Unit 9. $\mathcal{NP}$-complete Problems

Algorithms

EE3980

May 27, 2019

Algorithm Time Complexities

- Time complexity of an algorithm depicts the execution time as a function of the input size.
  - It is desirable to have the time complexity as a polynomial of the input size with a small degree.
    - $\mathcal{O}(n)$, $\mathcal{O}(n \lg n)$, $\mathcal{O}(n^2)$
  - For some problems the algorithms have been found are not polynomials.
    - For example, the traveling salesperson problem and 0/1 knapsack problem.
    - $\mathcal{O}(n^2 2^n)$, $\mathcal{O}(2^{n/2})$.
    - These problems can have extreme long execution time for a moderate size problem.
- The goal of the unit is to identify those problems that have no known algorithms with polynomial time complexity.

Nondeterministic Algorithms

- The algorithms described so far can always be executed with exact results – deterministic algorithms.
- A different class of algorithms, nondeterministic algorithms, allow the execution results to be not uniquely defined.
  - Three extra functions as following
    1. `Choice`(S): chooses one of the elements of set S arbitrarily.
    2. `Failure`$()$: signals an unsuccessful completion.
    3. `Success`$()$: signals an successful completion.
  - All three functions can be execute efficiently, i.e., $\mathcal{O}(1)$.
- Example
  - $x :=$ `Choice`$(1, n)$
  - x is assigned with an integer in the range $[1, n]$.

Nondeterministic Algorithms — Example

- Example: Nondeterministic search
  Given an array $A[1 : n]$ with n integers, the following algorithm will find the index j such that $A[j] = x$ or $j = 0$ if $x \notin A$.

Algorithm 9.1.1. Nondeterministic Search

```
   // A nondeterministic search algorithm.
   // Input: A with n elements, x; Output: j, A[j] = x, or 0 if x cannot be found.
Algorithm NDSearch(A, n, x)
{
      j := Choice (1, n);
      if (A[j] = x) then { write (j); Success (); }
      write (0); Failure ();
}
```

- It is assumed that the nondeterministic algorithm `NDSearch`(A, n, x) can find the correct index j such that $A[j] = x$ or 0 if no such x in $A[1 : n]$.
- And it takes $\mathcal{O}(1)$ time to execute.
- As compared to the deterministic algorithm that has time complexity of $\mathcal{O}(n)$.
- It can be assumed there are n processors to make choices then one of them will succeed.

Nondeterministic Algorithms — Example, II

- Nondeterministic sort algorithm:
  Given an n-integer array A, the following algorithm sorts A into a nondecreasing order.

Algorithm 9.1.2. Nondeterministic Sort

```
   // Sort n positive integers.
   // Input: Array A of n positive integers; Output: A in nondecreasing order.
1  Algorithm NDSort(A, n)
2  {
3      for i := 1 to n do B[i] := 0; // initialize B array.
4      for i := 1 to n do {
5          j := Choice (1, n);
6          if (B[j] ≠ 0) Failure (); // Repeated assignment.
7          B[j] := A[i];
8      }
9      for i := 1 to n − 1 do // Verify order.
10         if (B[i] > B[i + 1]) then Failure ();
11     write (B[1 : n]);
12     Success ();
13 }
```


Nondeterministic Algorithms — Example, III

- Note that an auxiliary array B is used.
- If the for loop on lines 4-8 is successfully executed, array B is a permutation of array A.
- Lines 9, 10 check if a nondecreasing order is achieved.
  If so, the sorting is done.
- The time complexity of NDSort algorithm is $\mathcal{O}(n)$.
  - As compared to $\mathcal{O}(n \lg n)$ in the deterministic case.
- There is no programming language or computer that can implement or execute the nondeterministic algorithms.
- The nondeterministic algorithms are tools for theoretical study in computer science.
- The primary objective of nondeterministic algorithm is whether an algorithm can result in a success
  – Verification Algorithms.

Decision and Optimization Problems

Definition. 9.1.3.

1. Any problem for which the answer is either one or zero (true or false) is called a decision problem.
2. An algorithm for a decision problem is termed a decision algorithm.
3. Any problem that involves the identification of an optimal (either minimum or maximum) value of a given cost function is known as an optimization problem.
4. An optimization algorithm is used to solve an optimization problem.

- The nondeterministic algorithms are mostly for studying decision problems.
- Though there might be many failures when a nondeterministic algorithm executes, the concern is whether a success can be achieved.
- If a decision problem can be solved in polynomial time, then the corresponding optimization problem can solve in polynomial time, too.
- On the other hand, if a optimization problem cannot be solved in polynomial time, then the corresponding decision problem cannot be solved in polynomial time, either.

Decision and Optimization Problems — Example

- Example: Maximum Clique Problem.
  - A maximal complete subgraph of a graph $G(V, E)$ is a clique.
  - The size of a clique is the number of vertices in the clique.
  - The maximum clique problem is an optimization problem that is to determine the largest clique in G.
  - The corresponding decision problem is to determine whether G has a clique of size at least k for some given k.
  - Let `DClique`(G, k) be the deterministic algorithm for the decision problem.
  - If the number of vertices in G is n, then the optimization problem can be solved by applying `DClique` repeatedly for different k, $k = n, n-1, \cdots$, until the output of `DClique` is 1.
  - If the time complexity of `DClique` is $f(n)$ then the optimization problem has the complexity less than or equal to $n \cdot f(n)$.
  - On the other hand, if the optimization problem can be solved in $g(n)$ time, then the decision problem can be solved in time $\leq g(n)$.
  - If the decision problem can be solved in polynomial time, then the optimization problem can also be solved in polynomial time.
  - If the optimization problem cannot be solved in polynomial time, then the corresponding decision problem cannot be solved in polynomial time, either.

Nondeterministic Algorithm Time Complexity

Definition 9.1.4.

The time required by a nondeterministic algorithm performing on any given input is the minimum number of steps needed to reach a successful completion if there exists a sequence of choices leading to such a completion. In case a successful completion is not possible, then the time required is $\mathcal{O}(1)$. A nondeterministic algorithm is of complexity $\mathcal{O}(f(n))$ if for all inputs of size n, $n \geq n_0$, that result in a successful completion, the time required is at most $c \cdot f(n)$ for some constants c and n_0.

- Note the difference to the time complexity of a deterministic algorithm.

Nondeterministic Algorithm Time Complexity Example

- Given n objects with profits $p[1:n]$ and weights $w[1:n]$, and numbers m and r, the following nondeterministic algorithm determined if there is an assignment $x[1:n]$, $x[i] = 0$ or 1, $1 \leq i \leq n$, such that

$$\sum_{i=1}^{n} x[i] \cdot p[i] \geq r \quad \text{and} \quad \sum_{i=1}^{n} x[i] \cdot w[i] \leq m.$$

Algorithm 9.1.5. 0/1 Knapsack Decision Algorithm.

```
   // Nondeterministic algorithm to solve 0/1 knapsack problem.
   // Input: p, w, n, m; Output: true if solution x exist, false otherwise.
Algorithm NDKP(p, w, n, m, r, x)
{
    W := 0; P := 0;
    for i := 1 to n do {
        x[i] := Choice (0,1); // assign x[i]
        W := W + x[i]× w[i]; P := P + x[i]× p[i];
    }
    if ((W > m) or (P < r)) then Failure ();
    else Success ();
}
```

- The time complexity of a successful completion of this algorithm is $\mathcal{O}(n)$.

Nondeterministic Algorithm Time Complexity Example, II

- Given a graph $G(V, E)$ with n vertices, the following algorithm determines if there is a clique of size k in G.

Algorithm 9.1.6. Nondeterministic Graph Clique Decision

```
   // To determine if G(V,E) contains a clique of size k.
   // Input: G(V,E), n, k; Output: true if yes, false otherwise.
1  Algorithm NDCK(V, E, n, k)
2  {
3        S := ∅ ; // initialize S to be empty set.
4        for i := 1 to k do { // find k distinct vertices
5              t := Choice (1, n) ;
6              if (t ∈ S) then Failure () ;
7              S := S ∪ {t} ; // Add t to set S.
8        }
9        for all (i, j) such that i, j ∈ S and i ≠ j do
10             if (i, j) ∉ E then Failure () ;
11       Success () ;
12 }
```

- Time complexity is dominated by the for loop on lines 9,10, $\mathcal{O}(k^2) \leq \mathcal{O}(n^2)$.
- There is no known polynomial time algorithm for the deterministic graph clique decision problem.

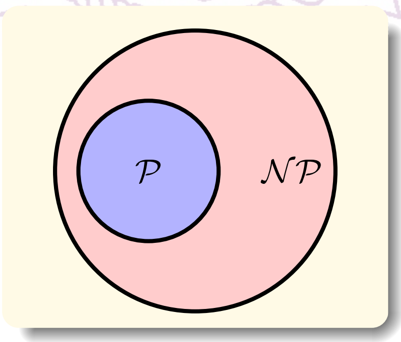
$\mathcal{P}$ and $\mathcal{NP}$

- An algorithm A is of polynomial complexity if there exists a polynomial p such that the computing time of A is $\mathcal{O}(p(n))$ for every input of size n.

Definition 9.1.7. $\mathcal{P}$ and $\mathcal{NP}$

$\mathcal{P}$ is the set of all decision problems solvable by deterministic algorithms in polynomial time. $\mathcal{NP}$ is the set of all decision problems solvable by nondeterministic algorithms in polynomial time.

- Since deterministic algorithms are special cases of nondeterministic algorithms, we have $\mathcal{P} \subseteq \mathcal{NP}$.
- It is not known which of the following is true: $\mathcal{P} = \mathcal{NP}$ or $\mathcal{P} \neq \mathcal{NP}$.
- The common belief of their relationship is shown below



Polynomial Time Transformation (Reducibility)

- Given two problems Q_1 and Q_2, if there is a polynomial time transformation such that Q_1 can be transformed into Q_2 we say that Q_1 transforms to Q_2 and denotes $Q_1 \propto Q_2$.
  - It is also commonly referred as Q_1 reduces to Q_2.
- Given the polynomial transformation $Q_1 \propto Q_2$, if Q_2 can be solved in polynomial time, then Q_1 can be solved in polynomial time as well.

Lemma 9.1.8.

If $Q_1 \propto Q_2$, then if $Q_2 \in \mathcal{P}$ then $Q_1 \in \mathcal{P}$ (and, equivalently, $Q_1 \notin \mathcal{P}$ then $Q_2 \notin \mathcal{P}$).

Lemma 9.1.9.

If $Q_1 \propto Q_2$ and $Q_2 \propto Q_3$, then $Q_1 \propto Q_3$.

$\mathcal{NP}$-complete

- A problem Q is said to be $\mathcal{NP}$-complete if $Q \in \mathcal{NP}$ and for all other $Q' \in \mathcal{NP}$, $Q' \propto Q$.
  - Thus, the $\mathcal{NP}$-complete problems are the hardest problems in $\mathcal{NP}$.
  - If any one can be solved in polynomial time, then all problems in $\mathcal{NP}$ can be solved in polynomial time.

Lemma 9.1.10.

If Q_1 and Q_2 belong to $\mathcal{NP}$, if Q_1 is $\mathcal{NP}$-complete and $Q_1 \propto Q_2$ then Q_2 is $\mathcal{NP}$-complete.

Definition 9.1.11. Polynomial equivalency.

Two problems Q_1 and Q_2 are said to be polynomial equivalent if and only if $Q_1 \propto Q_2$ and $Q_2 \propto Q_1$.

- To show a problem Q_2 is $\mathcal{NP}$-complete, it is adequate to show $Q_1 \propto Q_2$, where Q_1 is a problem already known to be $\mathcal{NP}$-complete.

Satisfiability Problem

- Let $x_1, x_2, \cdots, x_n$ be boolean variables such that x_i can be either *true* or *false*.
- Let $\overline{x_i}$ denote the negation of x_i.
- A literal is either a boolean variable or its negation.
- A formula in the propositional calculus is an expression that can be constructed using literals and the operators and and or.
- Examples of formulas

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

  The symbol $\vee$ denotes or and $\wedge$ denotes and.
- A formula is in conjunctive normal form (CNF) if and only if it is represented as $\bigwedge_{i=1}^{k} c_i$, where c_i are clauses each represented as $\bigvee l_{ij}$. The l_{ij} are literals.
  - Example of CNF: $(x_3 \vee \overline{x_4}) \wedge (x_1 \vee \overline{x_2})$.
- A formula is in disjunctive normal form if and only if it is represented as $\bigvee_{i=1}^{k} c_i$ and each clause c_i is represented as $\bigwedge l_{i,j}$.
  - Example of DNF: $(x_1 \wedge x_2) \vee (x_3 \wedge \overline{x_4})$.

Satisfiability Problem, II

- The satisfiability problem is to determine whether a formula is true for some assignment of truth values to the variables.
- The CNF-satisfiability is the satisfiability problem for CNF formula.
- Given an expression E and n boolean variables represented by the array $x[1:n] = (x_1, x_2, \cdots, x_n)$, the following nondeterministic algorithm find a set of truth value assignments that satisfies E, that is, $E(x_1, x_2, \cdots, x_n) =$ `true`.

Algorithm 9.1.12. Nondeterministic Satisfiability.

```
  // Nondeterministic algorithm for satisfiability problem.
  // Inputt: expression E, n; Output: true if E(x) = 1, false otherwise.
1 Algorithm NSat(E, n, x)
2 {
3     for i := 1 to n do // Choose a truth value assignment.
4         x[i] := Choice ( false , true );
5     if E(x) then Success ();
6     else Failure ();
7 }
```

- The time complexity is $\mathcal{O}(n)$ (`for` loop on lines 4-5) plus the time to evaluation expression E.

Cook's Theorem

- It is known from Algorithm (9.1.12) that the satisfiability decision problem is in $\mathcal{NP}$, and we have the following theorem by Cook.

Theorem 9.1.13. Cook's Theorem.

Satisfiability is in $\mathcal{P}$ if and only if $\mathcal{P} = \mathcal{NP}$.

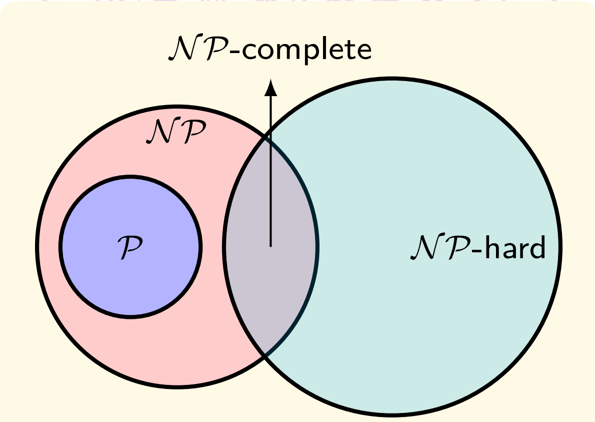
- Proof please see textbook [Horowitz], pp. 527-535, or [Cormen] pp. 1074-1077.
- S.A. Cook, "The complexity of theorem proving procedures." In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, pp. 151-158, 1971.
- In other words, satisfiability problem is $\mathcal{NP}$-complete.
- This is the first known $\mathcal{NP}$-complete problem.
  - Then others can be reduced from it.

$\mathcal{NP}$-Hard and $\mathcal{NP}$-Complete

Definition. 9.1.14. $\mathcal{NP}$-hard and $\mathcal{NP}$-complete.

A problem Q is $\mathcal{NP}$-hard if and only if satisfiability reduces to Q (satisfiability $\propto Q$). A problem Q is $\mathcal{NP}$-complete if and only if Q is $\mathcal{NP}$-hard and $Q \in \mathcal{NP}$.

- There are $\mathcal{NP}$-hard problems that are not $\mathcal{NP}$-complete.
- Only a decision problem can be $\mathcal{NP}$-complete.
- If Q_1 is a decision problem and Q_2 is the corresponding optimization problem, then it is quite possible that $Q_1 \propto Q_2$.
- An $\mathcal{NP}$-complete decision problem may have its corresponding optimization problem be $\mathcal{NP}$-hard.
- There are also decision problems that are $\mathcal{NP}$-hard.



3-Satisfiability Problem (3-SAT)

- 3-satisfiability problem is a special case of the CNF-satisfiability problem, where each clause has exactly three literals.
- A clause, C_k, of k literals can be converted into a CNF of 3 literals, C'_k, as the following. (y_i's are auxiliary variables.)

$$
\begin{aligned}
k = 1, \quad & C_1 = x_1, \\
& C'_1 = (x_1 \vee y_1 \vee y_2) \wedge (x_1 \vee \overline{y}_1 \vee y_2) \wedge (x_1 \vee y_1 \vee \overline{y}_2) \wedge (x_1 \vee \overline{y}_1 \vee \overline{y}_2), \\
k = 2, \quad & C_2 = x_1 \vee x_2, \\
& C'_2 = (x_1 \vee x_2 \vee y_1) \wedge (x_1 \vee x_2 \vee \overline{y}_1), \\
k = 3, \quad & C_3 = x_1 \vee x_2 \vee x_3, \\
& C'_3 = x_1 \vee x_2 \vee x_3, \\
k > 3, \quad & C_k = x_1 \vee x_2 \vee \cdots \vee x_k, \\
& C'_k = (x_1 \vee x_2 \vee y_1) \wedge (\overline{y}_1 \vee x_3 \vee y_2) \wedge \cdots \wedge (\overline{y}_{k-3} \vee x_{k-1} \vee x_k).
\end{aligned}
$$

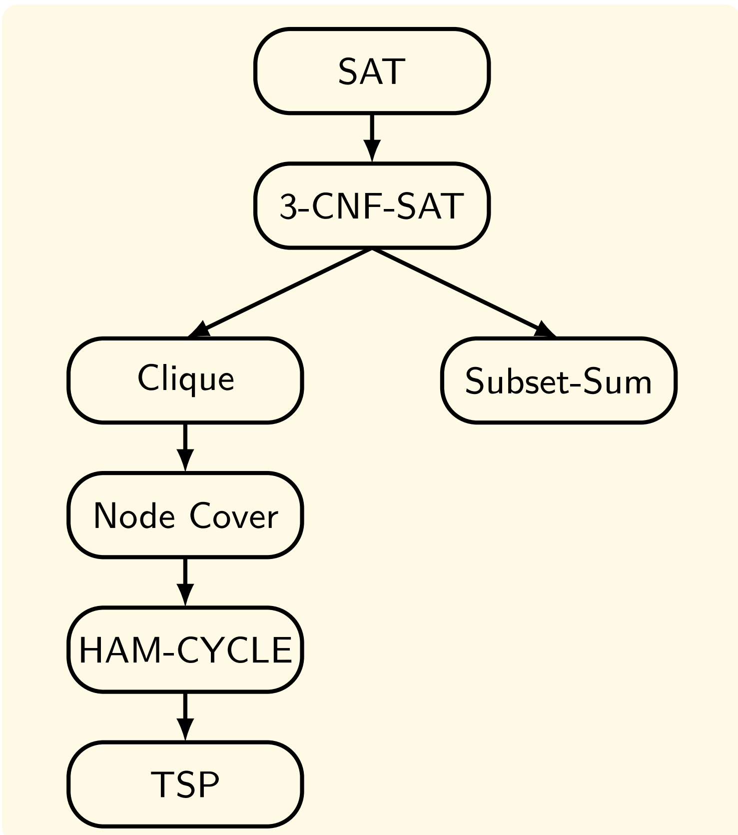
Theorem 9.1.15. 3-SAT

CNF-satisfiability problem $\propto$ 3-satisfiability problem.

- Thus, 3-satisfiability problem is $\mathcal{NP}$-complete.

Finding Other $\mathcal{NP}$-Complete Problems

- From the Satisfiability problem, more $\mathcal{NP}$-complete problems were identified.



Clique Decision Problem (CDP)

- A graph clique decision problem (CDP) is given a graph $G(V, E)$ to decide if there are cliques of size k in G.
- CDP is $\mathcal{NP}$-complete.

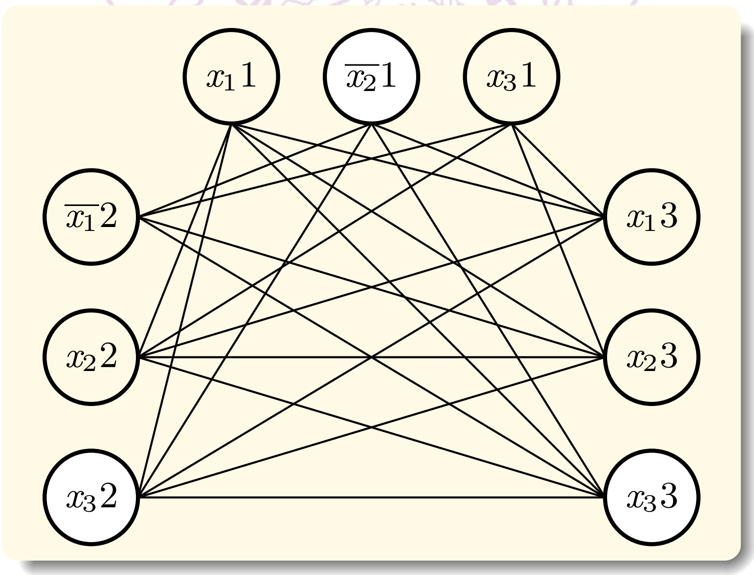
Theorem 9.1.16. CDF

CNF-satisfiability $\propto$ clique decision problem.

- Let $F = \bigwedge_{i=1}^{k} C_i$ be a propositional formula in CNF.
  - Let $x_i, 1 \leq i \leq n$ be a variable in F.
- Define $G = (V, E)$ as follows:
  - $V = \{\langle\sigma, i\rangle | \sigma$ is a literal in clause $C_i\}$.
  - $E = \{(\langle\sigma, i\rangle, \langle\delta, j\rangle) | i \neq j$ and $\sigma \neq \bar{\delta}\}$.
- The F is satisfiable if and only if G has a clique of size k.
- If the length of F is m, the sum variables of each clause, then G is obtainable from F in $\mathcal{O}(m^2)$ time.

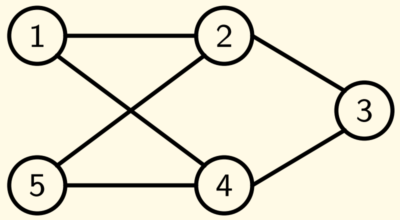
Clique Decision Problem (CDP), II

- Q_1: 3-Satisfiability.
  $\mathcal{I} = (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$.
- Q_2: Clique Decision problem.
  $G(V_{\mathcal{I}}, E_{\mathcal{I}})$ has a clique of size 3?

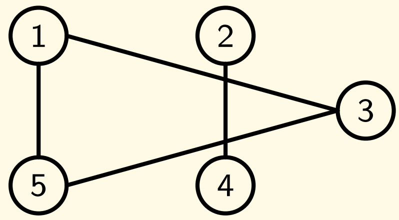


Node Cover Decision Problem (NCDP)

- A set $S \subseteq V$ is a node cover for a graph $G(V, E)$ if and only if all edges in E are incident to at least one vertex in S. The size $|S|$ of the cover is the number of vertices in S.
- The node cover decision problem is given a graph $G(V, E)$ and an integer k to determine if there is a node cover of size at most k.
- Example: Given a graph shown below.
  - $S_1 = \{2, 4\}$ is a node cover of size 2.
  - $S_2 = \{1, 3, 5\}$ is a node cover of size 3.



Graph G.



Graph G'.

Node Cover Decision Problem (NCDP), II

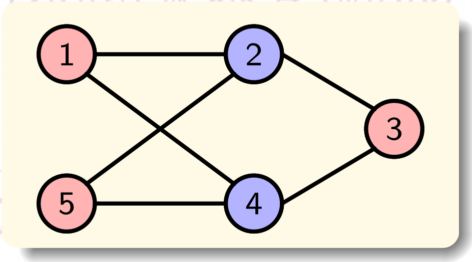
Theorem 9.1.17. NCDP

The clique decision problem $\propto$ the node cover decision problem.

- Given a $G(V, E)$ and an integer k, and instance of clique decision problem is defined. Assume that $|V| = n$.
- Construct a graph $G'(V, E')$, where $E' = \{(u, v) | u \in V, v \in V$ and $(u, v) \notin E\}$.
- This graph G' is known as the complement of G.
- If K is a clique in G, since there are no edges in E' connecting vertices in K, the remaining $n - |K|$ vertices in G' must cover all edges in E'.
- Thus if G has a clique of size at least k if and only if G' has a node cover of size at most $n - k$.
- Note that G' can be constructed from G in $\mathcal{O}(n^2)$ time, thus theorem is proved.
- Note also that since CNF-satisfiability $\propto$ CDP, and CDP $\propto$ NCDP, therefore NCDP is $\mathcal{NP}$-hard.
- NCDP is also $\mathcal{NP}$, so NCDP is $\mathcal{NP}$-complete.

Chromatic Number Decision Problem (CNDP)

- A coloring of a graph $G(V, E)$ is a function $f\colon V \to \{1, 2, \ldots, k\}$ defined for all $i \in V$. If $(u, v) \in E$, then $f(u) \neq f(v)$.
- The chromatic number decision problem is to determine whether G has a coloring for a given k.
- Example: a two-coloring graph.



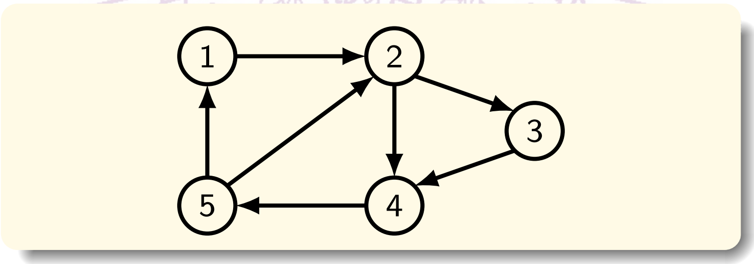
Theorem 9.1.18. CNDP

3-satisfiability problem $\propto$ chromatic number decision problem.

- Proof see textbook [Horowitz], pp. 540-541.

Directed Hamiltonian Cycle (DHC) Problem

- A directed Hamiltonian cycle in a directed graph $G(V, E)$ is a directed cycle of length $n = |V|$.
- The directed Hamiltonian cycle goes through every vertex exactly once and returns to the starting vertex.
- The DHC problem is to determine whether G has a directed Hamiltonian cycle.
- Example: $(1, 2, 3, 4, 5, 1)$ is a Hamiltonian cycle.



Theorem 9.1.19. DHC

CNF-satisfiability $\propto$ directed Hamiltonian cycle.

- Directed Hamiltonian cycle problem is $\mathcal{NP}$-complete.
- Proof please see textbook [Horowitz], pp. 542-545, or [Cormen], pp. 1091-1096 (for undirected graph).

Traveling Salesperson Decision Problem (TSP)

- The traveling salesperson decision problem (TSP) is to determine whether a complete directed graph $G(V, E)$ with edge cost $c(u, v)$, $u, v \in V$, has a tour of cost at most M.

Theorem 9.1.20. TSP

Directed Hamiltonian cycle (DHC) $\propto$ the traveling salesperson decision problem (TSP).

- Given a directed graph $G(V, E)$ for the DHC problem, construct a complete directed graph $G'(V, E')$, $E' = \{\langle i, j\rangle | i \neq j\}$ and $c(i, j) = 1$ if $\langle i, j\rangle \in E$; $c(i, j) = 2$ if $i \neq j$ and $\langle i, j\rangle \notin E$. In this case, G' has a tour of cost at most n if and only if G has a directed Hamiltonian cycle.
- TSP is an $\mathcal{NP}$-completeproblem.
- Both Hamiltonian Cycle and Travelling Salesperson Problem can be defined for undirected graph as well.
- Both undirected Hamiltonian Cycle and Travelling salesperson Problem are also $\mathcal{NP}$-complete.
- Proof please see textbook [Cormen], pp. 1091-1097.

Partition Problem

- Given a set $A = \{a_1, a_2, \cdots, a_n\}$ of n integers. The partition problem is to determine whether there is a partition P such that

$$\sum_{i \in P} a_i = \sum_{i \notin P} a_i.$$

Theorem 9.1.21. Partition Problem.

3-satisfiability problem $\propto$ partition problem.

- Proof see Garey and Johnson, *Computers and Intractability*, Freeman, 1979, p. 60.
- Thus, partition problem is a $\mathcal{NP}$-complete problem.

Sum of Subsets Problem

- Given a set $A = \{a_1, a_2, \cdots, a_n\}$ of n integers and an integer M. The sum of subsets problem is to determine whether there is a subset $S \subseteq A$ such that

$$\sum_{a_i \in S} a_i = M.$$

- Given the n-integer set A, an $n+2$ set B can be constructed as

$$b_i = a_i, \qquad 1 \le i \le n,$$
$$b_{n+1} = M + 1,$$
$$b_{n+2} = \left(\sum_{i=1}^{n} a_i\right) - M + 1,$$

Then $$b_{n+2} + \sum_{b_i \in S} b_i = b_{n+1} + \sum_{b_i \notin S} b_i.$$

The partition problem in B is equivalent to the sum of subsets problem in A.

Theorem 9.1.22. Sum of subsets.

Sum of subsets problem $\propto$ partition problem.

- The sum of subsets problem is $\mathcal{NP}$-complete.

Scheduling Identical Processors Problems

- Let P_i, $1 \le i \le m$, be m identical processors.
- Let J_i, $1 \le i \le n$, be n jobs. Each job J_i requires t_i processing time.
- A schedule S is an assignment of jobs to processors. For each job J_i, S specifies the time interval and the processor that processes J_i.
  - A job cannot be processed by more than one processor at any given time.
- Let f_i be the time at which job J_i complete processing. The mean finish time (MFT) of schedule S is

$$\text{MFT}(S) = \frac{1}{n}\sum_{i=1}^{n} f_i. \qquad (9.1.1)$$

- Let w_i be a weight associated with each job J_i. The weighted mean finish time (WMFT) of schedule S is

$$\text{WMFT}(S) = \frac{1}{n}\sum_{i=1}^{n} w_i \cdot f_i. \qquad (9.1.2)$$

- Let T_i be the time at which P_i finishes processing all jobs assigned to it. The finish time (FT) of schedule S is

$$\text{FT}(S) = \max_{i=1}^{m} T_i. \qquad (9.1.3)$$

- Schedule S is a nonpreemptive schedule if and only if each job J_i is processed continuously from start to finish on the same processor. Otherwise, it is preemptive.

Scheduling Problems – Complexities

Theorem 9.1.23. MFT

Partition problem $\propto$ minimum finish time nonpreemptive schedule problem.

- For $m = 2$ case, given the set $\{a_1, a_2, \cdots, a_n\}$ as an instance of the partition problem. Define n jobs with processing time $t_i = a_i$, $1 \le i \le n$. There is a nonpreemptive schedule for this set of jobs on two processors with finish time at most $\sum t_i/2$ if and only if there is a partition of the set $\{a_i | 1 \le i \le n\}$. It can also be proved for $m > 2$ cases.

Theorem 9.1.24. WMFT

Partition problem $\propto$ minimum WMFT nonpreemptive schedule problem.

- For $m = 2$ case, given the set $\{a_1, a_2, \cdots, a_n\}$ define a two-processor scheduling problem with $w_i = t_i = a_i$. Then there is a nonpreemptive schedule S with weighted mean finish time at most $1/2 \sum a_i^2 + 1/4(\sum a_i)^2$ if and only if the set $\{a_i | 1 \le i \le n\}$ has a partition.
The rest of the proof please see textbook [Horowitz], pp. 554-555.

Scheduling Problems – Complexities, II

Theorem 9.1.25. Flow Shop Scheduling

Partition problem $\propto$ the minimum finish time preemptive flow shop schedule with $m > 2$. (m is the number of processors.)

- Proof please see textbook [Horowitz], pp. 555-556.

Theorem 9.1.26. 2-processor Flow Shop Scheduling

2-processor flow shop schedule $\in \mathcal{P}$.

- Dynamic programming approach can solve this problem in polynomial time. Please see textbook [Horowitz], pp. 321-325.

Theorem 9.1.27. Job Shop Scheduling

Partition problem $\propto$ the minimum finish time preemptive job shop schedule with $m > 1$. (m is the number of processors.)

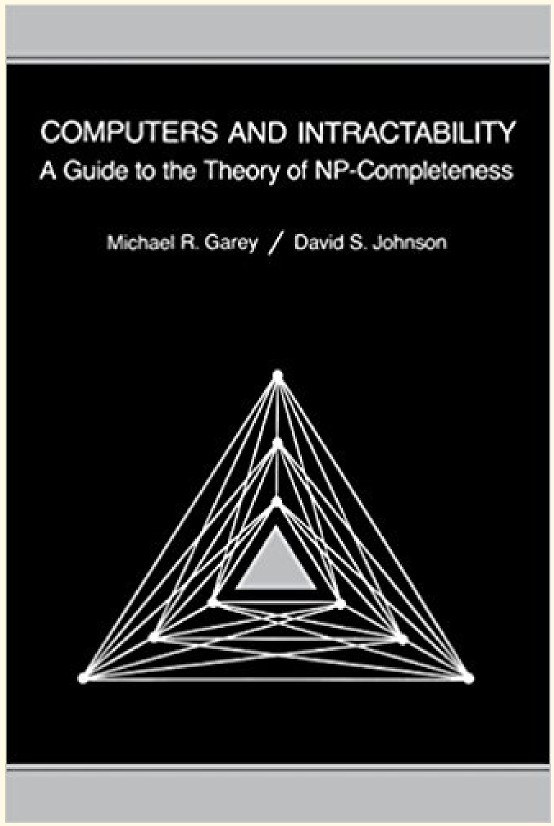
- Proof please see textbook [Horowitz], pp. 557-558.

Other $\mathcal{NP}$-complete Problems

- Since 1971, many $\mathcal{NP}$-complete problems have been found.
- A good source book is

  M.R. Garey and D.S. Johnson,
  Computers and Intractability – A Guide to the Theory of NP-Completeness,
  W.H. Freeman, 1979.

- More than 320 $\mathcal{NP}$-complete problems listed in its reference, pp. 190-288.



2-SAT Problem

- It has been shown that Satisfiability (SAT) and 3-SAT problems are $\mathcal{NP}$-complete.
- In the following we study 2-SAT problem.
- 2-SAT problem is also a special case of SAT problem. In this problem, each clause has exactly two literals.
- Example

$$F(x_1, x_2, x_3, x_4) = (x_1 \vee x_2) \wedge (x_2 \vee \overline{x_3}) \wedge (\overline{x_2} \vee \overline{x_4}) \wedge (x_2 \vee x_4) \wedge (x_4 \vee x_1).$$

  - Given formula shown above, is it satisfiable? That is, can one set $x_i =$`true` or $x_i =$`false` for each x_i such that the formula is evaluated to be `true`.

2-SAT Problem, II

- Example

$$F(x_1, x_2, x_3, x_4) = (x_1 \vee x_2) \wedge (x_2 \vee \overline{x_3}) \wedge (\overline{x_2} \vee \overline{x_4}) \wedge (x_2 \vee x_4) \wedge (x_4 \vee x_1).$$

- In evaluating $F(x_1, x_2, x_3, x_4)$, one can set $x_2 = 1$ (`true`), then $\overline{x_2} = 0$ (`false`) and the formula becomes

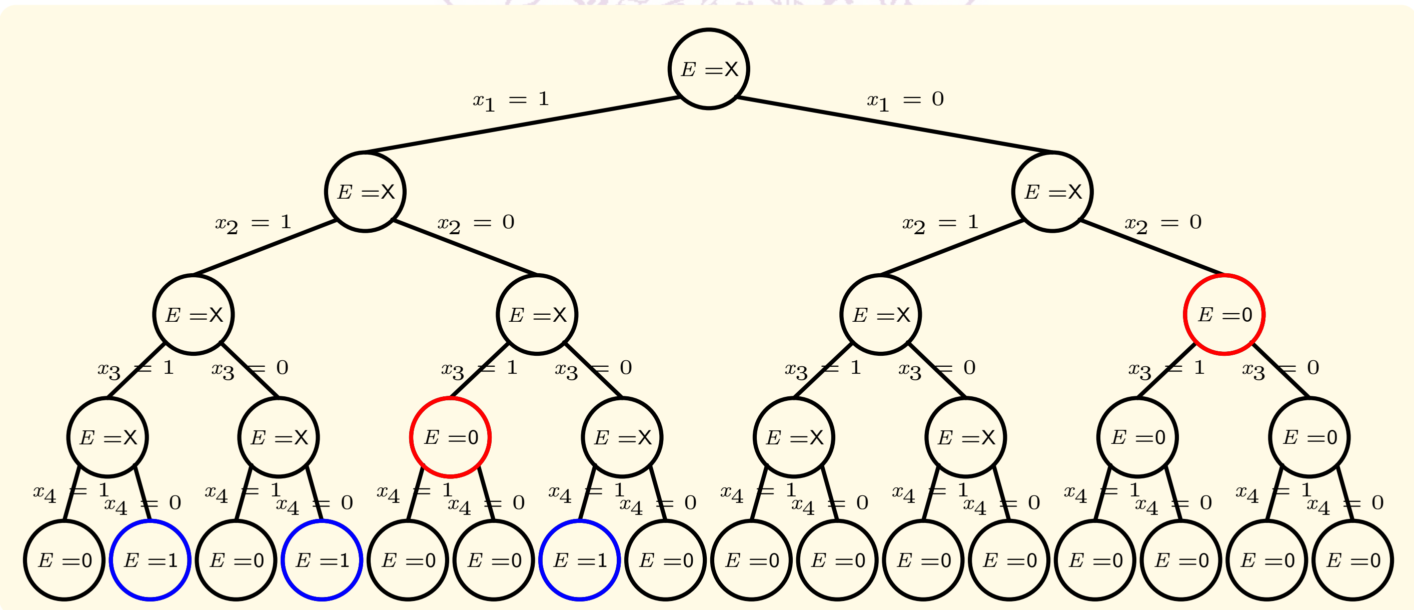
$$F(x_1, x_2 = 1, x_3, x_4) = (\overline{x_4}) \wedge (x_4 \vee x_1).$$

- Three clauses, $(x_1 \vee x_2)$, $(x_2 \vee \overline{x_3})$, and $(x_2 \vee x_4)$, become `true`, and thus can be eliminated from the formula.
- The clause $(\overline{x_2} \vee \overline{x_4})$ reduces to $(\overline{x_4})$ since $\overline{x_2} = 0$.
- In order $F(x_1, x_2, x_3, x_4) = 1$, one must have $x_4 = 0$ and $x_1 = 1$.
- The value of x_3 does not impact F and can be either 0 or 1 (don't care).
- This shows that $F(x_1, x_2, x_3, x_4)$ is satisfiable with $(x_1, x_2, x_3, x_4) = (1, 1, \times, 0)$.

2-SAT Problem, III

$$F(x_1, x_2, x_3, x_4) = (x_1 \vee x_2) \wedge (x_2 \vee \overline{x_3}) \wedge (\overline{x_2} \vee \overline{x_4}) \wedge (x_2 \vee x_4) \wedge (x_4 \vee x_1).$$

- The complete state space for the formula
  - Backtracking or branch-and-bound can be used to find the answer.
  - $\mathcal{O}(2^n)$, n is the number of boolean variables.



2-SAT Problem – Implicative Form

- In propositional calculus, the following two simple formulas are equivalent.

$$\begin{aligned} F_1 &= x_1 \vee x_2 \\ F_2 &= \overline{x_1} \rightarrow x_2 \end{aligned} \tag{9.1.4}$$

- Since $x_1 \vee x_2 = x_2 \vee x_1$, the following three are equivalent

$$\begin{aligned} F_1 &= x_1 \vee x_2 \\ F_2 &= \overline{x_1} \rightarrow x_2 \\ F_3 &= \overline{x_2} \rightarrow x_1 \end{aligned} \tag{9.1.5}$$

- It is easy to see the followings.

$$F_4 = x_1 \rightarrow x_1 \equiv \overline{x_1} \vee x_1 = \texttt{true}, \tag{9.1.6}$$

$$F_5 = \overline{x_1} \rightarrow \overline{x_1} \equiv x_1 \vee \overline{x_1} = \texttt{true}. \tag{9.1.7}$$

Yet,

$$F_6 = x_1 \rightarrow \overline{x_1} \equiv \overline{x_1} \vee \overline{x_1} \tag{9.1.8}$$

$$F_7 = \overline{x_1} \rightarrow x_1 \equiv x_1 \vee x_1 \tag{9.1.9}$$

F_6 can be `true` if $x_1 =$`false`, and F_7 can be `true` if $x_1 =$`true`.

2-SAT Problem – Implicative Form, II

- But,

$$\begin{aligned} F_8 &= (x_1 \rightarrow \overline{x_1}) \wedge (\overline{x_1} \rightarrow x_1) \\ &\equiv (\overline{x_1} \vee \overline{x_1}) \wedge (x_1 \vee x_1) \\ &= \overline{x_1} \wedge x_1 = \texttt{false}. \end{aligned} \tag{9.1.10}$$

- Using this equivalent relationship, the formulas in conjunctive normal form can be easily translated to the implicative form.

$$\begin{aligned} F(x_1, x_2, x_3, x_4) =& (x_1 \vee x_2) \wedge (x_2 \vee \overline{x_3}) \wedge (\overline{x_2} \vee \overline{x_4}) \wedge (x_2 \vee x_4) \wedge (x_4 \vee x_1) \\ \equiv& (\overline{x_1} \rightarrow x_2) \wedge (\overline{x_2} \rightarrow \overline{x_3}) \wedge (x_2 \rightarrow \overline{x_4}) \wedge (\overline{x_2} \rightarrow x_4) \wedge (\overline{x_4} \rightarrow x_1) \wedge \\ & (\overline{x_2} \rightarrow x_1) \wedge (x_3 \rightarrow x_2) \wedge (x_4 \rightarrow \overline{x_2}) \wedge (\overline{x_4} \rightarrow x_2) \wedge (\overline{x_1} \rightarrow x_4). \end{aligned}$$

- And, a directed graph $G(V, E)$ can be constructed from the conjunctive normal form $(F(x_1, x_2, \ldots, x_n) = \bigwedge_{j=1}^{m} (x_i \vee x_j))$.

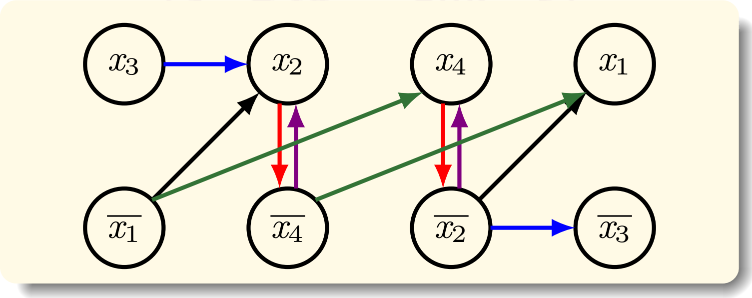
$$\begin{aligned} V &= \{y_i \,|\, y_i = x_i \text{ or } y_i = \overline{x_i}, i = 1, \ldots, n\}, \\ E &= \{(\overline{y_i}, y_j)(\overline{y_j}, y_i) \,|\, (y_i \vee y_j) \text{ is one clause in } F\}. \end{aligned}$$

Note that $|V| = 2n$ and $|E| = 2m$, where n is the number of variables and m is the number of clauses in F.

Implicative Graph

- Given the formula, the following graph is constructed.
  - Two strongly connected components, $\{x_2, \overline{x_4}\}$ and $\{\overline{x_2}, x_4\}$ can be observed.

$$F(x_1, x_2, x_3, x_4) = (x_1 \vee x_2) \wedge (x_2 \vee \overline{x_3}) \wedge (\overline{x_2} \vee \overline{x_4}) \wedge (x_2 \vee x_4) \wedge (x_4 \vee x_1)$$



Lemma 9.1.28. 2SAT

Given a formula $F(x_1, x_2, \ldots, x_n)$ and its implicative graph $G(V, E)$ then F is NOT satisfiable if and only if there is a strongly connected component in G that contains a boolean variable x_i and its complement $\overline{x_i}$.

- By the preceding lemma, the formula given above is satisfiable since those two strongly connected components contain no boolean variable together with its complement.

Solving 2-SAT Problems

- From the lemma, one can solve the 2-SAT problem by
  1. Construct the implicative graph, $G(V, E)$, of the formula $F(x_1, x_2, \ldots, x_n)$.
  2. Find all the strongly connected components, S_i, of $G(V, E)$.
  3. Check all the strongly connected components to see if any S_i contains both x_j and $\overline{x_j}$.
  4. If no such S_i and x_j exist, then $F(x_1, x_2, \ldots, x_n)$ is satisfiable; Otherwise, $F(x_1, x_2, \ldots, x_n)$ is not satisfiable.
- Note that
  1. $G(V, E)$ can be constructed in $\mathcal{O}(n + m)$ time, since $|V| = 2n$ and $|E| = 2m$. (n is the number of boolean variables and m is the number of clauses in F).
  2. The strongly connected graph can be find in $\mathcal{O}(|V| + |E|)$ time.
  3. Check for if both x_j and $\overline{x_j}$ are in S_i can be done in $\mathcal{O}(|S_i|)$ time.
  4. Thus, determine if $F(x_1, x_2, \ldots, x_n)$ is satisfiable can be done in $\mathcal{O}(n + m)$ time.

Lemma 9.1.29.

2-SAT $\in \mathcal{P}$.

Summary

- Nondeterministic algorithms
  - Examples
  - Complexity
- Decision and optimization problems
- Polynomial time transformation
- $\mathcal{P}$, $\mathcal{NP}$ and $\mathcal{NP}$-complete
- Satisfiability problem
- $\mathcal{NP}$-complete problems
  - 3-SAT
  - Graph clique problem
  - Node cover problem
  - Chromatic number problem
  - Hamiltonian cycle problem
  - Traveling salesperson problem
  - Partition problem
  - Sum of subsets problem
  - Scheduling identical processors problem
- 2-SAT problem