$3SAT$?

Class NP

or problems solvable in polynomial time with a 1-tape NTM. More formally

Definition

Language L is in class **NP** if there is a NTM M such that $L = L(M)$ and M has polynomial run time $t(n)$.

In some sources (e.g. book by M. Sipser) class **NP** is defined as class of problems with poly time **verifiers**: a verifier is an algo with poly run time and verifies if a given *witness* or *certificate* is a solution to a problem in **NP**.

Examples of problems in **NP**:
$HAMPATH$, $CLIQUE$, $SUBSETSUM$
$3SAT$?

More real-world applications: Cryptography!

Satisfiability Problem

A Boolean formula is **satisfiable** if some assignment of 0s and 1s to the variables makes the formula evaluate to 1.

Satisfiability Problem

A Boolean formula is **satisfiable** if some assignment of 0s and 1s to the variables makes the formula evaluate to 1.

Example

$$\phi = (\overline{x} \wedge y) \vee (x \wedge \overline{z})$$

Satisfiability Problem

A Boolean formula is **satisfiable** if some assignment of 0s and 1s to the variables makes the formula evaluate to 1.

Example

$$\phi = (\overline{x} \wedge y) \vee (x \wedge \overline{z})$$

The satisfiability problem is to test whether a Boolean formula is satisfiable.

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

Satisfiability Problem

A Boolean formula is **satisfiable** if some assignment of 0s and 1s to the variables makes the formula evaluate to 1.

Example

$$\phi = (\overline{x} \wedge y) \vee (x \wedge \overline{z})$$

The satisfiability problem is to test whether a Boolean formula is satisfiable.

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

Theorem (Cook-Levin Theorem)

$$SAT \in \mathbf{P} \ \textit{iff} \ \mathbf{P} = \mathbf{NP}.$$

Satisfiability Problem

A Boolean formula is **satisfiable** if some assignment of 0s and 1s to the variables makes the formula evaluate to 1.

Example

$$\phi = (\overline{x} \wedge y) \vee (x \wedge \overline{z})$$

The satisfiability problem is to test whether a Boolean formula is satisfiable.

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

Theorem (Cook-Levin Theorem)

$$SAT \in \mathbf{P} \ \textit{iff} \ \mathbf{P} = \mathbf{NP}.$$

SAT was the first problem to be identified to be **NP**-complete. Method to prove **NP**-completeness?

Satisfiability Problem

A Boolean formula is **satisfiable** if some assignment of 0s and 1s to the variables makes the formula evaluate to 1.

Example

$$\phi = (\overline{x} \wedge y) \vee (x \wedge \overline{z})$$

The satisfiability problem is to test whether a Boolean formula is satisfiable.

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

Theorem (Cook-Levin Theorem)

$$SAT \in \mathbf{P} \ \textit{iff} \ \mathbf{P} = \mathbf{NP}.$$

SAT was the first problem to be identified to be **NP**-complete.
Method to prove **NP**-completeness? Polynomial time reduction! ($\leq_p$)

Poly time reduction: an example

$3SAT$:

Poly time reduction: an example

$3SAT$: Each clause has exactly 3 variables, and entire formula is in CNF (*conjunctive normal form*) i.e. clauses connected with $\wedge$s and variables in clauses with $\vee$s.

Poly time reduction: an example

$3SAT$: Each clause has exactly 3 variables, and entire formula is in CNF (*conjunctive normal form*) i.e. clauses connected with $\wedge$s and variables in clauses with $\vee$s.

$3SAT = \{\langle \phi, k \rangle \mid \phi$ is a satisfiable 3CNF formula with k clauses$\}$.

Poly time reduction: an example

$3SAT$: Each clause has exactly 3 variables, and entire formula is in CNF (*conjunctive normal form*) i.e. clauses connected with $\wedge$s and variables in clauses with $\vee$s.

$3SAT = \{\langle \phi, k \rangle \mid \phi$ is a satisfiable 3CNF formula with k clauses$\}$.

Theorem: $3SAT \leq_p CLIQUE$.

Definition

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

1. B is in **NP**, and
2. every A in **NP** is poly time reducible to B.

Definition

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

1. B is in **NP**, and
2. every A in **NP** is poly time reducible to B.

Theorem

*If B is **NP**-complete and $B \in \mathbf{P}$, then $\mathbf{P} = \mathbf{NP}$.*

Definition

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

1. B is in **NP**, and
2. every A in **NP** is poly time reducible to B.

Theorem

*If B is **NP**-complete and $B \in \mathbf{P}$, then $\mathbf{P} = \mathbf{NP}$.*

Theorem

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

Definition

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

1. B is in **NP**, and
2. every A in **NP** is poly time reducible to B.

Theorem

*If B is **NP**-complete and $B \in \mathbf{P}$, then $\mathbf{P} = \mathbf{NP}$.*

Theorem

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

*Once we have one **NP**-complete problem, we may obtain others by poly time reduction from it!*

Cook-Levin Theorem[1]

Theorem

*SAT is **NP**-complete.*

[1] Alternate version.

Cook-Levin Theorem[1]

Theorem

*SAT is **NP**-complete.*

Proof:

▷ Show that SAT is in **NP**.

[1] Alternate version.

Cook-Levin Theorem[1]

Theorem

*SAT is **NP**-complete.*

Proof:
▷ Show that SAT is in **NP**.
i.e. An NTM N running in poly time *accepts* some input w iff boolean formula $\phi_{N,w}$ is *satisfiable*,

[1] Alternate version.

Cook-Levin Theorem[1]

Theorem

*SAT is **NP**-complete.*

Proof:

▷ Show that SAT is in **NP**.

i.e. An NTM N running in poly time *accepts* some input w iff boolean formula $\phi_{N,w}$ is *satisfiable*, i.e. $w \in L(N)$ iff $\phi_{N,w}$ is satisfiable.

[1] Alternate version.

Cook-Levin Theorem[1]

Theorem

*SAT is **NP**-complete.*

Proof:

▷ Show that SAT is in **NP**.

i.e. An NTM N running in poly time *accepts* some input w iff boolean formula $\phi_{N,w}$ is *satisfiable*, i.e. $w \in L(N)$ iff $\phi_{N,w}$ is satisfiable.

▷ Show that every language A in **NP** is poly time reducible to SAT.

[1] Alternate version.

Cook-Levin Theorem[1]

Theorem

*SAT is **NP**-complete.*

Proof:
▷ Show that SAT is in **NP**.
i.e. An NTM N running in poly time *accepts* some input w iff boolean formula $\phi_{N,w}$ is *satisfiable*, i.e. $w \in L(N)$ iff $\phi_{N,w}$ is satisfiable.

▷ Show that every language A in **NP** is poly time reducible to SAT.

Pick any language A in **NP**.
Let N be an NTM that decides A in $O(n^k)$ time for some constant k.

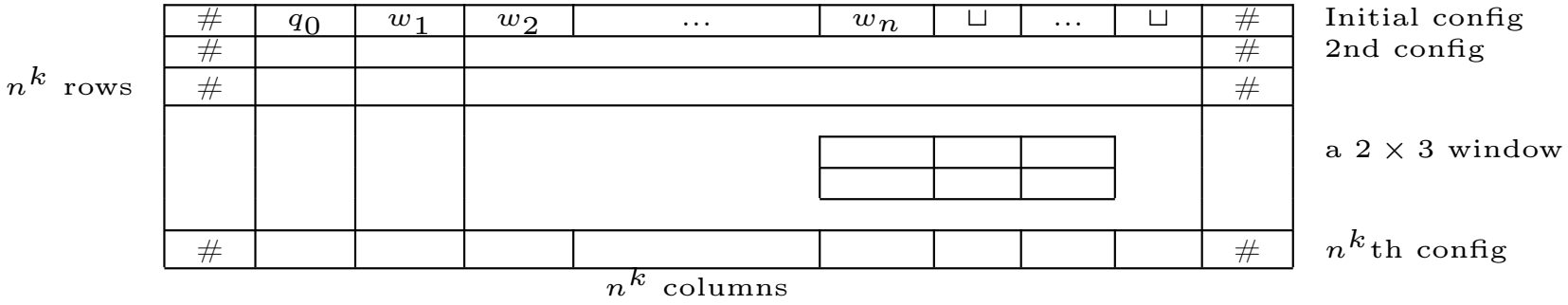
[1] Alternate version.

A tableau for N on w is an $n^k \times n^k$ table:

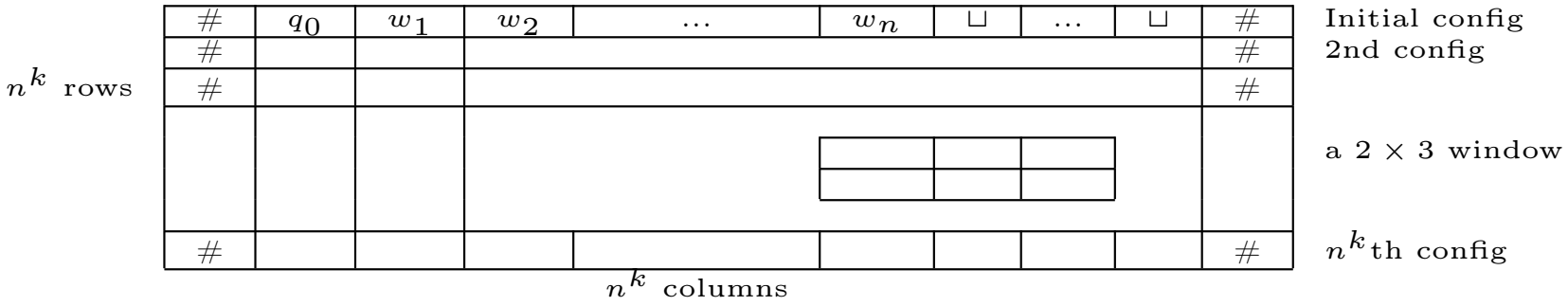
A tableau for N on w is an $n^k \times n^k$ table: *rows* are *configurations* of a branch of the computation of N on input w,

A tableau for N on w is an $n^k \times n^k$ table: *rows* are *configurations* of a branch of the computation of N on input w, *columns* are either states or symbols.

A tableau for N on w is an $n^k \times n^k$ table: *rows* are *configurations* of a branch of the computation of N on input w, *columns* are either states or symbols.



A tableau for N on w is an $n^k \times n^k$ table: *rows* are *configurations* of a branch of the computation of N on input w, *columns* are either states or symbols.



Problem of determining if N accepts $w \iff$ problem of determining if an accepting tableau for N on w exists.

Construct a poly time reduction f from A to SAT. On input w, reduction produces a formula $\phi_{N,w}$.

Construct a poly time reduction f from A to SAT. On input w, reduction produces a formula $\phi_{N,w}$.

- Let Q and Γ be the state set and tape alphabet of N, resp., and $C = Q \cup \Gamma \cup \{\#\}$.

Construct a poly time reduction f from A to SAT. On input w, reduction produces a formula $\phi_{N,w}$.

- Let Q and Γ be the state set and tape alphabet of N, resp., and $C = Q \cup \Gamma \cup \{\#\}$.
- For each $1 \leq i, j \leq n^k$ and $s \in C$, add variable $x_{i,j,s}$. $x_{i,j,s} = 1$ if $cell[i, j]$ of tableau contains s.

Construct a poly time reduction f from A to SAT. On input w, reduction produces a formula $\phi_{N,w}$.

- Let Q and Γ be the state set and tape alphabet of N, resp., and $C = Q \cup \Gamma \cup \{\#\}$.
- For each $1 \leq i, j \leq n^k$ and $s \in C$, add variable $x_{i,j,s}$. $x_{i,j,s} = 1$ if $cell[i, j]$ of tableau contains s.
- The formula $\phi_{N,w}$ is the AND of four parts, i.e.

$$\phi_{N,w} = \phi_{cell} \wedge \phi_{start} \wedge \phi_{move} \wedge \phi_{accept}.$$

$$\phi_{N,w} = \phi_{cell} \wedge \phi_{start} \wedge \phi_{move} \wedge \phi_{accept}.$$

Make sure assignment for $\phi_{N,w}$ corresponds to tableau entry:

$$\phi_{N,w} = \phi_{cell} \wedge \phi_{start} \wedge \phi_{move} \wedge \phi_{accept}.$$

Make sure assignment for $\phi_{N,w}$ corresponds to tableau entry: each $x_{i,j,s} = 1$ corresponds to one variable for each cell.

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

$$\phi_{N,w} = \phi_{cell} \wedge \phi_{start} \wedge \phi_{move} \wedge \phi_{accept}.$$

Make sure assignment for $\phi_{N,w}$ corresponds to tableau entry: each $x_{i,j,s} = 1$ corresponds to one variable for each cell.

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

Inside the brackets of ϕ_{cell}:

$$\phi_{N,w} = \phi_{cell} \wedge \phi_{start} \wedge \phi_{move} \wedge \phi_{accept}.$$

Make sure assignment for $\phi_{N,w}$ corresponds to tableau entry: each $x_{i,j,s} = 1$ corresponds to one variable for each cell.

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

Inside the brackets of ϕ_{cell}:

- 1st clause: at least one variable $x_{i,j,s} = 1$ corresponds to $cell[i,j] = s$.

$$\phi_{N,w} = \phi_{cell} \wedge \phi_{start} \wedge \phi_{move} \wedge \phi_{accept}.$$

Make sure assignment for $\phi_{N,w}$ corresponds to tableau entry: each $x_{i,j,s} = 1$ corresponds to one variable for each cell.

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

Inside the brackets of ϕ_{cell}:

- 1st clause: at least one variable $x_{i,j,s} = 1$ corresponds to $cell[i,j] = s$.
- 2nd clause: at most one $x_{i,j,s} = 1$ in corresponding cell (or for each pair of variables, at least one is 0).

$$\phi_{N,w} = \phi_{cell} \wedge \phi_{start} \wedge \phi_{move} \wedge \phi_{accept}.$$

Make sure assignment for $\phi_{N,w}$ corresponds to tableau entry: each $x_{i,j,s} = 1$ corresponds to one variable for each cell.

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

Inside the brackets of ϕ_{cell}:

- 1st clause: at least one variable $x_{i,j,s} = 1$ corresponds to $cell[i,j] = s$.
- 2nd clause: at most one $x_{i,j,s} = 1$ in corresponding cell (or for each pair of variables, at least one is 0).
- Both clauses: we ensure any satisfying assignment of $\phi_{N,w}$ corresponds to one symbol in each cell of tableau.

$$\phi_{N,w} = \phi_{cell} \wedge \phi_{start} \wedge \phi_{move} \wedge \phi_{accept}.$$

Formula ϕ_{start} ensures that 1st row of tableau is the *initial configuration* of N on $w = w_1 w_2 \ldots w_n$:

$$\phi_{start} = x_{1,1,\#} \wedge x_{1,2,q_0} \wedge x_{1,3,w_1} \wedge x_{1,4,w_2} \wedge \cdots \wedge x_{1,n+2,w_n} \wedge x_{1,n+3,\sqcup}$$

$$\wedge \cdots \wedge x_{1,n^k-1,\sqcup} \wedge x_{1,n^k,\#}$$

$$\phi_{N,w} = \phi_{cell} \wedge \phi_{start} \wedge \phi_{move} \wedge \phi_{accept}.$$

Formula ϕ_{start} ensures that 1st row of tableau is the *initial configuration* of N on $w = w_1 w_2 \ldots w_n$:

$$\phi_{start} = x_{1,1,\#} \wedge x_{1,2,q_0} \wedge x_{1,3,w_1} \wedge x_{1,4,w_2} \wedge \cdots \wedge x_{1,n+2,w_n} \wedge x_{1,n+3,\sqcup}$$

$$\wedge \cdots \wedge x_{1,n^k-1,\sqcup} \wedge x_{1,n^k,\#}$$

Formula ϕ_{accept} ensure that an *accepting configuration occurs* in tableau.

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

$$\phi_{N,w} = \phi_{cell} \wedge \phi_{start} \wedge \phi_{move} \wedge \phi_{accept}.$$

Finally, formula ϕ_{move} ensures row i of tableau *legally follows* from row $i-1$ based on transitions of N. Use concept of *legal windows*.

$$\phi_{N,w} = \phi_{cell} \wedge \phi_{start} \wedge \phi_{move} \wedge \phi_{accept}.$$

Finally, formula ϕ_{move} ensures row i of tableau *legally follows* from row $i-1$ based on transitions of N. Use concept of *legal windows*. A 2×3 window (split into upper and lower row) is legal if lower row can appear after a transition of N is applied to upper row.

$$\phi_{N,w} = \phi_{cell} \wedge \phi_{start} \wedge \phi_{move} \wedge \phi_{accept}.$$

Finally, formula ϕ_{move} ensures row i of tableau *legally follows* from row $i-1$ based on transitions of N. Use concept of *legal windows*. A 2×3 window (split into upper and lower row) is legal if lower row can appear after a transition of N is applied to upper row.
e.g. legal and illegal windows given: $\delta(q_1, a) = \{(q_1, b, R)\}$ and $\delta(q_1, b) = \{(q_2, c, L), (q_2, a, R)\}$

$$\phi_{N,w} = \phi_{cell} \wedge \phi_{start} \wedge \phi_{move} \wedge \phi_{accept}.$$

Finally, formula ϕ_{move} ensures row i of tableau *legally follows* from row $i-1$ based on transitions of N. Use concept of *legal windows*. A 2×3 window (split into upper and lower row) is legal if lower row can appear after a transition of N is applied to upper row.
e.g. legal and illegal windows given: $\delta(q_1, a) = \{(q_1, b, R)\}$ and $\delta(q_1, b) = \{(q_2, c, L), (q_2, a, R)\}$

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

$$\phi_{N,w} = \phi_{cell} \wedge \phi_{start} \wedge \phi_{move} \wedge \phi_{accept}.$$

Finally, formula ϕ_{move} ensures row i of tableau *legally follows* from row $i-1$ based on transitions of N. Use concept of *legal windows*. A 2×3 window (split into upper and lower row) is legal if lower row can appear after a transition of N is applied to upper row.
e.g. legal and illegal windows given: $\delta(q_1, a) = \{(q_1, b, R)\}$ and $\delta(q_1, b) = \{(q_2, c, L), (q_2, a, R)\}$

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

a	b	a
a	a	a

a	q_1	b
q_1	a	a

b	q_1	b
q_2	b	q_2

$$\phi_{N,w} = \phi_{cell} \wedge \phi_{start} \wedge \phi_{move} \wedge \phi_{accept}.$$

Finally, formula ϕ_{move} ensures row i of tableau *legally follows* from row $i-1$ based on transitions of N. Use concept of *legal windows*. A 2×3 window (split into upper and lower row) is legal if lower row can appear after a transition of N is applied to upper row.
e.g. legal and illegal windows given: $\delta(q_1, a) = \{(q_1, b, R)\}$ and $\delta(q_1, b) = \{(q_2, c, L), (q_2, a, R)\}$

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

a	b	a
a	a	a

a	q_1	b
q_1	a	a

b	q_1	b
q_2	b	q_2

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

Replace the text 'the (i, j) window is legal' with the following formula:

$$\bigwedge_{\substack{a_1, a_2, \ldots, a_6 \\ \text{is a legalwindow}}} \left(x_{i,j-1,a_1} \wedge x_{i,j,a_2} \wedge x_{i,j+1,a_3} \wedge x_{i+1,j-1,a_4} \wedge x_{i+1,j,a_5} \wedge x_{i+1,j+1,a_6}\right)$$

- Estimate total number of variables:
  - $n^k \times n^k$ table contains n^{2k} cells.

- Estimate total number of variables:
  - $n^k \times n^k$ table contains n^{2k} cells.
  - Each cell has $l = |C|$ variables associated with it.

- Estimate total number of variables:
  - $n^k \times n^k$ table contains n^{2k} cells.
  - Each cell has $l = |C|$ variables associated with it.
  - Since l depends only on TM N and not on input length, total number of variables is $O(n^{2k})$. Polynomial!

- Estimate total number of variables:
  - $n^k \times n^k$ table contains n^{2k} cells.
  - Each cell has $l = |C|$ variables associated with it.
  - Since l depends only on TM N and not on input length, total number of variables is $O(n^{2k})$. Polynomial!

- Estimate total number of variables:
  - $n^k \times n^k$ table contains n^{2k} cells.
  - Each cell has $l = |C|$ variables associated with it.
  - Since l depends only on TM N and not on input length, total number of variables is $O(n^{2k})$. Polynomial!
- Estimate size of each part of $\phi_{N,w}$:
  - ϕ_{cell} contains a fixed-size fragment of the formula for each cell, so its size is $O(n^{2k})$.

- Estimate total number of variables:
  - $n^k \times n^k$ table contains n^{2k} cells.
  - Each cell has $l = |C|$ variables associated with it.
  - Since l depends only on TM N and not on input length, total number of variables is $O(n^{2k})$. Polynomial!
- Estimate size of each part of $\phi_{N,w}$:
  - ϕ_{cell} contains a fixed-size fragment of the formula for each cell, so its size is $O(n^{2k})$.
  - ϕ_{start} has a fragment for each cell in the top row, so its size is $O(n^k)$.

- Estimate total number of variables:
  - $n^k \times n^k$ table contains n^{2k} cells.
  - Each cell has $l = |C|$ variables associated with it.
  - Since l depends only on TM N and not on input length, total number of variables is $O(n^{2k})$. Polynomial!
- Estimate size of each part of $\phi_{N,w}$:
  - ϕ_{cell} contains a fixed-size fragment of the formula for each cell, so its size is $O(n^{2k})$.
  - ϕ_{start} has a fragment for each cell in the top row, so its size is $O(n^k)$.
  - ϕ_{move} and ϕ_{accept} each contains a fixed-size fragment of the formula for each cell, so its size is $O(n^{2k})$.

- Estimate total number of variables:
  - $n^k \times n^k$ table contains n^{2k} cells.
  - Each cell has $l = |C|$ variables associated with it.
  - Since l depends only on TM N and not on input length, total number of variables is $O(n^{2k})$. Polynomial!
- Estimate size of each part of $\phi_{N,w}$:
  - ϕ_{cell} contains a fixed-size fragment of the formula for each cell, so its size is $O(n^{2k})$.
  - ϕ_{start} has a fragment for each cell in the top row, so its size is $O(n^k)$.
  - ϕ_{move} and ϕ_{accept} each contains a fixed-size fragment of the formula for each cell, so its size is $O(n^{2k})$.
  - Upper bound of $\phi_{N,w}$'s total size: $O(n^{2k})$.

- Estimate total number of variables:
  - $n^k \times n^k$ table contains n^{2k} cells.
  - Each cell has $l = |C|$ variables associated with it.
  - Since l depends only on TM N and not on input length, total number of variables is $O(n^{2k})$. Polynomial!
- Estimate size of each part of $\phi_{N,w}$:
  - ϕ_{cell} contains a fixed-size fragment of the formula for each cell, so its size is $O(n^{2k})$.
  - ϕ_{start} has a fragment for each cell in the top row, so its size is $O(n^k)$.
  - ϕ_{move} and ϕ_{accept} each contains a fixed-size fragment of the formula for each cell, so its size is $O(n^{2k})$.
  - Upper bound of $\phi_{N,w}$'s total size: $O(n^{2k})$.
  - This bound is good enough! Bound shows size of $\phi_{N,w}$ is polynomial in n. If more than poly, reduction f of A to SAT cannot have any chance of generating it in poly time.

- Estimate total number of variables:
  - $n^k \times n^k$ table contains n^{2k} cells.
  - Each cell has $l = |C|$ variables associated with it.
  - Since l depends only on TM N and not on input length, total number of variables is $O(n^{2k})$. Polynomial!
- Estimate size of each part of $\phi_{N,w}$:
  - ϕ_{cell} contains a fixed-size fragment of the formula for each cell, so its size is $O(n^{2k})$.
  - ϕ_{start} has a fragment for each cell in the top row, so its size is $O(n^k)$.
  - ϕ_{move} and ϕ_{accept} each contains a fixed-size fragment of the formula for each cell, so its size is $O(n^{2k})$.
  - Upper bound of $\phi_{N,w}$'s total size: $O(n^{2k})$.
  - This bound is good enough! Bound shows size of $\phi_{N,w}$ is polynomial in n. If more than poly, reduction f of A to SAT cannot have any chance of generating it in poly time.

- Estimate total number of variables:
  - $n^k \times n^k$ table contains n^{2k} cells.
  - Each cell has $l = |C|$ variables associated with it.
  - Since l depends only on TM N and not on input length, total number of variables is $O(n^{2k})$. Polynomial!
- Estimate size of each part of $\phi_{N,w}$:
  - ϕ_{cell} contains a fixed-size fragment of the formula for each cell, so its size is $O(n^{2k})$.
  - ϕ_{start} has a fragment for each cell in the top row, so its size is $O(n^k)$.
  - ϕ_{move} and ϕ_{accept} each contains a fixed-size fragment of the formula for each cell, so its size is $O(n^{2k})$.
  - Upper bound of $\phi_{N,w}$'s total size: $O(n^{2k})$.
  - This bound is good enough! Bound shows size of $\phi_{N,w}$ is polynomial in n. If more than poly, reduction f of A to SAT cannot have any chance of generating it in poly time.
- Each part of $\phi_{N,w}$ is composed of many nearly identical (repetitive) fragments: they *differ* only at the *indices* in a *simple way*. Hence, we can easily construct a reduction that produces $\phi_{N,w}$ in poly time given w.

- Estimate total number of variables:
  - $n^k \times n^k$ table contains n^{2k} cells.
  - Each cell has $l = |C|$ variables associated with it.
  - Since l depends only on TM N and not on input length, total number of variables is $O(n^{2k})$. Polynomial!
- Estimate size of each part of $\phi_{N,w}$:
  - ϕ_{cell} contains a fixed-size fragment of the formula for each cell, so its size is $O(n^{2k})$.
  - ϕ_{start} has a fragment for each cell in the top row, so its size is $O(n^k)$.
  - ϕ_{move} and ϕ_{accept} each contains a fixed-size fragment of the formula for each cell, so its size is $O(n^{2k})$.
  - Upper bound of $\phi_{N,w}$'s total size: $O(n^{2k})$.
  - This bound is good enough! Bound shows size of $\phi_{N,w}$ is polynomial in n. If more than poly, reduction f of A to SAT cannot have any chance of generating it in poly time.
- Each part of $\phi_{N,w}$ is composed of many nearly identical (repetitive) fragments: they *differ* only at the *indices* in a *simple way*. Hence, we can easily construct a reduction that produces $\phi_{N,w}$ in poly time given w.

Since any A in **NP** reduces to SAT, SAT is **NP**-complete.

Back to $3SAT$

Corollary

*3SAT is **NP**-complete*

Back to $3SAT$

Corollary

*$3SAT$ is **NP**-complete*

$\triangleright$ Obviously, $3SAT$ is in **NP**. Need to show every A in **NP** reduces to $3SAT$ in poly time.

Back to $3SAT$

Corollary

*$3SAT$ is **NP**-complete*

$\triangleright$ Obviously, $3SAT$ is in **NP**. Need to show every A in **NP** reduces to $3SAT$ in poly time.

- Modify proof of SAT to directly produce a 3CNF formula.

Back to $3SAT$

Corollary

*$3SAT$ is **NP**-complete*

$\triangleright$ Obviously, $3SAT$ is in **NP**. Need to show every A in **NP** reduces to $3SAT$ in poly time.

- Modify proof of SAT to directly produce a 3CNF formula. Note: $\phi_{cell}, \phi_{accept}, \phi_{start}$ are already in CNF.

Back to $3SAT$

Corollary

*$3SAT$ is **NP**-complete*

$\triangleright$ Obviously, $3SAT$ is in **NP**. Need to show every A in **NP** reduces to $3SAT$ in poly time.

- Modify proof of SAT to directly produce a 3CNF formula. Note: $\phi_{cell}, \phi_{accept}, \phi_{start}$ are already in CNF.
- ϕ_{move} is a big AND of subformulas, each of which is an OR of ANDs.

Back to $3SAT$

Corollary

*$3SAT$ is **NP**-complete*

$\triangleright$ Obviously, $3SAT$ is in **NP**. Need to show every A in **NP** reduces to $3SAT$ in poly time.

- Modify proof of SAT to directly produce a 3CNF formula. Note: $\phi_{cell}, \phi_{accept}, \phi_{start}$ are already in CNF.
- ϕ_{move} is a big AND of subformulas, each of which is an OR of ANDs. *Note: Distributive laws state that we can replace an OR of ANDs with an equivalent AND of ORs.*

Back to $3SAT$

Corollary

*$3SAT$ is **NP**-complete*

$\triangleright$ Obviously, $3SAT$ is in **NP**. Need to show every A in **NP** reduces to $3SAT$ in poly time.

- Modify proof of SAT to directly produce a 3CNF formula. Note: $\phi_{cell}, \phi_{accept}, \phi_{start}$ are already in CNF.
- ϕ_{move} is a big AND of subformulas, each of which is an OR of ANDs. *Note: Distributive laws state that we can replace an OR of ANDs with an equivalent AND of ORs.*
- If clause contains l variables, $(a_1 \vee a_2 \vee \cdots \vee a_l)$, we can replace it with the $l-2$ clauses
  $(a_1 \vee a_2 \vee z_1) \wedge (\overline{z_1} \vee a_3 \vee z_2) \wedge (\overline{z_2} \vee a_4 \vee z_3) \wedge \cdots \wedge (\overline{z_{l-3}} \vee a_{l-1} \vee a_l)$.
- Now we have a formula in CNF with 3 variables per clause!

NP-completeness

More on NP-completeness

NP-completeness is widespread!

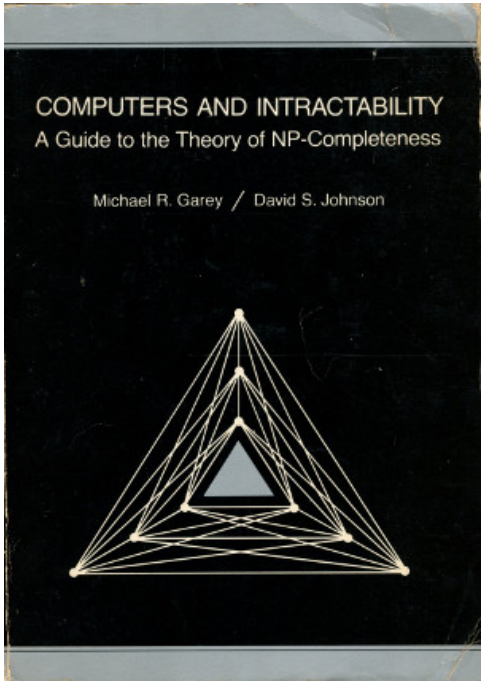
NP-complete problems are found in practically any field requiring computation, e.g.

`https://en.wikipedia.org/wiki/List_of_NP-complete_problems`

NP-completeness is widespread!

NP-complete problems are found in practically any field requiring computation, e.g.
`https://en.wikipedia.org/wiki/List_of_NP-complete_problems`

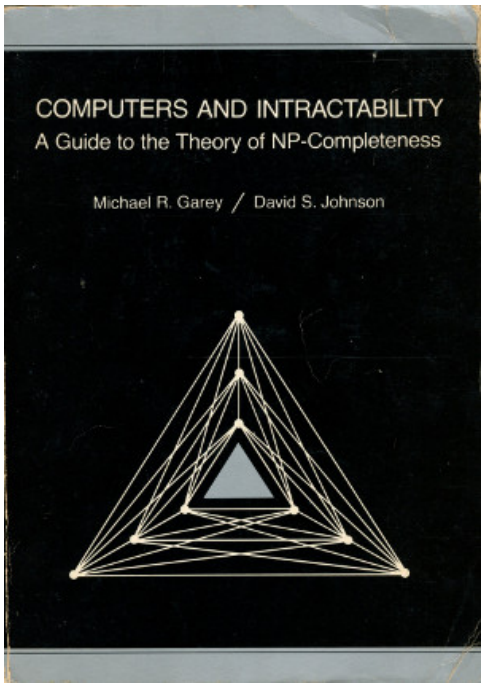
The "Bible":



NP-completeness is widespread!

NP-complete problems are found in practically any field requiring computation, e.g.
`https://en.wikipedia.org/wiki/List_of_NP-complete_problems`

The "Bible":



If you cannot find a poly time algo for a problem, good idea to check if problem is **NP**-complete.

Corollary

*CLIQUE is **NP**-complete.*

Corollary

*CLIQUE is **NP**-complete.*

The *vertex cover* (or *node cover*) problem:

A vertex cover of an undirected graph $G = (V, E)$ is a set $V' \subseteq V$ where all nodes in V' "cover" all edges of G,

Corollary

*CLIQUE is **NP**-complete.*

The *vertex cover* (or *node cover*) problem:

A vertex cover of an undirected graph $G = (V, E)$ is a set $V' \subseteq V$ where all nodes in V' "cover" all edges of G, i.e. if $(u, v) \in E$, then either $u \in V'$ or $v \in V'$.

Corollary

*CLIQUE is **NP**-complete.*

The *vertex cover* (or *node cover*) problem:

A vertex cover of an undirected graph $G = (V, E)$ is a set $V' \subseteq V$ where all nodes in V' "cover" all edges of G, i.e. if $(u, v) \in E$, then either $u \in V'$ or $v \in V'$.
The vertex cover problem asks whether a graph contains a vertex cover of a size k:

$VERTEX\text{-}COVER = \{\langle G, k\rangle \mid$
G is an undirected graph with a k-node vertex cover$\}$.

Corollary

*CLIQUE is **NP**-complete.*

The *vertex cover* (or *node cover*) problem:

A vertex cover of an undirected graph $G = (V, E)$ is a set $V' \subseteq V$ where all nodes in V' "cover" all edges of G, i.e. if $(u, v) \in E$, then either $u \in V'$ or $v \in V'$.
The vertex cover problem asks whether a graph contains a vertex cover of a size k:

$VERTEX\text{-}COVER = \{\langle G, k\rangle \mid$
G is an undirected graph with a k-node vertex cover$\}$.

Theorem

*$VERTEX\text{-}COVER$ is **NP**-complete.*

Corollary

*CLIQUE is **NP**-complete.*

The *vertex cover* (or *node cover*) problem:

A vertex cover of an undirected graph $G = (V, E)$ is a set $V' \subseteq V$ where all nodes in V' "cover" all edges of G, i.e. if $(u, v) \in E$, then either $u \in V'$ or $v \in V'$.
The vertex cover problem asks whether a graph contains a vertex cover of a size k:

$VERTEX\text{-}COVER = \{\langle G, k\rangle \mid$
G is an undirected graph with a k-node vertex cover$\}$.

Theorem

*$VERTEX\text{-}COVER$ is **NP**-complete.*

Need to show: $3SAT \leq_p VERTEX\text{-}COVER$.
Reduction maps a 3CNF ϕ to a graph G and number k, so ϕ is satisfiable only if G has a k-node vertex cover.

$3SAT \leq_p VERTEX\text{-}COVER$

Poly time reduction technique: from $3SAT$ *to some language* A*, look for structures in* A *that can simulate variables and clauses in Boolean formulas.*
Let us call such structures as gadgets.

$3SAT \leq_p VERTEX\text{-}COVER$

Poly time reduction technique: from $3SAT$ *to some language* A*, look for structures in* A *that can simulate variables and clauses in Boolean formulas.*
Let us call such structures as gadgets.

- For each variable x in ϕ, we produce an edge connecting two nodes.
  We label the 2 nodes in this gadget x and $\overline{x}$.

$3SAT \leq_p VERTEX\text{-}COVER$

Poly time reduction technique: from $3SAT$ *to some language* A*, look for structures in* A *that can simulate variables and clauses in Boolean formulas.*
Let us call such structures as gadgets.

- For each variable x in ϕ, we produce an edge connecting two nodes.
  We label the 2 nodes in this gadget x and $\overline{x}$.
- Each clause is a triple of three nodes that are labeled with the 3 literals of the clause.
  These nodes are connected to each other and to the nodes in the variables gadgets that have the identical labels.

$3SAT \leq_p VERTEX\text{-}COVER$

Poly time reduction technique: from $3SAT$ *to some language* A*, look for structures in* A *that can simulate variables and clauses in Boolean formulas.*
Let us call such structures as gadgets.

- For each variable x in ϕ, we produce an edge connecting two nodes.
  We label the 2 nodes in this gadget x and $\overline{x}$.
- Each clause is a triple of three nodes that are labeled with the 3 literals of the clause.
  These nodes are connected to each other and to the nodes in the variables gadgets that have the identical labels.
- Thus the total number of nodes that appear in G is $2m + 3l$ where ϕ has m variables and l clauses. Let $k = m + 2l$.

$3SAT \leq_p VERTEX\text{-}COVER$

Poly time reduction technique: from $3SAT$ *to some language* A*, look for structures in* A *that can simulate variables and clauses in Boolean formulas.*
Let us call such structures as gadgets.

- For each variable x in ϕ, we produce an edge connecting two nodes.
  We label the 2 nodes in this gadget x and $\overline{x}$.
- Each clause is a triple of three nodes that are labeled with the 3 literals of the clause.
  These nodes are connected to each other and to the nodes in the variables gadgets that have the identical labels.
- Thus the total number of nodes that appear in G is $2m + 3l$ where ϕ has m variables and l clauses. Let $k = m + 2l$.

E.g. $\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_2) \wedge (\overline{x}_1 \vee x_2 \vee x_2)$.

We need to show that ϕ is satisfiable if and only if G has a vertex cover with k nodes.

We need to show that ϕ is satisfiable if and only if G has a vertex cover with k nodes.

$\triangleright$ Suppose ϕ is satisfiable.

We need to show that ϕ is satisfiable if and only if G has a vertex cover with k nodes.

$\triangleright$ Suppose ϕ is satisfiable.

- Put the nodes of the variable gadgets that correspond to the TRUE literals in the assignment into the vertex cover.

We need to show that ϕ is satisfiable if and only if G has a vertex cover with k nodes.

$\triangleright$ Suppose ϕ is satisfiable.

- Put the nodes of the variable gadgets that correspond to the TRUE literals in the assignment into the vertex cover.
- Select one TRUE literal in every clause and put the remaining two nodes from every clause gadget into the vertex cover.

We need to show that ϕ is satisfiable if and only if G has a vertex cover with k nodes.

$\triangleright$ Suppose ϕ is satisfiable.

- Put the nodes of the variable gadgets that correspond to the TRUE literals in the assignment into the vertex cover.
- Select one TRUE literal in every clause and put the remaining two nodes from every clause gadget into the vertex cover.
- These $k = m + 2l$ nodes cover all edges. (Why?)

We need to show that ϕ is satisfiable if and only if G has a vertex cover with k nodes.

$\triangleright$ Suppose ϕ is satisfiable.

- Put the nodes of the variable gadgets that correspond to the TRUE literals in the assignment into the vertex cover.
- Select one TRUE literal in every clause and put the remaining two nodes from every clause gadget into the vertex cover.
- These $k = m + 2l$ nodes cover all edges. (Why?)
- Hence, G has a vertex cover with k nodes.

We need to show that ϕ is satisfiable if and only if G has a vertex cover with k nodes.

$\triangleright$ Suppose G has a vertex cover with k nodes.

We need to show that ϕ is satisfiable if and only if G has a vertex cover with k nodes.

$\triangleright$ Suppose G has a vertex cover with k nodes.

- Vertex cover must contain one node in each variable gadget and two in every clause gadget in order to cover all edges (edges of the variable and the edges within the clause gadgets).

We need to show that ϕ is satisfiable if and only if G has a vertex cover with k nodes.

$\triangleright$ Suppose G has a vertex cover with k nodes.

- Vertex cover must contain one node in each variable gadget and two in every clause gadget in order to cover all edges (edges of the variable and the edges within the clause gadgets). That accounts for all the k nodes, so none are left over.

We need to show that ϕ is satisfiable if and only if G has a vertex cover with k nodes.

$\triangleright$ Suppose G has a vertex cover with k nodes.

- Vertex cover must contain one node in each variable gadget and two in every clause gadget in order to cover all edges (edges of the variable and the edges within the clause gadgets). That accounts for all the k nodes, so none are left over.
- We take the nodes of the variable gadgets that are in the vertex cover and assign the corresponding literals TRUE.

We need to show that ϕ is satisfiable if and only if G has a vertex cover with k nodes.

$\triangleright$ Suppose G has a vertex cover with k nodes.

- Vertex cover must contain one node in each variable gadget and two in every clause gadget in order to cover all edges (edges of the variable and the edges within the clause gadgets). That accounts for all the k nodes, so none are left over.
- We take the nodes of the variable gadgets that are in the vertex cover and assign the corresponding literals TRUE. This assignment satisfies ϕ. (Why?)

We need to show that ϕ is satisfiable if and only if G has a vertex cover with k nodes.

$\triangleright$ Suppose G has a vertex cover with k nodes.

- Vertex cover must contain one node in each variable gadget and two in every clause gadget in order to cover all edges (edges of the variable and the edges within the clause gadgets). That accounts for all the k nodes, so none are left over.
- We take the nodes of the variable gadgets that are in the vertex cover and assign the corresponding literals TRUE. This assignment satisfies ϕ. (Why?)
- Hence, ϕ has a satisfying assignment.

Theorem

*SUBSETSUM is **NP**-complete.*

Theorem

*SUBSETSUM is **NP**-complete.*

We need to show that $3SAT \leq_p SUBSETSUM$.

Theorem

*SUBSETSUM is **NP**-complete.*

We need to show that $3SAT \leq_p SUBSETSUM$.

The reduction maps a 3CNF ϕ with variables $x_1, \cdots, x_l$ and clauses $c_1, \cdots c_k$ into an instance of the $SUBSETSUM$ problem $\langle S, t \rangle$ wherein the elements of S and the number t are the rows in the table on the next slide.

	1	2	3	4	...	l	c_1	c_2	...	c_k
y_1	1	0	0	0	...	0	1	0	...	0
z_1	1	0	0	0	...	0	0	0	...	0
y_2		1	0	0	...	0	0	1	...	0
z_2		1	0	0	...	0	1	0	...	0
$\vdots$					$\ddots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$
y_l						1	0	0	...	0
z_l						1	0	0	...	0
g_1							1	0	...	0
h_1							1	0	...	0
g_2								1	...	0
h_2								1	...	0
$\vdots$									$\ddots$	$\vdots$
g_k										1
h_k										1
t	1	1	1	1	...	1	3	3	...	3

The reduction maps a 3CNF ϕ with variables $x_1, \cdots, x_l$ and clauses $c_1, \cdots c_k$ into an instance of the $SUBSETSUM$ problem $\langle S, t \rangle$ wherein the elements of S and the number t are the rows in the table on the previous slide, expressed in ordinary decimal notation.

The reduction maps a 3CNF ϕ with variables $x_1, \cdots, x_l$ and clauses $c_1, \cdots c_k$ into an instance of the $SUBSETSUM$ problem $\langle S, t \rangle$ wherein the elements of S and the number t are the rows in the table on the previous slide, expressed in ordinary decimal notation. Constructed table has $2l + 2k$ decimal numbers (rows) and the final row for t. Each decimal number has at most $l + k$ digits (columns).

- S contains one pair of numbers, y_i, z_i for each variable x_i in ϕ.
- The decimal representation is in two parts.
  The left-hand part comprises a 1 followed by $l - i$ 0s.
  The right-hand part contains one digit for each clause.
  - the j^{th} digit of y_i is 1 if clause c_j contains literal x_i
  - the j^{th} digit of z_i is 1 if clause c_j contains literal $\overline{x_i}$
- S also contains one pair of numbers g_j, h_j for each clause c_j.
  These numbers are equal and consist of a 1 followed by $k - j$ 0s.
- The target number t, consists of l number of 1s followed by k number of 3s.

Suppose ϕ is satisfiable.

Suppose ϕ is satisfiable.

We construct a subset of S as follows:

- Select y_i if $x_i = 1$. Select z_i otherwise.
  - so far, the sum contains a 1 in each of the first l digits and a number between 1 and 3 in each of the last k digits
- Select enough of the g and h numbers to bring each of the last k digits up to 3.
- Sum would then be equal to t.

	1	2	3	4	...	l	c_1	c_2	...	c_k
y_1	1	0	0	0	...	0	1	0	...	0
z_1	1	0	0	0	...	0	0	0	...	0
y_2		1	0	0	...	0	0	1	...	0
z_2		1	0	0	...	0	1	0	...	0
$\vdots$					$\ddots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$
y_l						1	0	0	...	0
z_l						1	0	0	...	0
g_1							1	0	...	0
h_1							1	0	...	0
g_2								1	...	0
h_2								1	...	0
$\vdots$									$\ddots$	$\vdots$
g_k										1
h_k										1
t	1	1	1	1	...	1	3	3	...	3

Suppose that a subset of S sums to t.

Suppose that a subset of S sums to t.
Note: a "carry" into next column in the table never occurs when a subset of S is added, since each column sums up to at most 5.

Suppose that a subset of S sums to t.
Note: a "carry" into next column in the table never occurs when a subset of S is added, since each column sums up to at most 5.

We make the satisfying assignment as follows:

- To get a 1 in each of the first l columns, the subset must have either y_i or z_i for each i but not both.
  - if the subset contains y_i, assign $x_i = 1$; otherwise, $x_i = 0$.
- For each of the final k columns, at most 2 can come from g and h. To sum up to 3, thus matching the digit (column) of t, at least 1 (for each of these columns) must come from some y_i or z_i in the subset.
  - if it is y_i, then x_i appears in c_j and $x_i = 1$, so c_j is satisfied.
  - if it is z_i, then $\overline{x_i}$ appears in c_j and $x_i = 0$, so c_j is satisfied.
- Therefore, ϕ is satisfied.

More NP-complete problems

$HAMPATH, GRAPHCOLOR, KNAPSACK, LCSP$, Super Mario Bros., Candy Crush Saga, The Legend of Zelda, Pokémon et al.

`https://en.wikipedia.org/wiki/List_of_NP-complete_problems`

Fin (wakas)

Thanks for the attention.

Questions?

Reading assignment(s)

[Sipser 2005] Chapter 7.4, 7.5 or

[Hopcroft et al 2001] Chapter 10.2, 10.3, 10.4

References:

[Sipser 2005] M. Sipser. *Introduction to the Theory of Computation*: 2ed. PWS Publishing Company, 2005.

[Hopcroft, Ullman 1979] J. Hopcroft, J. Ullman. *Introduction to Automata Theory, Languages, and Computation*: Addison-Wesley, 1979.

[Hopcroft et al 2001] J. Hopcroft, R. Motwani, J. Ullman. *Introduction to Automata Theory, Languages, and Computation*: Addison-Wesley, 2001.

[Hernandez 2014] CS133 lecture slides of N.H.S. Hernandez, 2014