Complexity Theory

IE 661: Scheduling Theory

Fall 2003

Satyaki Ghosh Dastidar

Outline

- Goals
- Computation of Problems
  - Concepts and Definitions
- Complexity
  - Classes and Problems
- Polynomial Time Reductions
  - Examples and Proofs
- Summary

Goals of Complexity Theory

- To provide a method of quantifying *problem* difficulty in an absolute sense.
- To provide a method comparing the relative difficulty of two different *problems*.
- To be able to rigorously define the meaning of *efficient algorithm*.
  (e.g. Time complexity analysis of an algorithm).

Computation of Problems

Concepts and Definitions

Problems and Instances

A *problem* or *model* is an infinite family of *instances* whose objective function and constraints have a specific structure.

An *instance* is obtained by specifying values for the various problem parameters.

Measurement of Difficulty

<u>Instance</u>

- *Running time* (Measure the total number of elementary operations).

<u>Problem</u>

- *Best case* (No guarantee about the difficulty of a given instance).
- *Average case* (Specifies a probability distribution on the instances).
- *Worst case* (Addresses these problems and is usually easier to analyze).

Time Complexity

***Θ*-notation** *(asymptotic tight bound)*

$$\Theta(g(n)) = \left\{ \begin{array}{l} f(n): \text{ there exist positive constants } c_1, c_2, \text{ and } n_0 \text{ such that} \\ \qquad 0 \le c_1 g(n) \le f(n) \le c_2 g(n) \text{ for all } n \ge n_0 \end{array} \right.$$

***O*-notation** *(asymptotic upper bound)*

$$O(g(n)) = \left\{ \begin{array}{l} f(n): \text{ there exist positive constants } c \text{ and } n_0 \text{ such that} \\ \qquad 0 \le f(n) \le cg(n) \text{ for all } n \ge n_0 \end{array} \right.$$

***Ω*-notation** *(asymptotic lower bound)*

$$\Omega(g(n)) = \left\{ \begin{array}{l} f(n): \text{ there exist positive constants } c \text{ and } n_0 \text{ such that} \\ \qquad 0 \le cg(n) \le f(n) \text{ for all } n \ge n_0 \end{array} \right.$$

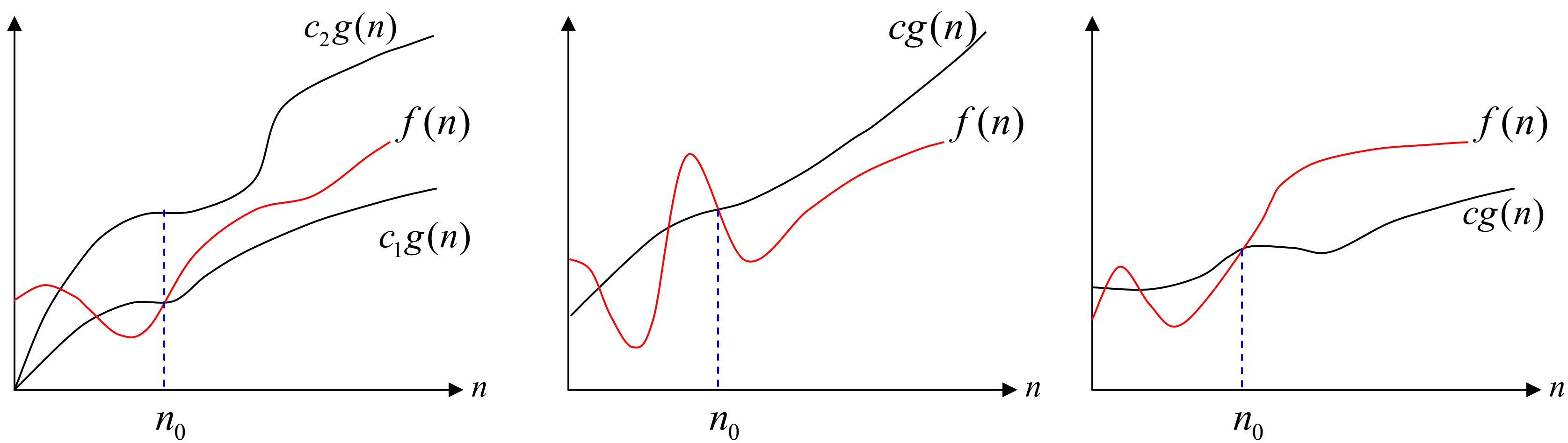
***o*-notation** *(asymptotic "loose" upper bound)*

$$o(g(n)) = \left\{ \begin{array}{l} f(n): \text{ for any positive constant } c > 0, \text{ there exists a constant} \\ \qquad n_0 > 0 \text{ such that } 0 \le f(n) < cg(n) \text{ for all } n \ge n_0 \end{array} \right.$$

…Time Complexity *(contd.)*

ω-notation ***(asymptotic "loose" lower bound)***

$$\omega(g(n)) = \left\{ \begin{array}{l} f(n): \text{ for any positive constant } c > 0, \text{ there exists a constant} \\ \quad n_0 > 0 \text{ such that } 0 \le cg(n) < f(n) \text{ for all } n \ge n_0 \end{array} \right.$$



$f(n) = \Theta(g(n))$ $f(n) = O(g(n))$ $f(n) = \Omega(g(n))$

Algorithm Types

- **Polynomial Time Algorithm:**

  An algorithm whose running time is bounded by a polynomial function is called a *polynomial time algorithm*.

  Example: Shortest path problem with nonnegative weights. Running Time: $O(n^2)$

- **Exponential Time Algorithm:**

  An algorithm that is bounded by an exponential function is called an exponential time algorithm.

  Example: Check every number of *n* digits to find a solution. Running Time: $O(10^n)$

- **Pseudopolynomial Time Algorithm:**

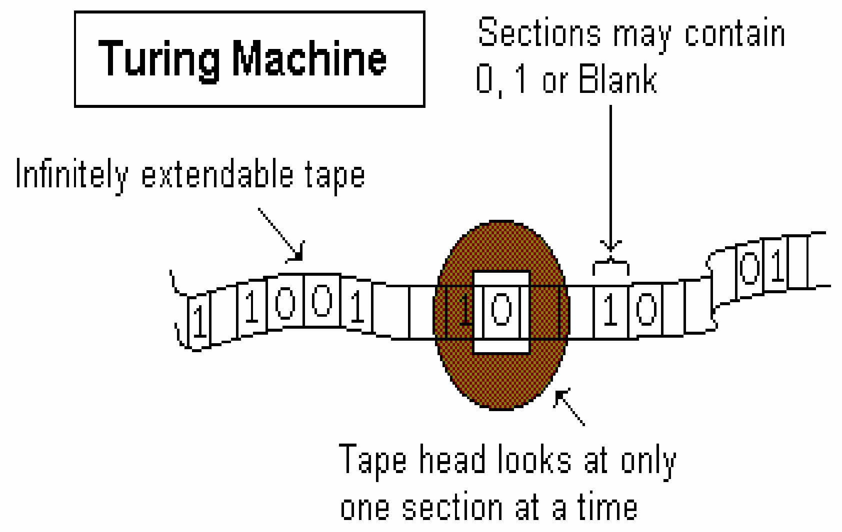
  A *pseudopolynomial time algorithm* is one that is polynomial in the length of the data when encoded in *unary*.

  Example: Integer Knapsack Problem. Running time: $O(nb)$

Turing Machine

- A Turing machine is an abstract representation of a computing device.

The behavior of a TM is completely determined by:



- The ***state*** the machine is in,
- The ***number*** on the square it is scanning, and
- A table of instructions or the ***transition table***.

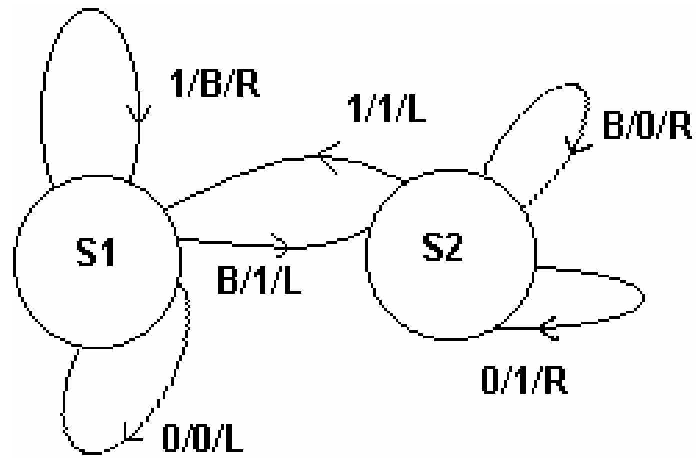
"A function is computable if it can be computed by a Turing Machine."

- Church-Turing Hypothesis

Finite State Machine

State	Read	Write	Move	Next State
S1	0	0	L	S1
	B	1	L	S2
	1	B	R	S1
S2	0	1	R	S2
	B	0	R	S2
	1	1	L	S1

State Transition Table for a Turing Machine



Transition State Diagram for Turing Machine

Decision Problem

Decision problems are those that have a **TRUE/FALSE** answer.

- **SATISFIABILITY**: *Given a set of variables and a collection of clauses defined over the variables, is there an assignment of values to the variables for which each of the clauses is true?*

  Example:

  Consider the expression

$$(x_1 + \overline{x_4} + x_3 + \overline{x_2})(\overline{x_1} + \overline{x_2} + x_4 + \overline{x_3})(\overline{x_2} + \overline{x_3} + x_1 + \overline{x_5})(\overline{x_5} + \overline{x_1} + x_4 + \overline{x_2})$$

  It can be easily verified that the assignment x_1=0, x_2=0, x_3=0, x_4=0, and x_5=0 gives a truth assignment to each one of the four clauses.

Decision Problems and Reductions

- For every *optimization* problem there is a corresponding *decision* problem.

  Example: $Fm||C_{max}$ minimize makespan (*optmization*).

  Is there a schedule with a makespan $\leq z$? (*decision*).

Problem Reduction:

Problem P *reduces* to problem P′ if for any instance of P an equivalent instance of P′ can be constructed.

Polynomial Reducibility:

Problem P *polynomially reduces* to problem P′ if a polynomial time algorithm for P′ implies polynomial time algorithm for P.

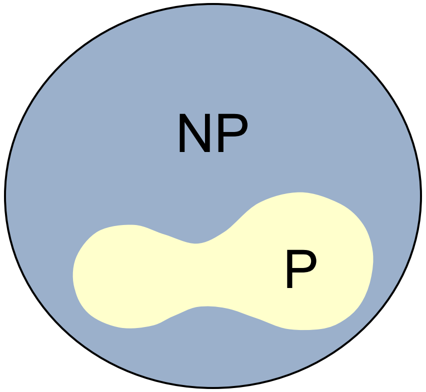
$$P \propto P'$$

Complexity

Classes and Problems

Complexity Classes

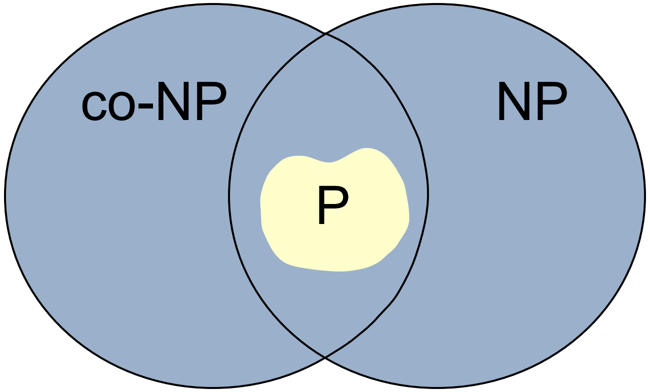
- **Definition: (Class *P*)** *The class P contains all decision problems for which there exists a Turing machine algorithm that leads to the right "yes/no" answer in a number of steps bounded by a polynomial in the length of the encoding.*
- **Definition: (Class *NP*)** *The class NP contains all decision problems for which, given a proper guess, there exists a polynomial time "proof" or "certificate" C that can verify if the guess is the right "yes/no" answer.*



A tentative view of the world of NP

… Complexity Classes *(contd.)*

- **Definition: (Class co-*P*)** *The class co-P contains all decision problems for which there exists a polynomial time algorithm that can determine what all "yes/no" answers are incorrect.*
- **Definition: (Class co-*NP*)** *The class co-NP contains all decision problems such that there exists a polynomial time "proof" or "certificate" C that can verify if the problem does not have the right "yes/no" answer.*



A view of the world of NP and co-NP

Important Results

- ***P = co-P***
- ***NP ≠ co-NP***
- ***P ≠ NP***

It turns out that almost all interesting problems lie in *NP* and *P* is the set of easy problems. So are all interesting problems easy, i.e. do we have *P* = *NP*?

This is the main open question in Computer Science. It is like other great questions

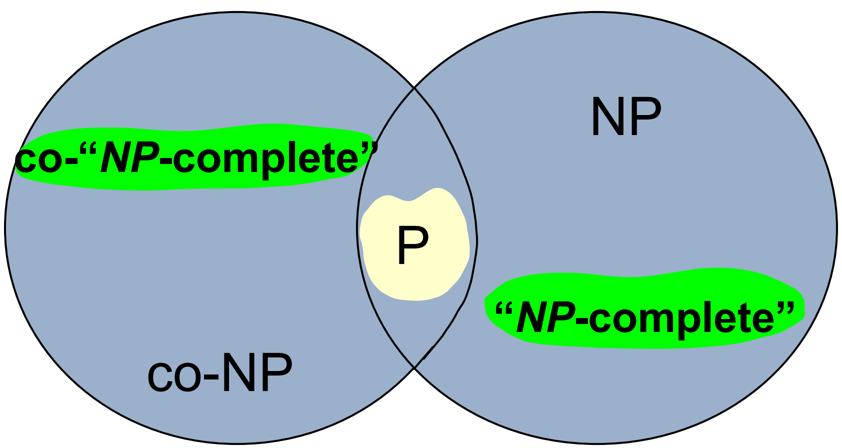
- *Is there intelligent life in the universe?*
- *What is the meaning of life?*
- *Will you get a job when you graduate?*

NP-Complete Problems

- **Definition: (*NP*-complete)** *A decision problem D is said to be NP-complete if D™NP and, for all other decision problems D′™NP, there exists a polynomial transformation from D′ to D, i.e., D′∝ D.*

Assumption: *P≠NP.*

Result: *If any single NP-complete problem can be solved in polynomial time, then all problems in NP can be solved.*



The world of NP, revisited

Cook's Theorem

A problem is *NP*-complete if:

(i) The problem is in *NP*

(ii) ***All*** other problems in *NP* polynomially transforms into the above problem.

NP-Hard Problems

- **Definition: (*NP*-hard)** *A decision problem whether a member of NP or not, to which we can transform a NP-complete problem is at least as hard as the NP-complete problem. Such a decision problem is called NP-hard.*

  Example:

  K^TH^ LARGEST SUBSET: *Given a set* $A \in \{a_1, a_2, \ldots a_t\}$ *,* $b \leq \sum_{j \in A} a_j$*, and* $k \leq 2^{|A|}$*, do there exist at least K distinct subsets where* $A' \in \{S_1, S_2, \ldots S_K\}$ *and* $A' \subseteq A$ *such that* $\sum_{j \in A'} S_j \leq b$ *?*

Six Basic *NP*-Complete Problems

- **3-SATISFIABILITY**: *Given a collection* $C = \{c_1, c_2, \ldots, c_m\}$ *of clauses on a finite set U of variables such that* $|c_i|=3$ *for* $1 \le i \le m$, *is there a truth assignment for U that satisfies all the clauses in C?*
- **3-DIMENSIONAL MATCHING**: *Given a set* $M \subseteq W \times X \times Y$, *where W, X, and Y are disjoint sets having the same number q of elements, does M contain a matching, i.e., a subset* $M' \subseteq M$ *such that* $|M'| = q$ *and no two elements of* M' *agree in any coordinate?*
- **PARTITION**: *Given positive integers* $a_1, \ldots, a_t$ *and* $b = \frac{1}{2}\sum_{j=1}^{t} a_j$, *do there exist two disjoint subsets* S_1 *and* S_2 *such that*
$$\sum_{j \in S_i} a_j = b \quad \text{for } i = 1, 2\ ?$$

…Six Basic Problems *(contd.)*

- **VERTEX COVER**: *Given a graph G=(V,E) and a positive integer K ≤ |V|, is there a vertex cover of size K or less for G, i.e., a subset* $V' \subseteq V$ *such that* $|V'| \leq K$ *and, for each edge* $\{u,v\} \in E$*, at least one of u and v belongs to* V'*?*

- **HAMILTONIAN CIRCUIT**: *For a graph G = (N, A) with node set N and arc set A, does there exist a circuit (or tour) that connects all the N nodes exactly once?*

- **CLIQUE**: *For a graph G = (N, A) with node set N and arc set A, does there exist a clique of size c? i.e., does there exist a set* $N^* \subset N$*, consisting of c nodes such that for each distinct pair of nodes* $u, v \in N^*$*, the arc {u,v} is an element of A?*

Transformation Topology

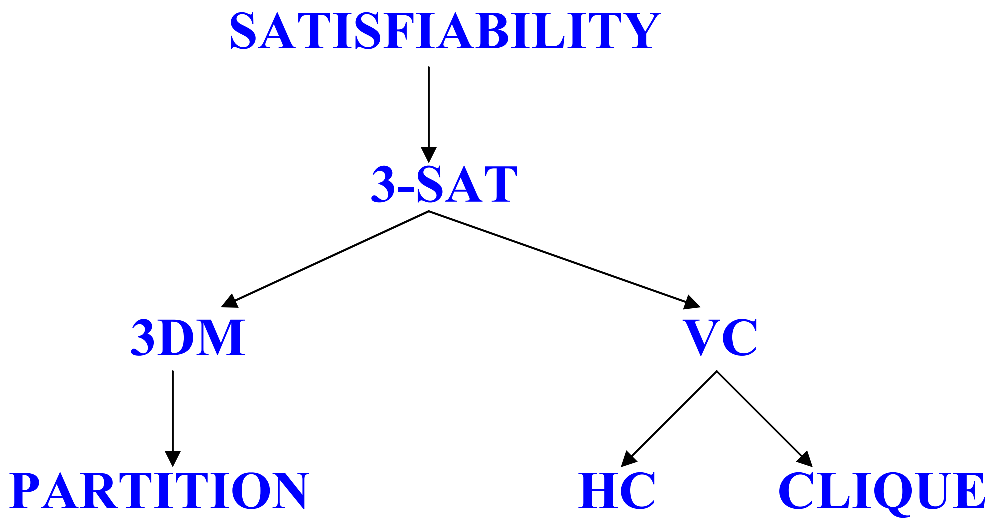


Diagram of the sequence of transformations used to prove that the six basic problems are *NP*-complete.

Problems of which the complexity is established through a reduction from **PARTITION** typically have pseudopolynomial time algorithms and are therefore *NP*-hard in the ordinary sense.

Other Popular Problems

- **3-PARTITION**: *Given positive integers $a_1, \ldots, a_{3t}$ and b with* $\frac{b}{4} < a_j < \frac{b}{2},\ j = 1,\ldots,3t,$ *and* $\sum_{j=1}^{3t} a_j = tb$ *, do there exist t pairwise disjoint three element subsets* $S_i \subset \{1,\ldots,3t\}$ *such that* $\sum_{j \in S_i} a_j = b$ *for* $i = 1,\ldots,t$?

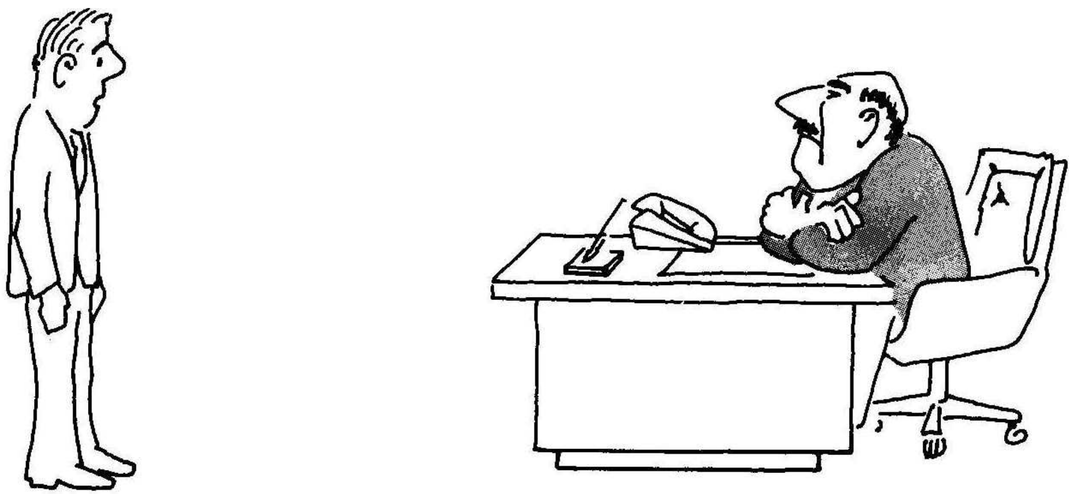
- **TRAVELING SALESMAN PROBLEM**: *For a set of cities $C=\{c_1, c_2, \ldots, c_m\}$ does there exist a "tour", of all the cities in C, of length $\leq b$ such that one city is visited exactly once?*

Polynomial Time Reductions

Examples and Proofs

Dealing with Hard Problems

You: Give up!



"I can't find an efficient algorithm, I guess I'm just too dumb"

Boss: Fires you!

Still Dealing!!

You: Challenge Boss!

"I can't find an efficient algorithm, as no such algorithm is possible!"

Boss: Asks for proof!

You: Cannot prove!

Boss: Gives you a rise?…..very unlikely!

Better Strategy

You: Prove that the problem is "hard" and that everyone else has failed.

"I can't find an efficient algorithm, but neither can all these famous guys!"

Boss: At least he gets no benefit out of firing you!

Problem Reduction – Example 1

- **KNAPSACK PROBLEM**

  KNAPSACK problem is equivalent to the scheduling problem $1|d_j=d|\sum w_j U_j$. The value d refers to size of the knapsack and the jobs are the items that have to be put into the knapsack. The size of the item j is p_j and the weight (value) of the item j is w_j. It can be shown that **PARTITION** reduces to **KNAPSACK** by taking $n = t,\ p_j = a_j,\ w_j = a_j,$

$$d = \frac{1}{2}\sum_{j=1}^{t} a_j = b,\ z = \frac{1}{2}\sum_{j=1}^{t} a_j = b.$$

  It can be verified that there exists a schedule with an objective value $\le \frac{1}{2}\sum_{j=1}^{n} w_j$ iff there exists a solution for the **PARTITION** problem.

Problem Reduction – Example 2

- **MINIMIZE MAKESPAN ON PARALLEL MACHINES ($P2||C_{max}$)**

  Consider $P2||C_{max}$. It can be shown that **PARTITION** reduces to this problem by taking $n = t,\ p_j = a_j,\ w_j = a_j,$

  $$z = \frac{1}{2}\sum_{j=1}^{t} a_j = b.$$

  It is trivial to verify that there exists a schedule with an objective value $\leq \frac{1}{2}\sum_{j=1}^{n} p_j$ iff there exists a solution for the **PARTITION** problem.

Problem Reduction – Example 3

- **MINIMIZE MAKESPAN IN A JOB SHOP**

  Consider $J2|recrc, prmp|C_{max}$. It can be shown that **3-PARTITION** reduces to $J2|recrc, prmp|C_{max}$ by taking the following transformation. If the number of jobs be n, take

  $$n = 3t+1, \quad p_{1j}=p_{2j}=a_j, \quad for\ j=1, \ldots 3t.$$

  Each of these $3t$ jobs has to be processed on machine 1 and then on machine 2. These $3t$ jobs do *not* recirculate. The last job, job $3t+1$, has to start its processing on machine 2 and then alternate between machines 1 and 2. It has to be processes in this way t times on machine 2 and t times on machine 1, and each of these $2t$ processing times = b. For a schedule to have a makespan C_{max}=$2tb$, this last job has to be scheduled without preemption. The remaining slots can be filled without idle times by jobs 1, ..., $3t$ iff **3-PARTITION** has a solution.

Problem Reduction – Example 4

- **SEQUENCE-DEPENDENT SETUP TIMES**

  Consider the **TRAVELING SALESMAN PROBLEM (TSP)** or in scheduling terms $1|s_{jk}|C_{max}$ problem. That the **HAMILTONIAN CIRCUIT (HC)** can be reduced to $1|s_{jk}|C_{max}$ can be shown as follows. Let each node in a **HC** correspond to a city in a **TSP**. Let the distance between two cities equal 1 if there exists an arc between two corresponding nodes in the **HC**. Let the distance between two cites be 2 if such an arc does *not* exist. The bound on the objective is equal to the number of nodes in the **HC**. It is easy to show that the two problems are equivalent.

Summary

Observation

- Present research is in the boundary of polynomial time problems and *NP*-hard problems.
- If a problem is *NP*-complete (or *NP*-hard), there is no polynomial time algorithm that solves it unless $P=NP$. (No pseudopolynomial time algorithms for *strong* *NP*-complete problems).

Why all these analyses?

- Determine the boundary of polynomial time problems and *NP*-hard problems.
- For which decision problems do algorithms exist?
- Develop better algorithms in *cryptography*.

Beyond *NP*-completeness

- Try to prove that $P=NP$ (AMS will give one million dollars).
- Randomized Algorithms.
- Approximation Algorithms.
- Heuristics.