1. NP and Computational

**Polynomial-Time Reductions**

Definition:

- Loose

If problem X is at least as hard as problem Y, it means that if we could solve X, we could solve Y.

- Formal: *Y≤P X (Y is polynomial time reducible to X)*

If Y can be solved using a polynomial number of standard computational steps plus a polynomial number of cells to a black-box that solves X.

Fact:





**Independent Set (packing problem)**

Definition:

In a graph G = (V, E), we say that a set of nodes S V is independent if no two nodes in S are joined by an edge.

Problem statement:

- *optimization version*: Find the largest independent set in graph G.

- *decision version*: Given a graph G and a number k, does G contain an independent set of size at least k.

- Given a method to solve the optimization version, we automatically solve the decision version (for any k) as well.

**Vertex Cover**

Definition:

Given a graph G = (V, E), we say that a set of nodes S V is a vertex cover if every edge in E has at least one end in S.

Problem statement:

- *optimization version*: Find the smallest vertex cover set in graph G.

- *decision version*: Given a graph G and a number k, does G contain an vertex cover of size at most k.

Fact:



Claim: Independent set ≤P Vertex cover and Vertex cover ≤P Independent set

**Set Cover**

Problem statement:

Given a set U of n elements, a collection S1, S2,... , Sm of subsets of U, and a number k, does there exist a collection of at most k of these sets whose union is equal to all of U.

Claim: Vertex cover ≤P Set cover

- G has a vertex cover of size k, if the corresponding set cover instance has k sets whose union contains all edges in G.

- Proof:

1. If l have a vertex cover set of size k in G, l can find a collection of k sets whose union contains all edges in G.
2. If l have k sets whose union contains all edges in G, l can find a vertex cover set of size k in G.

**Reduction using Gadgets**

Definition:

- *clause*: a disjunction of terms t1t2... tl, where ti∈{x1...xn, ...}. (xi is boolean variable)

- *truth assignment for X*: an assignment of values 0 or 1 to each xi

- an assignment satisfies a clause C if it causes C to evaluate to 1.

- an assignment satisfies a collection of clauses if it causes C1C2...Ck evaluate to 1.

**3-SAT(Satisfiability Problem)**

Problem statement: Given a set of clauses C1...Ck each of length 3 over a set of variable X = {x1...xn}. does there exist a satisfying truth assignment?

Claim: 3-SAT ≤P Independent set.

- Given an instance of 3-SAT with k clauses, build a graph G that has an independent set of size k if the 3-SAT instance is satisfiable.

- Proof: The 3-SAT instance is satisfiable if G has an independent set of size k.

1. If the 3-SAT instance is satisfiable, then there is at least one node label per triangle that evaluates to 1. Let S be a set containing one such node from each triangle k.
2. Suppose G has an independent set S of size at least k. If xn appears as a label in S then set xn to 1. If appears as a label in S, then set to 0. If neither xn nor appears as a label in S, then set xn to either 0 or 1.

**Efficient Certification**

Condition: Polynomial length certificate and Polynomial time certifier

- 3-SAT: the certificate t is an assignment of truth values to the variables, the certifier evaluates the given set of clauses with respect to this assignment.

- Independent set: the certificate t is a set of nodes of size at least k in G, the certifier check each edge to make sure no edges joins any pair of these nodes.

- Set Cover: the certificate t is a list of k sets from the given collection, the certifier checks that the union of these sets is equal to the underlying set U.

**NP-Complete Problem**

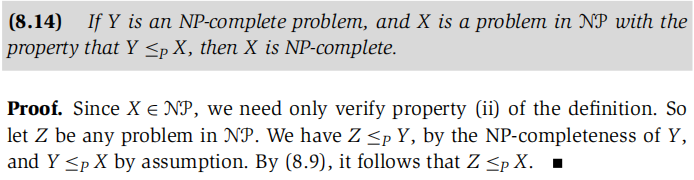
*P*: there exists an algorithm A with a polynomial running time that solves X.

*NP*: the set of all problems for which there exists an efficient certifier.

*NP-Complete*: a problem X∈NP and for all Y∈NP, Y≤PX. (hardest problem in NP)

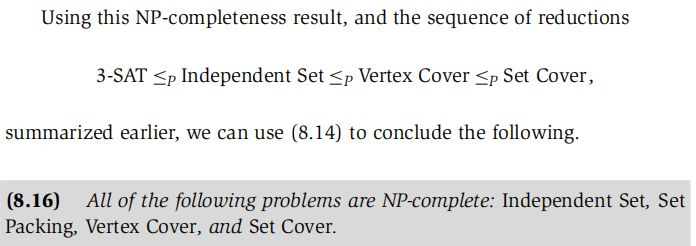


Statement:

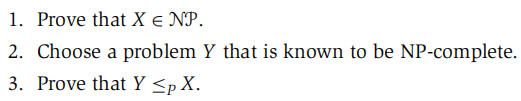


- If X∈NP, then Y∈NP.

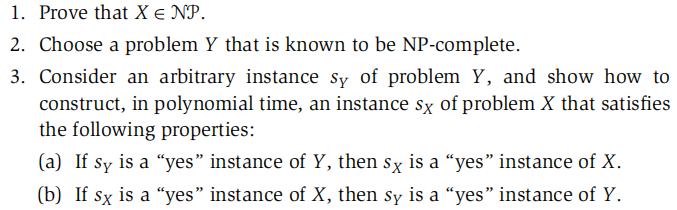




Basic Strategy to prove a problem X is NP-Complete



Refine step 3:



**Hamiltonian Cycle Problem**

*Hamiltonian Cycle*: a cycle in G which visits each vertex exactly once.

Problem Statement: Given a undirected graph G, is there a Hamiltonian Cycle in G?

NPC Proof:

1. Hamiltonian Cycle is in NP.

- Certificate: there is a solution would be the ordered list of the vertices on the cycle.

- Certifier: every node is in the list and no node is repeated. And each consecutive pair in the ordering is joined by an edge.

1. Choose Vertex cover for our reduction
2. Show that Vertex cover ≤P Hamiltonian Cycle.

- Plan: Given an undirected graph G = (V, E) and an integer k, we construct G` = (V`, E`) that has a Hamiltonian cycle if G has a vertex cover of size at most k.

- Construction of G`: For each edge (v, u) in G, G` will have one gadget wvu with following node labeling. There are k vertices in G` which is selector vertices. For each vertex v∈V, adding edges to join pairs of gadgets in order to form a path going through all the gadgets corresponding to edges incident on v in G. And finding set of edges in G` join the first vertex and last vertex of each of these paths to each of the selector vertices.

- Proof of correctness for the reduction step.

**Travelling Salesman Problem (TSP)**

Problem Statement: Given the set of distances, order n cities in a tour with i1 = 1, so it minimizes .

Decision version:

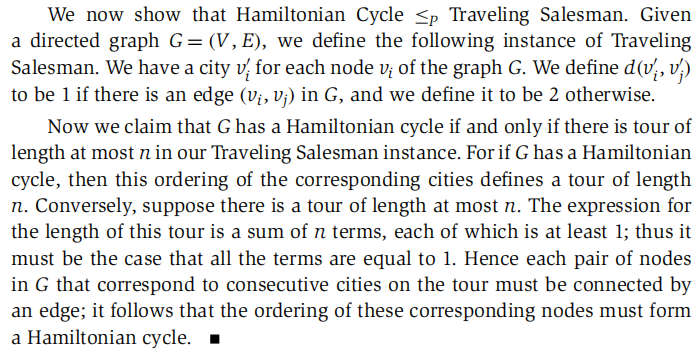
Given a set of distances on n cities, and a bound D, is there a tour of length at most D?

NPC Proof:

1. TSP is in NP.

- Certificate: a tour of length at most D.

- Certifier: all check we did for Hamiltonian Cycle and the cost of tour≤D.

1. Choose Hamiltonian Cycle for our reduction
2. 

- Proof of correctness for the reduction step: G has a Hamiltonian cycle if and only if there is tour of length at most n in our TSP.

Theorem:

If P≠NP, then for any constant 1, there is no polynomial time approximation algorithm with approximation ratio for the general TSP.

- Plan: Assume that such an approximation algorithm exists, we will then use it to solve the HC problem.

1. Given an instance of the HC problem on graph G, we will construct a fully connected graph G` as follows: G` has the same set nodes as in G. Edges in G` that are also in G have a cost of 1, and other edges in G have a cost of |v|+1.
2. If G has a HC then G` will have a tour of cost |v|, and if G` has a tour of cost≤ |v| then G must have a HC.
3. Approximation algorithm

**Load Balancing Problem**

Input:

- m resources with equal processing power

- n jobs where job j takes tj to process

- Ti : load on machine i

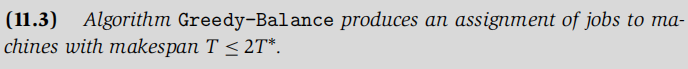
- T\* :value of the optimal solution

Objective: Assignment of jobs to resources such that the maximum load on any machine is minimized.

Greedy algorithm:

- Assigning j to the machine whose load is smallest so far.

- Analysis:





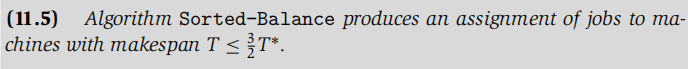




Improved approximation to Greedy balancing:

- Initially sort jobs in decreasing order of length then use same greedy balancing.

- Analysis:





 (tj is the last job and j≥m+1)





**Vertex Cover Problem**

Problem Statement: Find the smallest vertex cover in graph G.

1. approximation algorithm:

- Algorithm

Start with s=null

While S is not a vertex cover

Select an edge e not covered by s

Add both ends of e to s

Endwhile

- Independent set ≤P Vertex, while we can’t use this algorithm for vertex cover to find a .5-approximation algorithm to independent set. Unless P=NP, there is no approximation for the maximum independent set problem for any >0 when n is the number of nodes in the graph.

- Since vertex cover ≤P Set cover, we can use 2-approximation algorithm for set cover to find a 2-approximation algorithm for vertex cover.

**Max-3SAT Problem**

Problem Statement: Given a set of clauses of length 3, find a truth assignment that satisfies the largest number of clauses.

.5-approximation:

- set everything to true, if less than 50% of clauses evaluate to true, then set everything to false.

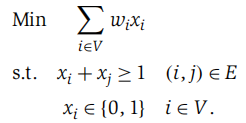
- more sophisticated approximation methods can get to within a factor of 8/9 of the optimal solution.

**Weighted Vertex Cover Problem**

- Each vertex i∈V has a weight wi≥0, with the weight of a set S of vertices denoted

- xi is a decision variable for each node i∈V, xi=0 indicates that node i is not in the vertex cover.

Integer Programming: discrete variables.



Linear Programming: continuous variables.

- Drop the requirement that xi∈{0,1} and solve the LP in polynomial time to find {xi\*} between 0 & 1

- Assume S` is the optimal vertex cover set and w(S`) is the weight of the opt solution.

- Round values at least 1/2 up, and those below 1/2 down.

WLP = 





