Theory of NP-Completeness

For many problems, algorithms for solving them can be divided into 2 groups:
1) Problems solvable by a polynomial time algorithm. E.g., Fractional knapsack problem.
2) Problems for which no polynomial time algorithm is known. E.g., TSP. Dynamic Programming takes $O(n^2 2^n)$.

No one has been successful in developing polynomial time algorithms for those in the second group.
Theory of NP-completeness
1) Does not provide a method for obtaining polynomial time algorithms for problems in the second group.
2) Does not say such algorithms do not exist
Then what?
It shows that many of the problems in the second group are related.

Nondeterministic algorithm:

Outcome of every operation is not uniquely defined but limited to a specified set of choices. A machine capable of executing nondeterministic algorithms is a nondeterministic machine which does not exists in practice.

Whenever there is a set of choices that lead to a successful termination, then one such set of choices is always made and the algorithm terminates successfully. A nondeterministic algorithm terminates unsuccessfully iff there exists no set of choices leading to a success signal.

Advantage of non-deterministic algorithms:

Often what could be very complex to write deterministically is very easy to write non-deterministically. In fact, it is very easy to obtain polynomial time non-deterministic algorithm for many problems that can be deterministically solved by a systematic search of a solution space of exponential size.

Any problem for which the answer is either 'yes' or 'no' is called a decision problem. An algorithm for a decision problem is called a decision algorithm.

Any problem for which we have to find out the optimal value of a function is called an optimization problem. An algorithm for an optimization problem is called an optimization algorithm.

Many optimization problems can be recast into decision problems so that the decision problem can be solved in polynomial time iff the corresponding optimization can.

Knapsack decision problem

//Assign values to xi's and see of the resulting profit is at least R.

```
Procedure DKP(P,W,n,M,R,X)
begin
for i = 1 to n do
        X(i) = choice (0,1)
endfor
if                  ❑       or                  ❑       then failure
else success;
endif
end
```

Time complexity is O(n).

Here, choice(0,1) chooses a value between 0 and 1; failure and success are like stop statement.

1. Use the decision algorithm to solve the optimization problem: Start with R=1, and apply the decision algorithm. Increase R until the answer is 'no'.
2. Use the optimization algorithm to solve the decision problem: Find the optimal value, V of the problem. Check whether V>=R.

Let P be the set of all decision problems solvable by a deterministic algorithm in polynomial time.
Let NP be the set of all decision problems solvable by a nondeterministic algorithm in polynomial time.

Deterministic algorithms are a special case of nondeterministic algorithms.
So, P is a subset of NP.

Most famous unsolved problem in Computer science:
Whether P=NP or P ≠ NP?

a) For all problems in NP, is it possible that there exists polynomial time deterministic algorithms which have remained undiscovered?
Unlikely

b) The answer to whether P ≠ NP? Is also elusive.
Cook's question:
Is there a problem in NP such that if we showed that it is in P, then it would imply that P=NP?
Yes. It is the Satisfiability problem.
Satisfiability problem:

Determine of a propositional calculus formula is true for some assignment of truth values to the variables.
E.g., (x3 or x4') AND (x1 or x2') - a formula in CNF

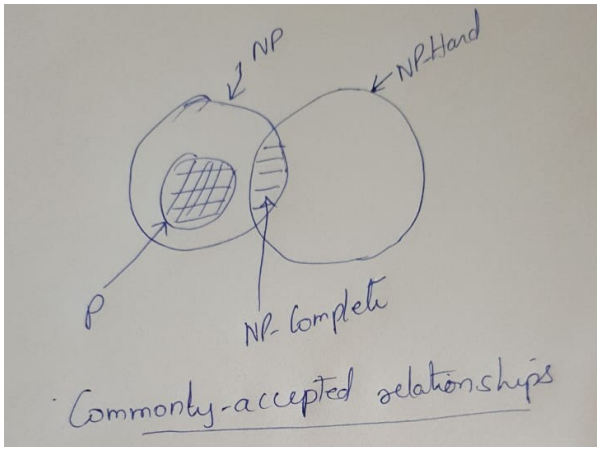
```
Procedure EVAL(E,n)
        //Determine if the propositional formula is satisfiable
begin
for i = 1 to no do
        x(i) = choose (true, false)
endfor
if E(x(1), x(2),…,x(n)) is true then success
else failure
endif
end
```

Cook's Theorem: Satisfiability is in P iff P=NP.

Reducibility: L1 reduces to L2 iff there is a way to solve L1 by a deterministic polynomial time algorithm using a deterministic algorithm that solves L2 in polynomial time.

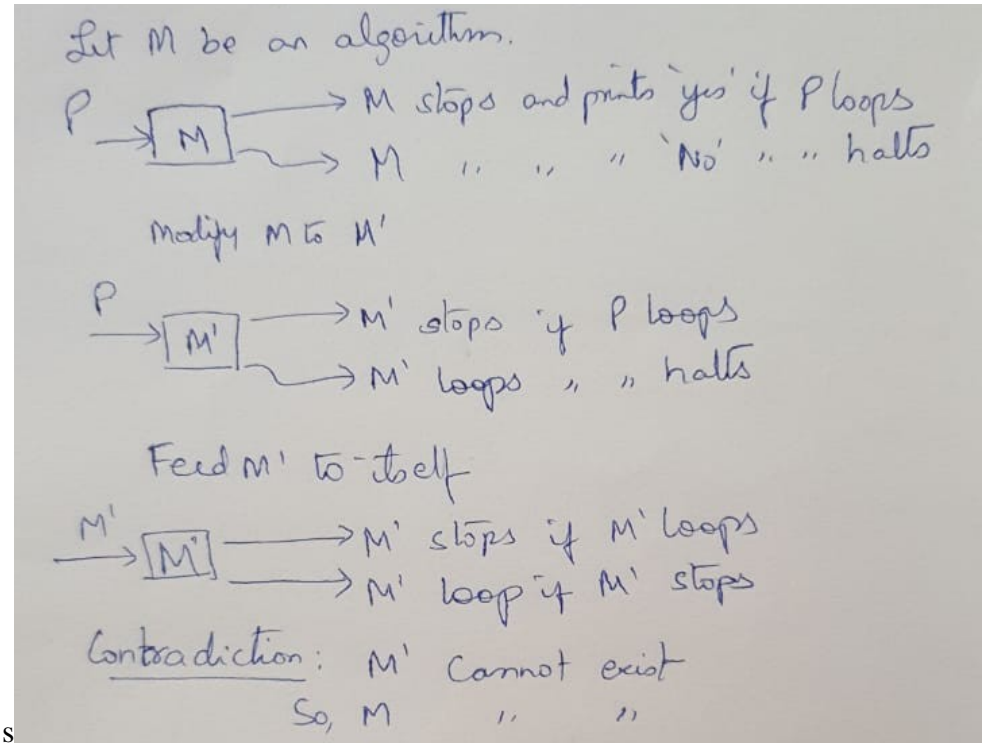
This defn. implies that:
If we have a polynomial time algorithm for L2, then we can solve L1 in polynomial time.

Defn. A problem L is NP-hard iff Satisfiability reduces to L.
Defn. A problem L is NP-complete iff L is NP-hard and L ϵNP.



Undecidable problem:
No algorithm exists of any complexity to solve this problem. The halting problem is an example of an undecidable problem. The proof is given in the picture below.

Halting problem: Will a program P terminate or not?
Let M be an algorithm that solves this problem.



S

Hence, there is no algorithm that solves the Halting problem. Consequently, there is no non-deterministic time algorithm that solves the Halting problem. Thus, Halting problem is not in NP.

However, Satisfiability reduces to the Halting problem (proof of this is not given here). Therefore Halting problem is NP-hard but not in NP.
