Chapter 9: "The Theory of *NP*" Categories of problems:

- problems with know polynomial-time algorithm(s), e.g., sorting, searching, etc.
- problems that have been proven to have **no** polynomial-time algorithm, called *intractable*
  e.g., Halting problem - input: <an algorithm, algorithm's input>
  output: Yes/No will the algorithm halt?
- problems that have not been proven to be intractable, but have no known polynomial-time algorithm. e.g., TSP, 0-1 Knapsack, graph-coloring, etc.

1. The Theory of *NP* phrases these optimization problems as decision problems which only output: Yes or No. Phrase the 0-1 Knapsack problem in terms of a decision problem (Yes/No answer).

2. One of the biggest open-questions in Computer Science is whether $P = NP$.

a) What would need to be done to show that $P = NP$?

b) What would need to be done to show that $P \neq NP$?

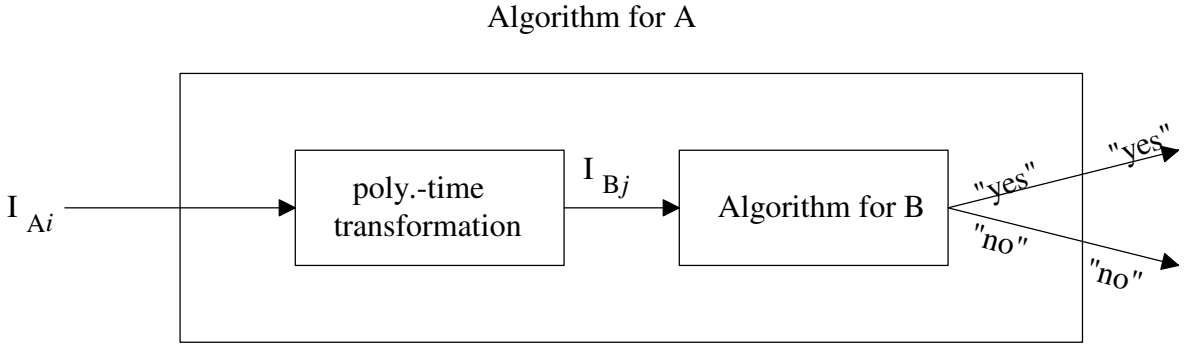
Definition of polynomial-time reducibility

Decision problem A reduces to decision problem B if any instance of problem A, say I_{Ai}, can be transformed into an instance of problem B, say I_{Bj}, such that

a) transformation time is a polynomial in the size of I_{Ai} (call it n_A),

b) the size of $I_{Bj} \leq$ polynomial w.r.t. to size of I_{Ai} , and

c) Algorithm B with I_{Bj} as input answers "yes" if and only if algorithm A with I_{Ai} as input answers "yes".



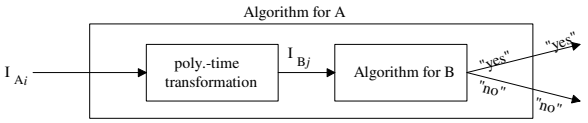
We say "A reduces to problem B," or "A$\propto$ B"

3. Assume there exists a polynomial-time transformation from decision problem A to decision problem B.

a) If we can solve B in poly. time, then how fast can we solve A?

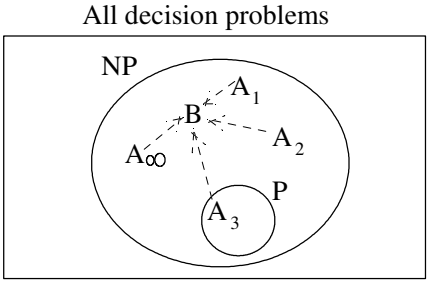
b) If A is known to be "hard", say best worst-case algorithm of $\Theta(2^n)$, then what can we conclude about B?

Theorem 9.1: If decision problem B is in *P* and problem A reduces to problem B (A∝ B), then decision problem A is in *P*.



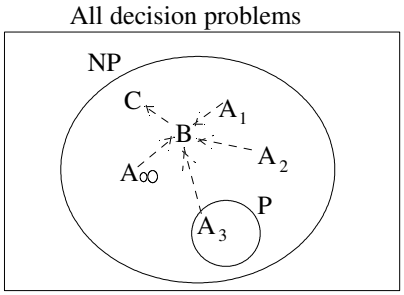
Definition: a problem B is called *NP-complete* if
1) B is in *NP*, and
2) for **every other** problem A in *NP*, A ∝ B.

4. If we find a polyinomal-time algorithm for any *NP*-complete problem B, what can we conclude?



5. If
- B is known to be an *NP*-complete problem,
- a problem C in shown to be in *NP,* and
- B ∝ C,

then what can we conclude? (Theorem 9.3)



Theorem 9.2: (Cook's Theorem 1971) CNF-Satisfiability is *NP*-complete.
(Proof uses common properties of NP problems to show all of them reduce to CNF-SAT.)

CNF-Satisfiability Problem
$x_1, x_2, ..., x_n$: Boolean variables
$\overline{x}_1, \overline{x}_2, ..., \overline{x}_n$: complement of the Boolean variables
$\overline{x}_1, \overline{x}_2, ..., \overline{x}_n, x_1, x_2, ..., x_n$:set of "literals"

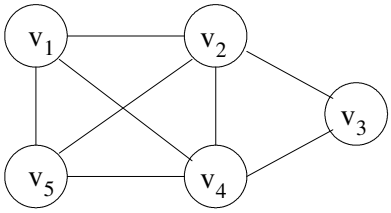
A *clause* is the "or" (∨)of a set of literals, e.g., $(\overline{x_1} \vee \overline{x_2} \vee x_4)$.

A Boolean expression is in *CNF (Conjunctive Normal Form)* if it is the conjunction/"anding" (∧) of one or more clauses, e.g., $(x_1 \vee x_2) \wedge x_3 \wedge (\overline{x_1} \vee \overline{x_2} \vee x_4)$.

The CNF-Satisfiability Decision problem is to determine for a given CNF expression whether there is some truth assignment that makes the CNF expression "TRUE."

For the CNF-SAT Boolean formula B = $(x_1 \vee x_2) \wedge x_3 \wedge (\overline{x_1} \vee \overline{x_2} \vee x_4)$, the answer is "Yes" since the true assignment of x_1=TRUE, x_2= FALSE, x_3=TRUE, x_4=TRUE makes B true.

Clique Problem
A *clique* in an undirected graph G=(V, E) is a subset W vertices such that each vertex W has an edge to all other vertices of W in the graph G.



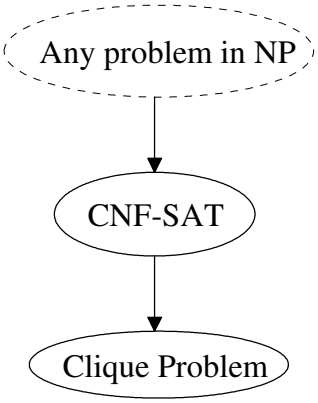
Several cliques exist:

{v2, v3, v4}

{v1, v2, v4 , v5 }

The optimization clique problem is to find a maximal clique, i.e., a clique with largest number of vertices

The textbook shows the details of reducing CNF-Satisfiability Decision problem to the clique problem. So we have:



6. What can we conclude about the clique problem?