Chapter 8

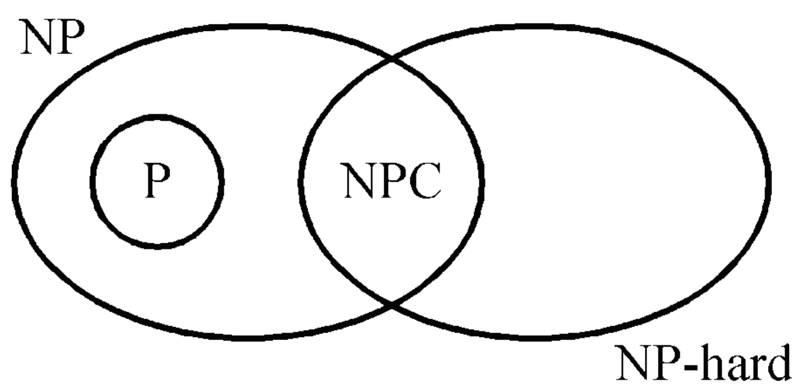
The Theory of NP-Completeness

Outlines

- **An Informal discussion of the Theory of NP-completeness**
- **The Decision Problem**
- **The Satisfiability Problem**
- **The NP problems**
- **NP-complete problems**
- **Example of Proving NP-completeness**
- **2-satisfiability problem**

An Informal discussion of the Theory of NP-completeness

- Cook's Theorem
- Turing Award
- NP-completeness
- Not all NP problems are difficult, e.g. searching problem, MST problem, …



- **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, like TSP, bin packing problem, ….

NP-complete problem

- Up to now, *none of the NP-complete problems can be solved by any polynomial time algorithm, in worst cases*.
- Up to now, the best algorithm to solve any NP-complete problem has exponential time complexity in the worst case.
- It is possible that an NP-complete problem can already be solved by an algorithm with polynomial average case time complexity.

Some concepts of NPC

- Definition of reduction: 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 theory of NP-completeness always considers the worst case.
- The lower bound of any NPC problem seems to be in the order of an exponential function.
- Not all NP problems are difficult. (e.g. the MST problem is an 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)

8.2 Decision problems

- Decision problem: the solution is simply “Yes” or “No”.
- <u>Optimization</u> problems are more difficult.
- e.g. the traveling salesperson problem (TSP)
- 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 :

- TSP optimization problem is more difficult than TSP decision problem.
- Solving TSP optimization problem by decision algorithm :
  - Give c_1 and test (decision algorithm)
    Give c_2 and test (decision algorithm)
    ⋮
    Give c_n and test (decision algorithm)
  - We can easily find the smallest c_i
- If we can solve the TSP **optimization** problem, then we can solve the TSP **decision** problem but not vice versa.

0/1 Knapsack decision problem

- Given $M>0$, $R>0$, $W_i>0$ and $P_i>0$, $i=1, 2, \ldots, n$, determine where there exist $x_i=1$ or 0 such that

$$\sum_{i=1}^{n} P_i x_i \geq R \text{ and } \quad \sum_{i=1}^{n} W_i x_i \leq M$$

- In general, optimization problems are more difficult to solve than their corresponding decision problems.
- If the decision problem cannot be solved by polynomial algorithms, the optimization cannot be solved by polynomial algorithms.
- In Np-complete discussion, only decision problem is considered.

8.3 The satisfiability problem

- The satisfiability problem (first NP-complete 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 an assignment makes a formula true, we shall say that this assignment satisfies the formula; otherwise, it falsifies the formula.

- 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 :

$$
\begin{aligned}
& x_1 \vee x_2 \\
\& \; & x_1 \vee -x_2 \\
\& \; & -x_1 \vee x_2 \\
\& \; & -x_1 \vee -x_2
\end{aligned}
$$

$$
\begin{aligned}
& x_1 \\
\& \; & -x_1
\end{aligned}
$$

The satisfiability problem

- Definition of 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 (**CNF**)

$$C_1 \& C_2 \& \ldots \& C_m$$

- Every Boolean formula can be transformed into the CNF.
- A formula G is a logical consequence of a formula F if and only if whenever F is true, G is true

Resolution Principle

c_1: $\boxed{L_1} \vee L_2 \ldots \vee L_j$

and c_2: $\boxed{\neg L_1} \vee L_2' \ldots \vee L_k'$.

we can deduce a clause

$$L_2 \vee \ldots \vee L_j \vee L_2' \vee \ldots \vee L_k'$$

as a logical consequence of c_1 & c_2, if the clause

$$L_2 \vee \ldots \vee L_j \vee L_2' \vee \ldots \vee L_k'$$

does not contain any pair of literals which are complementary to each other.

The resolution principle

- Resolution principle

  C_1 : $-x_1 \vee -x_2 \vee x_3$

  C_2 : $x_1 \vee x_4$

  $\Rightarrow C_3$: $-x_2 \vee x_3 \vee x_4$ (C_1 and C_2)

- Given a set of clauses, we may repeatedly apply the resolution principle to deduce new clause. If no new clauses can be deduced, then it is 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, then it is unsatisfiable.

$$
\begin{array}{lll}
 & -x_1 \lor -x_2 \lor x_3 & (1) \\
 & x_1 \lor -x_2 & (2) \\
 & x_2 & (3) \\
 & -x_3 & (4) \\
 & \Downarrow \text{deduce} & \\
(1)\ \&\ (2) & -x_2 \lor x_3 & (5) \\
(4)\ \&\ (5) & -x_2 & (6) \\
(6)\ \&\ (3) & \square & (7)
\end{array}
$$

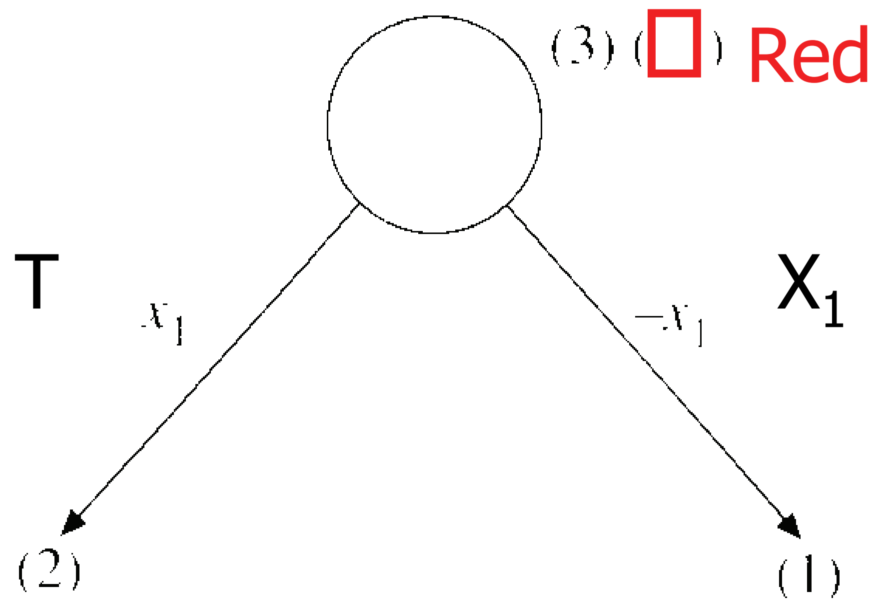
Empty clause →unsatisfiable.

- If an empty clause is deduced, then it is unsatisfiable.
- If no new clauses can be deduced when the process is terminate, the set of clauses is **satisficable**.

Semantic tree

Example:
x_1 (1)
$-x_1$ (2)

FIGURE 8–3 A semantic tree.



Reduce to empty

X_1 is assigned T

X_1 is assigned F

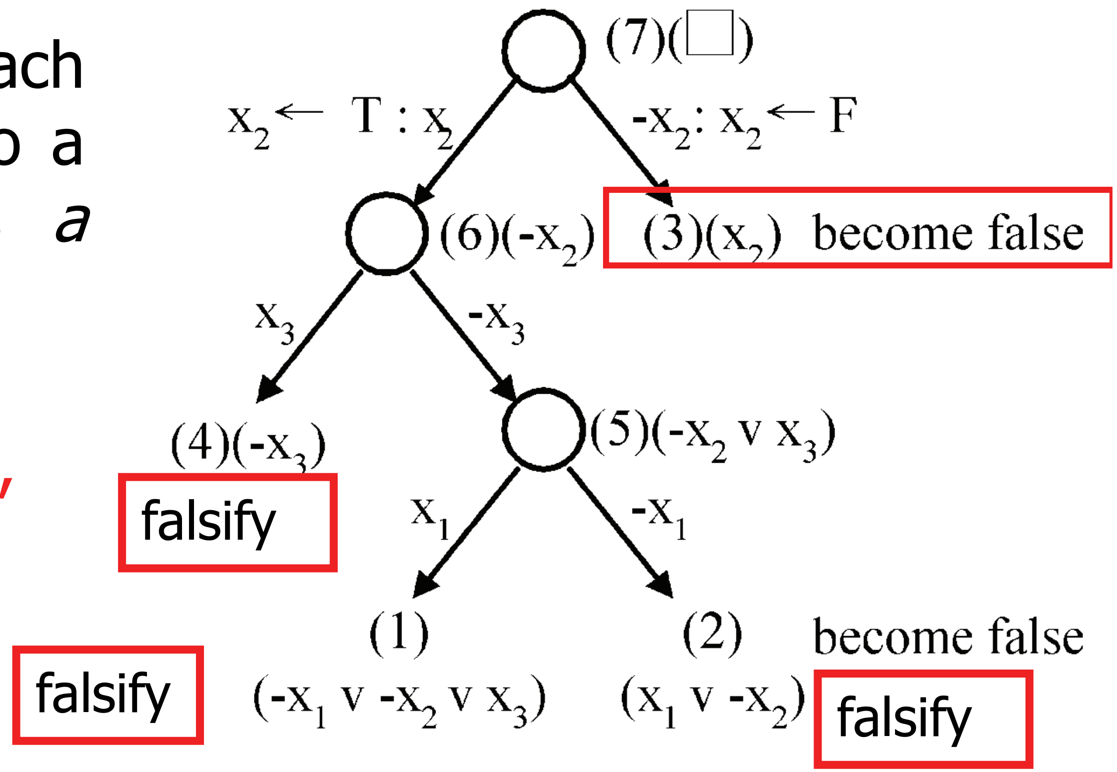
Falsifies clause (2)

Falsifies clause (1)

Semantic tree

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

- 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, then it is unsatisfiable.



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

(3) Terminate by clause (3)

Rule of constructing the semantic tree

- From each internal node of the semantic tree, there are two branches branching out of it. One of them is labeled with x_i, and the other one is labeled with $-x_i$. where x_i is a variable occurring in the set of clauses.
- A node is terminated as soon as the assignment corresponding to the literals occurring in the path from the root of the tree to this node falsifies a clause *(j)* in the set. Mark this node as a terminal node and attach clause *(j)* to this terminal node.
- No path in the semantic tree may contain complementary pair so that each assignment is consistent.

Determine unsatisfiable or unsatisfiable?

- It is obvious that each semantic tree is finite.
- *If each terminal is attached with clause, then no assignment satisfying all clauses exists. This means that this set clauses is unsatisfiable.*
- *Otherwise, there exists at least one assignment, satisfying all clauses and this set of clauses is satisfiable.*

Consider the following set of clauses:

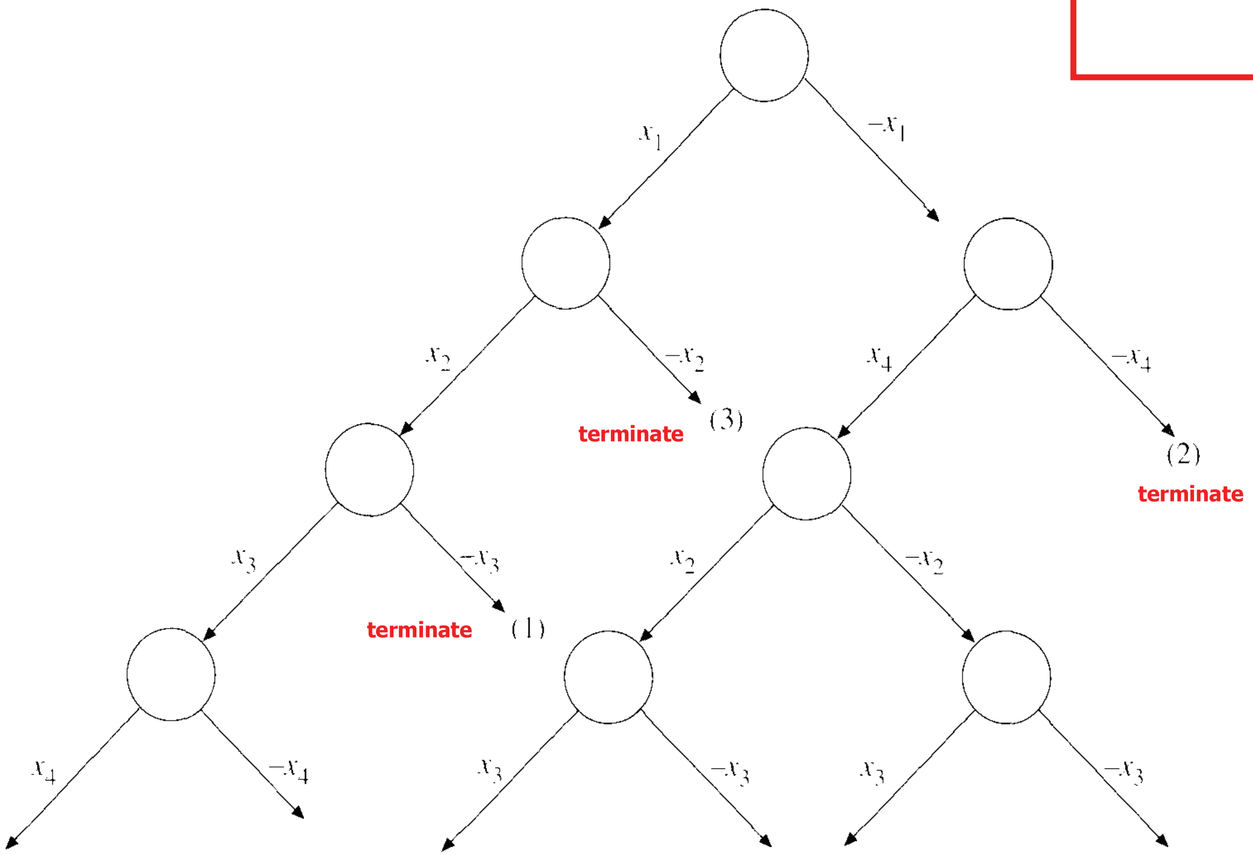
(1) $-x_1 \vee -x_2 \vee x_3$
(2) $x_1 \vee x_4$
(3) $x_2 \vee -x_1$.

We may construct a semantic tree as shown in Figure 8–5 and

(x_1, x_2, x_3, x_4),
$(x_1, x_2, x_3, -x_4)$,
$(-x_1, -x_2, x_3, x_4)$,
$(-x_1, -x_2, -x_3, x_4)$,
$(-x_1, x_2, x_3, x_4)$,
$(-x_1, x_2, -x_3, x_4)$.

Can satisfy the formula

FIGURE 8–5 A semantic tree.



Resolution principle

- If a set of clauses is unsatisfiable, then every semantic tree corresponds to the deduction of an empty clause using the resolution principle.
- This deduction is extracted from the semantic tree as follows:
  - Consider an internal node whose descendants are terminal nodes. Let the clauses attached to these two terminal nodes be c_i and c_j respectively.
  - Apply the resolution principle to these two clauses and attach the resolvent to this parent node.
  - Delete the descendant nodes altogether. The original internal node now becomes a terminal node.
  - Repeat the above step until the tree becomes empty and an empty clause is deduced.

FIGURE 8–6 A semantic tree.

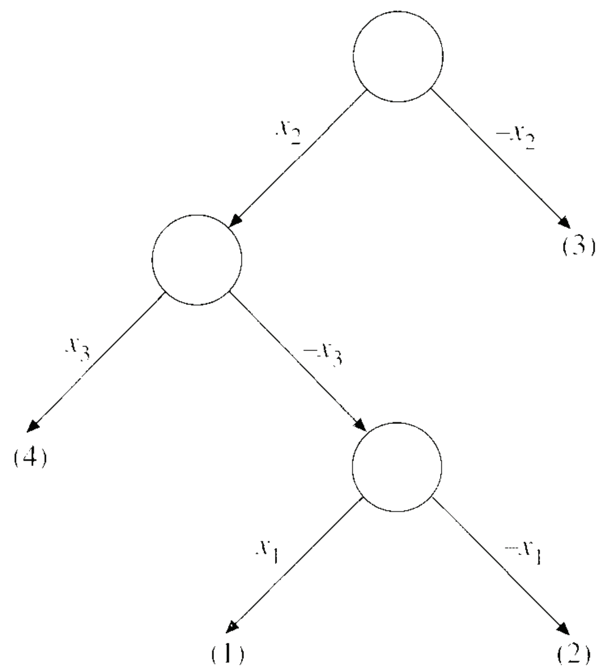
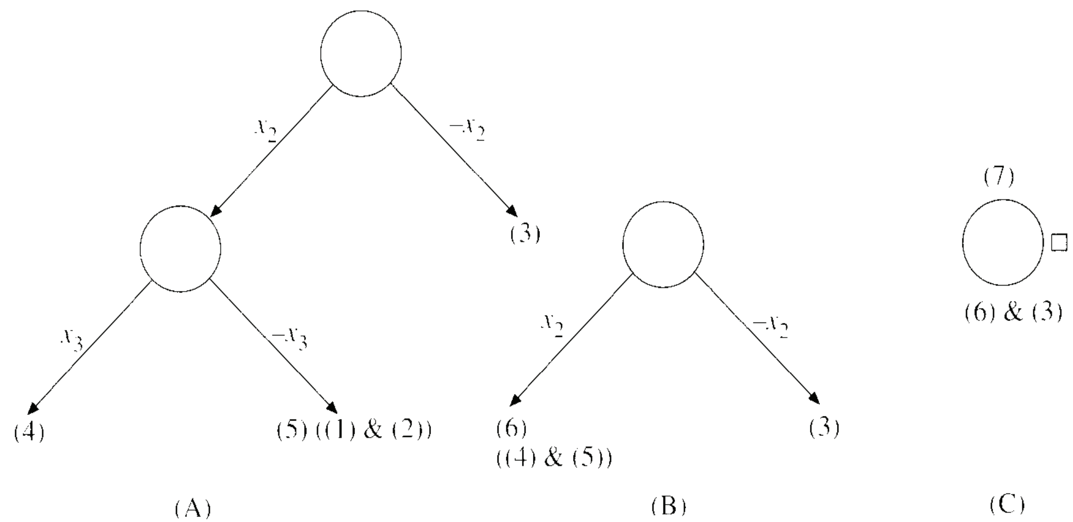


FIGURE 8–7 The collapsing of the semantic tree in Figure 8–6.



The deduction is as follows:

(1) & (2) $-x_2 \vee x_3$ (5)

(5) & (4) $-x_2$ (6)

(6) & (3) $\square$. (7)

(1) $-x_1 \vee -x_2 \vee x_3$

(2) $x_1 \vee x_3$

(3) x_2

(4) $-x_3$.

SAT problem is exponential time

- Even as we use the deduction approach, we are actually finding assignments satisfying all clauses. **If there are *n* variables, then there are 2^n possible assignments. Up to now, for the best available algorithm, in worst cases, we must examine an exponential number of possible assignments before we can make any conclusion.**
- Is there any possibility that the satisfiability problem can be solved in polynomial time?
- The theory of NP-completeness does not rule out this possibility. However, it does make the following claim.
- *If the **satisfiability problem** can be solved in polynomial number of steps, then **all NP problems** can be solved in polynomial number of steps.*

8.4 NP problems

- A nondeterminstic algorithm consists of

  phase 1: guessing

  phase 2: checking.

Furthermore. it is assumed that a non-deterministic algorithm always makes a correct guessing.

- 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)

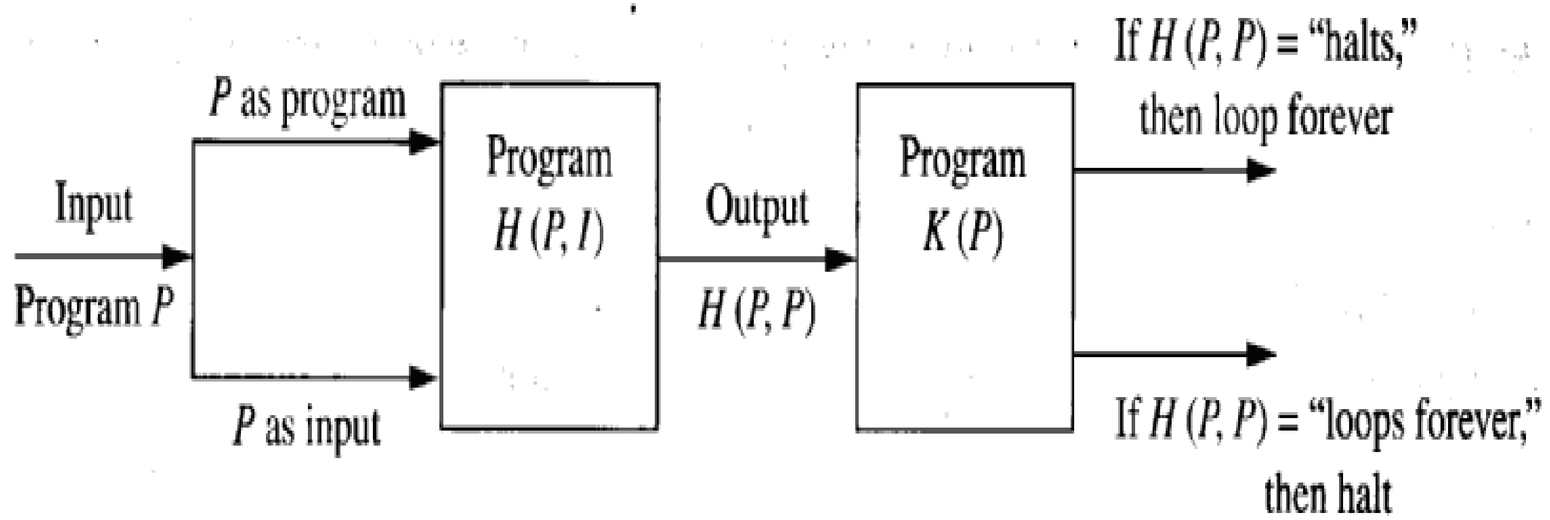
Decision problems

- Decision version of sorting:

  Given $a_1, a_2, \ldots, a_n$ and c, is there a permutation of a_i's ($a_1', a_2', \ldots, a_n'$) such that $|a_2'-a_1'| + |a_3'-a_2'| + \ldots + |a_n'-a_{n-1}'| < c$?
- 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
  - E.g.First-order predicate calculus satisfiability problem.
  - Upper bound never exist for undiciable problems.

Halting problem

- The desired function is *Halts*(P,I) = the truth value of the statement 'Program P, given input I, eventually halts'.
- Implies general impossibility of predictive analysis of arbitrary computer programs.



Nondeterministic operations and functions

[Horowitz 1998]

- Choice(S) : arbitrarily chooses one of the elements in set S
- Failure : an unsuccessful completion
- Success : a successful completion
- Nonderministic searching algorithm:

```
j ← choice(1 : n)   /* guessing */
if A(j) = x then success  /* checking */
                 else failure
```

nondeterministic algorithm

- A nondeterministic algorithm terminates unsuccessfully iff there exist no a 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 unbounded parallelism in computation.

Nondeterministic sorting

```
B ← 0
/* guessing */
for i = 1 to n do
    j ← choice(1 : n)
    if B[j] ≠ 0 then failure
    B[j] = A[i]
/* checking */
for i = 1 to n-1 do
    if B[i] > B[i+1] then failure
    success
```

Nondeterministic SAT

/* guessing */

for i = 1 to n do

  $x_i \leftarrow$ choice(true, false)

/* checking */

if $E(x_1, x_2, \ldots, x_n)$ is true then success

else failure

8.5 Cook's theorem

- NP = P iff the satisfiability problem is a P problem.
- SAT is NP-complete.
- It is the first NP-complete problem.
- Every NP problem reduces to SAT.

Cook's Theorem

- Suppose that we have an NP problem A which is quite difficult to solve. Instead of solving this problem directly, we shall create another problem A' and by solving that problem A'. We shall obtain the solution of A.
- Since **problem A** is an NP problem, there must exist an NP **algorithm B** which solves this problem. An NP algorithm is a non-deterministic polynomial algorithm. It is physically impossible and therefore we cannot use it. However as we shall see, we can still use B conceptually in the following steps.
- We shall construct a **Boolean formula C** corresponding to **B** such that <u>C is satisfiable if and only if the non-deterministic algorithm B terminates successfully and returns an answer "yes". If C is unsatisfiable, then algorithm B would terminate unsuccessfully and return the answer "no".</u>

Transforming searching to SAT

- Does there exist a number in { x(1), x(2), …, x(n) }, which is equal to 7?
- Assume n = 2. x(1)=7, x(2) ≠7

  nondeterministic algorithm:

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

Boolean formula corresponding the ND algorithm

$i=1 \vee i=2$

& $i=1 \rightarrow i \neq 2$

& $i=2 \rightarrow i \neq 1$

& $x(1)=7$ & $i=1 \rightarrow$ SUCCESS

& $x(2)=7$ & $i=2 \rightarrow$ SUCCESS

& $x(1) \neq 7$ & $i=1 \rightarrow$ FAILURE

& $x(2) \neq 7$ & $i=2 \rightarrow$ FAILURE

& FAILURE $\rightarrow$ -SUCCESS

& SUCCESS (Guarantees a successful termination)

& $x(1)=7$ (Input Data)

& $x(2) \neq 7$

Boolean formula -> CNF

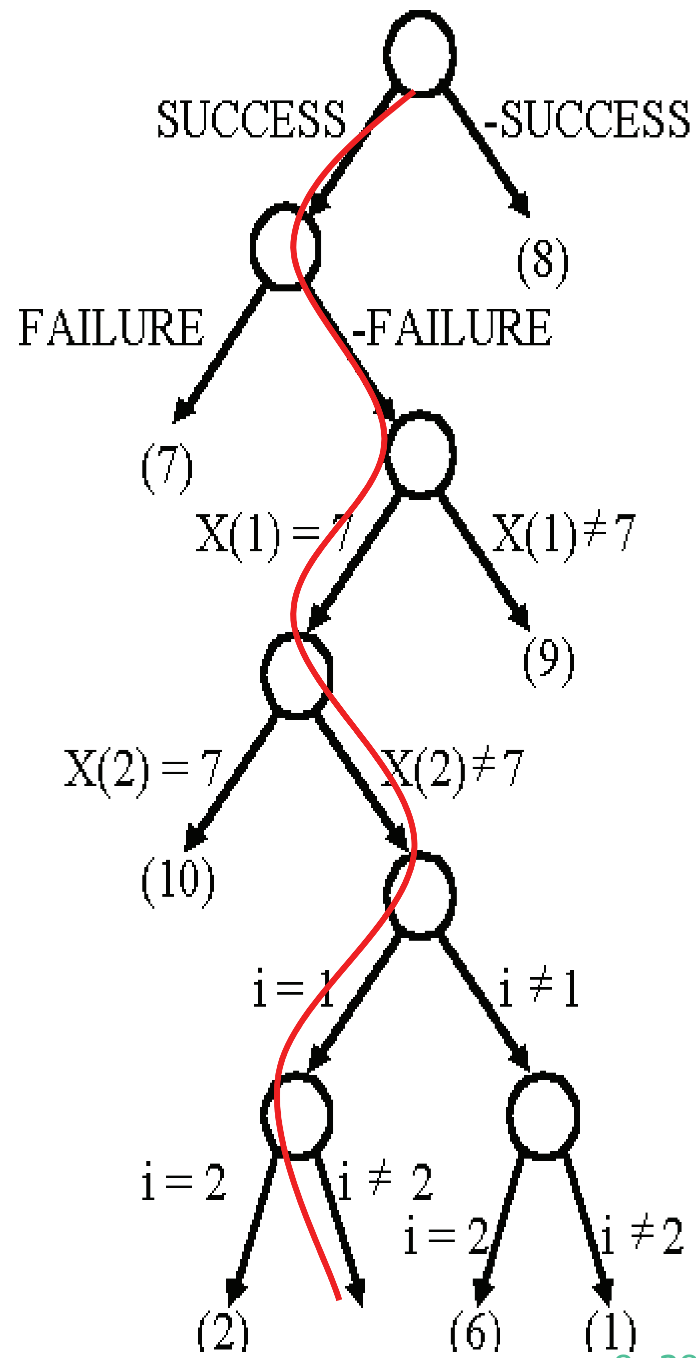
- **CNF (conjunctive normal form)** :

$i=1 \vee i=2$ (1)
$i\neq 1 \vee i\neq 2$ (2)
$x(1)\neq 7 \vee i\neq 1 \vee SUCCESS$ (3)
$x(2)\neq 7 \vee i\neq 2 \vee SUCCESS$ (4)
$x(1)=7 \vee i\neq 1 \vee FAILURE$ (5)
$x(2)=7 \vee i\neq 2 \vee FAILURE$ (6)
$-FAILURE \vee -SUCCESS$ (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 $x(1)\neq 7$, $x(2)\neq 7$

- CNF (conjunctive normal form) :

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

- Apply resolution principle :

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

We get an empty clause $\Rightarrow$ unsatisfiable

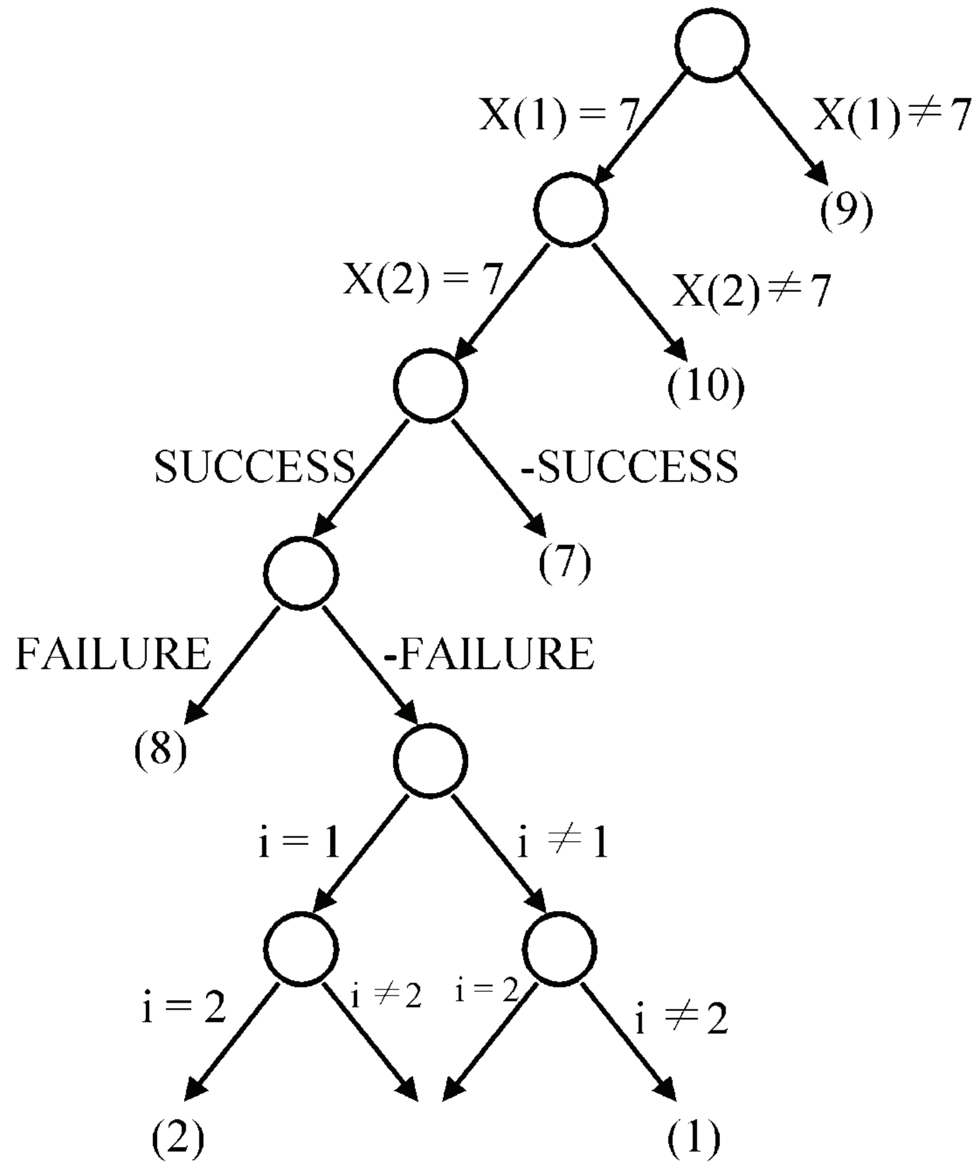
$\Rightarrow$ 7 does not exit in x(1) or x(2).

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

- CNF:

i=1	v	i=2					(1)
i≠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)=7							(9)
x(2)=7							(10)

The semantic tree



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

SAT problem

Let us consider the following set of clauses:

$$x_1 \tag{1}$$

$$-x_2. \tag{2}$$

We shall try to determine whether the above set of clauses is satisfiable or not. A non-deterministic algorithm to solve this problem is as follows:

Do $\quad i = 1, 2$

$\qquad x_i = choice(T, F)$

If x_1 and x_2 satisfy clauses 1 and 2, then *SUCCESS*, else *FAILURE*.

We shall show how the algorithm can be transformed into a Boolean formula. First of all, we know that in order for the non-deterministic algorithm to terminate with *SUCCESS*, we must have clauses 1 and 2 being true. Thus,

$$
\begin{array}{lcll}
-SUCCESS & \vee & c_1 = T & (1)\\
-SUCCESS & \vee & c_2 = T & (2)\\
-c_1 = T & \vee & x_1 = T & (3)\\
-c_2 = T & \vee & x_2 = F & (4)\\
x_1 = T & \vee & x_1 = F & (5)\\
x_2 = T & \vee & x_2 = F & (6)\\
x_1 \neq T & \vee & x_1 \neq F & (7)\\
x_2 \neq T & \vee & x_2 \neq F & (8)\\
SUCCESS. & & & (9)
\end{array}
$$

$$\left.\right\} (SUCCESS \rightarrow c_1 = T \;\&\; c_2 = T)$$

$$(c_1 = T \rightarrow x_1 = T)$$

$$(c_2 = T \rightarrow x_2 = F)$$

It is easy to see that the following assignment satisfies all the clauses.

$c_1 = T$	satisfying	(1)
$c_2 = T$	satisfying	(2)
$x_1 = T$	satisfying	(3) and (5)
$x_2 = F$	satisfying	(4) and (6)
$x_1 \neq F$	satisfying	(7)
$x_2 \neq T$	satisfying	(8)
$SUCCESS$	satisfying	(9).

X_1
$-X_1$

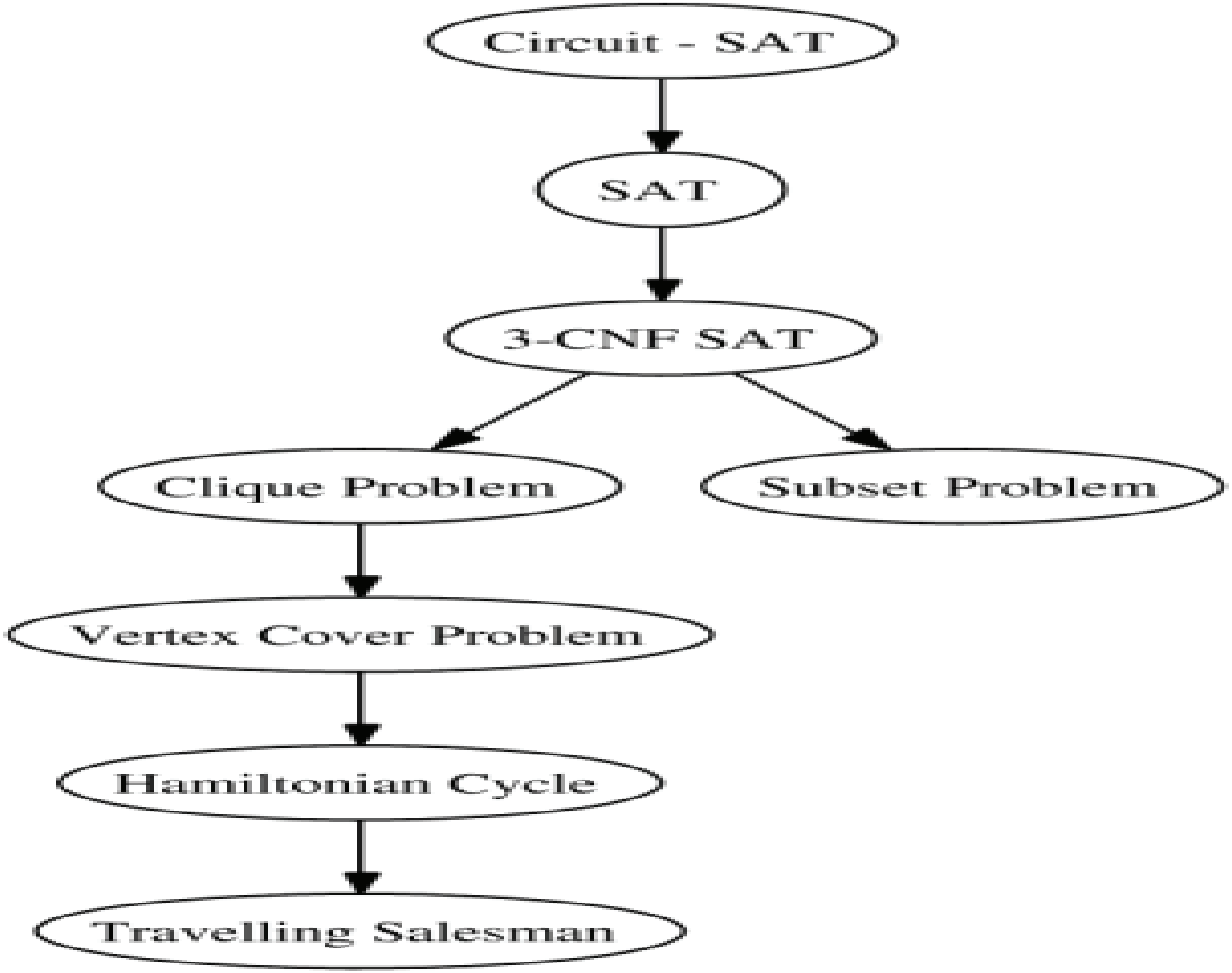
SAT

For the above set of clauses, we may construct the following Boolean formula:

$$-SUCCESS \vee c_1 = T \quad (1)$$
$$-SUCCESS \vee c_2 = T \quad (2)$$
$$-c_1 = T \vee x_1 = T \quad (3)$$
$$-c_2 = T \vee x_1 = F \quad (4)$$
$$x_1 = T \vee x_1 = F \quad (5)$$
$$x_1 \neq T \vee x_1 \neq F \quad (6)$$
$$SUCCESS. \quad (7)$$

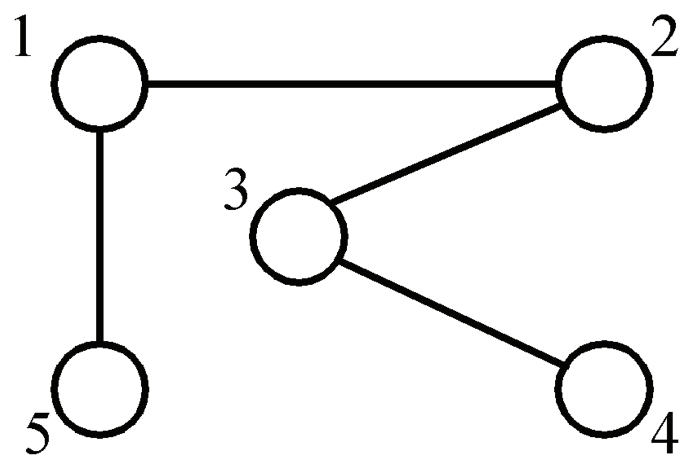
That the above set of clauses is unsatisfiable can be proved by using the resolution principle:

$$(1)\ \&\ (7) \quad c_1 = T \quad (8)$$
$$(2)\ \&\ (7) \quad c_2 = T \quad (9)$$
$$(8)\ \&\ (3) \quad x_1 = T \quad (10)$$
$$(9)\ \&\ (4) \quad x_1 = F \quad (11)$$
$$(10)\ \&\ (6) \quad x_1 \neq F \quad (12)$$
$$(11)\ \&\ (12) \quad \boxed{}. \quad (13)$$


Circuit - SAT
SAT
3-CNF SAT
Clique Problem
Subset Problem
Vertex Cover Problem
Hamiltonian Cycle
Travelling Salesman

The node cover problem

- **Def**: Given a graph G=(V, E), S is the node cover 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 : $\exists S \ni |S| \leq K$?

Transforming the node cover problem to SAT

BEGIN

$i_1 \leftarrow$ choice($\{1, 2, \ldots, n\}$)

$i_2 \leftarrow$ choice($\{1, 2, \ldots, n\} - \{i_1\}$)

$\vdots$

$i_k \leftarrow$ choice($\{1, 2, \ldots, n\} - \{i_1, i_2, \ldots, i_{k-1}\}$).

For $j=1$ to m do

BEGIN

if e_j is not incident to one of v_{i_t} $(1 \le t \le k)$

then FAILURE

END

SUCCESS

CNF:

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

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

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

$\vdots$

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

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

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

$\vdots$

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

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

(To be continued)

-SUCCESS v -FAILURE

$\nu_{r_1} \in e_1$

$\nu_{s_1} \in e_1$

$\nu_{r_2} \in e_2$

$\nu_{s_2} \in e_2$

$\vdots$

$\nu_{r_m} \in e_m$

$\nu_{s_m} \in e_m$

Note about Cook's Theorem

- It is important to note that Cook's theorem is valid under one constraint: It takes polynomial number of steps to transform an NP problem into a corresponding Boolean formula.
- If it takes exponential number of steps to construct the corresponding Boolean formula, Cook's theorem cannot be established.
- Although we can construct a Boolean formula describing the original problem, we are still unable to easily solve the original problem because the satisfiability of the Boolean formula cannot be determined easily.

Cook's Theorem

- Note that when we prove that a formula is satisfiable, we are finding an assignment satisfying this formula. This work is equivalent to finding a solution of the original problem.
- The non-deterministic algorithm irresponsibly ignores the time needed to find this solution as it claims that it always makes a correct guess.
- A deterministic algorithm to solve the satisfiability problem cannot ignore this time needed to find an assignment.
- Cook's theorem indicates that if we can find an assignment satisfying a Boolean formula in polynomial time, then we can really correctly guess a solution in polynomial time.
- Unfortunately, up to now, we cannot find an assignment in polynomial time. Therefore, we cannot guess correctly in polynomial time.

Np-complete

- Cook's theorem informs us that the satisfiability problem is a very difficult problem among all NP problems because if it can be solved in polynomial time, then all NP problems can he solved in polynomial time.
- But, is the satisfiability problem the only problem in NP with this kind of property?
- We shall see that there is a class of problems which are equivalent to one another in the sense that if any of them can he solved in polynomial time, then all NP problems can be solved in polynomial time. They are called the class of NP-complete problems.

8-6 NP-complete problems

- Problem A reduces to problem B ($A \propto B$)
  iff A can be solved by using any algorithm which solves B.

  If there is a polynomial time algorithm solving B, then there is a polynomial time algorithm to solve A.

  If $A \propto B$, B is more difficult.

$$
\begin{tikzcd}
\text{instance of A} \arrow[r, "\text{transformation}", "T(tr_1)"'] \arrow[d, "T(A)"'] & \text{instance of B} \arrow[d, "T(B)"', "\text{solver of B}"] \\
\text{answer of A} & \text{answer of B} \arrow[l, "\text{transformation}"', "T(tr_2)"]
\end{tikzcd}
$$

Note: $T(tr1) + T(tr2) < T(B)$

$T(A) \le T(tr1) + T(tr2) + T(B) \sim O(T(B))$

- **If there is a polynomial time algorithm to solve B then there is a polynomial time algorithm to solve A.**
- **Every NP problem reduces to the SAT problem.**

Reduction example

- **The n-Tuple Optimization Problem**
  - **We are given a positive integer C, $C > 1$ and a positive integer n.**
  - **Our problem is to determine whether there exist positive integers $c_1, c_2, \ldots, c_n$ such that.**

$$\prod c_i = C \text{ and } \sum_{i=1}^{n} c_i \text{ is minimized}$$

- **Prime number problem**
  - **Determine whether a positive integer C is a prime number or not.**
- **Prime number problem $\propto$ n-tuple optimization problem.**

- **After solving the n-tuple optimization problem, we examine the solution $c_1, c_2, \ldots, c_n$.**
- **C is a prime number if and only if there is exactly one not equal to 1, and all other c_i's are equal to 1.**
- **This examination process takes only n steps and is therefore a polynomial process.**
- **In summary, if the n-tuple optimization problem can be solved in polynomial time, then the prime number problem can be solved in polynomial time.**

Example 8–8 The Bin Packing Problem and Bucket Assignment Problem

- **The bin packing decision problem**
  - We are given a set *of* n objects which will be put into B bins.
  - Each bin has capacity C and each object requires $\mathbf{c_i}$ units of capacity.
  - The bin packing decision problem is to determine whether we can divide these n objects into k, $1 \leqq k \leqq B$, groups such that each group of objects can be put into a bin.
- For instance, let $(c_1, c_2, c_3, c_4) = (1, 4, 7, 4)$, C = 8 and $B = 2$. Then we can divide the objects into two groups: objects 1 and 3 in one group and objects 2 and 4 in one group.
- If $(c_1, c_2, c_3, c_4) = (1, 4, 8, 4)$, C = 8 and $B = 2$, then there is no way to divide the objects into two groups or one group such that each group of objects can be put into a bin without exceeding the capacity of that bin.

bucket assignment decision problem

- **Bucket Assignment Problem**
- We are given n records which are all characterized by one key.
- This key assumes h distinct values: $v_1, v_2, \ldots, v_h$ and there are n_i records corresponding to v_i. That is. $n_1+n_2+\ldots n_h=n$.
- The bucket assignment decision problem is to determine whether we can put these n records into k buckets such that records with the same v, are within one bucket and no bucket contains more than C records.

bucket assignment decision problem

For instance, let the key assume values a, b, c and d, $(n_a, n_b, n_c, n_d) = (1, 4, 2, 3)$, $k = 2$ and $C = 5$. Then we can put the records into two buckets as follows:

Bucket 1	Bucket 2
a	c
b	c
b	d
b	d
b	d

V=2 for each bucket

If $(n_a, n_b, n_c, n_d) = (2, 4, 2, 2)$, then there is no way for us to assign the records into buckets without exceeding the capacity of each bucket and keeping the records with the same key value in the same bucket.

The Bin Packing Problem ∝ Bucket Assignment Problem

Reduction

- From the definition of "reduce to", we can easily see the following: If A1 $\propto$ A2 *and A2* $\propto$ *A3, then A1* $\propto$ *A3*.
- **Definition: A problem A is NP-complete if A $\in$ NP and every NP problem reduces to** A.
- IF A is an NP-complete problem and A can be solved in polynomial time, then every NP problem can be solved in polynomial time.
- Clearly. the satisfiability problem is an NP-complete problem because of Cook's theorem.
- By definition, if any NP-complete problem can be solved in polynomial time, hen NP = P.

SAT is NP-complete

(1) SAT is an NP algorithm.

(2) SAT is NP-hard:

- Every NP algorithm can be transformed in polynomial time to SAT [Horowitz 1998] such that SAT is satisfiable if and only if the answer for the original NP problem is “YES”.
- That is, every NP problem $\propto$ SAT .

- By (1) and (2), SAT is NP-complete.

SAT is the first NP-complete problem

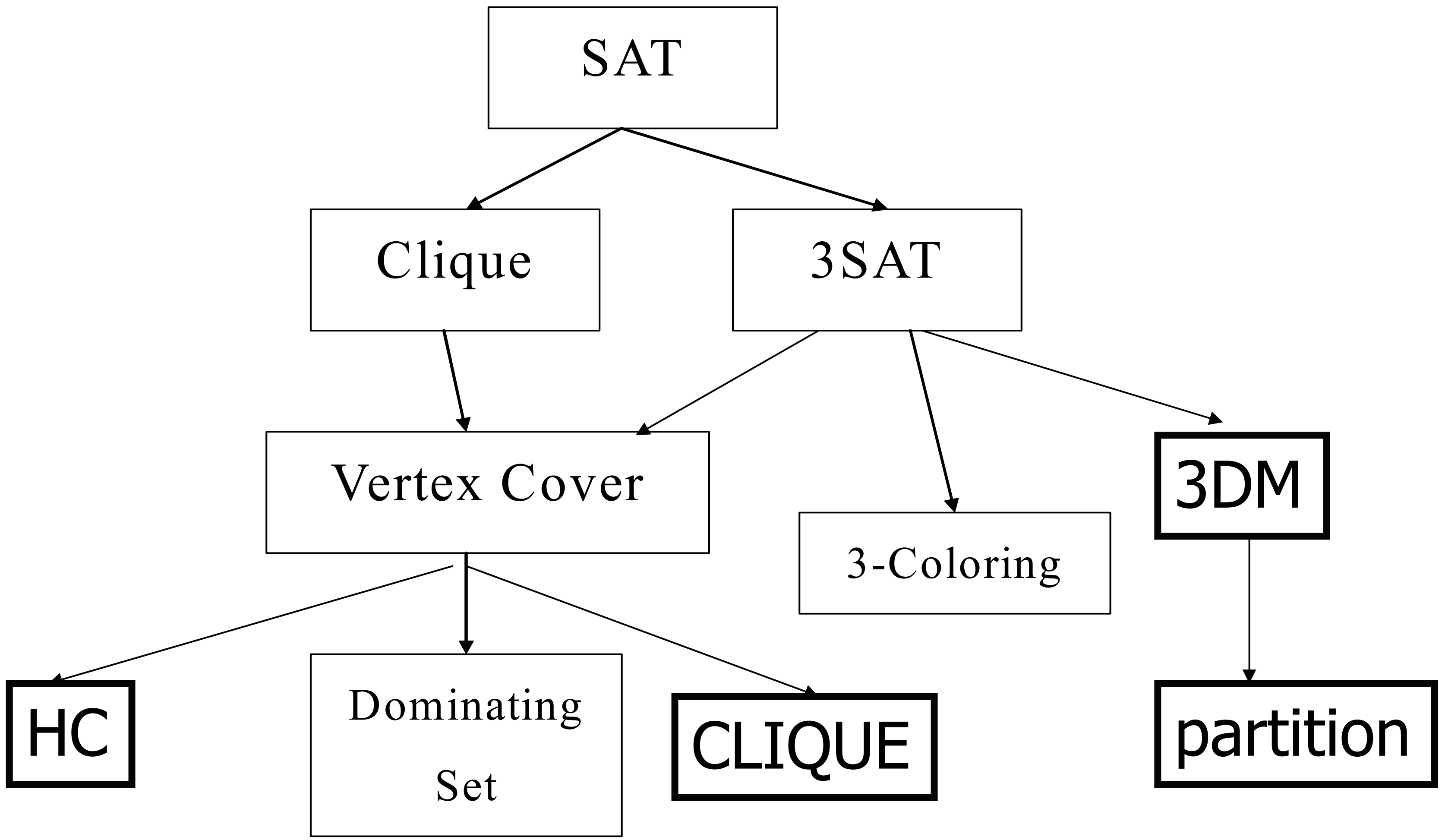
Proof of NP-Completeness

- To show that A is NP-complete
  (I) **Prove that A is an NP problem.**
  (II) **Prove that $\exists$ B $\in$ NPC, B $\propto$ A.**
  $\Rightarrow$ A $\in$ NPC
- Why ?
- If B is an NP-complete problem, then all NP problems reduce to *B.*
- If B $\propto$ A, then all NP problems reduce to A because of the transitive property of "reduce to".
- Therefore *A* must be NP-complete.

equivalent class of NPC

- If A is an NP-complete problem, then by definition every NP problem, say B, reduces to A.
- If we further prove that B is NP-complete by proving $A \underline{\propto} B$, then A and B are equivalent to each other.
- In summary, all NP-complete problems form an **equivalent class**.

8.7 Examples of NP-Complete Proofs



3-satisfiability problem (3-SAT)

- **Def**: Each clause contains exactly three literals.
- (I) 3-SAT is an NP problem (obviously)
- (II) SAT $\propto$ 3-SAT : the satisfiability problem reduces to the 3-satisfiability problem.
- We shall show that for every ordinary Boolean formula F_1, we can create another Boolean formula F_2, in which every clause contains exactly three literals, such that F_1 is satisfiable if and only if F_2 satisfiable.

Proof:

(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 : <u>remain unchanged</u>.

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

in 3-SAT :

$L_1 \vee L_2 \vee y_1$

$L_3 \vee -y_1 \vee y_2$

$\vdots$

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

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

Example of transforming 3-SAT to SAT

- an instance S in SAT :

  $x_1 \vee x_2$

  $-x_3$

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

- The instance S′ in 3-SAT :

  $x_1 \vee x_2 \vee y_1$

  $x_1 \vee x_2 \vee -y_1$

  $-x_3 \vee y_2 \vee y_3$

  $-x_3 \vee -y_2 \vee y_3$

  $-x_3 \vee y_2 \vee -y_3$

  $-x_3 \vee -y_2 \vee -y_3$

  $x_1 \vee -x_2 \vee y_4$

  $x_3 \vee -y_4 \vee y_5$

  $-x_4 \vee x_5 \vee -y_5$

$$\begin{array}{c} \text{SAT} \\ S \end{array} \xrightarrow{\textbf{transform}} \begin{array}{c} \text{3-SAT} \\ S' \end{array}$$

- Proof : S(SAT) is satisfiable $\Leftrightarrow$ S′ (3SAT) 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 : $y_{i-1} = F$

- $y_j = T \quad \forall j < i-1$

  $y_j = F \quad \forall j > i-1$

  $(\because L_i \vee -y_{i-2} \vee y_{i-1})$

  $\Rightarrow$ 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).

Thus, 3-SAT is NP-complete.

Comment for 3-SAT

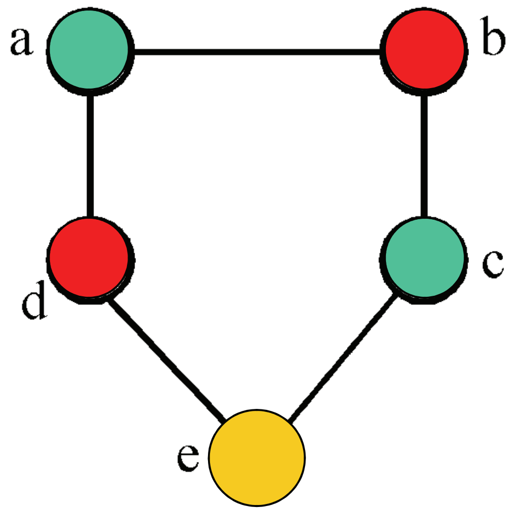
- If a problem is NP-complete, its special cases may or may not be NP-complete.

Chromatic number decision problem (CN)

- **Def**: A coloring of a graph G=(V, E) is a function

f : V $\rightarrow$ { 1, 2, 3,…, k } such that if (u, v) $\in$ E, then f(u)$\neq$f(v). The CN problem is to determine if G has a coloring for k.

- E.g.



3-colorable

f(a)=1, f(b)=2, f(c)=1

f(d)=2, f(e)=3

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

3SAT $\propto$ CN

- We shall now prove that the satisfiability problem with at most three literals per clause reduces to the chromatic number decision problem.
- Essentially, we shall show that for every satisfiability problem with at most three literals per clause (n variables), we can construct a corresponding graph such that the original Boolean formula is satisfiable if and only if the constructed graph can be colored by using $n + 1$ colors where n is the number of variables occurring in the Boolean formula.

3SAT $\propto$ CN

Proof :

instance of SATY :

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

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

instance of CN :

$G=(V, E)$

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

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

Example of 3SAT $\propto$ CN

$x_1 \vee x_2 \vee x_3$ (1)

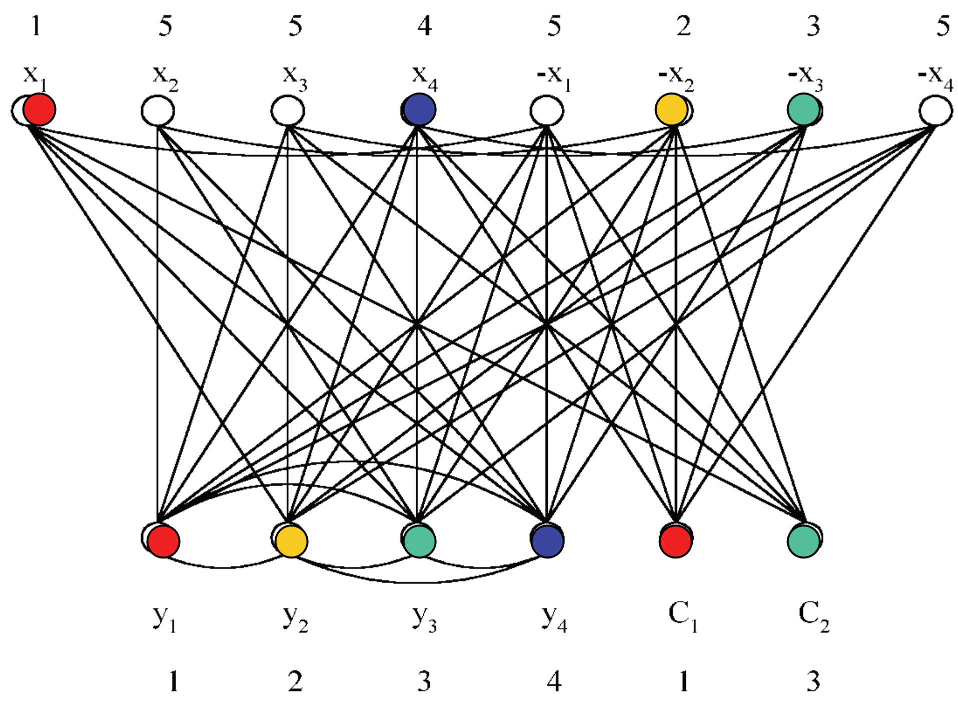
$-x_3 \vee -x_4 \vee x_2$ (2)

$\Downarrow$

True assignment:

x_1=T x_2=F

x_3=F x_4=T



Proof of 3SAT $\propto$ CN

- Satisfiable $\Leftrightarrow$ n+1 colorable
  - "$\Rightarrow$"

    (1) $f(y_i) = i$ (y_i is colored with color 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, $\ni$ 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

Ex 8-11 Set cover decision problem

- **Def:** $F = \{S_i\} = \{ S_1, S_2, \ldots, S_k \}$

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

  T is a set cover 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.

- example

$$F = \{\underset{s_1}{(a_1, a_3)}, \underset{s_2}{(a_2, a_4)}, \underset{s_3}{(a_2, a_3)}, \underset{s_4}{(a_4)}, \underset{s_5}{(a_1, a_3, a_4)}\}$$

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

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

Exact cover problem

(Notations same as those in set cover.)

Def: To determine if F has an exact cover T, which is a cover of F and the sets in T are pairwise disjoint.

<Theorem> CN $\propto$ exact cover

(No proof here.)

CN $\propto$ exact cover

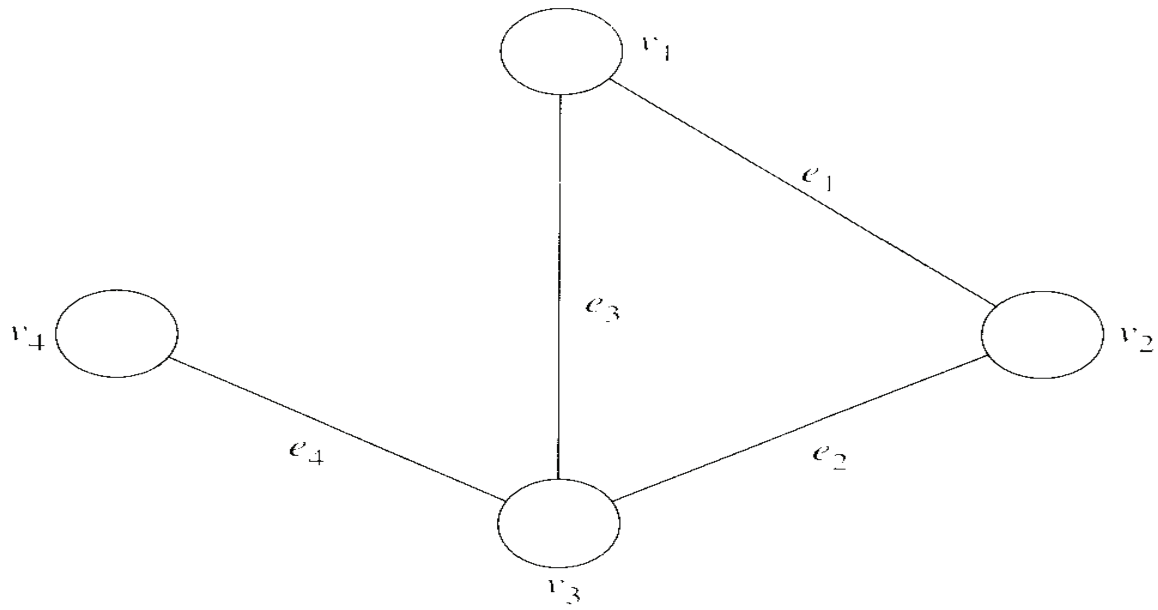
Let the set of vertices of the given graph in the chromatic coloring problem be $\boxed{V = \{v_1, v_2, \ldots, v_n\}}$ and the set of edges be $\boxed{E = \{e_1, e_2, \ldots, e_m\}}$. Together with the integer k, we transform this chromatic coloring problem instance into an exact cover problem instance $S = \{v_1, v_2, \ldots, v_n, E_{11}, E_{12}, \ldots, E_{1k}, E_{21}, E_{22}, \ldots, E_{2k}, \ldots, E_{m1}, E_{m2}, \ldots, E_{mk}\}$, where $E_{i1}, E_{i2}, \ldots, E_{ik}$ correspond to e_i, $1 \le i \le m$, and a family F of subsets $F = \{C_{11}, C_{12}, \ldots, C_{1k}, C_{21}, C_{22}, \ldots, C_{2k}, \ldots, C_{n1}, C_{n2}, \ldots, C_{nk}, D_{11}, D_{12}, \ldots, D_{1k}, D_{21}, D_{22}, \ldots, D_{2k}, \ldots, D_{m1}, D_{m2}, \ldots, D_{mk}\}$. Each C_{ij} and D_{ij} are determined according to the following rule:

(1) If edge e_i has vertices v_a and v_b as its ending terminals, then C_{ad} and C_{bd} will both contain E_{id} for $d = 1, 2, \ldots, k$.

(2) $D_{ij} = \{E_{ij}\}$ for all i and j.

(3) C_{ij} contains v_i for $j = 1, 2, \ldots, k$.

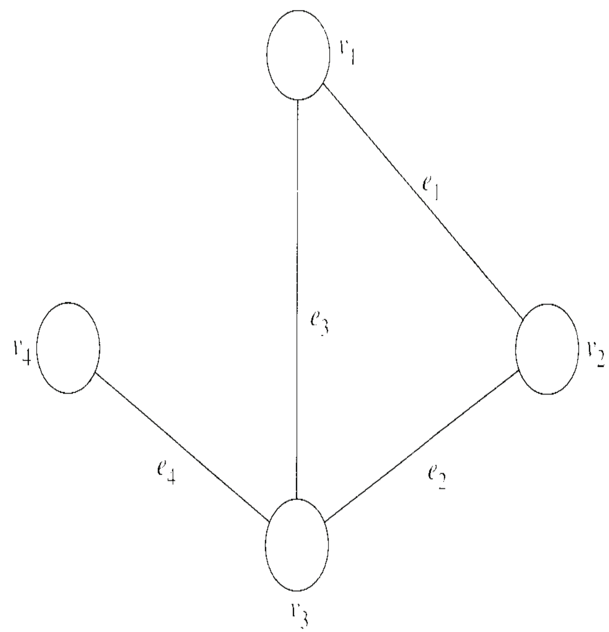
FIGURE 8–14 A graph illustrating the transformation of a chromatic coloring problem to an exact cover problem.



In this case, $n = 4$ and $m = 4$. Suppose $k = 3$. We therefore have $S = \{v_1, v_2, v_3, v_4, E_{11}, E_{12}, E_{13}, E_{21}, E_{22}, E_{23}, E_{31}, E_{32}, E_{33}, E_{41}, E_{42}, E_{43}\}$ and $F = \{C_{11}, C_{12}, C_{13}, C_{21}, C_{22}, C_{23}, C_{31}, C_{32}, C_{33}, C_{41}, C_{42}, C_{43}, D_{11}, D_{12}, D_{13}, D_{21}, D_{22}, D_{23}, D_{31}, D_{32}, D_{33}, D_{41}, D_{42}, D_{43}\}$. Each D_{ij} contains exactly one E_{ij}. For C_{ij}, we shall illustrate its contents by one example. Consider e_1, which is connected by v_1 and v_2. This means that C_{11} and C_{21} will both contain E_{11}. Similarly, C_{12} and C_{22} will both contain E_{12}. C_{13} and C_{23} will also both contain E_{13}.

$$
\begin{aligned}
C_{11} &= \{E_{11}, E_{31}, v_1\}, \\
C_{12} &= \{E_{12}, E_{32}, v_1\}, \\
C_{13} &= \{E_{13}, E_{33}, v_1\}, \\
C_{21} &= \{E_{11}, E_{21}, v_2\}, \\
C_{22} &= \{E_{12}, E_{22}, v_2\}, \\
C_{23} &= \{E_{13}, E_{23}, v_2\}, \\
C_{31} &= \{E_{21}, E_{31}, E_{41}, v_3\}, \\
C_{32} &= \{E_{22}, E_{32}, E_{42}, v_3\}, \\
C_{33} &= \{E_{23}, E_{33}, E_{43}, v_3\}, \\
C_{41} &= \{E_{41}, v_4\}, \\
C_{42} &= \{E_{42}, v_4\}, \\
C_{43} &= \{E_{43}, v_4\}.
\end{aligned}
$$

FIGURE 8–14 A graph illustrating the transformation of a chromatic coloring problem to an exact cover problem.



$$
\begin{aligned}
D_{11} &= \{E_{11}\}, D_{12} = \{E_{12}\}, D_{13} = \{E_{13}\}, \\
D_{21} &= \{E_{21}\}, D_{22} = \{E_{22}\}, D_{23} = \{E_{23}\}, \\
D_{31} &= \{E_{31}\}, D_{32} = \{E_{32}\}, D_{33} = \{E_{33}\}, \\
D_{41} &= \{E_{41}\}, D_{42} = \{E_{42}\}, D_{43} = \{E_{43}\}.
\end{aligned}
$$

Sum of subsets problem

- **Def:** 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. $A = \{ 7, 5, 19, 1, 12, 8, 14 \}$
  - $C = 21$, $A' = \{ 7, 14 \}$
  - $C = 11$, no solution

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

Exact cover $\propto$ sum of subsets

- Proof :

  instance of exact cover :

  $F = \{ S_1, S_2, \ldots, S_k \}$ $\qquad \bigcup_{S_i \in F} S_i = \{u_1, u_2, \ldots, u_n\}$

  instance of sum of subsets :

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

  $a_i = \sum_{1 \le j \le n} e_{ij}(k+1)^j$ where $e_{ij} = 1$ if $u_j \in S_i$

  $e_{ij} = 0$ if otherwise.

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

- Why k+1? (See the example on the next page.)

Example of Exact cover $\propto$ sum of subsets

- Valid transformation:

  $u_1=1$, $u_2=2$, $u_3=3$, $n=3$

  EC: $\mathbf{S_1=\{1,2\}, S_2=\{3\}}$, $S_3=\{1,3\}$, $S_4=\{2,3\}$

  $F=\{u_1, u_2, u_3\}=\{1,2,3\}$

  $k=4$

  SS: $a_1=5^1+5^2=30$

  $a_2=5^3=125$

  $a_3=5^1+5^3=130$

  $a_4=5^2+5^3=150$

  $C=5^1+5^2+5^3=155$

$$a_i = \sum_{1\le j\le n} e_{ij}(k+1)^j$$

- Invalid transformation:

  EC: $S_1=\{1,2\}$, $S_2=\{2\}$, $S_3=\{2\}$, $S_4=\{2,3\}$. $K=4$

  Suppose $k-2=2$ is used.

  SS: $a_1=2^1+2^2=6$

  $a_2=2^2=4$

  $a_3=2^2=4$

  $a_4=2^2+2^3=12$

  $C=2^1+2^2+2^3=14$

Partition problem

- **Def**: Given 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. $A = \{3, 6, 1, 9, 4, 11\}$
  partition : $\{3, 1, 9, 4\}$ and $\{6, 11\}$

<Theorem> sum of subsets $\propto$ partition

Sum of subsets $\propto$ partition

proof :

instance of sum of subsets :

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

instance of partition :

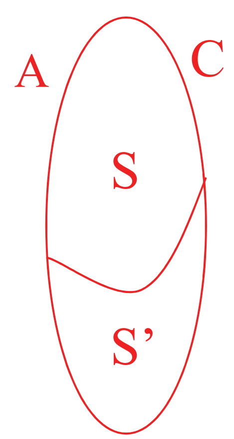
$B = \{ b_1, b_2, \ldots, b_{n+2} \}$, where $b_i = a_i,\ 1 \le i \le n$

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

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

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

$\Leftrightarrow$ 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

- **Def:** n items, each of size c_i , $c_i > 0$, integer
  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 $\propto$ bin packing.

VLSI discrete layout problem

- Given: n rectangles, each with height h_i (integer)

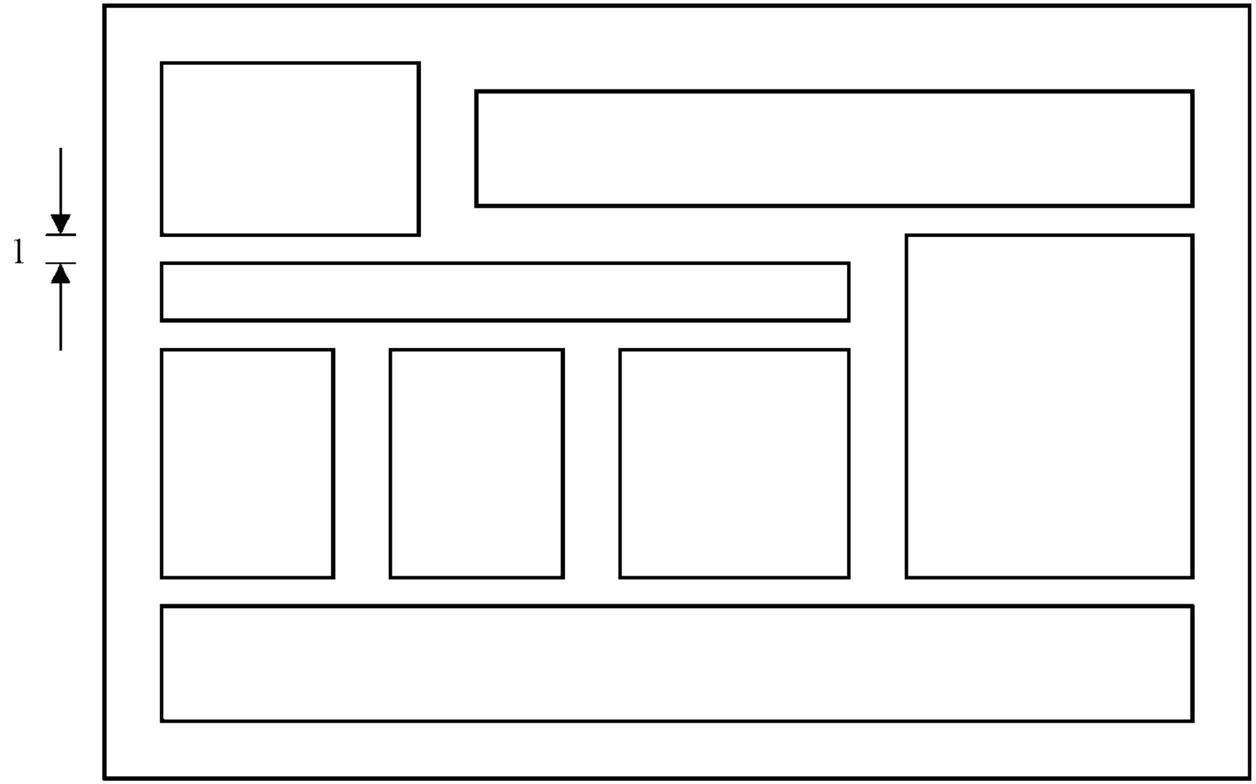
  width w_i

  and an area A

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

  1. Boundaries of rectangles parallel to x axis or y axis.
  2. Corners of rectangles lie on integer points.
  3. No two rectangles overlap.
  4. Two rectangles are separated by at least a unit distance.

  (See the figure on the next page.)

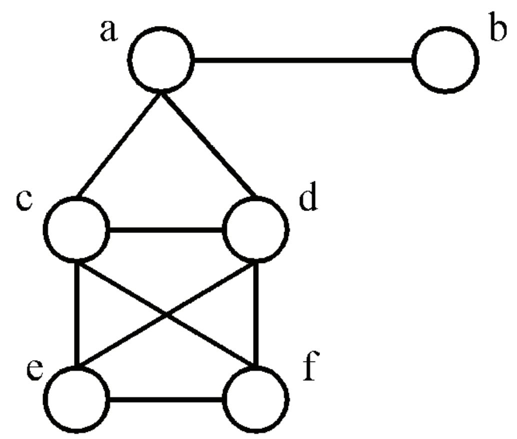


A Successful Placement

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

Max clique problem

- **Def**: 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

- **Def**: 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.

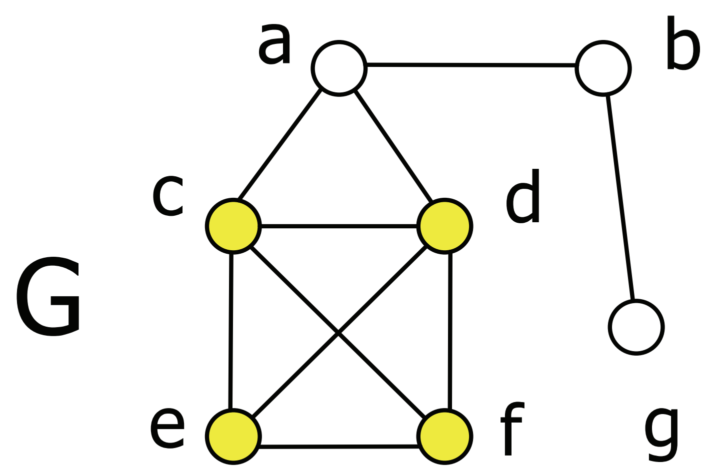
(See proof on the next page.)

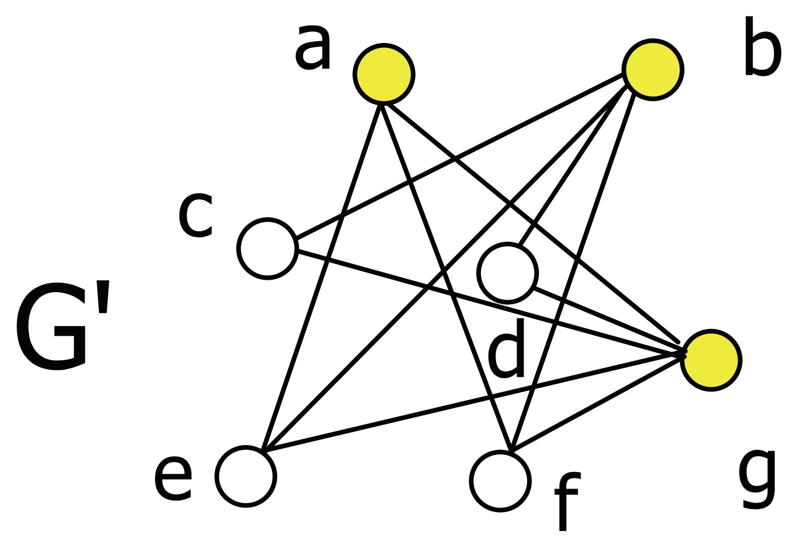
Clique decision $\propto$ node cover decision

- $G=(V,E)$: clique Q of size k ($Q \subseteq V$)

  $\updownarrow$

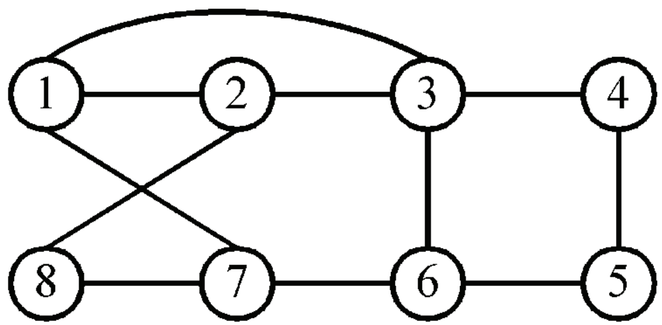
  $G'=(V,E')$: node cover S of size $n-k$, $S=V-Q$
  where $E'=\{(u,v) \mid u \in V, v \in V \text{ and } (u,v) \notin E\}$





Hamiltonian cycle problem

- **Def**: 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)

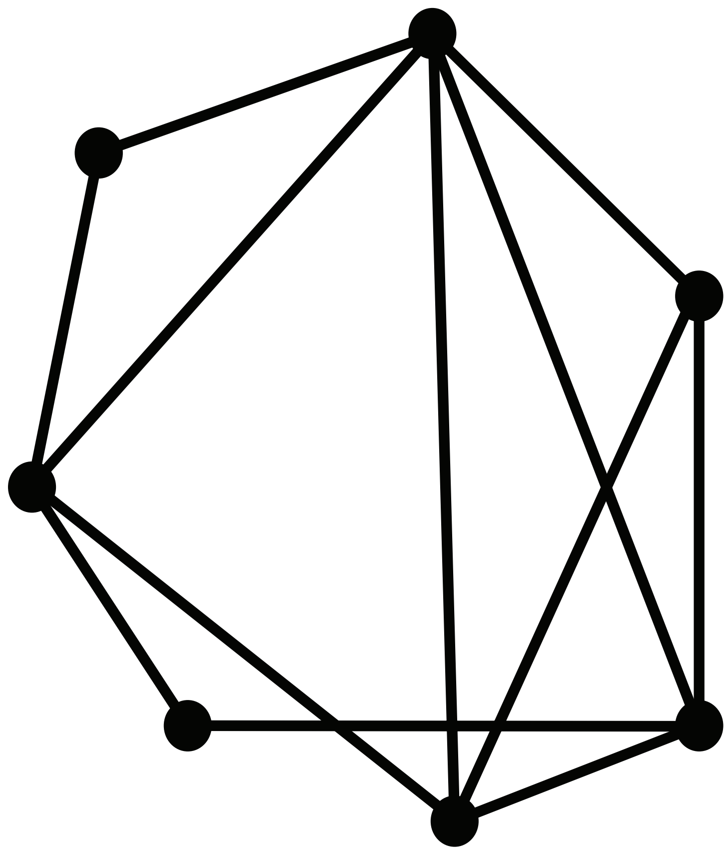
Traveling salesperson problem

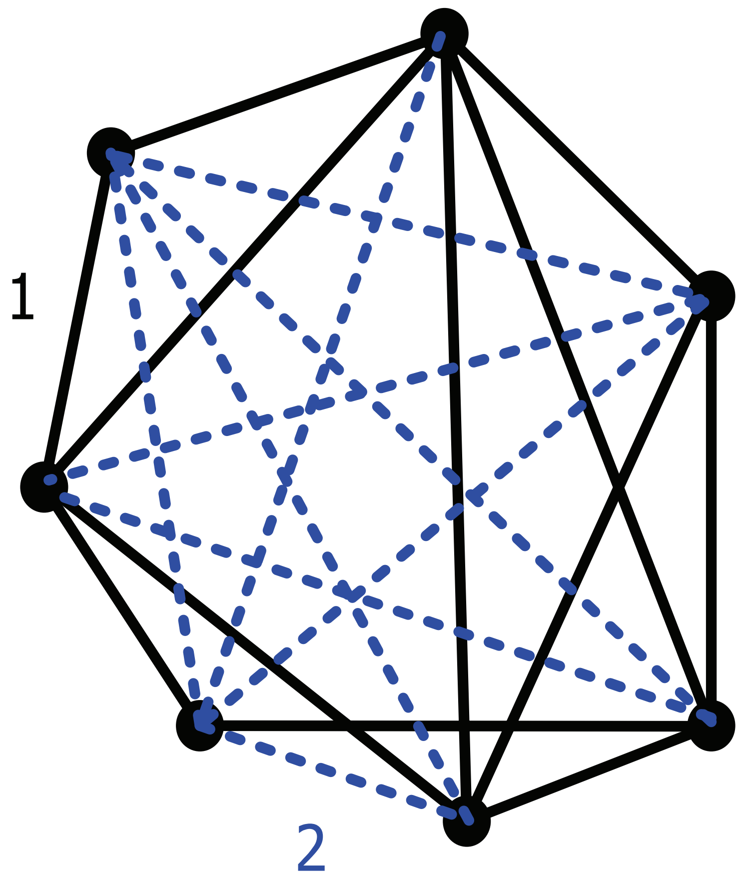
- **Def:** A tour of a directed graph G=(V, E) is a directed cycle that includes every vertex in V. The problem is to find a tour of minimum cost.

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

(See proof on the next page.)

Proof of Hamiltonian ∝ TSP





0/1 knapsack problem

- **Def**: 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.

- Refer to Sec. 11.3, Sec. 11.4 and its exercises of [Horowitz 1998] for the proofs of more NP-complete problems.