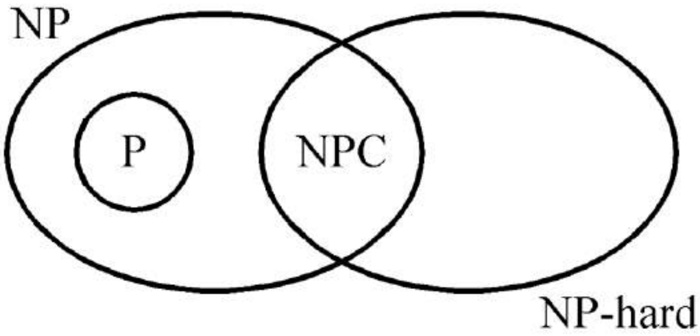
§The Theory of NP-Completeness



NP: the class of decision problem which can be solved by a non-deterministic polynomial algorithm.

P: the class of problems which can be solved by a deterministic polynomial algorithm.

NP-hard: the class of problems to which every NP problem reduces.

NP-complete (**NPC**): the class of problems which are NP-hard and belong to NP.

Def: Problem A reduces to problem B ($A \propto B$) iff A can be solved by a deterministic polynomial time algorithm using a deterministic algorithm that solves B in polynomial time.

- Up to now, none of the NPC problems can be solved by a deterministic polynomial time algorithm in the worst case.
- It does not seem to have any polynomial time algorithm to solve the NPC problems.
- The lower bound of any NPC problem seems to be in the order of an exponential function.

- The theory of NP-completeness always considers the worst case.
- Not all NP problems are difficult. (e.g. the MST problem is a NP problem.)
- If A, B $\in$ NPC, then A $\propto$ B and B $\propto$ A.

- Theory of NP-completeness
  If any NPC problem can be solved in polynomial time, then all NP problems can be solved in polynomial time. (NP = P)

- Decision problems
  the solution is simply "Yes" or "No".
  $\begin{cases} \text{optimization problem: more difficult} \\ \text{decision problem} \end{cases}$

e.g. the traveling salesperson problem
  optimization version:
    find the shortest tour
  decision version:
    Is there a tour whose total length is less than or equal to a constant c.

- Solving an optimization problem by a decision algorithm :

$$\begin{array}{l} \text{give } c_1 \text{ and test} \\ \qquad c_2 \\ \qquad \vdots \\ \qquad c_n \end{array}$$

The satisfiability problem

the logical formula :

$$x_1 \vee x_2 \vee x_3$$
$$\& -x_1$$
$$\& -x_2$$

the assignment :

$$x_1 \leftarrow F, x_2 \leftarrow F, x_3 \leftarrow T$$

will make the above formula true

$(-x_1, -x_2, x_3)$ represents

$$x_1 \leftarrow F, x_2 \leftarrow F, x_3 \leftarrow T$$

- If there is at least one assignment which satisfies a formula, then we say that this formula is satisfiable; otherwise, it is unsatisfiable.
- An unsatisfiable formula :

$$x_1 \vee x_2$$
$$\& x_1 \vee -x_2$$
$$\& -x_1 \vee x_2$$
$$\& -x_1 \vee -x_2$$

- The satisfiability problem: given a Boolean formula, determine whether this formula is satisfiable or not.

- A **literal** : x_i or $-x_i$
- A **clause** : $x_1 \vee x_2 \vee -x_3 \equiv c_i$
- A **formula** : conjunctive normal form

$$c_1 \& c_2 \& \cdots \& c_m$$

- resolution principle

c_1 : $-x_1 \vee -x_2 \vee x_3$

c_2 : $x_1 \vee x_4$

$\Rightarrow c_3$: $-x_2 \vee x_3 \vee x_4$

- if no new clauses can be deduced

  $\Rightarrow$ satisfiable

	$-x_1 \vee -x_2 \vee x_3$	(1)
	x_1	(2)
	x_2	(3)
(1) & (2)	$-x_2 \vee x_3$	(4)
(4) & (3)	x_3	(5)
(1) & (3)	$-x_1 \vee x_3$	(6)

- If an empty clause is deduced

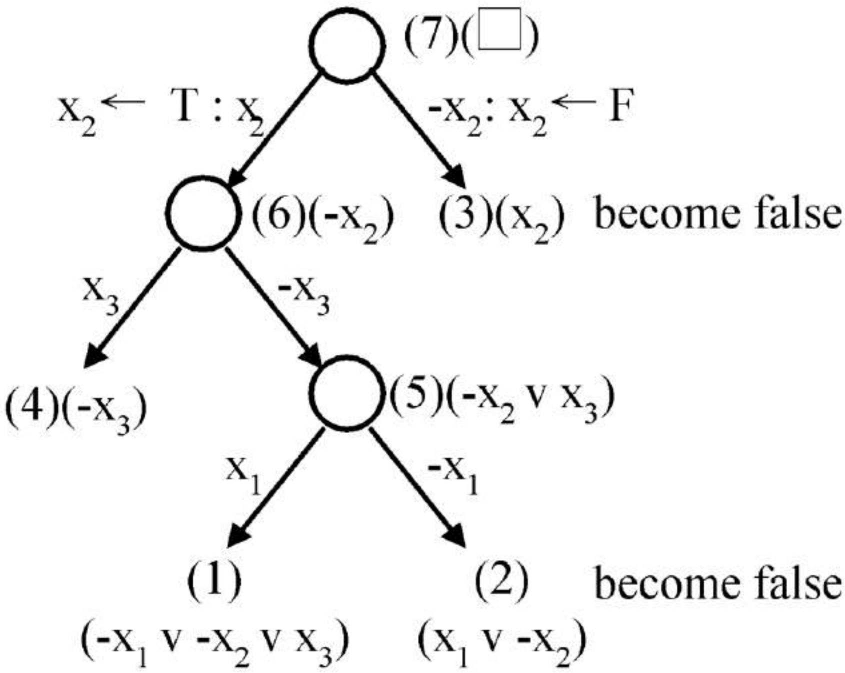
  $\Rightarrow$ unsatisfiable

$-x_1 \vee -x_2 \vee x_3$	(1)
$x_1 \vee -x_2$	(2)
x_2	(3)
$-x_3$	(4)

$\Downarrow$ deduce

(1) & (2)	$-x_2 \vee x_3$	(5)
(4) & (5)	$-x_2$	(6)
(6) & (3)	$\square$	(7)

semantic tree :



In a semantic tree, each path from the root to a leaf node represents a class of assignments.

If each leaf node is attached with a clause $\Rightarrow$unsatisfiable.

□**Nondeterministic** algorithms

(1) **guessing**

(2) **checking**

If the checking stage of a nondeterministic algorithm is of polynomial time-complexity, then this algorithm is called an NP (nondeterministic polynomial) algorithm.

NP problems : (must be decision problems)

e.g. searching, MST
sorting
satisfiability problem (SAT)
traveling salesperson problem (TSP)

- decision version of sorting :
Given $a_1, a_2, \cdots, a_n$ and c, is there a permutation of a_i's ($a_1', a_2', \cdots, a_n'$) such that $|a_2' - a_1'| + |a_3' - a_2'| + \cdots + |a_n' - a_{n-1}'| < c$?

- Not all decision problems are NP problems
e.g. halting problem :
Given a program with a certain input data, will the program terminate or not?
NP-hard
Undecidable

- 〔**Horowitz and sahni 1978**〕
(i) choice(s): arbitrarily chooses one of the elements in set S
(ii) failure: an unsuccessful completion

(iii) success: a successful completion

<u>nondeterministic searching</u> :

$j \leftarrow$ choice(1 : n)

if A(j) = x then success

else failure

- A nondeterministic algorithm terminates unsuccessfully iff there exist no set of choices leading to a success signal.
- The time required for choice(1 : n) is O(1).
- A deterministic interpretation of a non-deterministic algorithm can be made by allowing <u>unbounded parallelism</u> in computation.
- <u>nondeterministic sorting</u> :

$B \leftarrow 0$

$$\text{guessing}\left\{\begin{array}{l}\text{for } i = 1 \text{ to } n \text{ do}\\ \quad j \leftarrow \text{choice}(1:n)\\ \quad \text{if } B[j] \neq 0 \text{ then failure}\\ \quad B[j] = A[i]\end{array}\right.$$

$$\text{checking}\left\{\begin{array}{l}\text{for } i = 1 \text{ to } n\text{-}1 \text{ do}\\ \quad \text{if } B[i] > B[i+1] \text{ then failure}\end{array}\right.$$

success

- <u>nondeterministic SAT</u>

$$\text{guessing}\left\{\begin{array}{l}\text{for } i = 1 \text{ to } n \text{ do}\\ \quad x_i \leftarrow \text{choice(true, false)}\end{array}\right.$$

$$\text{checking}\left\{\begin{array}{l}\text{if } E(x_1, x_2, \cdots, x_n) \text{ is true}\\ \quad \text{then success}\\ \quad \text{else failure}\end{array}\right.$$

□ Cook's theorem

NP = P iff the satisfiability problem is a P problem.

- SAT is NP-complete.
- Every NP problem reduces to SAT.
- transforming searching to SAT :
  Does there exist a number in { x(1), x(2), …, x(n) }, which is equal to 7?
  Assume n = 2

```
i = choice(1, 2)
if x(i) = 7 then SUCCESS
else FAILURE
```

$$
\begin{array}{ll}
 & i=1 \quad \vee \quad i=2 \\
\& & i=1 \rightarrow i\neq 2 \\
\& & i=2 \rightarrow i\neq 1 \\
\& & x(1)=7 \ \& \ i=1 \rightarrow \text{SUCCESS} \\
\& & x(2)=7 \ \& \ i=2 \rightarrow \text{SUCCESS} \\
\& & x(1)\neq 7 \ \& \ i=1 \rightarrow \text{FAILURE} \\
\& & x(2)\neq 7 \ \& \ i=2 \rightarrow \text{FAILURE} \\
\& & \text{FAILURE} \rightarrow -\text{SUCCESS} \\
\& & \text{SUCCESS} \quad \text{(Guarantees a successful termination)} \\
\& & x(1)=7 \quad \text{(Input Data)} \\
\& & x(2)\neq 7
\end{array}
$$

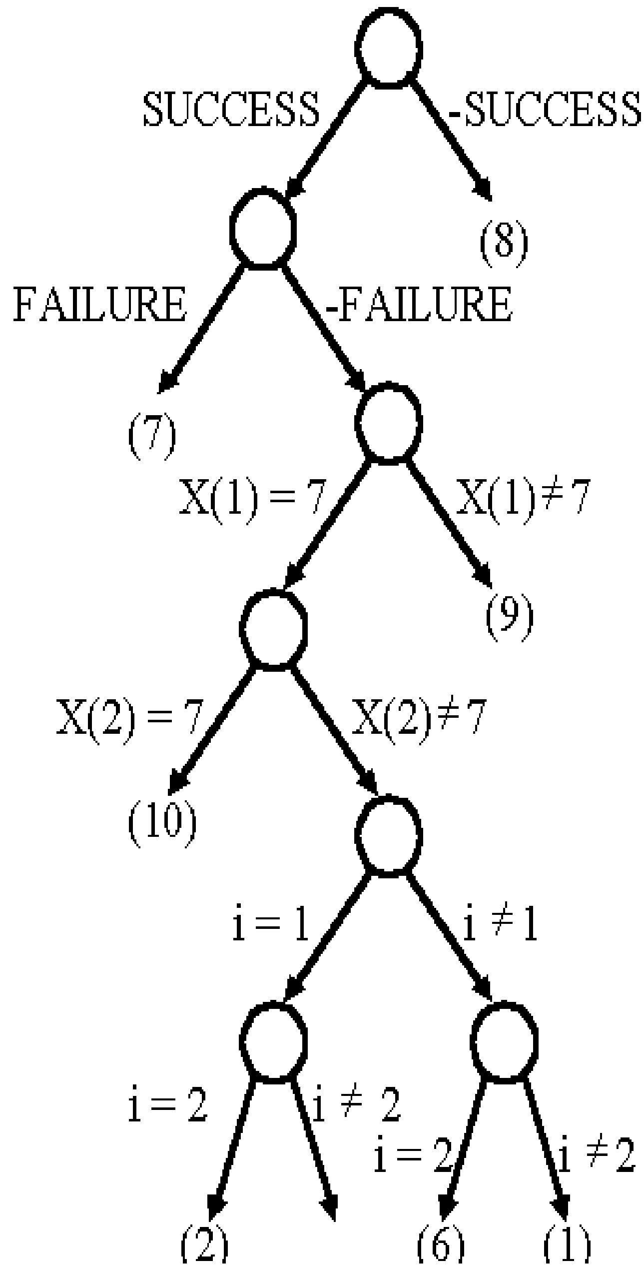
CNF (conjunctive normal form) :

$i=1$	v	$i=2$				(1)
$i\neq 1$	v	$i\neq 2$				(2)
$x(1)\neq 7$	v	$i\neq 1$	v	SUCCESS		(3)
$x(2)\neq 7$	v	$i\neq 2$	v	SUCCESS		(4)
$x(1)=7$	v	$i\neq 1$	v	FAILURE		(5)
$x(2)=7$	v	$i\neq 2$	v	FAILURE		(6)
-FAILURE	v	-SUCCESS				(7)
SUCCESS						(8)
$x(1)=7$						(9)
$x(2)\neq 7$						(10)

satisfiable at the following assignment :

$i=1$	satisfying	(1)
$i\neq 2$	satisfying	(2), (4) and (6)
SUCCESS	satisfying	(3), (4) and (8)
–FAILURE	satisfying	(7)
$x(1)=7$	satisfying	(5) and (9)
$x(2)\neq 7$	satisfying	(4) and (10)

The semantic tree :



Searching for 7, but x(1)≠7, x(2)≠7

CNF :

i=1	v	i=2					(1)
i≠1	v	i≠2					(2)
x(1)≠7	v	i≠1	v	SUCCESS			(3)
x(2)≠7	v	i≠2	v	SUCCESS			(4)
x(1)=7	v	i≠1	v	FAILURE			(5)
x(2)=7	v	i≠2	v	FAILURE			(6)

SUCCESS	(7)
-SUCCESS v -FAILURE	(8)
$x(1) \neq 7$	(9)
$x(2) \neq 7$	(10)

apply resolution principle :

(9) & (5)	$i\neq1$ v FAILURE	(11)
(10) & (6)	$i\neq2$ v FAILURE	(12)
(7) & (8)	-FAILURE	(13)
(13) & (11)	$i\neq1$	(14)
(13) & (12)	$i\neq2$	(15)
(14) & (1)	$i=2$	(11)
(15) & (16)	$\square$	(17)

We get an empty clause $\Rightarrow$ unsatisfiable

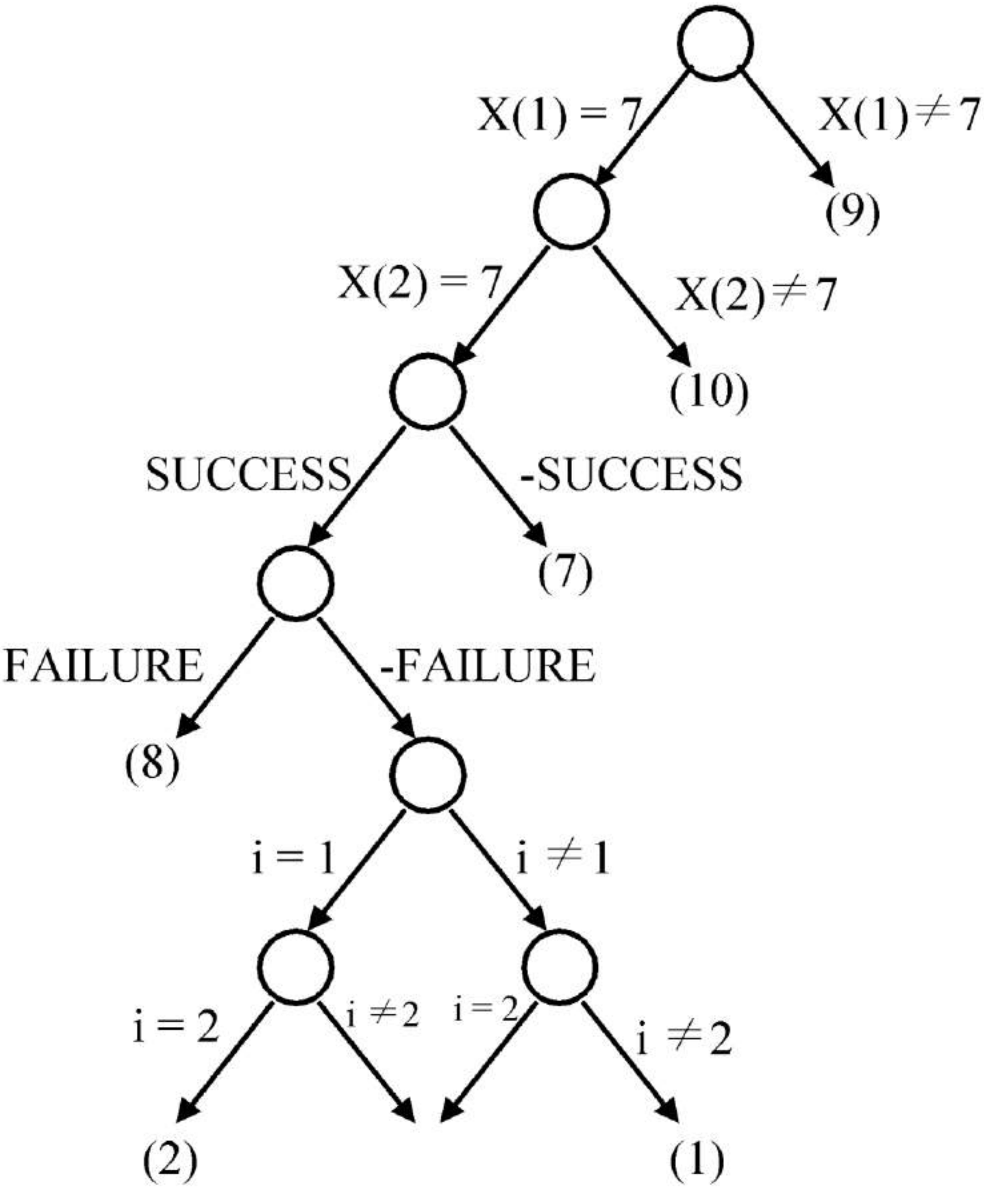
$\Rightarrow$ 7 does not exit in x(1) or x(2).

Searching for 7, where x(1)=7, x(2)=7

CNF :

$i=1$	v	$i=2$					(1)
$i\neq1$	v	$i\neq2$					(2)
$x(1)\neq7$	v	$i\neq1$		v		SUCCESS	(3)
$x(2)\neq7$	v	$i\neq2$		v		SUCCESS	(4)
$x(1)=7$	v	$i\neq1$		v		FAILURE	(5)
$x(2)=7$	v	$i\neq2$		v		FAILURE	(6)
SUCCESS							(7)
-SUCCESS v -FAILURE							(8)
$x(1)=7$							(9)
$x(2)=7$							(10)

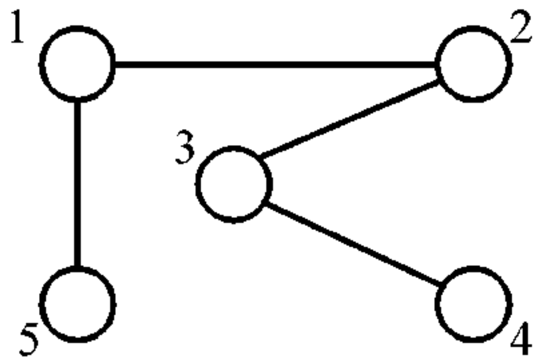
The semantic tree :



It implies that both assignments (i=1, i=2) satisfy the clauses.

● Transforming the node cover problem to SAT

<u>Def</u>: Given a graph $G=(V, E)$, S is the <u>node cover</u> if $S \subseteq V$ and for every edge $(u, v) \in E$, either $u \in S$ or $v \in S$.



node cover:
{1, 3}
{5, 2, 4}

decision problem : $\exists S \ni |S| \leq K$?

```
BEGIN
    i1 ← choice({1, 2, …, n})
    i2 ← choice({1, 2, …, n} − {i1})
    ⋮
    ik ← choice({1, 2, …, n} − {i1, i2, …, ik-1}).
    For j=1 to m do
        BEGIN
          if ej is not incident to one of v_{i_t} (1≤t≤k)
          then FAILURE
        END
     SUCCESS
```

CNF (conjunctive normal form) :

$i_1 = 1 \quad \vee \quad i_1 = 2 \ldots \quad \vee \quad i_1 = n$

$(i_1 \neq 1 \rightarrow i_1 = 2 \ \vee \ i_1 = 3 \ldots \vee \ i_1 = n)$

$i_2 = 1 \quad \vee \quad i_2 = 2 \ldots \quad \vee \quad i_2 = n$

$\vdots$

$i_k = 1 \quad \vee \quad i_k = 2 \ldots \quad \vee \quad i_k = n$

$i_1 \neq 1 \quad \vee \quad i_2 \neq 1 \qquad (i_1=1 \rightarrow i_2 \neq 1 \ \& \ \ldots \ \& \ i_k \neq 1)$

$i_1 \neq 1 \quad \vee \quad i_3 \neq 1$

$\vdots$

$i_{k-1} \neq n \quad \vee \quad i_k \neq n$

$v_{i_1} \in e_1 \ \vee \ v_{i_2} \in e_1 \ \vee \ldots \vee \ v_{i_k} \in e_1 \ \vee$ FAILURE

$(v_{i_1} \notin e_1 \ \& \ v_{i_2} \notin e_1 \ \& \cdots \& \ v_{i_k} \notin e_1 \rightarrow \text{Failure})$

$v_{i_1} \in e_2 \ \vee \ v_{i_2} \in e_2 \ \vee \ldots \vee \ v_{i_k} \in e_2 \ \vee$ FAILURE

$\vdots$

$v_{i_1} \in e_m \ \vee \ v_{i_2} \in e_m \ \vee \ldots \vee \ v_{i_k} \in e_m \ \vee$ FAILURE

SUCCESS

-SUCCESS $\vee$ -FAILURE

$v_{r_1} \in e_1$

$v_{s_1} \in e_1$

$v_{r_2} \in e_2$

$v_{s_2} \in e_2$

$\vdots$

$v_{r_m} \in e_m$

$v_{s_m} \in e_m$

☐ **Proof of NP-Completeness**

(I) prove that A is an NP problem

(II) prove that $\exists$ B $\in$ NPC, B $\propto$ A

$\Rightarrow$ A $\in$ NPC

- 3-satisfiability problem (3-SAT)
  Each clause contains exactly three literals.

(I) 3-SAT is an NP problem. (obviously)

(II) SAT $\propto$ 3-SAT

(1) one literal L_1 in a clause in SAT :
in 3-SAT :

$L_1 \vee y_1 \vee y_2$
$L_1 \vee -y_1 \vee y_2$
$L_1 \vee y_1 \vee -y_2$
$L_1 \vee -y_1 \vee -y_2$

(2) two literals L_1, L_2 in a clause in SAT :
in 3-SAT :

$L_1 \vee L_2 \vee y_1$
$L_1 \vee L_2 \vee -y_1$

(3) three literals in a clause :
remain unchanged

(4) more than 3 literals $L_1, L_2, \ldots, L_k$ in a clause :
in 3-SAT :

$L_1 \vee L_2 \vee y_1$
$L_3 \vee -y_1 \vee y_2$
$\vdots$
$L_{k-2} \vee -y_{k-4} \vee y_{k-3}$
$L_{k-1} \vee L_k \vee -y_{k-3}$

e.g.

an instance in SAT :

$x_1 \vee x_2$

$-x_3$

$x_1 \vee -x_2 \vee x_3 \vee -x_4 \vee x_5$

transform to an instance in 3-SAT :

$$
\begin{array}{lllll}
x_1 & v & x_2 & v & y_1 \\
x_1 & v & x_2 & v & -y_1 \\
-x_3 & v & y_2 & v & y_3 \\
-x_3 & v & -y_2 & v & y_3 \\
-x_3 & v & y_2 & v & -y_3 \\
-x_3 & v & -y_2 & v & -y_3 \\
x_1 & v & -x_2 & v & y_4 \\
x_3 & v & -y_4 & v & y_5 \\
-x_4 & v & x_5 & v & -y_5
\end{array}
$$

$$\text{SAT} \xrightarrow{\textbf{transform}} \text{3-SAT}$$

$$S \longrightarrow S'$$

Proof : S is satisfiable $\Leftrightarrow$ S′ is satisfiable

"$\Rightarrow$"

≤ 3 literals in S (trivial)

consider ≥ 4 literals

$S : L_1 \vee L_2 \vee \ldots \vee L_k$

$S': L_1 \vee L_2 \vee y_1$

$L_3 \vee -y_1 \vee y_2$

$L_4 \vee -y_2 \vee y_3$

$\vdots$

$L_{k-2} \vee -y_{k-4} \vee y_{k-3}$

$L_{k-1} \vee L_k \vee -y_{k-3}$

S is satisfiable $\Rightarrow$ at least $L_i = T$

Assume : $L_j = F \quad \forall\, j \neq i$
assign : $y_{i-1} = F$
$y_j = T \quad \forall\, j < i-1$
$y_j = F \quad \forall\, j > i-1$
($\because$ L_i v $-y_{i-2}$ v y_{i-1})
$\Rightarrow$ S′ is satisfiable.

"$\Leftarrow$"

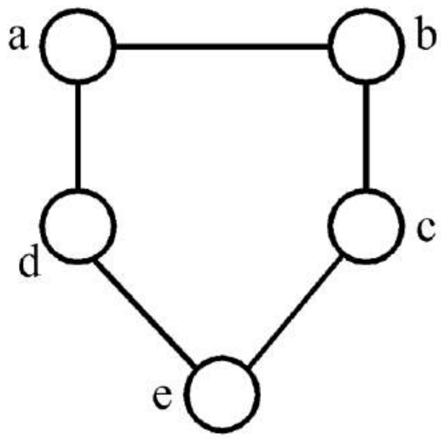
If S′ is satisfiable, then assignment satisfying S′ can not contain y_i's only
$\Rightarrow$ at least L_i must be true.
(We can also apply the resolution principle).
$\therefore$ 3-SAT is NP-complete #

- If a problem is NP-complete, its special cases may or may not be NP-complete.

- chromatic number decision problem
A coloring of a graph G=(V, E) is a function $f : V \rightarrow \{ 1, 2, 3, \ldots, k \} \ni$ if $(u, v) \in E$, then $f(u) \neq f(v)$.

The chromatic number decision problem (CN) :
determine if G has a coloring for k.

e.g.



3-colorable

$f(a)=1, \quad f(b)=2, \quad f(c)=1$

$f(d)=2, \quad f(e)=3$

<Theorem> Satisfiability with at most 3 literals per clause (SATY) $\propto$ CN.

Proof :

instance of SATY :

variable : $x_1, x_2, \ldots, x_n$, $n \ge 4$

clause : $c_1, c_2, \ldots, c_r$

instance of CN :

$G=(V, E)$

$$V=\{ x_1, x_2, \ldots, x_n \}\cup\{ -x_1, -x_2, \ldots, -x_n \} \cup \underbrace{\{ y_1, y_2, \ldots, y_n \}}_{\text{newly added}}\cup\{ c_1, c_2, \ldots, c_r \}$$

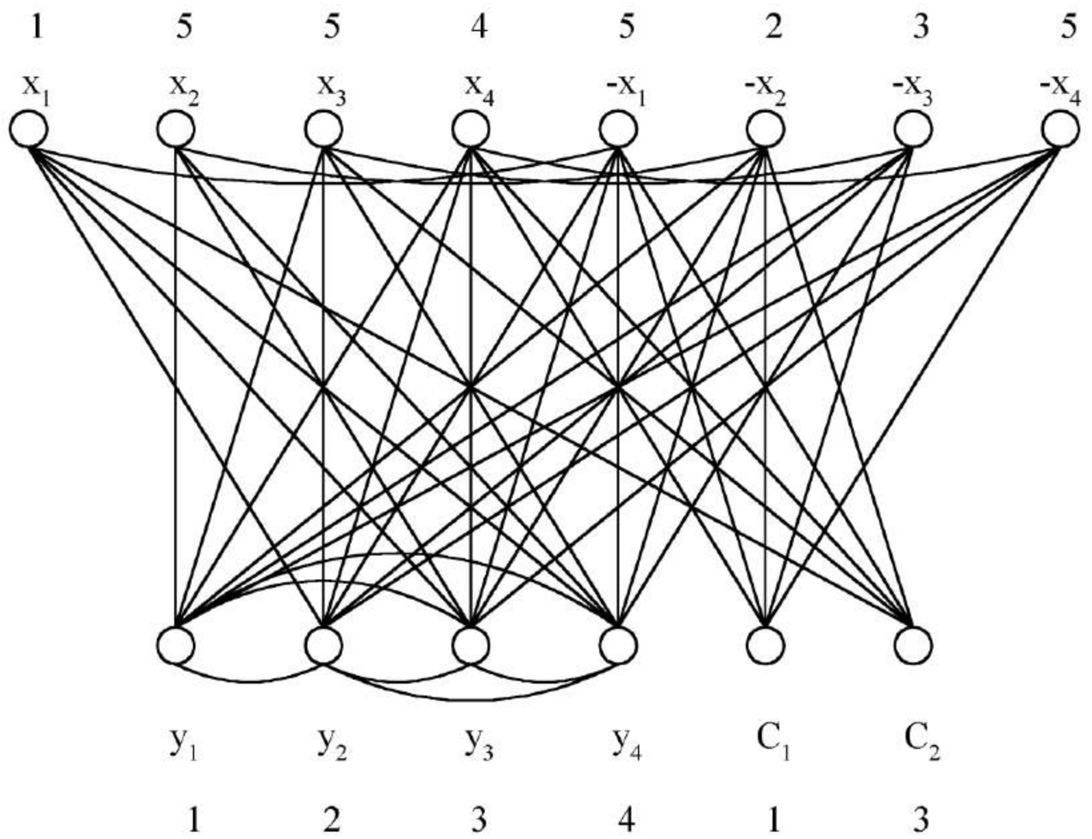
$$E=\{ (x_i, -x_i) \mid 1\le i \le n \}\cup\{ (y_i, y_j) \mid i \neq j \} \cup\{ (y_i, x_j) \mid i \neq j \}\cup\{ (y_i, -x_j) \mid i \neq j \} \cup\{ (x_i, c_j) \mid x_i \notin c_j \}\cup\{ (-x_i, c_j) \mid -x_i \notin c_j \}$$

e.g.

$x_1 \quad v \quad x_2 \quad v \quad x_3 \qquad (1)$

$-x_3 \quad v \quad -x_4 \quad v \quad x_2 \qquad (2)$

$\Downarrow$



Satisfiable $\Leftrightarrow$ n+1 colorable

"$\Rightarrow$"

(1) $f(y_i) = i$

(2) if $x_i = T$, then $f(x_i) = i$, $f(-x_i) = n+1$
    else $f(x_i) = n+1$, $f(-x_i) = i$

(3) if x_i in c_j and $x_i = T$, then $f(c_j) = f(x_i)$
    if $-x_i$ in c_j and $-x_i = T$, then $f(c_j) = f(-x_i)$
    (at least one such x_i)

"$\Leftarrow$"

(1) y_i must be assigned with color i.

(2) $f(x_i) \neq f(-x_i)$
    either $f(x_i) = i$ and $f(-x_i) = n+1$
    or $f(x_i) = n+1$ and $f(-x_i) = i$

(3) at most 3 literals in c_j and $n \geq 4$
    $\Rightarrow$ at least one x_i, ∍ x_i and $-x_i$ are not in c_j

$\Rightarrow f(c_j) \neq n+1$

(4) if $f(c_j) = i = f(x_i)$, assign x_i to T

if $f(c_j) = i = f(-x_i)$, assign $-x_i$ to T

(5) if $f(c_j) = i = f(x_i) \Rightarrow (c_j, x_i) \notin E$

$\Rightarrow x_i$ in $c_j \Rightarrow c_j$ is true

if $f(c_j) = i = f(-x_i) \Rightarrow$ similarly #

- set cover decision problem

$F = \{S_i\} = \{ S_1, S_2, \ldots, S_k \}$

$\bigcup_{S_i \in F} S_i = \{ u_1, u_2, \ldots, u_n \}$

T is a <u>set cover</u> of F if $T \subseteq F$ and

$\bigcup_{S_i \in T} S_i = \bigcup_{S_i \in F} S_i$

The set cover decision problem is to determine if F has a cover T containing no more than c sets.

e.g.

$F = \{(a_1, a_3), (a_2, a_4), (a_2, a_3), (a_4)\}$

$\quad\quad s_1 \quad\quad s_2 \quad\quad s_3 \quad\quad s_4$

$T = \{ s_1, s_3, s_4 \}$ set cover

$T = \{ s_1, s_2 \}$ set cover, exact cover

- exact cover problem

notations same as those in set cover. to determine if F has an <u>exact cover</u> T, which is a cover of F and the sets in T are pairwise disjoint.

<Theorem> CN $\propto$ exact cover

- sum of subsets problem

  a set of positive numbers $A = \{ a_1, a_2, \ldots, a_n \}$

  a constant C

  determine if $\exists A' \subseteq A \ni \sum_{a_i \in A'} a_i = C$

e.g. $A = \{ 7, 5, 19, 1, 12, 8, 14 \}$

(i) $C = 21$, $A' = \{ 7, 14 \}$

(ii) $C = 11$, no solution

<Theorem> Exact cover $\propto$ sum of subsets

Proof :

instance of exact cover :

$F = \{ S_1, S_2, \ldots, S_k \}$

$$\bigcup_{S_i \in F} S_i = \{u_1, u_2, \ldots, u_n,\}$$

instance of sum of subsets :

$A = \{ a_1, a_2, \ldots, a_k \}$ where

$a_j = \sum_{1 \le i \le n} e_{ji}(k+1)^i$ where $e_{ji} = 1$ if $u_i \in S_j$

$e_{ji} = 0$ if otherwise.

$C = \sum_{1 \le i \le n} (k+1)^i = (k+1)((k+1)^n - 1) / k$ #

---

why k+1?

- Partition problem
  a set of positive numbers $A = \{ a_1, a_2, \ldots, a_n \}$
  determine if $\exists$ a partition P, $\ni \sum_{i \in p} a_i = \sum_{i \notin p} a_i$

e.g. $A = \{1, 3, 8, 4, 10\}$
partition : $\{1, 8, 4\}$ and $\{3, 10\}$

<Theorem> sum of subsets $\propto$ partition

proof :

instance of sum of subsets :

$A = \{ a_1, a_2, \ldots, a_n \}$

C

instance of partition :

$B = \{ b_1, b_2, \ldots, b_{n+2} \}$ where $b_i = a_i$, $1 \le i \le n$

$b_{n+1} = C+1$

$b_{n+2} = (\sum_{1 \le i \le n} a_i)+1-C$

$C = \sum_{a_i \in S} a_i \quad \Leftrightarrow \quad (\sum_{a_i \in S} a_i)+b_{n+2} = (\sum_{a_i \notin S} a_i)+b_{n+1}$

$\Leftrightarrow$ partition : $\{ b_i \mid a_i \in S \} \cup \{b_{n+2}\}$
and $\{ b_i \mid a_i \notin S \} \cup \{b_{n+1}\}$ #

why $b_{n+1} = C+1$? why not $b_{n+1} = C$?

to avoid b_{n+1} and b_{n+2} to be partitioned into the same subset.

- bin packing problem
  n items, each of size c_i , $c_i > 0$
  bin capacity : c
  determine if we can assign the items into k bins, $\ni \sum_{i \in bin_j} c_i \le c$, $1 \le j \le k$.

<Theorem> partition ∝ bin packing.

- VLSI discrete layout problem

  n rectangles, each with height h_i (integer)

  width w_i

  an area A

  determine if there is a placement of the n rectangles within the area A according to the rules :

(i) boundaries of rectangles parallel to x axis or y axis.

(ii) corners of rectangles lie on integer points.

(iii) no two rectangles overlap.

(iv) two rectangles are separated by at least a unit distance.

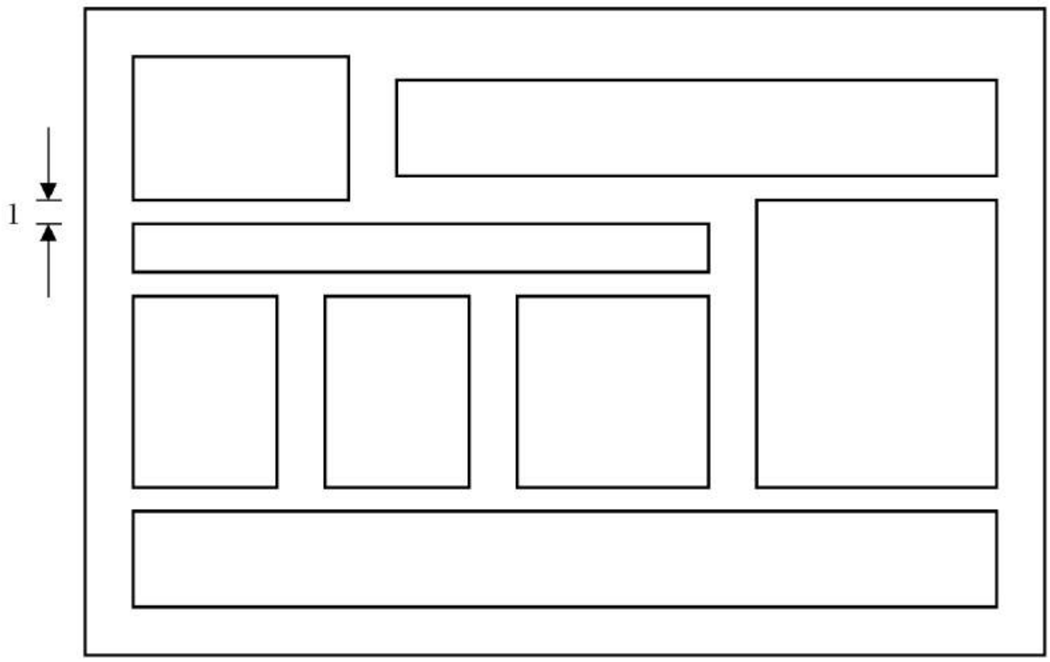


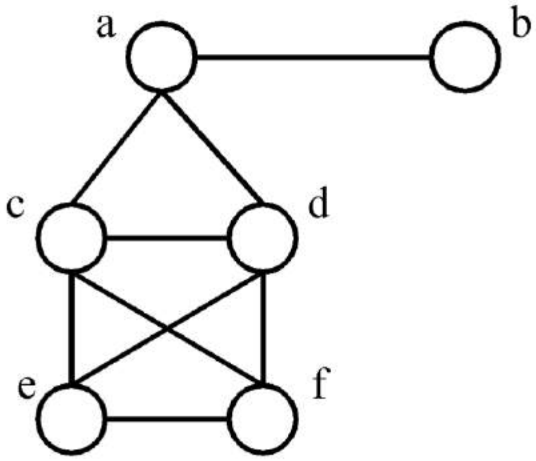
Fig. 3-14 A Successful Placement

<Theorem> bin packing $\propto$ VLSI discrete layout.

- max clique problem

  A maximal complete subgraph of a graph G=(V, E) is a clique. The max (maximum) clique problem is to determine the size of a largest clique in G.

e.g.



maximal cliques :
{a, b}, {a, c, d}
{c, d, e, f}
maximum clique :
(largest)
{c, d, e, f}

<Theorem> SAT $\propto$ clique decision problem.

- node cover decision problem

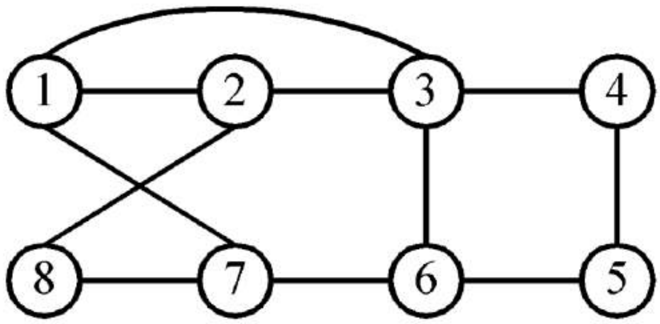
  A set $S \subseteq V$ is a node cover for a graph G = (V, E) iff all edges in E are incident to at least one vertex in S. $\exists$ S, $\ni$ $|S| \leq K$?

<Theorem> clique decision problem $\propto$ node cover decision problem.

- Hamiltonian cycle problem

  A Hamiltonian cycle is a round trip path along n edges of G which visits every vertex once and returns to its starting vertex.

e.g.



Hamiltonian cycle : 1, 2, 8, 7, 6, 5, 4, 3, 1.

<Theorem> SAT $\propto$ directed Hamiltonian cycle (in a directed graph)

- traveling salesperson problem

A tour of a directed graph G=(V, E) is a directed cycle that includes every vertex in V. The problem is to find a tour of minimum cost.

<Theorem> Directed Hamiltonian cycle $\propto$ traveling salesperson decision problem.

- 0/1 knapsack problem

n objects, each with a weight $w_i > 0$
a profit $p_i > 0$

capacity of knapsack : M

maximize $\sum_{1 \le i \le n} p_i x_i$

subject to $\sum_{1 \le i \le n} w_i x_i \le M$

$x_i = 0$ or 1, $1 \le i \le n$

decision version :

given K, $\exists \sum_{1 \le i \le n} p_i x_i \ge K$?

knapsack problem : $0 \le x_i \le 1$, $1 \le i \le n$.

<Theorem> partition $\propto$ 0/1 knapsack decision problem.