Lecture 01 : Overview of NP-completeness and How to Tackle It.

Welcome to this first lecture of Approximation Algorithm Design course. Let me give a brief overview of what we will study in this course. In today's world we know that we use computer programs or apps for various decision making in daily life for example, finding the route to take when we want to go from one city to another or maybe we want apps to help us recommend which news or news item we would like to see or which items we want to buy from various other applications like scheduling classes in a large institute like IITs and so on. But it often turns out that most of the problems are hard. Now, what do you mean by hard? So, let us formalize. So, broadly this area is called discrete optimization where the goal is to maximize or minimize some function over discrete space ok.

Once we have a discrete optimization problem we want to have an efficient algorithm. So, we want an efficient algorithm for discrete optimization problems. Now, it turns out that for all problems do not admit an efficient algorithm. By efficient we mean run time is polynomial in the input size. So, there is a complexity class called P is the set of problems that admit ah polynomial time algorithm.

So, here we are slightly abusing the notation because when we define complexity classes like P, NP we deal with decision version of the problem. So, the decision version of the corresponding optimization problem which admits a polynomial time algorithm. That set of problems is denoted by P and there is another famous class NP loosely speaking the set of problems which has a polynomial time verifiable certificate for every yes instance. You see in the definition of NP we are crucially using the fact that it is a decision version the answer is either yes or no. It is typically believed we believe that P is not equal to NP.

What are some examples of ah problems which are in P? For example, in the 2SAT problem we are given bunch of CNF clauses each CNF clause has exactly two literals, and we need to find out whether there exists an assignment to this variables which satisfy all clauses. That is 2SAT. Then we have shortest path with no negative weight cycles ok. Then we have matching. Given a graph G compute the maximum matching, what is the matching? It is a set of edges where no two edge share any end point. So, these are all problems and so on many more these problems admit polynomial time algorithm and hence this problem belongs to P the decision version. Similarly, examples of problems NP complete problems ok. So, NP complete class is the set of problems which are hardest in the class NP of course, every problem in P belongs to NP because P is a subset of NP.

The NP complete class is the set of all problems in NP which are hardest in the sense that if any one of them admits any polynomial time algorithm, then every problem in NP admits a polynomial time algorithm. So, it is typically believed that $P \neq NP$. So, to show $P = NP$ it is enough to show that some NP-complete problem admits a polynomial time algorithm. So, what are some examples? it is the 3 satisfiability problem where we are given n Boolean variables and clauses m clauses each clause is an or of 3 literals and we need to

find out whether there exist an assignment to this Boolean variables which satisfy all these clauses. Then we have longest path. Given a graph G and 2 vertices x and y, find the longest path between x and y, then 3 dimensional matching and so on. So, it turns out that most of the real world problems are NP complete. The examples of real world problems which are not NP complete which belongs to P are few. So, in this course we will design tools to tackle NP complete problems. Now, how do we typically tackle NP complete problems? way outs from NP completeness. For any problem which is NP complete there are 3 things which you cannot get simultaneously.

So, what are the 3 things? First one is finding an optimal solution, then running in polynomial time. And 3 is works for all instances ok. So, need to leave at least one unless $P=NP$ ok, which we consider to be unlikely. So, what are the approaches typically? The first approach is heuristics. So, can we have algorithms which work well in practice, but have no probable guarantee.

Algorithms which seem to. work well in practice and take small amount of time to execute ok. So, examples of such approaches are genetic algorithms, A^* search etc.

The problem with this approach is that it does not come with a guarantee. So, for a new instance it may happen that the algorithm either takes too long time or it does not give good solution, close to optimal solution. So, drawback does not have provable guarantee on both performance on both quality of solution and computation time. So, some heuristics may have guarantee on one of them may be it transfers, but it may not always give good solution for example, genetic algorithms or some heuristic like A^* may guarantee quality of solution it will find the optimal solution, but the runtime on some instances it may be prohibitive. So, these are the is the first approach.

The second approach is fixed parameter tractability. what is it? So, here you designate a parameter call k and design an algorithm which runs in time some function of k times polynomial in input size. So, the idea is if the parameter is small then your algorithm effectively runs in polynomial time and design algorithm which runs in polynomial time and always output a correct solution. The third approach which we will see in this course is approximation algorithm. Here the goal is to design an algorithm which runs in polynomial time and outputs are an approximately optimal solution with provable guarantee.

So, here definition what is the definition of an approximation factor? This is the provable guarantee approximation factor. So, an α approximation algorithm for a minimization problem. Alternatively maximization is a polynomial time algorithm which outputs a solution whose value is at most for minimization and for maximization at least respectively alpha times the value of an optimal solution. So, for minimization problem α will be greater than 1 and for maximization problem α will be less than 1. α is greater than equal to 1 and for maximization problem α is less than equal to 1 ok.

Now and as α goes towards 1, we have the better solution it is more close to optimal. So, how close we can go? So, the one target is what is called to have a polynomial time approximation scheme. petas for short. So, what is a PTAS? It is a family of algorithms A_ϵ for every ϵ greater than 0, where A_ϵ outputs or $1+\epsilon$ approximate solution for minimization problems and $1-\epsilon$ approximate solution for maximization problems. and runs in time $n^{f(\epsilon)}$.

So, for every constant ϵ the runtime is polynomial time and this is the fixed this is the polynomial time approximation scheme or PTAS for short. Now, there are some problems which admits PTAS and there is another thing is called FPTAS fully polynomial time approximation scheme. It is also a family of algorithms A_ϵ ϵ greater than 0 and it is a $1\pm\epsilon$ approximation algorithm depending on whether it is a maximization problem or minimization problem. Approximate solution in time polynomial in input size and $\frac{1}{\epsilon}$ ok. So, examples of problems in peters for example, Euclidean travelling salesman problem admits a PTAS.

On the other hand Knapsack problem admits an FPTAS. So, we will continue our study of approximation algorithm in the next class. So, let us stop