Chapter 9: Computational Complexity and intractability: An Introduction to the Theory of NP

Contents

9.1 Intractability

9.2 Input Size Revisited

9.3 The Three General Categories of Problems

9.4 The Theory of NP

- **9.4.1 The Sets P and NP**
- **9.4.2 NP-Complete Problems**
- **9.4.3 NP-Hard, NP-Easy, and NP-Equivalent Problems**

9.1 Intractability

- **Def.) polynomial-time algorithm**

  $$W(n) \in O(p(n))$$

  where n: input size, $p(n)$: polynomial in n

- **Ex. 9.1** $n,\ n\log n,\ n^k\ (k:\ \text{constant})$

  not polynomial: $2^n,\ 2^{\sqrt{n}},\ n!$

- **A problem is intractable if it is impossible to solve it with a polynomial-time algorithm.**

9.2 Input Size Revisited

- **Def.) Input size**
  - **The # of characters it takes to write the input**
- **Ex) sort n positive integers $\leq L$**

binary encoding $n\log_2 L$

decimal encoding $n\log_{10} L = \underbrace{\log_{10} 2}_{\text{constant}} \cdot n\log_2 L$

- **"reasonable encoding": not affect the determination of whether an algorithm is polynomial-time.**

Examples – Prime checking

```
bool prime(int n)
{
   int i; bool switch;
   switch = true;
   i = 2;
   while (switch && i <= n^(1/2))
      if(n % i == 0)
         // % returns the remainder
         // when n is divided by i.
         switch = false;
      else
         i++;
   return switch;
}
```

- Input size $\log n = r$
- Magnitude n
- Complexity $\Theta(2^r) = \Theta(n)$ $\rightarrow$ not linear

- n-th Fibonacci number (Algorithm 1.7)

  Same as prime checking

```
int fib2 (int n)
{
   index i;
   int f[0..n];
   f[0] = 0;
   if (n > 0) {
      f[1] = 1;
      for (i = 2; i <= n; i++)
         f[i] = f[i – 1] + f[i – 2];
   }
      return f[n];
}
```

Examples – 0/1 Knapsack

- **Complexity** $\Theta(nW)$
  - ***n*****: measure of size (# of items)**
  - ***W*****: magnitude (log *W* : size)**
  - **So polynomial in terms of magnitude and size but exponential in terms of size alone.**

- **Def.) Pseudopolynomial-time algorithm**
  - **An algorithm whose worst-case time complexity is bounded above by a polynomial function of its size and magnitude.**

9.3 The Categories of Problems

(1) Problems for which polynomial-time algorithms have been found.

(2) Intractable problems

- **Nonpolynomial amount of outputs**
  - **Ex) (n-1)! Hamiltonian circuits in K_n**
- **Undecidable problems**
  - **Ex) halting problems**
- **Decidable decision problems proven to be intractable**
  - **Presburger arithmetic**

(3) Problems proven to be neither intractable nor polynomially solved

- **Ex) 0/1 knapsack, TSP, sum-of-subsets, …**

9.4 The Theory of NP

- **Decision problem: A problem for which the answer is either zero (no) or one (yes) is called a decision problem.**
- **Ex) TSDP: determine for a given positive *d* whether there is a tour having total weight $\leq d$.**
- **Ex) 0/1 knapsack DP: determine whether there is a 0/1 assignment of values to x_i s.t. $\sum p_i x_i \geq P$ and $\sum w_i x_i \leq W$**

Examples – continued

- **Ex) Graph coloring DP**
  - **Chromatic number: min # of colors need to color a graph so that no two adjacent vertices are colored the same color.**
  - **Determine, for an integer *m*, whether there is a coloring that uses at most *m* colors.**
- **Ex) Clique DP**
  - **Clique: maximal complete subgraph**
  - **Determine, for a positive integer *k*, whether there is a clique of size ≥ *k***
  - **See Fig. 9.1**

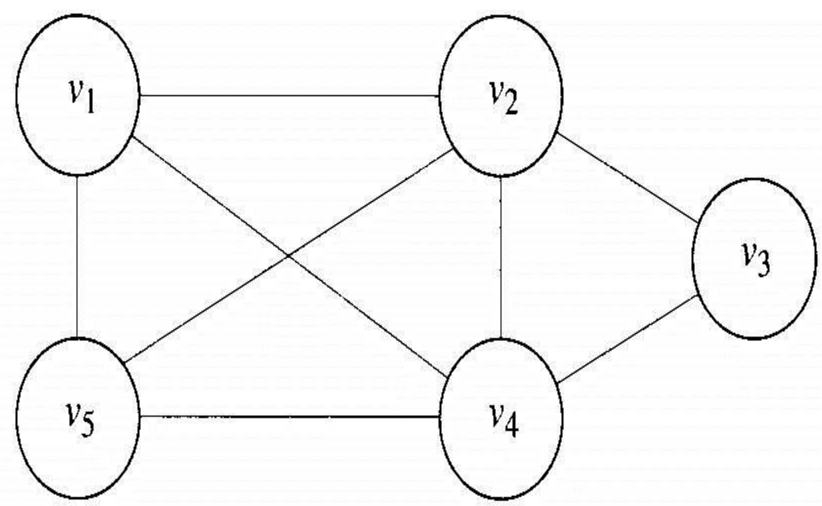


Figure 9.1 The maximal clique is $\{v_1, v_2, v_4, v_5\}$.

9.4.1 The Sets P and NP

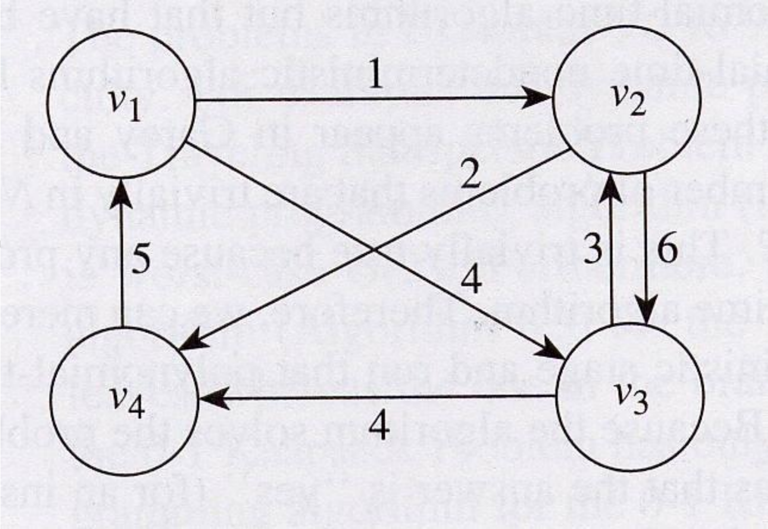
- **Def.) P is the set of all decision problems that can be solved by polynomial-time algorithms**
- **Nondeterministic algorithm**

  (1) Guessing (nondeterministic) stage

  generate some string *S* (guess a solution)

S	Output	Reason
$[v_1, v_2, v_3, v_4, v_1]$	False	Total weight is greater than 15
$[v_1, v_4, v_2, v_3, v_1]$	False	*S* is not a tour
#@12*&%a_1\	False	*S* is not a tour
$[v_1, v_3, v_2, v_4, v_1]$	True	*S* is a tour with total weight no greater than 15

Verification stage



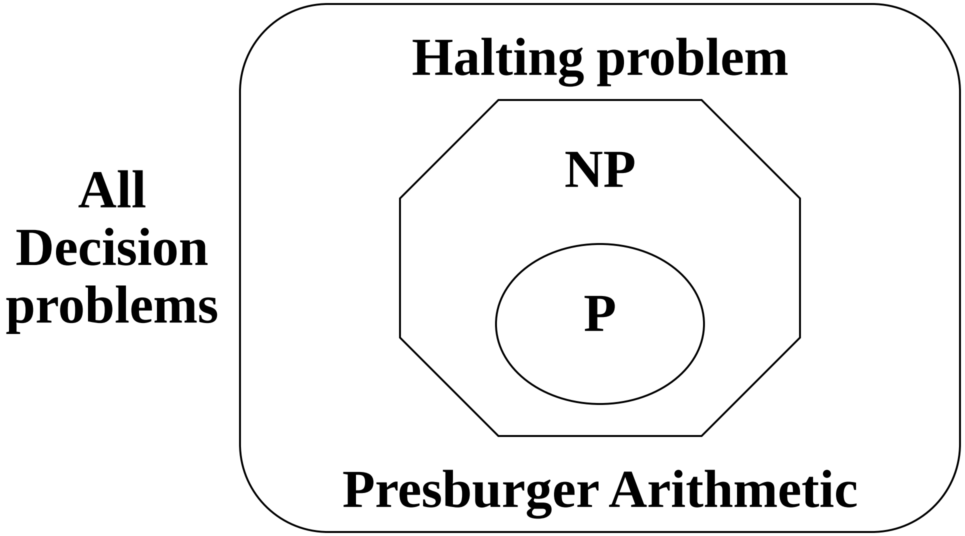
(2) Verification (deterministic) stage

```
bool verify(weighted_digraph G,
            number d,
            claimed tour S)
{
   if (S is a tour && the total weight of the edges in S is <= d)
      return true;
   else
      return false;
}
```

S	Output	Reason
$[v_1, v_2, v_3, v_4, v_1]$	False	Total weight is greater than 15
$[v_1, v_4, v_2, v_3, v_1]$	False	*S* is not a tour
#@12*&%a$_1$\	False	*S* is not a tour
$[v_1, v_3, v_2, v_4, v_1]$	True	*S* is a tour with total weight no greater than 15

Definition – NP

- **Def.) NP is the set of all decision problems that can be solved by polynomial-time nondetermistic algorithms.**



- **P = NP (?) open**
- **No one has ever proven that there is a problem in NP that is not in P.**

9.4.2 NP-Complete Problems

- **CNF-satisfiability Problem (SAT)**

  x_i : **logical (boolean) variable, true or false**

  $\bar{x}_i$: **negation of** x_i

  CNF (Conjunctive Normal Form)

  ex) $(\bar{x}_1 \vee x_2 \vee \bar{x}_3) \wedge (x_1 \vee \bar{x}_4) \wedge (\bar{x}_2 \vee x_3 \vee x_4)$

  - **To determine whether a formula is true for some assignment of truth values to the variables.**

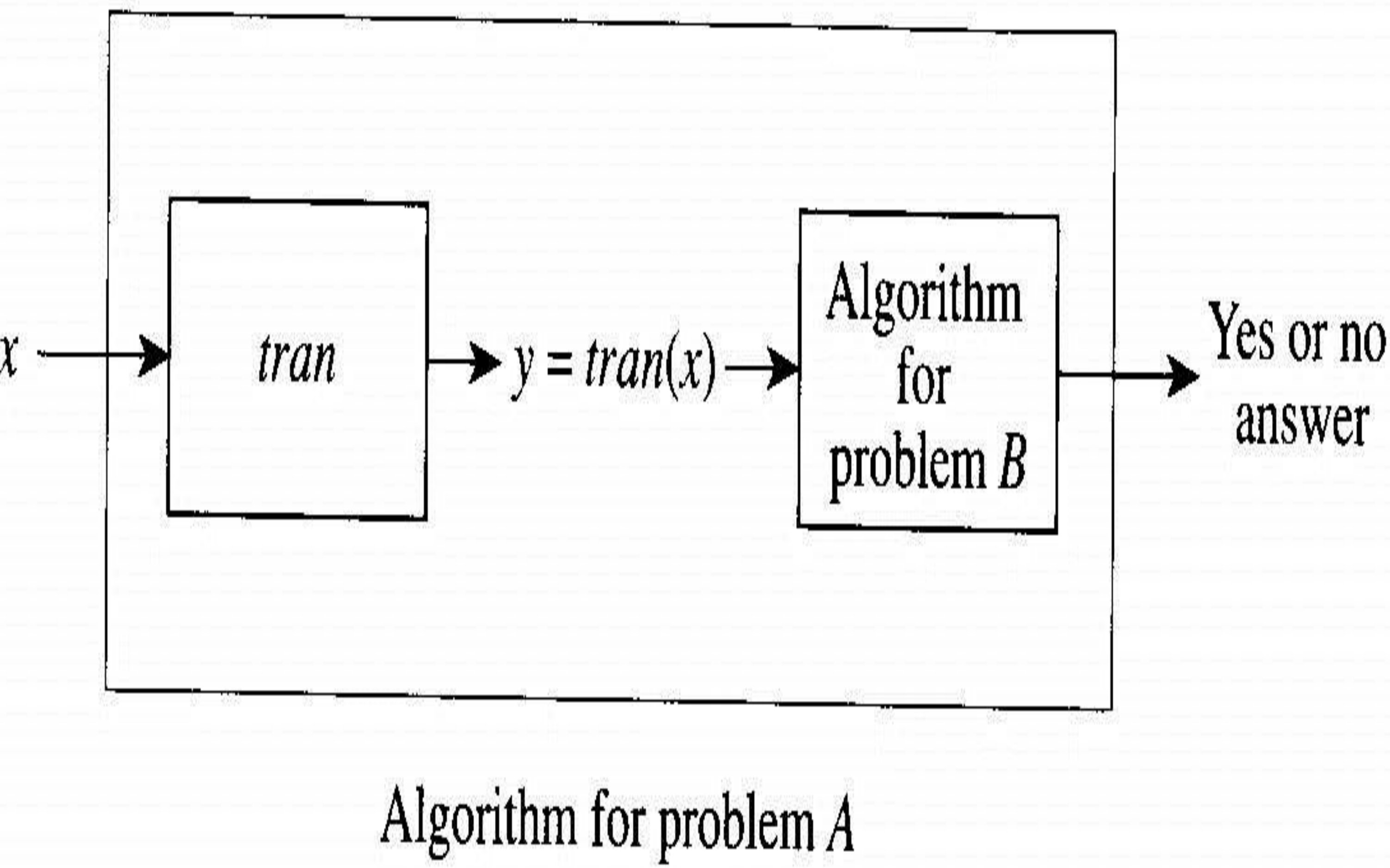
- **Transformation** $(A \propto B)$

  - **Want to solve (decision) problem** A

    Assume we have transformation (algorithm) $I_B = tran(I_A)$ **and (decision) algorithm for solving** B

    Then, we can solve problem A

Figure 9.4 Algorithm *tran* is a transformation algorithm that maps each instance x of decision problem A to an instance y of decision problem B. Together with the algorithm for decision problem B, it yields an algorithm for decision problem A.



Theorem 9.1

- **If decision problem B is in P and $A \propto B$ then decision problem A is in P.**
- **(Proof)**

  p: polynomial-time complexity of transformation

  q : polynomial-time complexity algorithm for B.

$$I_A \text{ of size } n \xrightarrow[p(n)]{trans} I_B \text{ of size} \le p(n) \xrightarrow[q(p(n))]{\text{algorithm B}} \text{yes/no}$$

Note that $p(n) + q(p(n))$ is polynomial-time in n

Definition and Theorems

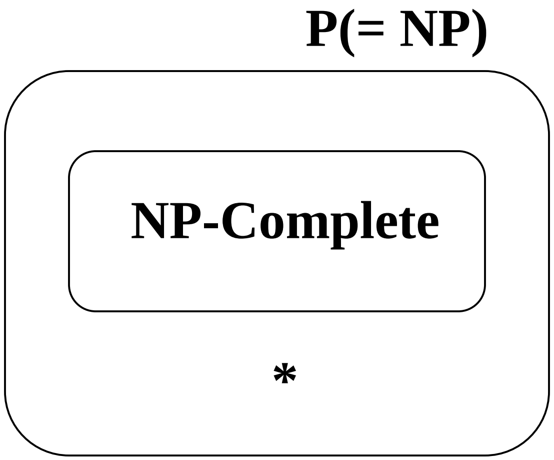
- **A problem B is called NP-complete if**
  1. It is in NP and
  2. $\forall A$ **in NP,** $A \propto B$

- **Theorem 9.2 (Cook's Theorem)**
  SAT is NP-complete. (SAT is in P iff P = NP)

- **Theorem 9.3 A problem C is NP-complete if**
  1. $C \in NP$
  2. $\exists B \in \text{NP} - \text{complete} \;\; B \propto C$
  (Proof) $\propto$ **is transitive (B can be Sat)**

Discussion – The state of NP

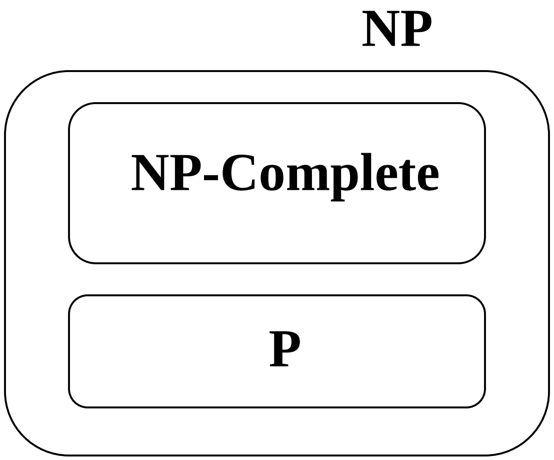
- **P = NP**



*** trivial decision problems that always answers yes(or no)**

(nontrivial decision problem can't be transformed to it)

- $P \neq NP$



$P \cap NP-complete = \emptyset$

(if not, by Th. 9.1, we could solve any problem in NP in polynomial-time)

9.4.3 NP-Hard, NP-Easy, NP-Equivalent Problems

- **Extend to general problem**
- **Def.)** $A \propto_T B$

If problem A can be solved in polynomial-time using a "hypothetical" polynomial-time algorithm for B, A is polynomial-time Turing reducible to B. (A Turing reduces to B).

$\propto_T$ **is transitive**

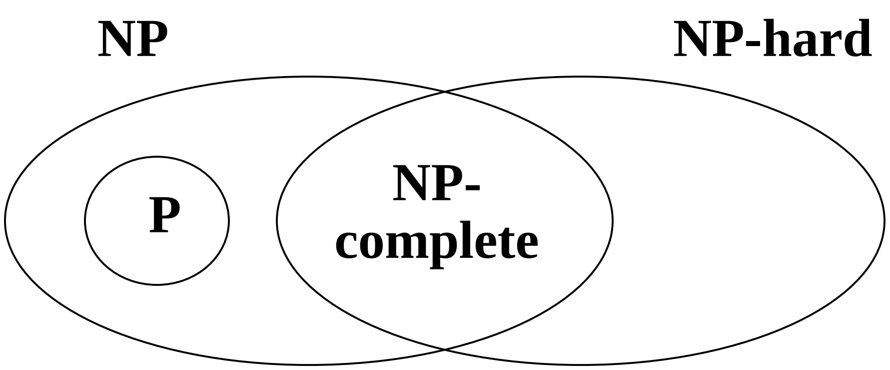
$A \propto B$ **implies** $A \propto_T B$

Definition

- **A problem B is called NP-hard if for some NP-complete problem A, $A \propto_T B$**

Every NP-complete problem is NP-hard.

The optimization problem corresponding to any NP-complete problem is NP-hard.



If a problem is NP-hard, it is at least as hard (for finding polynomial-time algorithm) as the NP-complete problems.