Chapter 3

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.

Some concepts of NPC

- Definition of reduction: Problem A reduces to problem B (A $\preceq B$) 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 $A \preceq B$ and $B \preceq A$.

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”.
- Optimization problems are more difficult.
- 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 $c$?

The satisfiability problem (SAT)

- The satisfiability problem
  - The logical formula:
    \[
    x_1 \lor x_2 \lor x_3 \\
    \land -x_1 \\
    \land -x_2
    \]
  - the assignment:
    \[
    x_1 \false, x_2 \false, x_3 \true
    \]
    will make the above formula true.
    \(-x_1, -x_2, x_3\) represents $x_1 \false, x_2 \false, x_3 \false$.

Solving an optimization problem by a decision algorithm:

- Solving TSP optimization problem by decision algorithm:
  - Give $c_1$ and test (decision algorithm)
  - Give $c_2$ and test (decision algorithm)
    \[
    M
    \]
  - Give $c_n$ and test (decision algorithm)
  - We can “easily” find the smallest $c_i$.

- 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:
  \[
  x_1 \lor x_2 \\
  \land x_1 \lor -x_2 \\
  \land -x_1 \lor x_2 \\
  \land -x_1 \lor -x_2
  \]
Definition of the satisfiability problem: Given a Boolean formula, determine whether this formula is satisfiable or not.

- A literal: \( x_i \) or \( \neg x_i \)
- A clause: \( x_1 \lor x_2 \lor \neg x_3 \equiv C_i \)
- A formula: conjunctive normal form \( C_1 \land C_2 \land \ldots \land C_m \)

Resolution principle

\[ C_1 : \neg x_1 \lor \neg x_2 \lor x_3 \]
\[ C_2 : x_1 \lor x_4 \]
\[ \Rightarrow C_3 : \neg x_2 \lor x_3 \lor x_4 \] resolvent

If no new clauses can be deduced, then it is satisfiable.

\[ \neg x_1 \lor \neg x_2 \lor x_3 \] (1)
\[ x_1 \] (2)
\[ x_2 \] (3)
(1) & (2) \[ \neg x_2 \lor x_3 \] (4)
(4) & (3) \[ x_3 \] (5)
(1) & (3) \[ \neg x_1 \lor x_3 \] (6)

If an empty clause is deduced, then it is unsatisfiable.

\[ \neg x_1 \lor \neg x_2 \lor x_3 \] (1)
\[ x_1 \lor \neg x_2 \] (2)
\[ x_2 \] (3)
\[ \neg x_3 \] (4)
\[ \emptyset \]

\[ (1) \lor (2) \neg x_2 \lor x_3 \] (5)
\[ (4) \lor (5) \neg x_2 \] (6)
\[ (6) \lor (3) \] (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, then it is unsatisfiable.

\[ (\neg x_1 \lor \neg x_2 \lor x_3) \] (7)
\[ (\neg \neg x_2 \lor \neg x_3) \] (6)
\[ (\neg x_1 \lor \neg x_2 \lor x_3) \] (0)
\[ (\neg x_1 \lor x_2 \lor \neg x_3) \] (0)
\[ (\neg x_1 \lor \neg x_3) \] (0)
\[ (\neg x_1 \lor \neg x_3) \] (0)
Nondeterministic algorithms

- A nondeterministic algorithm consists of
  - Phase 1: guessing
  - Phase 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 problems

- Decision version of sorting:
  - Given \( a_1, a_2, \ldots, a_n \) and \( c \), is there a permutation of \( a'_1, a'_2, \ldots, a'_n \) such that \( a'_2 - a'_1 + a'_3 - a'_2 + \ldots + a'_{n-1} - a'_n \) \( < c \) ?
- 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

Non-deterministic operations and functions

[Horowitz 1998]

- **Choice(S)**: arbitrarily chooses one of the elements in set \( S \)
- **Failure**: an unsuccessful completion
- **Success**: a successful completion
- Non-deterministic searching algorithm:
  \[ j \leftarrow \text{choice}(1 : n) \quad /\ast \text{ guessing } /\ast \]
  \[ \text{if } A(j) = x \text{ then success } /\ast \text{ checking } /\ast \]
  \[ \text{else failure} \]

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

\[ B \leftarrow 0 \]

/* guessing */

for \( i = 1 \) to \( n \) do

\[ j \leftarrow \text{choice}(1 : n) \]

if \( B[j] \neq 0 \) then failure

\[ B[j] = A[i] \]

/* checking */

for \( i = 1 \) to \( n-1 \) do

if \( B[i] > B[i+1] \) then failure

success

Cook’s theorem

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

- Every NP problem reduces to SAT.
- SAT is NP-complete.
- It is the first NP-complete problem.

Transforming Searching to SAT

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

nondeterministic algorithm:

\[ i = \text{choice}(1,2) \]

if \( x(i) = 7 \) then SUCCESS
else FAILURE
i=1 \lor i=2 \\
& i=1 \not= i=2 \\
& i=2 \not= i=1 \\
& x(1)=7 \land i=1 \quad \rightarrow \quad \text{SUCCESS} \\
& x(2)=7 \land i=2 \quad \rightarrow \quad \text{SUCCESS} \\
& x(1) \not= 7 \land i=1 \quad \rightarrow \quad \text{FAILURE} \\
& x(2) \not= 7 \land i=2 \quad \rightarrow \quad \text{FAILURE} \\
& \text{FAILURE} \\
& \text{-SUCCESS} \\
& \text{SUCCESS} \quad (\text{Guarantees a successful termination}) \\
& x(1)=7 \quad (\text{Input Data}) \\
& x(2) \not= 7

\textbf{CNF (conjunctive normal form):}

i=1 \lor i=2 \\
i \not= 1 \lor i \not= 2 \\
x(1)=7 \lor i \not= 1 \lor \text{SUCCESS} \\
x(2)=7 \lor i \not= 2 \lor \text{SUCCESS} \\
x(1)=7 \lor i \not= 1 \lor \text{FAILURE} \\
x(2)=7 \lor i \not= 2 \lor \text{FAILURE} \\
\text{-FAILURE} \lor \text{-SUCCESS} \\
\text{SUCCESS} \\
x(1)=7 \\
x(2) \not= 7

\textbf{Satisfiable at the following assignment:}

i=1 \quad \text{satisfying (1)} \\
i \not= 2 \quad \text{satisfying (2), (4) and (6)} \\
\text{SUCCESS} \quad \text{satisfying (3), (4) and (8)} \\
\text{-FAILURE} \quad \text{satisfying (7)} \\
x(1)=7 \quad \text{satisfying (5) and (9)} \\
x(2) \not= 7 \quad \text{satisfying (4) and (10)}
Searching for 7, but $x(1) \neq 7$, $x(2) \neq 7$

- CNF (conjunctive normal form):

  \begin{align*}
  &i=1 \lor i=2 \quad (1) \\
  &i \neq 1 \lor i \neq 2 \quad (2) \\
  &x(1)=7 \lor i \neq 1 \lor \text{SUCCESS} \quad (3) \\
  &x(2)=7 \lor i \neq 2 \lor \text{SUCCESS} \quad (4) \\
  &x(1)=7 \lor i \neq 1 \lor \text{FAILURE} \quad (5) \\
  &x(2)=7 \lor i \neq 2 \lor \text{FAILURE} \quad (6) \\
  &\text{SUCCESS} \quad (7) \\
  &\text{SUCCESS} \lor \text{FAILURE} \quad (8) \\
  &x(1) \neq 7 \\
  &x(2) \neq 7 \quad (10)
  \end{align*}

- Apply resolution principle:

  \begin{align*}
  &\text{(9) } \land \text{ (5) } \quad i \neq 1 \lor \text{FAILURE} \quad (11) \\
  &\text{(10) } \land \text{ (6) } \quad i \neq 2 \lor \text{FAILURE} \quad (12) \\
  &\text{(7) } \land \text{ (8) } \quad \text{FAILURE} \quad (13) \\
  &\text{(13) } \land \text{ (11) } \quad i \neq 1 \quad (14) \\
  &\text{(13) } \land \text{ (12) } \quad i \neq 2 \quad (15) \\
  &\text{(14) } \land \text{ (1) } \quad i = 2 \quad (11) \\
  &\text{(15) } \land \text{ (16) } \quad \Box \quad (17)
  \end{align*}

We get an empty clause $\Rightarrow$ unsatisfiable
$\Rightarrow 7$ does not exist in $x(1)$ or $x(2)$.


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

- CNF:

  \begin{align*}
  &i=1 \lor i=2 \quad (1) \\
  &i \neq 1 \lor i \neq 2 \quad (2) \\
  &x(1)=7 \lor i \neq 1 \lor \text{SUCCESS} \quad (3) \\
  &x(2)=7 \lor i \neq 2 \lor \text{SUCCESS} \quad (4) \\
  &x(1)=7 \lor i \neq 1 \lor \text{FAILURE} \quad (5) \\
  &x(2)=7 \lor i \neq 2 \lor \text{FAILURE} \quad (6) \\
  &\text{SUCCESS} \quad (7) \\
  &\text{SUCCESS} \lor \text{FAILURE} \quad (8) \\
  &x(1)=7 \lor \text{FAILURE} \\
  &x(2)=7 \quad (10)
  \end{align*}

The semantic tree

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

- **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 $v \in S$.

- Decision problem: $\exists S \ni |S| \leq K$?

Reducing Node Covering to SAT

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

FOR $j=1$ to $m$ DO
    BEGIN
        if $e_j$ is not incident to one of $v_{i_t}$ $(1 \leq t \leq k)$ then FAILURE
    END
SUCCESS

CNF:

$$
\begin{align*}
\text{SUCCESS} & \lor \text{FAILURE} \\
& \lor \left( v_{i_1} \in e_1 \lor \ldots \lor v_{i_n} \in e_m \right) \\
& \lor \left( v_{i_1} \in e_1 \lor \ldots \lor v_{i_n} \in e_m \right) \\
& \lor \left( v_{i_1} \in e_1 \lor \ldots \lor v_{i_n} \in e_m \right) \\
& \lor \left( v_{i_1} \in e_1 \lor \ldots \lor v_{i_n} \in e_m \right)
\end{align*}
$$
Important Definition and Concepts

- Problem $A$ is NP-complete if $A$ is in NP and every NP problem reduces to $A$.
- If $A_1$ is NP-complete, $A_2$ is in NP and we can prove that $A_1$ reduces to $A_2$, then $A_2$ is NP-complete. (Why?)
- SAT is the most difficult in NP.
- To prove that problem $A$ is NP-complete, we reduce $A$ to SAT.

If any NP-complete problem can be solved in polynomial time, then all NP-complete problems are P-solvable. Namely, NP=P.

NP-complete problems constitute an equivalence class.

Given any two NP-complete problems $A$ and $B$, $A$ reduces to $B$ and vice versa.

“I can’t find an efficient algorithm, I guess I’m just too dumb.”
SAT is NP-complete

(1) SAT is an NP problem.
(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 $\preceq$ SAT.
   - By (1) and (2), SAT is NP-complete.

Proof of NP-Completeness

- To show that problem A is NP-complete
  (I) Prove that A is an NP problem.
  (II) Prove that $\exists \ B \in \text{NPC}, B \preceq A$.
  $\Rightarrow A \in \text{NPC}$
- Why?
- Note the time required by the reduction!
Supplementary for Time Complexity

- Time complexity is defined in terms of input length instead of the number of items.
- Consider the binary encoding of some input instance for sorting problem: \(x_1, x_2, \ldots, x_n\).
- The input length is \(\log x_1 + \log x_2 + \cdots + \log x_n\).
- Clearly, the length is no less than \(n\).

The time complexity is exponential!
Complexity Hierarchy

- Polynomial
- Pseudo-polynomial
- Exponential
- Far beyond

3-satisfiability problem (3-SAT)

- Each clause contains exactly three literals.
- (I) 3-SAT is an NP problem (obviously)
- (II) SAT \( \propto \) 3-SAT

Proof:
1. One literal \( L_1 \) in a clause in SAT:
   - in 3-SAT:
     \( L_1 \lor L_2 \lor y_1 \)
     \( L_1 \lor L_2 \lor -y_1 \)

2. Two literals \( L_1, L_2 \) in a clause in SAT:
   - in 3-SAT:
     \( L_1 \lor L_2 \lor y_1 \)
     \( L_1 \lor L_2 \lor -y_1 \)

3. Three literals in a clause: remain unchanged.

4. More than 3 literals \( L_1, L_2, \ldots, L_4 \) in a clause:
   - in 3-SAT:
     \( L_1 \lor L_2 \lor y_1 \)
     \( L_3 \lor -y_1 \lor y_2 \)
     \( M \)
     \( L_{k-2} \lor -y_k \lor y_{k+1} \)
     \( L_{k-1} \lor L_k \lor -y_k \)

Example of transforming 3-SAT to SAT

- an instance \( S \) in SAT:
  \( x_1 \lor x_2 \)
  \( -x_3 \)
  \( x_1 \lor -x_2 \lor x_3 \lor -x_4 \lor x_5 \)

- The instance \( S' \) in 3-SAT:
  \( x_1 \lor x_2 \lor y_1 \)
  \( x_1 \lor x_2 \lor -y_1 \)
  \( -x_3 \lor y_2 \lor y_3 \)
  \( -x_3 \lor -y_2 \lor y_3 \)
  \( -x_3 \lor -y_2 \lor -y_3 \)
  \( x_1 \lor -x_2 \lor y_4 \)
  \( x_3 \lor -y_4 \lor y_5 \)
  \( -x_4 \lor x_5 \lor -y_5 \)

SAT \( \xrightarrow{\text{transform}} \) 3-SAT

\( S \) \hspace{1cm} 3-SAT \hspace{1cm} \( S' \)
Proof: $S$ is satisfiable $\iff S'$ is satisfiable

$\Rightarrow$

$\leq 3$ literals in $S$ (trivial)

consider $\geq 4$ literals

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

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

$L_3 \lor \neg y_1 \lor y_2$

$L_4 \lor \neg y_2 \lor y_3$

$\vdots$

$L_{k-2} \lor \neg y_{k-4} \lor y_{k-3}$

$L_{k-1} \lor L_k \lor \neg y_{k-3}$

Comment for 3-SAT

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

3-SAT is NP-complete.

Chromatic number decision problem (CN)

- **Def**: A coloring of a graph $G=(V, E)$ is a function $f: V \rightarrow \{1, 2, 3, \ldots, k\}$ 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.**

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

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

  3-colorable

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

Proof:

instance of SATY:
variable: x₁, x₂, ..., xₙ, n \geq 4
clause: c₁, c₂, ..., cₙ

instance of CN:
G=(V,E)
V={x₁, x₂, ..., xₙ} \cup {-x₁, -x₂, ..., -xₙ}
∪ {y₁, y₂, ..., yₙ} \cup {c₁, c₂, ..., cₙ}
| 1 4 2 4 3 |
newly added
E={ (xᵢ, -xᵢ) | 1 \leq i \leq n } \cup { (yᵢ, yⱼ) | i \neq j }
∪ { (yᵢ, cⱼ) | i \neq j } \cup { (yᵢ, -cⱼ) | i \neq j }
∪ { (xᵢ, cⱼ) | xᵢ \neq cⱼ } \cup { (-xᵢ, cⱼ) | -xᵢ \neq cⱼ }

Example of SATY \propto CN

x₁ \lor x₂ \lor x₃ (1) True assignment:
x₁=T
-x₁ \lor -x₂ \lor x₄ (2)

Proof of SATY \propto CN

- Satisfiable \iff n+1 colorable
- "\Rightarrow"
  (1) f(yᵢ) = i
  (2) if xᵢ = T, then f(xᵢ) = i, f(-xᵢ) = n+1
     else f(xᵢ) = n+1, f(-xᵢ) = i
  (3) if xᵢ in cⱼ and xᵢ = T, then f(cⱼ) = f(xᵢ)
     if -xᵢ in cⱼ and -xᵢ = T, then f(cⱼ) = f(-xᵢ)
      (at least one such xᵢ)

- "\Leftarrow"
  (1) yᵢ must be assigned with color i.
  (2) f(xᵢ) \neq f(-xᵢ)
   either f(xᵢ) = i and f(-xᵢ) = n+1
   or f(xᵢ) = n+1 and f(-xᵢ) = i
  (3) at most 3 literals in cⱼ and n \geq 4
      \Rightarrow at least one xᵢ, xⱼ, and -xᵢ are not in cⱼ
      \Rightarrow f(cⱼ) \neq n+1
  (4) if f(cⱼ) = i = f(xᵢ), assign xᵢ to T
      if f(cⱼ) = i = f(-xᵢ), assign -xᵢ to T
  (5) if f(cⱼ) = i = f(xᵢ) \Rightarrow (cⱼ, xᵢ) \notin E
      \Rightarrow xᵢ in cⱼ \Rightarrow cⱼ is true
      if f(cⱼ) = i = f(-xᵢ) \Rightarrow similarly
Two-Machine Flowshop

Example

Three-Machine Assembly-Type Flowshop

Set Cover Decision Problem

- **Def:** \( F = \{ S_i \} = \{ S_1, S_2, \ldots, S_k \} \)
  \[ Y_S = \{ u_1, u_2, \ldots, u_n \} \]
  \( T \) is a set cover of \( F \) if \( T \subseteq F \) and \( \forall S \in F \) \( S \subseteq \cup_{i \in T} S_i \)

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

- **example**
  \[ F = \{ (a_1, a_2), (a_2, a_3), (a_3), (a_1, a_3, a_4) \} \]
  \[ S = \{ s_1, s_2, s_3, s_4, s_5 \} \]
  \( \{ s_3 \} \) set cover
  \( \{ s_2 \} \) set cover, exact cover
Exact cover problem

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

\(<\text{Theorem}>\) CN \( \propto \) exact cover
(No proof here.)

Sum of subsets problem

- Def: A set of positive numbers \( A = \{ a_1, a_2, \ldots, a_n \} \) a constant \( C \)
  - Determine if \( \exists A' \subseteq A \) \( \sum a_i = C \)
  - e.g. \( A = \{ 7, 5, 19, 1, 12, 8, 14 \} \) \( \sum a_i^A = \)
    - \( C = 21 \), \( A' = \{ 7, 14 \} \)
    - \( C = 11 \), no solution

\(<\text{Theorem}>\) Exact cover \( \propto \) sum of subsets.

Example of Exact cover \( \propto \) sum of subsets

- Valid transformaiton:
  - \( u_1 = 1, u_2 = 2, u_3 = 3, n = 3 \)
  - EC: \( S_1 = \{ 1, 2 \}, S_2 = \{ 3 \}, S_3 = \{ 1, 3 \}, S_4 = \{ 2, 3 \} \)
  - \( F = \{ u_1, u_2, u_3 \} = \{ 1, 2, 3 \} \)
  - \( k = 4 \)
  - \( SS: a_1 = 5^1 + 5^2 = 30 \)
  - \( a_2 = 5^3 = 125 \)
  - \( a_3 = 5^4 + 5^5 = 130 \)
  - \( a_4 = 5^6 + 5^7 = 150 \)
  - \( C = 5^4 + 5^5 + 5^6 = 155 \)

- Invalid transformaiton:
  - \( EC: S_1 = \{ 1, 2 \}, S_2 = \{ 2 \}, S_3 = \{ 2 \} \)
  - \( S_4 = \{ 2, 3 \} \)
  - \( k = 4 \)
  - Suppose \( k - 2 = 2 \) is used.
  - \( SS: a_1 = 2^1 + 2^2 = 6 \)
  - \( a_2 = 2^3 = 8 \)
  - \( a_3 = 2^4 = 16 \)
  - \( a_4 = 2^5 + 2^6 = 32 \)
  - \( C = 2^4 + 2^5 + 2^6 = 14 \)
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 $b_i = a_i$, $1 \leq i \leq n$
    - $b_{n+1} = C+1$
    - $b_{n+2} = (\sum_{i \leq n} a_i) + 1 - C$
    - $C = \sum_{a \in S} a \iff (\sum_{a \in S} a) + b_{n+2} = (\sum_{a \notin S} a) + b_{n+1}$
    - $\iff$ partition: $\{ b_i \mid a_i \in S \} \cup \{ b_{n+2} \}$ and $\{ b_i \mid a_i \notin S \} \cup \{ b_{n+1} \}$

Bin packing problem

- **Def:** n items, each of size $c_i$, $c_i \geq 0$
  - bin capacity: $C$
- Determine if we can assign the items into $k$ bins, $\forall \sum_{i \in S} c_i \leq C$, $1 \leq j \leq k$

**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.)

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.

  \begin{itemize}
  \item maximal cliques: \{a, b\}, \{a, c, d\}
  \item \{c, d, e, f\}
  \end{itemize}

  maximum clique: \{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. $\exists S, \exists |S| \leq K$ ?

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

Hamiltomian 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 $\propto$ TSP

- 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)
0/1 knapsack problem

- **Def:** n objects, each with a weight $w_i > 0$
  - a profit $p_i > 0$
  - capacity of knapsack : $M$
  - Maximize $\sum_{i=1}^{n} p_i x_i$
  - Subject to $\sum_{i=1}^{n} w_i x_i \leq M$
  - $x_i = 0$ or $1$, $1 \leq i \leq n$
- **Decision version:**
  - Given $K$, \( \exists \sum_{i=1}^{n} x_i \geq K \) ?
- **Knapsack problem:** $0 \leq x_i \leq 1$, $1 \leq i \leq n$.

<Theorem> partition \( \approx \) 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.