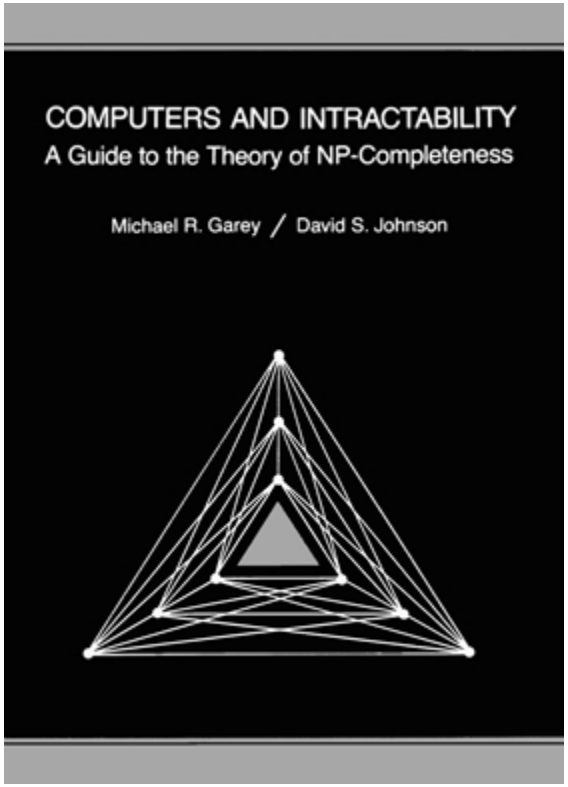
NP-Complete Reductions



Theory of Computation

CISC 603, Spring 2020, Daqing Yun

Reductions

- *NP*-Complete problems
- Cook's theorem
- SAT
- Independent Set

Recall: Polynomial-Time Reductions and *NP*-Completeness

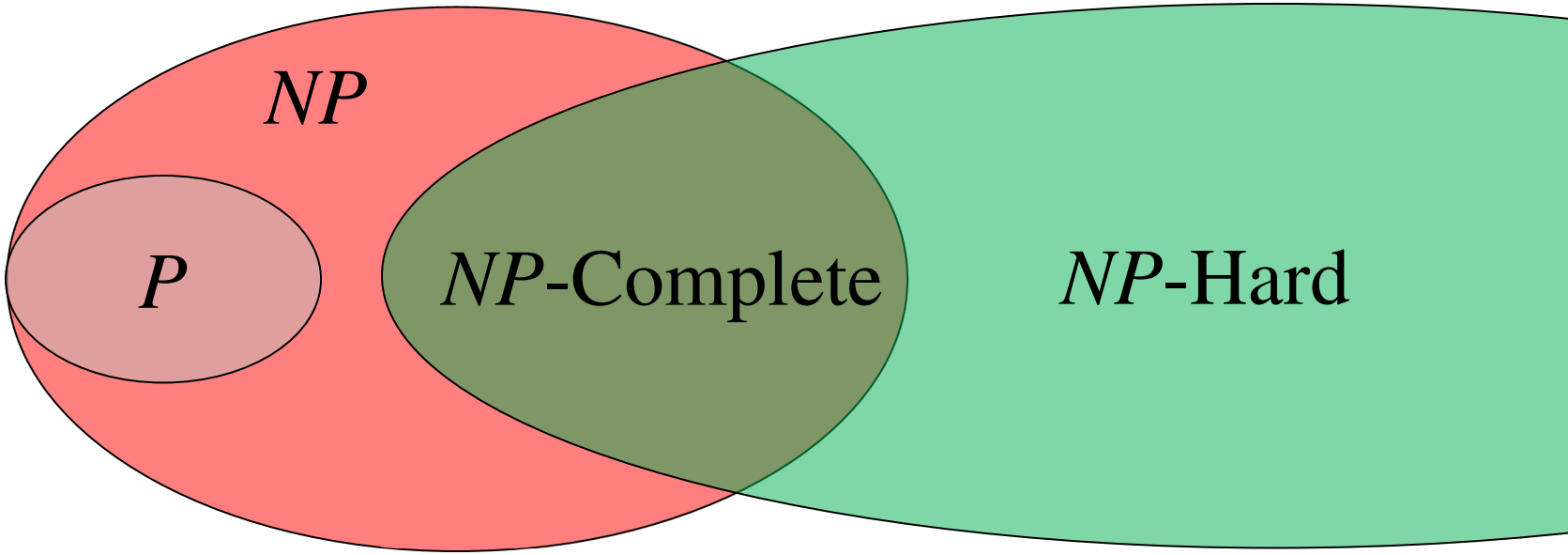
- A language L is *NP-hard* if $L_1 \leq_p L$ for each $L_1 \in NP$
- L is *NP-complete* if $L \in NP$ and L is *NP*-hard
- If L and L_1 are languages such that L is *NP*-hard and $L \leq_p L_1$, then L_1 is also *NP*-hard
- If L is any *NP*-complete language, then $L \in P$ if and only if $P = NP$
- We do not know if $P = NP$, most people assume that *NP* is a larger set, but no one has been able to demonstrate that $P \neq NP$

Recall

- *P* problems can be solved efficiently (i.e., in polynomial time)
- *NP* problems can be verified efficiently (i.e., in polynomial time)
- *NP-hard* problems are all harder than *NP* problems
- *NP*-Complete problems are the hardest ones in *NP*

Recall

If $P \neq NP$
(Many believe this)

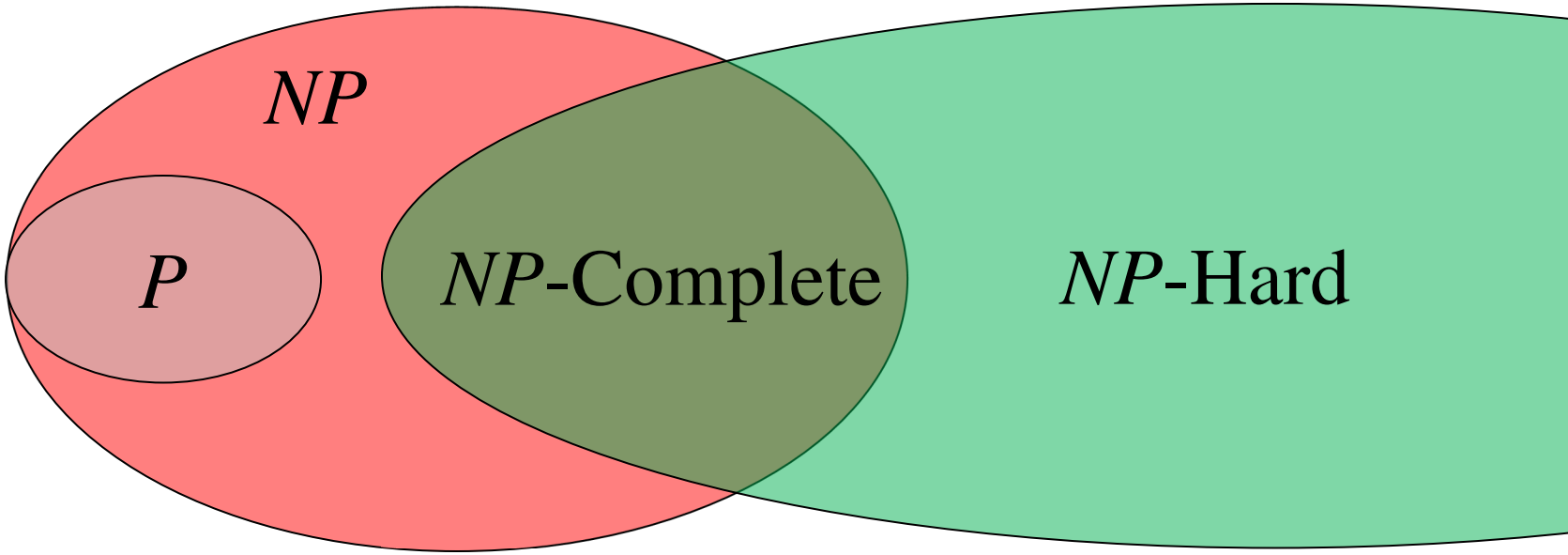


Question

- How to prove *NP*-completeness?
- Reduction

Theorem

- If A and B are problems in NP, A is NP-Complete, and A polynomially reduces to B, denoted as $A \leq_p B$, then B is NP-complete.



Process of Devising an NP-Completeness Proof for a Decision Problem *p*

1. Showing that *p* is in *NP*
2. Selecting a known *NP*-complete problem *p'*
3. Constructing a transformation *f* from *p'* to *p* (i.e., proving *p'* reduces to *p*)
4. Proving that *f* is a (polynomial) transformation

Wait a minute

- Reduce from what?
- Do we need the first *NP*-complete problem?
- The honor goes to Steve Cook, who proved that the *SAT* problem is (the first) *NP*-Complete problem

Input: given a formula in format of $\psi = (x_1 \vee \overline{x_2} \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_1 \vee x_2) \wedge \cdots \wedge (\overline{x_1} \vee \overline{x_2} \vee x_3)$ in which every clause contains at most 3 literals (boolean variables).
Output: the decision whether there is an assignment to x_1, x_2, x_3 so that ψ is true or not.

Karp's 21 NP-complete problems

- **Satisfiability**: the boolean satisfiability problem for formulas in conjunctive normal form (often referred to as SAT)
  - **0–1 integer programming** (A variation in which only the restrictions must be satisfied, with no optimization)
  - **Clique** (see also independent set problem)
    - **Set packing**
    - **Vertex cover**
      - **Set covering**
      - **Feedback node set**
      - **Feedback arc set**
      - **Directed Hamilton circuit** (Karp's name, now usually called **Directed Hamiltonian cycle**)
        - **Undirected Hamilton circuit** (Karp's name, now usually called **Undirected Hamiltonian cycle**)
  - **Satisfiability with at most 3 literals per clause** (equivalent to 3-SAT)
    - **Chromatic number** (also called the Graph Coloring Problem)
      - **Clique cover**
      - **Exact cover**
        - **Hitting set**
        - **Steiner tree**
        - **3-dimensional matching**
        - **Knapsack** (Karp's definition of Knapsack is closer to Subset sum)
          - **Job sequencing**
          - **Partition**
            - **Max cut**

Independent Set

DEFINITION 4.1. An *independent set* of vertices of a graph $G(V, E)$ is defined as a set of vertices of G such that there is no edge between any two of these vertices.

DEFINITION 11.1. **Independent Set Problem**
Instance: a graph $G(V, E)$, integer $k \leq |V|$.
Question: does G contain an independent set of size at least k?

Independent Set

Definition An *independent set* of vertices of a graph $G(V, E)$ is defined as a set of vertices of G such that there is no edge between any two of these vertices.

Definition **Independent Set Problem**
Instance: a graph $G(V, E)$, integer $k \leq |V|$.
Question: does G contain an independent set of size at least k?

Note that this decision version problem can be easily converted to the optimization problem asking for the Largest IS (LIS) as one just needs to keep increasing the value of k until it reaches $|V|$, which runs in linear time.

Recall These Two Theorems

- Theorem. If A and B are problems in NP, A is NP-Complete, and A polynomially reduces to B, denoted as $A \leq_p B$, then B is NP-complete.
- Cook-Levin Theorem. SAT is NP-Complete
- We can prove the NP-Completeness of the IS problem by reducing from 3-SAT

Show IS is in *NP*

- It is quite straightforward to show the IS problem is in NP since if we are given an independent set, we can simply count the number of vertices (denoted as *m*) in it to see if $m <= k$ or not
- This verification could be done in *linear* (thus polynomial) time

Reduction from 3-SAT

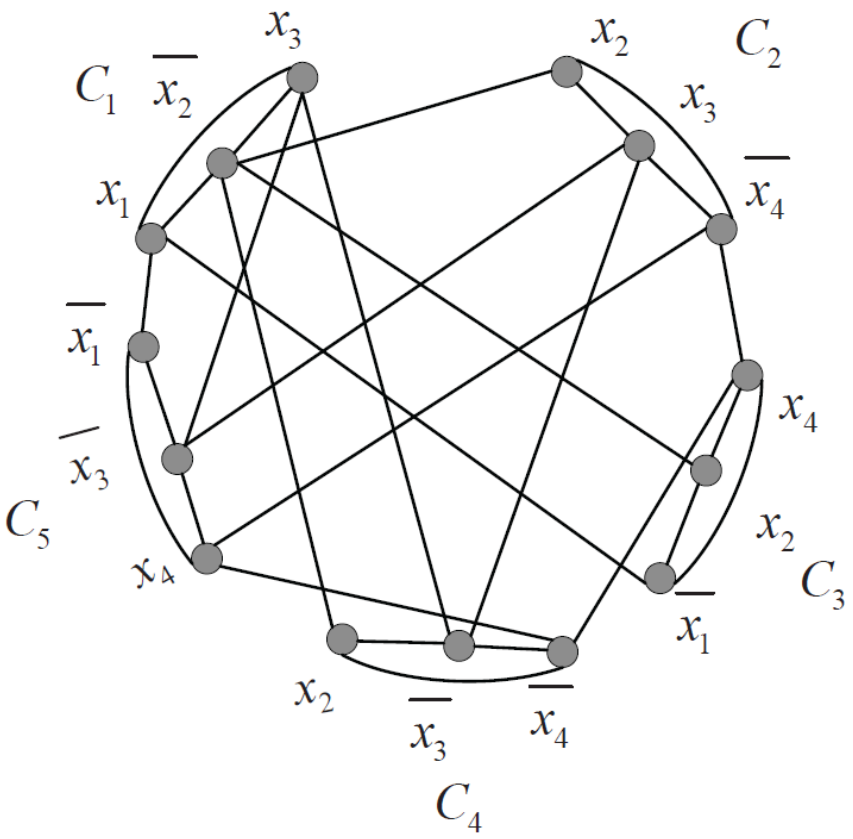
- Show that the IS problem is *NP*-complete by reducing 3-SAT to it, i.e., 3-SAT $\leq_p$ IS

Suppose we are given a boolean formula φ with clauses $C_1, C_2, \ldots, C_k$, we build a graph G so that φ is satisfiable if and only if G contains an IS of size k.

For each clause C_i, we create a vertex for each of the literals and have them connected by three edges. This forms a "triangle". We then denote the vertices of each "triangle" with the literals of the clause. Among the k "triangles", we add edges between all the contradictory literals (i.e., x_i and $\overline{x_i}$).

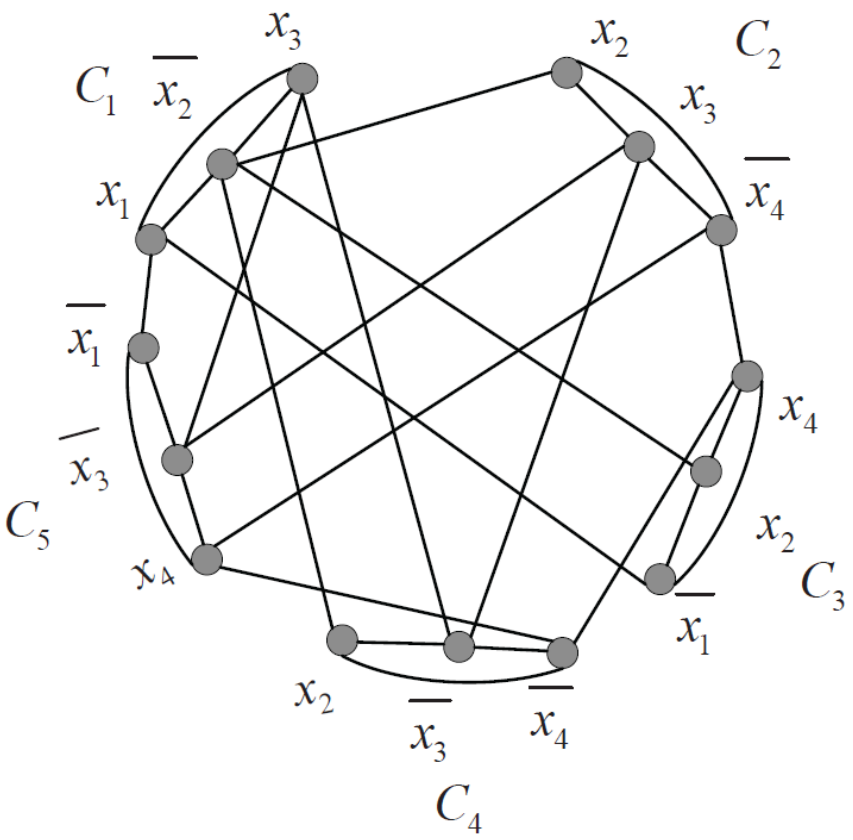
Reduction from 3-SAT

For example, the resulted graph G from formula $\varphi = (x_1 \vee \overline{x_2} \vee x_3) \wedge (x_2 \vee x_3 \vee \overline{x_4}) \wedge (\overline{x_1} \vee x_2 \vee x_4) \wedge (x_2 \vee \overline{x_3} \vee \overline{x_4}) \wedge (\overline{x_1} \vee \overline{x_3} \vee x_4)$ is shown



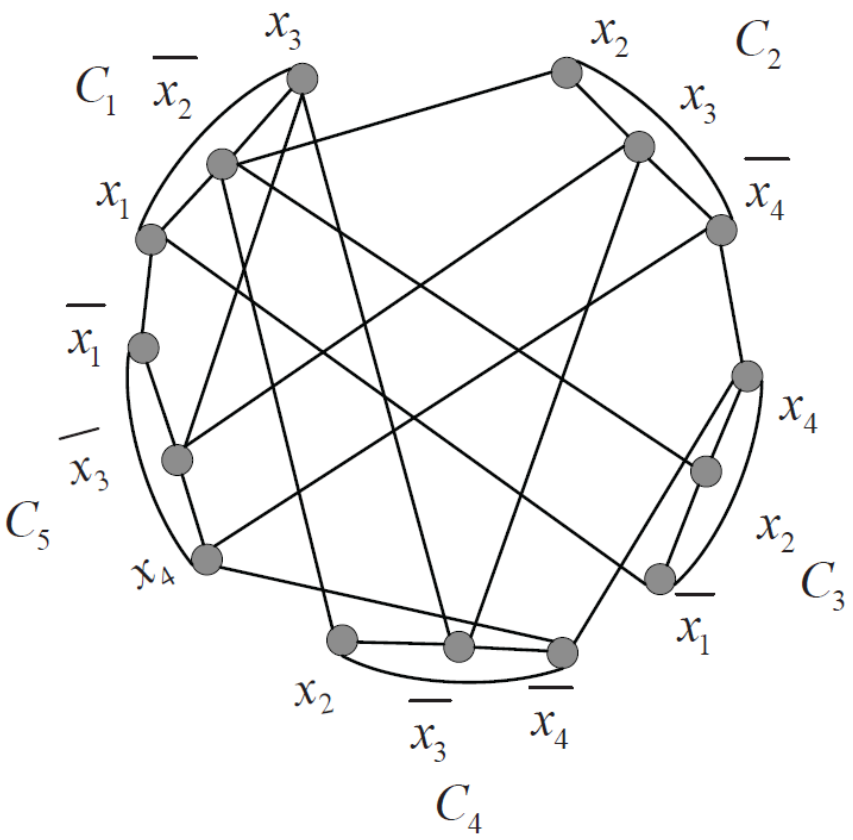
Reduction from 3-SAT

- 3-SAT → IS



Reduction from 3-SAT

- IS → 3-SAT



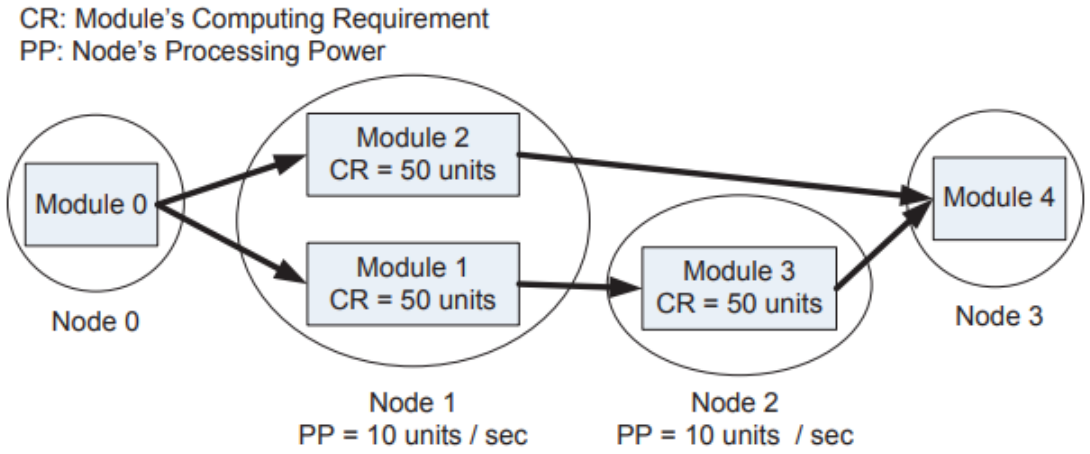
Process of Devising an NP-Completeness Proof for a Decision Problem *p*

- How to prove the *NP*-completeness?
- Special Case method: find a known *NP*-Complete problem, and show that it is a special case of the problem being proven

Process of Devising an NP-Completeness Proof for a Decision Problem *p*

Definition 1. (***ONWS***) *Given a DAG-structured computing workflow* $G_w = (V_w, E_w)$, *a heterogeneous overlay computer network* $G_c = (V_c, E_c)$ *and a mapping scheme* M : $G_w \rightarrow G_c$, *we wish to find a job scheduling policy on every computer node with concurrent modules such that the mapped workflow achieves:*

$$\text{MED} = \min_{\text{all possible job scheduling policies}} (T_{\text{ED}}). \tag{2}$$



Yun *et al.*, On Workflow Scheduling for End-to-End Performance Optimization in Distributed Network Environments. In JSSPP '12.

Definition 2. (***ONWS-Decision***) *Given a set T of tasks t, each having a length(t) and executed by a specific processor p(t), a number* $m \in Z^+$ *of processors, partial order* $\prec$ *on T, and an overall deadline* $D \in Z^+$*, is there an m-processor preemptive schedule for T that obeys the precedence constraints and meets the overall deadline?*

ONWS-Decision is *NP*-Complete is proven by showing that the *m-processor bound unit execution time* (MBUET) system scheduling problem, an existing *NP*-complete problem, is a special case of ONWS-Decision

Definition 3. (***MBUET***) *Given a set T of tasks, each having length(t) = 1 (unit execution time) and executed by a specific processor p(t), an arbitrary number* $m \in Z^+$ *of processors, partial order* $\prec$ *of a forest on T, and an overall deadline* $D \in Z^+$*, is there an m-processor schedule for T that obeys the precedence constraints and meets the overall deadline?*

Yun *et al.*, On Workflow Scheduling for End-to-End Performance Optimization in Distributed Network Environments. In JSSPP '12.

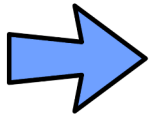
Theorem 1. *The ONWS-Decision problem is NP-complete.*

Proof. Obviously, the MBUET scheduling problem is a special type of ONWS-Decision problem with $length(t)$ restricted to be 1 for all tasks $t \in T$ and partial order $\prec$ restricted to be a forest. For preemptive scheduling as considered in ONWS-Decision, a task is not required to finish completely without any interruption once it starts execution. However, if we restrict the length of all tasks to be 1 (the smallest unit of time) in the input of ONWS-Decision, each task would finish in its entirety once it is assigned to the designated processor for execution. Moreover, the partial order $\prec$ of a forest is a special DAG structure of workflow topology. Therefore, the MBUET problem polynomially transforms to the preemptive ONWS-Decision scheduling problem. Since MBUET is NP-complete [16], the general ONWS-Decision problem is also NP-complete. The validity of the NP-completeness proof by restriction is established in [15], where "restriction" constrains the given, not the question of a problem.

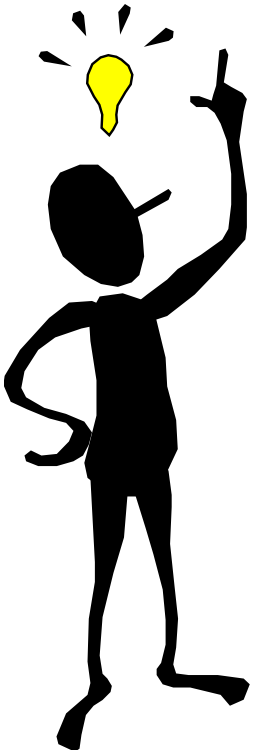
15. Garey, M., Johnson, D.: Computers and Intractability: A Guide to the Theory of NP-completeness. W.H. Freeman and Company, San Francisco (1979)
16. Goyal, D.: Scheduling processor bound systems. Tech. rep., Computer Science Department, Washington State University (1976)

Problem Solving

1. Problem formulation – what to solve?
2. Complexity analysis – how hard?
3. Algorithm design – abstract solution
4. Implementation – practical solution
5. Evaluation – performance

Next 

- What we can do to tackle the “hard” problems
  - Coping with the *NP*-Complete problems



Thanks ! ☺

Questions ?