P,NP, NP-Hard and NP-Complete

- We can categorize the problem space into two parts
- Solvable Problems
- Unsolvable problems

Halting Problem

- *Given a description of a program and a finite input, decide whether the program finishes running in a finite time or will run forever, given that input*
- The halting problem is famous because it was one of the first problems proved undecidable, which means there is no computer program capable of correctly answering the question for all possible inputs

Post Correspondence Problem

- U1 U2 U3 U4
- aba bbb aab bb

- V1 v2 v3 v4
- a aaa abab babba

- A solution to this problem would be the sequence 1, 4, 3, 1 because
- $u_1u_4u_3u_1 = aba + bb + aab + aba = ababbaababa = a + babba + abab + a = v_1v_4v_3v_1$
- *The problem is to find whether such a solution exists or not*

Moving towards solution

- From not solvable we come to the concept of how well we can solve the problems
- logn- logaritihmic time
- n- linear time
- nlogn – near linear time
- n^2 – square time
- n^3 cubic time
- n^4 --- n^k - higher polynomial or super polynomial

P Class of Problems

- All those algorithms that can be solved in polynomial time with worst case running time as $O(n^k)$. So these problems are regarded as tractable.
- A problem that can be solved polynomially in one model can also be solved polynomially in other model.

Determinism

- If the result of every operation is uniquely defined then we call that algorithm as deterministic algorithm.
- If we remove this notion and we say that the output of an operation is not unique but limited to a set of possibilities
- In theoretical computer science, a non-deterministic Turing machine (NTM) is a Turing machine (TM) in which state transitions are not uniquely defined.

Non determinism

- A Non deterministic algorithm for searching an element x in a given set of elements A[1..n]
- 1. j= choice(1,n)
- 2. if A[j]=x then write (j); Success
- 3. else write(0); Failure

Non determinsitic sorting

- Algorithm Nsort
- For i=1 to n do B[i]=0;
- for i=1to n do
- { j= choice(1,n);
- if b[j]<>0 then failure();
  - B[j] = a[i]
  - }
  - for i= 1 to n-1 do
  - if B[i] > B[i+1] then failure;
  - success();

Non-determininstic Polynomial (NP)

- It is the set of all problems that can be solved in polynomial time with non-deterministic concept.
- However this concept only exists in theory. There are no implementations what so ever.

Decision Problems

- A decision problem is that problem which can have only two options as its solution space i.e. yes or no.
- Whether x is a prime number is also a decision problem that will have answer yes or no.
- For a Travelling salesperson problem the question that whether a tour of cost < k exists is a decision problem.

Guess & verify

- So for many problems it will be like guess & verify situation.
- We will guess a particular solution and then verify that whether this is the solution (yes) or not (no)
- If this guessing and verifying can be done in polynomial time with the help of a non deterministic concept then we say that the problem is in NP

Satisfiability Problem

- Satisfiability is the problem of determining if the variables of a given Boolean formula can be assigned in such a way as to make the formula evaluate to TRUE. Equally important is to determine whether no such assignments exist, which would imply that the function expressed by the formula is identically FALSE for all possible variable assignments. In this latter case, we would say that the function is unsatisfiable; otherwise it is satisfiable. To emphasize the binary nature of this problem, it is frequently referred to as Boolean or propositional satisfiability.

Satisfiability Problem

- if $f(x_1,x_2,\ldots\ldots x_n)$ = True
- x_i = 0 or 1
- we have to find $x_1,x_2,x_3 \ldots x_n$
- such that $f(b_1,b_2,b_3\ldots\ldots b_n)$=1 where b1= 0 or 1,b2= 0 or 1…….. bn = 0 or 1
- 2^n different permutations, someone out of that can give answer as 1 (true)
- guess
- ?f(1,0,1,0,0,1 ……,1,0) = 1
- verify (evaluate the function)
- ability to guess and verify in polynomial time

COOK theorem

- The complexity class P is contained in NP but the vice versa has not been proved and proving whether p= NP remains the biggest research question.
- Cook Formulated a question that is there any single problem in NP such that if we showed it to be in P, then that would imply P=NP

reducibility

- Given an instance x of a problem A, use a polynomial time reduction algorithm to transform it to instance y of problem B.
- Run the polynomial time algorithm for B on instance y
- use the answer for y as the answer for x.

NP Hard

- This problem A can be decision problem for some problem or a decision version for optimization problem. It may be or may not be part of NP class of problems. We call such problems as NP hard problems if they can be reduced to another problem B in polynomial time and then then that problem B can be solved in polynomial time and then that result of B can be used for giving solution of A in polynomial time.

NP complete

- if Problem is NP-hard and problem belongs to NP then it is NP-complete.
- Problems are designated "NP-complete" if their solutions can be quickly checked for correctness, and if the same solving algorithm used can solve all other NP problems. They are the most difficult problems in NP in the sense that a deterministic, polynomial-time solution to any NP-complete problem would provide a solution to every other problem in NP

Final judgement

- So we are able to correlate one problem which is NP hard with the another problem which is also NP hard and so we can say that if one of them can be solved in polynomial time then another can also be solved in polynomial time. Similarly it can be proved for all problems in NP.
- But the problem is that we don't have any polynomial solution for any of the problems in NP included both discussed. And also there is almost no hope. Only thing we can say here is that it has not been proved that P=NP and neither it has been proved that P<> NP so that can give us a little hope to work on and on.

- The time and resources of researchers taken by P=NP question exceeds any other problem in computer science.
- Most of the people are coming out with proof P ≠ NP which is yet to be verified. So there are more voters of P ≠ NP theory.
- The scientists are not waiting for the final word on this question to come and then think about the next strategy.
- The Problems that are in question under this dilemma are so important in our daily life and for the society that these can not be left unsolved.

Finding optimal solutions for specific categories

- Looking at the characteristics of the problem it should be explored whether it can be treated as some special case of a NP-complete problem or we can take advantage of any special feature or additional information given in the problem .

- Vertex cover is NP-complete but finding vertex cover for a bipartite graph can be done in polynomial time.

- Pseudo polynomial time algorithms may be better then their exponential counterparts.
  (An algorithm is said to run in pseudo polynomial time if its runtime is polynomial in the size of the input instance when the numbers in the input are represented in unary. Decimal 9 Binary 1001 Unary 111111111

Finding optimal solutions to NP-Complete problems

- Exponential algorithms may be adequate if problem size is small or problem is to be solved once.
- Some heuristics may be used which “work well in practice” but does not give any guarantees.
- Given the available resources, If exact solution is not available then we will try to find out a solution which is near to the optimal solution and this may be done in polynomial time.
- Difficult part is that at least we need to prove a solution’s value is close to optimum (and to what extent), without even knowing what optimum value is!

Approximation factor

If $SV_A(i)$: Solution value using Algorithm A on instance i

$SV_{OPT}(i)$: Solution value using Algorithm OPT on instance i

Assuming cost is positive always.

For Minimization problems : $SV_{OPT}(i) < SV_A(i)$

And for Maximization Problems : $SV_{OPT}(i) > SV_A(i)$

Approximation $factor_i$ or $ratio_i$ =

$SV_A(i)/ SV_{OPT}(i)$ *for Minimization*

$SV_{OPT}(i)/ SV_A(i)$ *for Maximization*

Approximation factor of A(n) = max{ approx. factors of all the instances of i of size n}

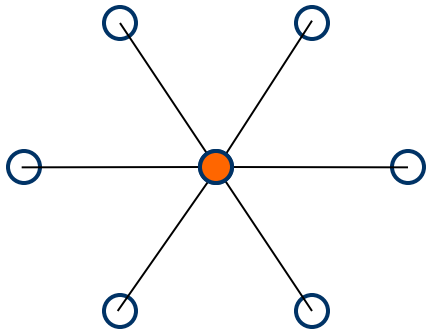
Objective of approximation

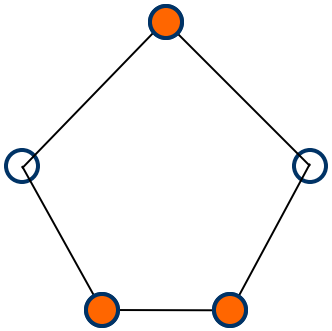
- Goal of the approximation algorithms is to design an algorithm such that approximation factor is as small as possible which means that we can go as close to the exact solution as possible in the polynomial time.
- It will be good if in some cases the complexity of the algorithm can be calculated in terms of the input size as well as the approximation factor.

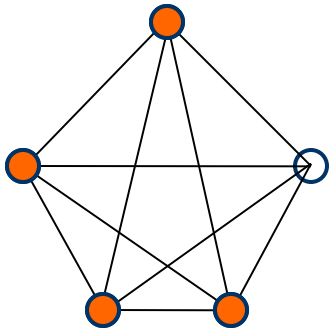
PTAS and FPTAS

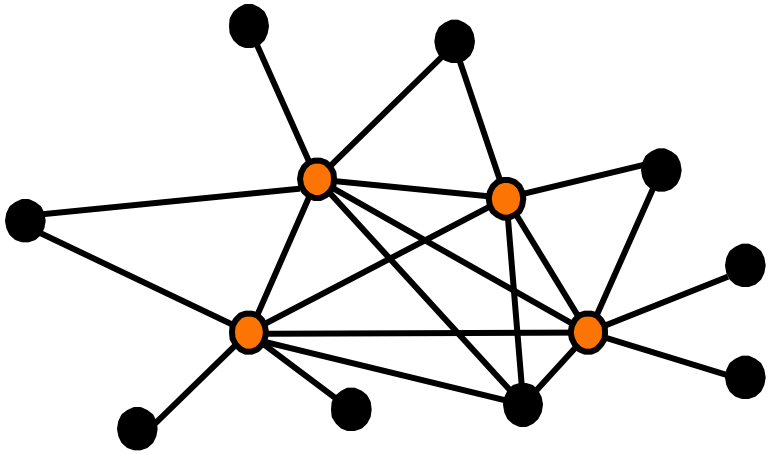
- Polynomial Time Approximation Scheme: It takes two arguments , an instance of the problem and a parameter $\varepsilon > 0$ and, in polynomial time, produces a solution that is within a factor $1 + \varepsilon$.
- For the TSP, a PTAS would produce a tour with length at most $(1 + \varepsilon)L$, where L is the length of the shortest tour.
- The running time of a PTAS is to be polynomial in n for every fixed ε but can be different for different ε.
- If the time is also polynomial in $1/ \varepsilon$ then it is called fully polynomial time approximation scheme (FPTAS).
- FPTAS for knapsack problem is n^3/ ε

Vertex cover









2-Approximation for vertex cover

1. Pick any edge {u,v} from set E of G
2. add both u & v to S
3. Delete u & v and all the incident edges from G
4. Repeat until any edge remains in G

Time taken by the above greedy algorithm is (V+E)

Output depends on the order we visit edges

Proof for 2-approximation vertex cover

- Every chosen edge e has both ends in S
- But e must be covered by an optimal cover; hence, one end of e must be in OPT
- Thus, there is at most twice as many vertices in C as in OPT.
- That is, S is a 2-approximation of OPT

- Best approximation ratio
- 2- (1/sqrt(logn)) [G Karakostas 2009 ACM trans on Algorithms]
- Best inapproximability
- (1+1/6) approximation algorithm unless P=NP [hastad Elseveir Science Direct Theoretical Computer Science 97]

Further approximation itself will be NP-Hard

- Best approximation ratio 1.5 [christofedes,CMU]
- Inapproximability 1+1/3812 [papadimitriou & yannakakis 93]
- General Approximation bound technique
- - Find lower bound on OPT
- - relate approximation algorithm to lower bound
- Other technique can be competing with the optimal

Techniques for Approximation algorithms

- 1. Greedy Algorithms
- 2. Primal Dual Technique
- 3. Linear Programming and Rounding
- 4. Integer Programming