Chapter 8

NP-Completeness

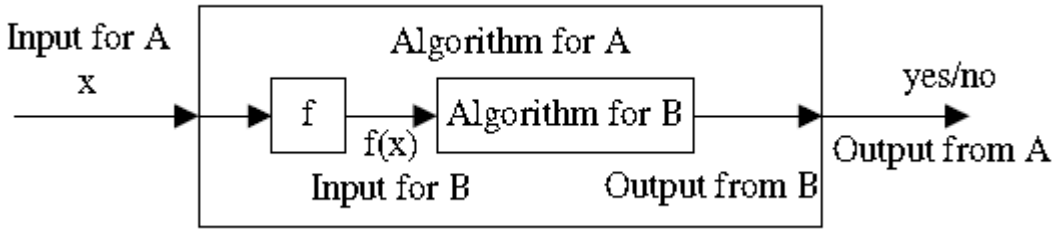
Most of the problems considered up to now can be solved by algorithms in worst-case polynomial time. There are many problems and it is not necessary that all the problems have the apparent solution. This concept, somehow, can be applied in solving the problem using the computers. The computer can solve: some problems in limited time e.g. sorting, some problems requires unmanageable amount of time e.g. Hamiltonian cycles, and some problems cannot be solved e.g. Halting Problem. In this section we concentrate on the specific class of problems called NP complete problems (will be defined later).

Tractable and Intractable Problems

We call problems as tractable or easy, if the problem can be solved using polynomial time algorithms. The problems that cannot be solved in polynomial time but requires superpolynomial time algorithm are called intractable or hard problems. There are many problems for which no algorithm with running time better than exponential time is known some of them are, traveling salesman problem, Hamiltonian cycles, and circuit satisfiability, etc.

Polynomial time reduction

Given two problems A and B, a polynomial time reduction from A to B is a polynomial time function f that transforms the instances of A into instances of B such that the output of algorithm for the problem A on input instance x must be same as the output of the algorithm for the problem B on input instance f(x) as shown in the figure below. If there is polynomial time computable function f such that it is possible to reduce A to B, then it is denoted as $A \leq_p B$. The function f described above is called reduction function and the algorithm for computing f is called reduction algorithm.



P and NP classes and NP completeness

The set of problems that can be solved using polynomial time algorithm is regarded as class P. The problems that are verifiable in polynomial time constitute the class NP. The class of NP complete problems consists of those problems that are NP as well as they are as hard as any problem in NP (more on this later). The main concern of studying NP completeness is to understand how hard the problem is. So if we can find some problem as NP complete then we try to solve the problem using methods like approximation, rather than searching for the faster algorithm for solving the problem exactly.

Complexity Class P

P is the class of problems that can be solved in polynomial time on a deterministic effective computing system (ECS). Loosely speaking, all computing machines that exist in the real world are deterministic ECSs. So P is the class of things that can be computed in polynomial time on real computers.

Complexity Class NP

NP is the class of problems that can be solved in polynomial time on a non-deterministic effective computing system (ECS) or we can say that "NP is the class of problems that can be solved in super polynomial time on a deterministic effective computing system (ECS)". Loosely speaking, all computing machines that exist in the real world are deterministic ECSs. So NP is the class of problem that can be computed in super polynomial time on real computers. But problem of class NP are verifiable in polynomial time. Using the above idea we say the problem is in class NP (nondeterministic polynomial time) if there is an algorithm for the problem that verifies the problem in polynomial time. For e.g. Circuit satisfiability problem (SAT) is the question "Given a Boolean combinational circuit, is it satisfiable? i.e. does the circuit has assignment sequence of truth values that produces the output of the circuit as 1?" Given the circuit satisfiability problem take a circuit x and a certificate y with the set of values that produce output 1, we can verify that whether the given certificate satisfies the circuit in polynomial time. So we can say that circuit satisfiability problem is NP.

Complexity Class NP-Complete

NP complete problems are those problems that are hardest problems in class NP. We define some problem say A, is NP-complete if

a. $A \in NP$, and
b. $B \leq_p A$, for every $B \in NP$.

The problem satisfying property b is called NP-hard.

NP-Complete problems arise in many domains like: boolean logic; graphs, sets and partitions; sequencing, scheduling, allocation; automata and language theory; network design; compilers, program optimization; hardware design/optimization; number theory, algebra etc.

Cook's Theorem

SAT is NP-complete

Proof

To prove that SAT is NP-complete, we have to show that

- SATεNP
- SAT is NP-Hard

SATεNP

Circuit satisfiability problem (SAT) is the question "Given a Boolean combinational circuit, is it satisfiable? i.e. does the circuit has assignment sequence of truth values that produces the output of the circuit as 1?" Given the circuit satisfiability problem take a circuit x and a certificate y with the set of values that produce output 1, we can verify that whether the given certificate satisfies the circuit in polynomial time. So we can say that circuit satisfiability problem is NP.

SAT is NP-hard

Take a problem V ε NP, let A be the algorithm that verifies V in polynomial time (this must be true since V ε NP). We can program A on a computer and therefore there exists a (huge) logical circuit whose input wires correspond to bits of the inputs x and y of A and which outputs 1 precisely when A(x,y) returns yes. For any instance x of V let Ax be the circuit obtained from A by setting the x-input wire values according to the specific string x. The construction of Ax from x is our reduction function.

Approximation Algorithms

An approximate algorithm is a way of dealing with NP-completeness for optimization problem. This technique does not guarantee the best solution. The goal of an approximation algorithm is to come as close as possible to the optimum value in a reasonable amount of time which is at most polynomial time. If we are dealing with optimization problem (maximization or minimization) with feasible solution having positive cost then it is worthy to look at approximate algorithm for near optimal solution.

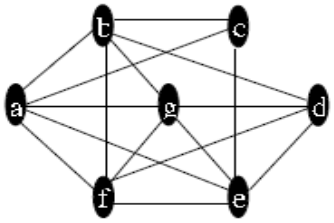
Vertex Cover Problem

A **vertex cover** of an undirected graph G =(V,E) is a subset V’ □ V such that for all edges (u,v) εE either uεV’ or vεV’ or u and v ε V’. The problem here is to find the vertex cover of minimum size in a given graph G. Optimal vertex-cover is the optimization version of an NP-complete problem but it is not too hard to find a vertex-cover that is near optimal.

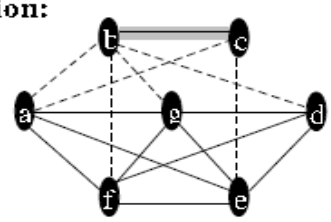
Algorithm

```
ApproxVertexCover (G)
{
        C ={ } ;
        E’ = E
        while E` is not empty
                do Let (u, v) be an arbitrary edge of E`
                C = C U {u, v}
                Remove from E` every edge incident on either u or v
        return C
}
```

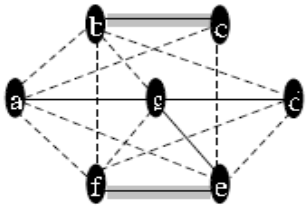
Example: (vertex cover running example for graph below)



Solution:



Edge chosen is (b,c) C = {b,c}



Edge chosen is (f,e) C = {b,c,e,f}