NP-complete problems

Chapter 8 from reference book

Success of algorithmic endeavors

- So far in this course, we have seen efficient algorithmic solutions of variety of problems
  - Here "efficient" means run time of such algorithms are polynomial in problem size n;- for example O(n), $O(n^2)$, $O(n^3)$ etc.
  - List of such problems include
    - Finding shortest path, finding minimum spanning tree, matching in bipartite graph, maximum increasing subsequence, maximum flow in networks etc…
- To better appreciate such achievement, consider the alternatives
  - For all these problems we are searching for a solution from exponential possibilities
    - n boys can be matched with n girls in n! possible ways
    - A graph with n vertices can have n^{n-2} spanning trees
    - A typical graph has exponential number of paths from s to t
  - All these problems could in principle be solved in exponential time by checking through exponential number of candidate solutions
  - Quest of efficient algorithm is about finding clever ways to bypass the process of exhaustive search!

Failures of algorithmic endeavors

- Now is the time to meet the quest's most embarrassing and persistent failures
- The setting is still same as before
  - We are still searching for efficient solutions from exponential possibilities for some other search problems
  - But for these new problems no clever solutions exists
  - In spite of over 50 years of research by the best the mathematicians and computer scientists of this planet, the best known and fastest solutions for these problems are still exponential in problem size – not substantially better than exhaustive search
- Now is the time to introduce some of these "hard" problems

Hard problems

- Satisfiability (SAT)
  - Given a Boolean formula in the Conjunctive Normal Form (CNF), can we find a satisfying truth assignment to each variable so that the formula is true?

$$(x \vee y \vee z)\ (x \vee \overline{y})\ (y \vee \overline{z})\ (z \vee \overline{x})\ (\overline{x} \vee \overline{y} \vee \overline{z}).$$

  - With a little thought it is not hard to see that that the above problem has no possible truth assignment
  - SAT is a typical *search problem (or decision problem)* where we are given an instance I of the problem and is asked to find a solution S
- **Definition:** *A search problem (or decision problem) is specified by an algorithm C that takes two inputs, an instance I and a proposed solution S and runs in time polynomial in |I| (size of the problem). We say S is a solution to I if C(I,S)=true.*

Hard problems

- Traveling salesman problem (TSP)
  - We are given n vertices 1,…,n and all n(n-1)/2 distances between them, as well as a budget b. We are asked to find a *tour*, a cycle that passes through each vertex exactly once of total cost b or less– or to report that no such tours exists
- In reality one seeks to find the shortest possible tour which is an optimization problem. So why are we formulating TSP is as a search problem?
- We do this for two reasons
  - Our plan in this chapter is to compare and relate problems. The framework of search problem is helpful in this regard as it encompasses optimization problem like TSP in addition to search problems like SAT
  - Turning an optimization problem into a search problem does not increases its difficulty at all, because the two versions reduce one to another

Hard problems

- Reduction from search problem to optimization problem and vice versa
  - An algorithm that solves an optimization problem can readily be used to solve a search problem
    - Find the optimal cost of the shortest tour. If it is less than b return the tour, otherwise report no such tour exists
  - An algorithm that solves a search problem can also be used to solve an optimization problem
    - Start with an arbitrary cost b
    - If the search problem reports no solution exists double b and repeat
    - If the search problem finds a solution halve b and repeat
    - At the end, we get the cost of the optimal tour
    - This is simply an application of binary search!

Hard problems

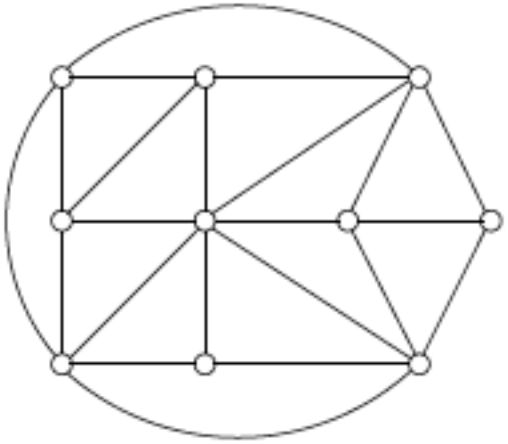
- HAMILTONIAN PATH
  - Given a graph and start vertex s and an end vertex t, find a path from s to t that goes through each vertex exactly once
- HAMILTONIAN CYCLE
  - Given a graph, find a cycle that visits each vertex exactly once.

Hard problems

- Three dimensional matching
  - There are n boys, n girls and n pets and compatibility among them is specified by a set of triples, each containing a boy, a girl and a pet. A triple (b,g,p) means boy b, girl g and pet p get along well together. We want to find n disjoint triples and thereby create n harmonious households.

Hard problems

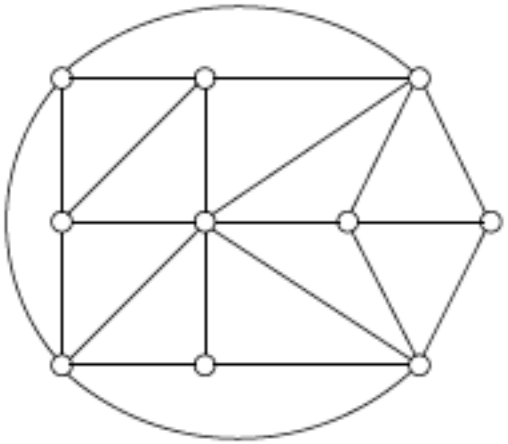
- INDEPENDENCE SET
  - In the INDEPENDENCE SET problem we are given a graph and an integer g, and the aim is to find g vertices that are independent, that is, now two of which have an edge between them



  - Can you find an independent set of 3 vertices in the above figure? Four vertices?

Hard problems

- VERTEX COVER
  - Here the input is a graph and a budget b, and we seek to find b vertices that cover (touch) every edge



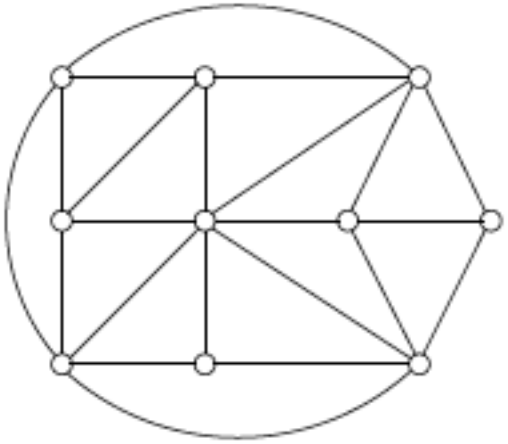
  - Can you cover all edges with seven vertices in the above figure? With six vertices?

Hard problems

- VERTEX COVER, SET COVER and 3D MATCHING
  - VERTEX COVER is a special case of SET COVER where we are given a set E and several of its subsets S_1, S_2,…,S_m and a budget b, and we are asked to select b of these subsets so that their union is E.
  - VERTEX COVER is a special case of SET COVER in which E consists of the edges of a graph and there is a subset S_i for each vertex, containing the edges adjacent to that vertex.
  - 3D MATCHING is also a special case of SET COVER (can you see why?)

Hard problems

- CLIQUE
  - Given a graph and a goal g, find a set of g vertices such that all possible edges between them are present.



  - What is the largest clique in the above figure?

Hard problems

- LONGEST PATH
  - Given a graph G with non-negative edge weights and two distinct vertices s and t, along with a goal g, we seek to find a simple path from s to t with total weight at least g.
    - We have seen that for directed acyclic graph efficient solution requiring O(V+E) time exists. But no efficient solution is known for general graph.

Hard problems

- SUBSET SUM
  - Given a set of integers and a separate integer W, that task is to find a subset of these integers that sum to W.
    - This is a special case of knapsack problem

Hard problems vs easy problems

- The world is full of search problems some are easy while others seem to be hard

Hard problems (NP-complete)	Easy problems (in P)
3SAT	2SAT, HORN SAT
TRAVELING SALESMAN PROBLEM	MINIMUM SPANNING TREE
LONGEST PATH	SHORTEST PATH
3D MATCHING	BIPARTITE MATCHING
KNAPSACK	UNARY KNAPSACK
INDEPENDENT SET	INDEPENDENT SET on trees
INTEGER LINEAR PROGRAMMING	LINEAR PROGRAMMING
RUDRATA PATH	EULER PATH
BALANCED CUT	MINIMUM CUT

- Observation
  - Problems on the right can be solved efficiently using different techniques, such as dynamic programming, network flow, graph search , greedy etc.
  - In stark contrast the problems on the left are *all difficult for the same reason.* At their core they are all the same problem just in different disguise. In fact, they are all equivalent.

P and NP

- The defining characteristic of a search problem is that any proposed solution can be checked quickly for correctness in the sense that there is an efficient checking algorithm C that takes as input the give instance I as well as a proposed solution S and outputs true if and only if S really is a solution for instance I. Moreover running time of C(I,S) is bounded by a polynomial in |I|, the length of the instance. We denote this class of all search problems by NP.
- We have seen many examples of NP search problems are indeed are solvable in polynomial (in |I|) time. In such cases, there is an algorithm that takes as input an instance I, runs in time polynomial in |I| and returns a solution if I has a solution and reports none otherwise. The class of all search problems that can be solved in polynomial time is denoted by P

P≠ NP?

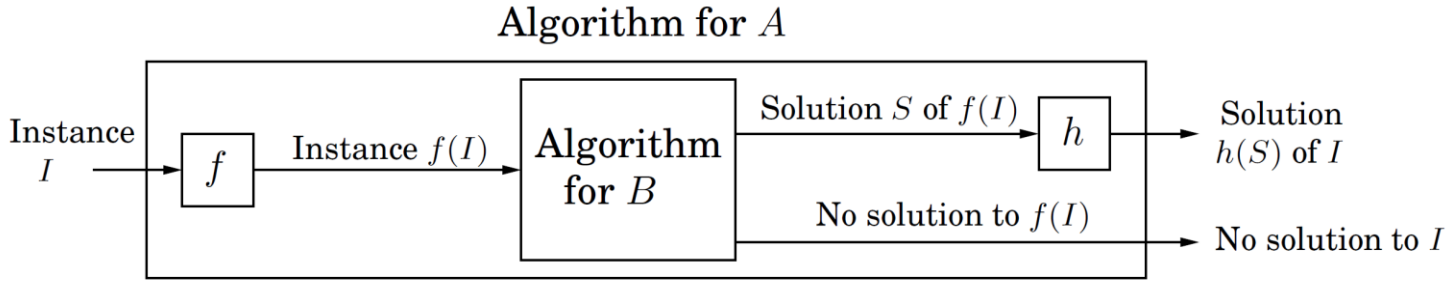
- Are there search problems that can not be solved in polynomial time, in other words, is P≠ NP?
  - Most algorithm researchers believe so
  - It is hard to believe that exponential search can always be avoided, in other words, a simple trick will crack all these hard problems famously unsolved for decades and centuries
  - However proving that P≠NP (or P=NP for that matter) has turned out to be extremely difficult and this is one of the deepest and most important puzzles in mathematics

Reductions

- Even if we accept P≠NP, what about the specific problems on the left side of the table show earlier?
- On the basis of what *evidence* do we believe that these particular problems have no efficient algorithm?
- Such evidence is provided by *reductions*, which translates one search problem into another.
  - What they demonstrate is that the problem on the left side of the table are all, in some sense, *exactly the same problem,* except that they are stated in different language.
- In fact we will use reductions to show that
  - These search problems are hardest search problems in NP
  - If even one of them has a polynomial time algorithm then every problem in NP has a polynomial time algorithm
- Thus if we believe P≠NP then all these search problems are hard

NP-completeness

- A reduction from search problem A to search problem B is a polynomial-time algorithm f that transforms any instance I of A into an instance f(I) of B, together with another polynomial-time algorithm h that maps any solution S of f(I) back to solution h(S) of I



  - If f(I) has no solution then neither does I
  - These two translation process f and h imply that any algorithm for B can be converted to an algorithm for A.
- A search problem is NP-complete if
  - It is in NP, and
  - All problems in NP can be reduced in polynomial-time to it

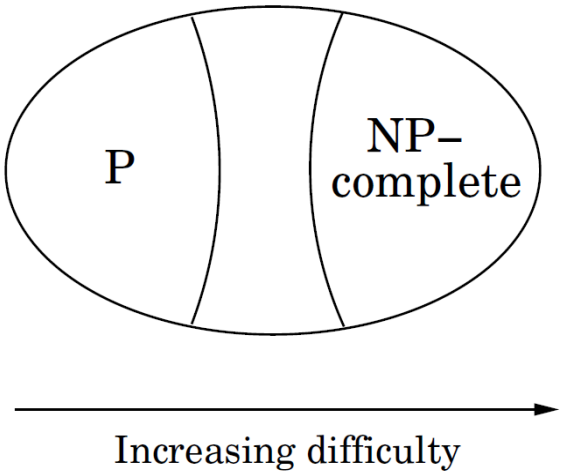
It is really surprising that NP-complete problems exist!

Two ways to use reduction

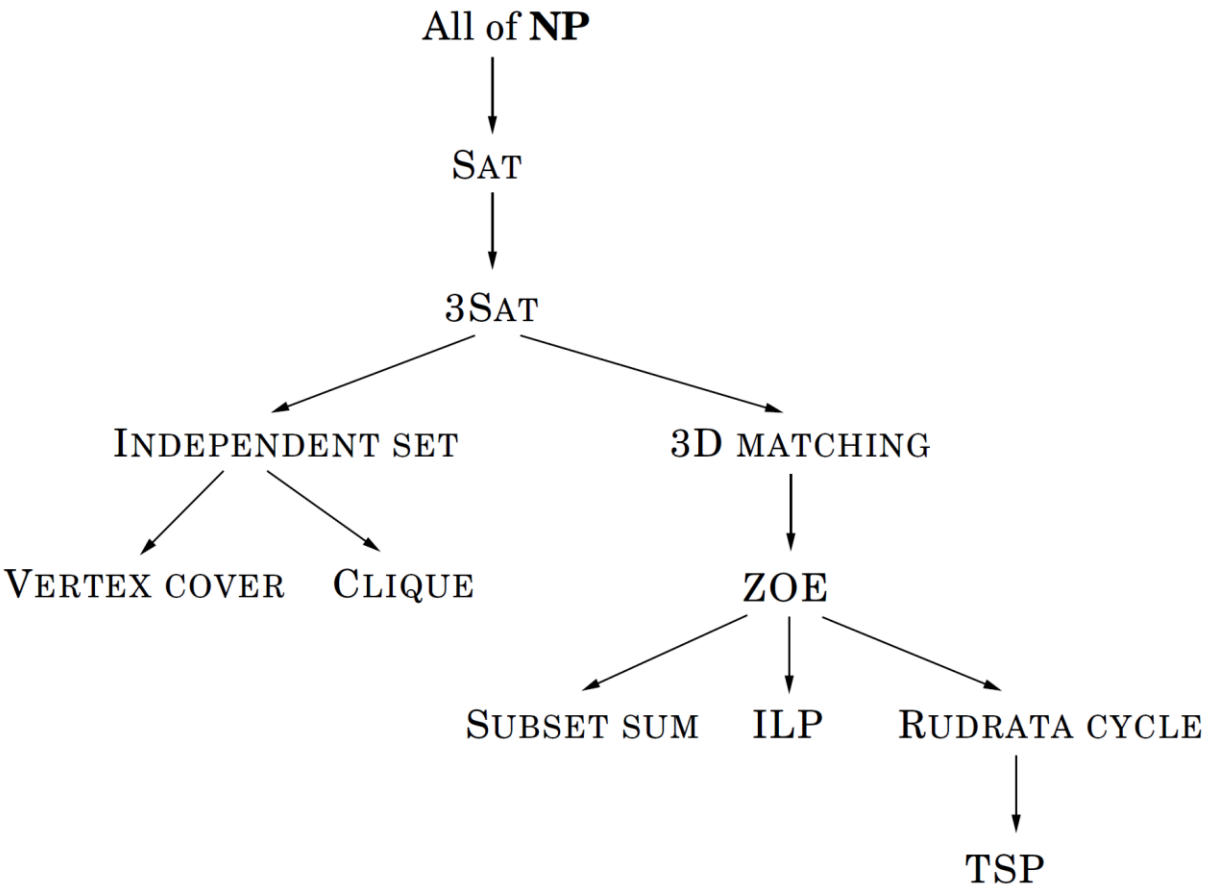
- There are two ways to use reduction that translates a search problem A to a search problem B
  - If B has an efficient algorithm then that algorithm can be used to solve A efficiently
  - Reduction from A to B also is also used to show that if we know that A is hard then we use the reduction to prove that B is hard as well.
    - The argument here is if we have an efficient (polynomial-time) algorithm for B, we could have used that algorithm to solve A efficiently, contradicting that A is hard and does not have an efficient (polynomial-time) algorithm
- Reductions also compose
  - If we have a reduction from A to B and another reduction from B to C, together they imply a reduction from A to C
  - If (f_{AB},h_{AB}) and (f_{BC},h_{BC}) define the reductions from A to B and from B to C, respectively, then a reduction from A to C is given by compositions of these functions
    - $f_{BC}(f_{AB})$ maps an instance of A to an instance of C and $h_{AB}(h_{BC})$ send the solution of C back to solution of A

Using reduction

- This means once we know a problem A is NP-complete, we can use to prove that a new search problem B is also NP-complete, simply by reducing A to B.
- Such a reduction establishes that all problems in NP reduce to B via A.



Reductions between search problems



NP-complete vs NP-hard

- A search problem is NP-complete if
  - It is in NP, and
  - All problems in NP can be reduced in polynomial-time to it
- A search problem is NP-hard if
  - All problems in NP can be reduced in polynomial-time to it
- That means NP-hard problems need not be in NP
  - They are at least as hard as NP-complete problems
  - Example: Halting problem, Given a program P and an Input I will it halt?
- Each NP-complete problem is also NP-hard

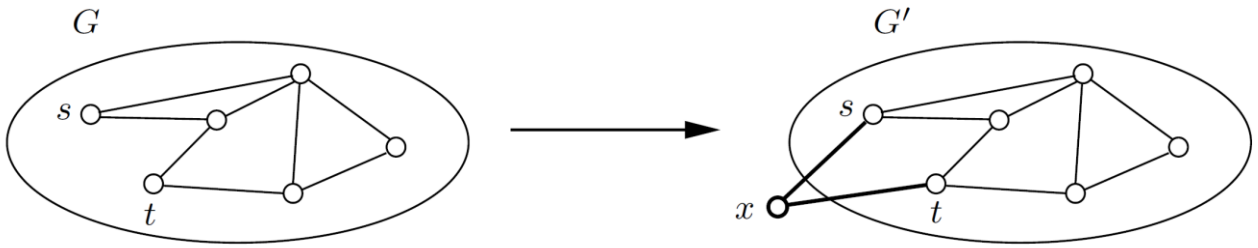
HAMILTONIAN CYCLE IS NP-COMPLETE

- We will use that fact that HAMILTONIAN PATH is NP-complete
- Now, we will simply reduce HAMILTONIAN PATH to HANILTONIAN CYCLE
  - Since HAMILTONIAN PATH is NP-complete, all problems in NP can be reduced to HAMILTONIAN PATH
  - If we can reduce HAMILTONIAN PATH to HAMILTONIAN CYCLE, this would mean all problems in NP can be reduced to HAMILTONIAN CYCLE

- To prove that a problem B is NP-complete, often the trick is to choose the correct NP-complete problem A such that reduction from A to B is simple.

HAMILTONIAN PATH to HAMILTONIAN CYCLE

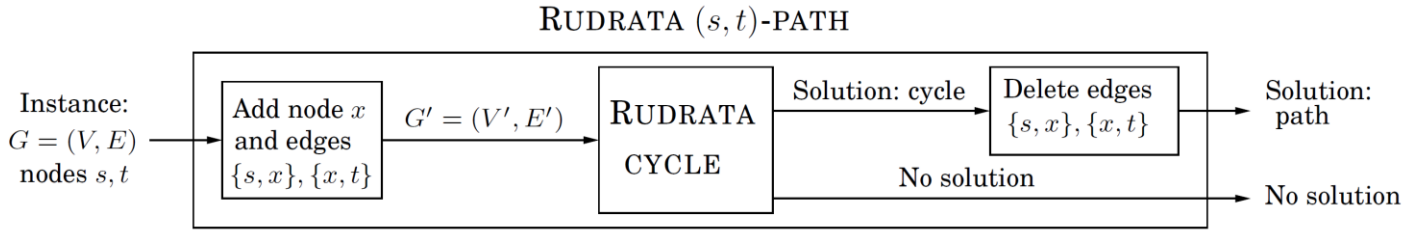
- Let (G=(V,E),s,t) be an instance of a HAMILTONIAN PATH from vertex s to vertex t that goes through each vertex. The reduction maps an instance (G=(V,E),s,t) of HAMILTONIAN PATH to an instance G'=(V',E') of HAMILTONIAN CYCLE as follows:
  - G' is simply G with an additional vertex x and two additional edges {s,x} and {x,t}. For instance:



  - So V'=V U {x} and E'=E U{{s,x},{x,t}}
  - How do we recover HAMILTONIAN PATH in G given any HANILTONIAN CYCLE in G'?
    - Easy, just delete the edge {s,x} and {x,t} from the cycle

HAMILTONIAN PATH to HAMILTONIAN CYCLE

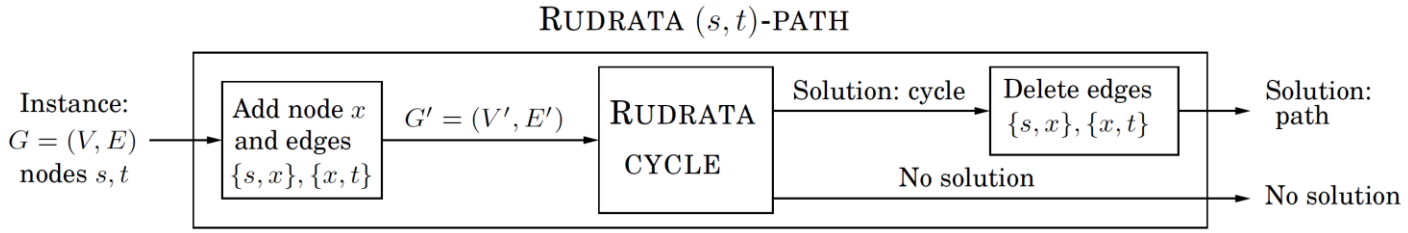
- To confirm validity of reduction, we have to show that works in case of either outcome



  - When the instance of HAMILTONIAN CYCLE has a solution
    - Since the new vertex x has only two neighbors s and t, any HAMILTONIAN CYCLE G' must consecutively traverse the edges {t,x} and {x,s}. Thus deleting the two edges from the HAMILTONIAN CYCLE gives a HAMILTONIAN PATH from s to t in the original graph G

HAMILTONIAN PATH to HAMILTONIAN CYCLE

- To confirm validity of reduction, we have to show that works in case of either outcome



  - When the instance of HAMILTONIAN CYCLE does not have a solution
    - In this case we must show that the original instance of HAMILTONIAN PATH can not have a solution either. It is usually easier to prove the contra positive, that is to show that if there is a HAMILTONIAN PATH from s to t in G, then there is also a HAMILTONIAN CYCLE in G'. But this is easy: just add two edges {t,x} and {x,s} to the HAMILTONIAN PATH to close the cycle.
    - Recall contraposition says "if A implies B" then "negation of B implies negation of A".

Other reductions

- For the other reductions please read the reference books.

- We will do the following reductions in class. Here the notation "A to B" means, we seek to prove that B is NP-complete. To achieve this, we simply pick a known NP-complete problem A and reduce A to B.

  - 3SAT to INDEPENDENCE SET
  - INDEPENDENCE SET to VERTEX COVER
  - INDEPENDENCE SET to CLIQUE
  - HAMILTONIAN CYCLE to TSP
  - 3COLOR