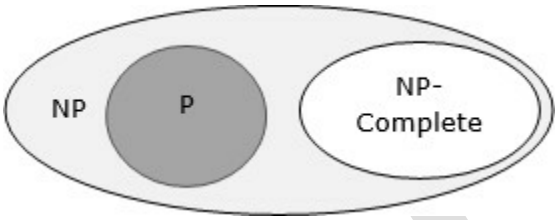
Q.Explain the proof of NP complete?

PROOF OF NP-COMPLETE AND NP-HARD PROBLEM

A problem is in the class NPC if it is in NP and is as **hard** as any problem in NP. A problem is **NP-hard** if all problems in NP are polynomial time reducible to it, even though it may not be in NP itself.



If a polynomial time algorithm exists for any of these problems, all problems in NP would be polynomial time solvable. These problems are called **NP-complete**. The phenomenon of NP-completeness is important for both theoretical and practical reasons.

Definition of NP-Completeness

A language **B** is ***NP-complete*** if it satisfies two conditions

- **B** is in NP
- Every **A** in NP is polynomial time reducible to **B**.

If a language satisfies the second property, but not necessarily the first one, the language **B** is known as **NP-Hard**. Informally, a search problem **B** is **NP-Hard** if there exists some **NP-Complete** problem **A** that Turing reduces to **B**.

The problem in NP-Hard cannot be solved in polynomial time, until **P = NP**. If a problem is proved to be NPC, there is no need to waste time on trying to find an efficient algorithm for it. Instead, we can focus on design approximation algorithm.

NP-Complete Problems

Following are some NP-Complete problems, for which no polynomial time algorithm is known.

- Determining whether a graph has a Hamiltonian cycle
- Determining whether a Boolean formula is satisfiable, etc.

NP-Hard Problems

The following problems are NP-Hard

- The circuit-satisfiability problem

- Set Cover
- Vertex Cover
- Travelling Salesman Problem

In this context, now we will discuss TSP is NP-Complete

- **TSP is NP-Complete**
- The traveling salesman problem consists of a salesman and a set of cities. The salesman has to visit each one of the cities starting from a certain one and returning to the same city. The challenge of the problem is that the traveling salesman wants to minimize the total length of the trip

- **Proof**
- To prove ***TSP is NP-Complete***, first we have to prove that ***TSP belongs to NP***. In TSP, we find a tour and check that the tour contains each vertex once. Then the total cost of the edges of the tour is calculated. Finally, we check if the cost is minimum. This can be completed in polynomial time. Thus ***TSP belongs to NP***.
- Secondly, we have to prove that ***TSP is NP-hard***. To prove this, one way is to show that ***Hamiltonian cycle*** $\leq_p$ ***TSP*** (as we know that the Hamiltonian cycle problem is NP-complete).
- Assume ***G = (V, E)*** to be an instance of Hamiltonian cycle.
- Hence, an instance of TSP is constructed. We create the complete graph $G' = (V, E')$, where
- E′={(i,j):i,j∈Vandi≠jE′={(i,j):i,j∈Vandi≠j Thus, the cost function is defined as follows −
- t(i,j)={01if(i,j)∈Eotherwiset(i,j)={0if(i,j)∈E1otherwise
- Now, suppose that a Hamiltonian cycle h exists in G. It is clear that the cost of each edge in h is **0** in G' as each edge belongs to E. Therefore, h has a cost of **0** in G'. Thus, if graph G has a Hamiltonian cycle, then graph G' has a tour of **0** cost.
- Conversely, we assume that G' has a tour h' of cost at most **0**. The cost of edges in E' are **0** and **1** by definition. Hence, each edge must have a cost of **0** as the cost of h' is **0**. We therefore conclude that h' contains only edges in E.We have thus proven that G has a Hamiltonian cycle, if and only if G' has a tour of cost at most **0**. TSP is NP-complete.

Q)Discuss in detail the components of linear programming and mention its applications?

LINEAR PROGRAMMING

Linear programming (LP) is a method to achieve the optimum outcome under some requirements represented by linear relationships. LP is an optimization technique for a system of linear constraints and a linear objective function. An objective function defines the quantity to be optimized and the goal of linear programming is to find the values of the variables that maximize or minimize the objective function subject to some linear constraints. In general, the standard form of LP consists of

- Variables: $\boldsymbol{x} = (x_1, x_2, \ldots, x_d)^\mathsf{T}$
- Objective function: $\boldsymbol{c} \cdot \boldsymbol{x}$
- Inequalities (constraints): $A\boldsymbol{x} \leq \boldsymbol{b}$, where A is a $n \times d$ matrix

and we maximize the objective function subject to the constraints and $x \geq 0$.

LP has many different applications, such as flow, shortest paths, and even politics. In this lecture, we will be covering different examples of LP, and present an algorithm for solving them. Linear programming is part of an important area of mathematics called "optimization techniques" as it is literally used to find the most optimized solution to a given problem. A very basic example of linear optimization usage is in logistics or the "method of moving things around efficiently."

For example, suppose there are 1000 boxes of the same size of 1 cubic meter each; 3 trucks that are able to carry 100 boxes, 70 boxes and 40 boxes respectively; several possible routes and 48 hours to deliver all the boxes. Linear programming provides the mathematical equations to determine the optimal truck loading and route to be taken in order to meet the requirement of getting all boxes from point A to B with the least amount of going back and forth and of course the lowest cost at the fastest time possible.

The basic components of linear programming are as follows:

- Decision variables - These are the quantities to be determined.
- Objective function - This represents how each decision variable would affect the cost, or, simply, the value that needs to be optimized.
- Constraints - These represent how each decision variable would use limited amounts of resources.
- Data - These quantify the relationships between the objective function and the constraints.

Applications of Linear Programming

Linear programming and Optimization are used in various industries. The manufacturing and service industry uses linear programming on a regular basis. In this section, we are going to look at the various applications of Linear programming.

Manufacturing industries use linear programming for **analyzing their supply chain operations**. Their motive is to maximize efficiency with minimum operation cost. As per the recommendations from the linear programming model, the manufacturer can reconfigure their storage layout, adjust their workforce and reduce the bottlenecks.

1. Linear programming is also used in organized retail for **shelf space optimization**. Since the number of products in the market has increased in leaps and bounds, it is important to understand what does the customer want. Optimization is aggressively used in stores like Walmart, Hypercity, Reliance, Big Bazaar, etc. The products in the store are placed strategically keeping in mind the customer shopping pattern. The objective is to make it easy for a customer to locate & select the right products. This is subject to constraints like limited shelf space, a variety of products, etc.
2. Optimization is also used for **optimizing Delivery Routes**. This is an extension of the popular traveling salesman problem. The service industry uses optimization for finding the best route for

multiple salesmen traveling to multiple cities. With the help of clustering and greedy algorithm, the delivery routes are decided by companies like FedEx, Amazon, etc. The objective is to minimize the operation cost and time.

3. Optimizations are also used in **Machine Learning**. Supervised Learning works on the fundamental of linear programming. A system is trained to fit on a mathematical model of a function from the labeled input data that can predict values from an unknown test data

Q)What are NP-hardness and NP-completeness problems explain with examples?

NP –HARD AND NP – COMPLETE PROBLEMS

Group2 – contains problems whose best known algorithms are non-polynomial.

Example – knapsack problem 0(2n/2) etc. There are two classes of non-polynomial time problems

1. NP- hard
2. NP-complete

A problem which is NP complete will have the property that it can be solved in polynomial time if all other NP – complete problems can also be solved in polynomial time. The class NP (meaning non-deterministic polynomial time) is the set of problems that might appear in a puzzle magazine: ``Nice puzzle."

What makes these problems special is that they might be hard to solve, but a short answer can always be printed in the back and it is easy to see that the answer is correct once you see it.

Example... Does matrix A have LU
decomposition No guarantee if answer is ``no".

Exponential Upper bound

- Another useful property of the class NP is that all NP problems can be solved in exponential time (EXP).
- This is because we can always list out all short certificates in exponential time and check all O (2nk) of them.
- Thus, P is in NPand NP is in EXP. Although we know that P is not equal to EXP, it is possible that NP = P, or EXP, or neither.

NP-hardness

Some problems are at least as hard to solve as any problem in NP. We call such problems NP-hard.

How might we argue that problem X is at least as hard (to within a polynomial factor) as problem Y?

If X is at least as hard as Y, how would we expect an algorithm that is able to solve X to behave?

Example of NP-Complete and NP-Hard

Problem Chromatic Number Decision Problem (CNP)

i. A coloring of a graph G = (V,E) is a function f : V { 1,2, …, k} i V

ii. If (U, V) E then f(u) f(v).
iii. The CNP is to determine if G has a coloring for a given K.
iv. Satisfiability with at most three literals per clause chromatic number problem. CNP is NP-hard.

Directed Hamiltonian Cycle(DHC)

i. Let G=(V,E) be a directed graph and length n=1V1
ii. The DHC is acycle that goes through every vertex exactly once and then returns to the starting vertex.
iii. The DHC problem is to determine if G has a directed Hamiltonian Cycle.
iv. Theorem: CNF (Conjunctive Normal Form) satisfiability DHC is NP-hard.

Travelling Salesperson Decision Problem (TSP) :

i. The problem is to determine if a complete directed graph G = (V,E) with edge costs C(u,v) has a tour of cost at most M Theorem: Directed Hamiltonian Cycle (DHC) TSP
ii. But from problem (2) satisfiability DHC Satisfiability TSP, TSP is NP-hard.

- CNF-Satisfiability with at most three literals per clause is NP-hard. If each clause is restricted to have at most two literals then CNF-satisfiability is polynomial solvable.
- Generating optimal code for a parallel assignment statement is NP-hard, However if the expressions are restricted to be simple variables, then optimal code can be generated in polynomial time.
- Generating optimal code for level one directed a-cyclic graphs is NP-hard but optimal code for trees can be generated in polynomial time. Determining if a planner graph is three colorable is NP-Hard. To determine if it is two colorable is a polynomial complexity problem. (It only have to see if it is bipartite)

Q)Write the approximation algorithm and explain in detail?

APPROXIMATION ALGORITHM:

Introduction:

An Approximate Algorithm is a way of approach **NP-COMPLETENESS** for the optimization problem.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 the most polynomial time. Such algorithms are called approximation algorithm or heuristic algorithm.

- For the traveling salesperson problem, the optimization problem is to find the shortest cycle, and the approximation problem is to find a short cycle.
- For the vertex cover problem, the optimization problem is to find the vertex cover with fewest vertices and the approximation problem is to find the vertex cover with few vertices.

Performance Ratios:

Suppose we work on an optimization problem where every solution carries a cost. An Approximate Algorithm returns a legal solution, but the cost of that legal solution may not be optimal.

- For Example, suppose we are considering for a **minimum size vertex-cover (VC)**. An approximate algorithm returns a VC for us, but the size (cost) may not be minimized.
- Another Example is we are considering for a **maximum size Independent set (IS)**. An approximate algorithm returns an IS for us, but the size (cost) may not be maximum. Let C be the cost of the solution returned by an approximate algorithm and C* is the cost of the optimal solution. We say the approximate algorithm has an approximate ratio P (n) for an input size n, where

$$\max\left(\frac{C}{C^*}, \frac{C^*}{C}\right) \leq P(n)$$

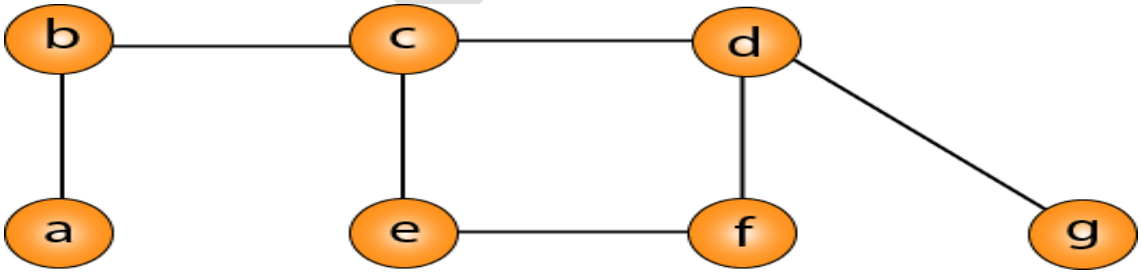
Intuitively, the approximation ratio measures how bad the approximate solution is distinguished with the optimal solution. A large (small) approximation ratio measures the solution is much worse than (more or less the same as) an optimal solution.

Observe that P (n) is always ≥ 1, if the ratio does not depend on n, we may write P. Therefore, a 1-approximation algorithm gives an optimal solution. Some problems have polynomial-time approximation algorithm with small constant approximate ratios, while others have best-known polynomial time approximation algorithms whose approximate ratios grow with n.

Vertex Cover

A Vertex Cover of a graph G is a set of vertices such that each edge in G is incident to at least one of these vertices. The decision vertex-cover problem was proven NPC. Now, we want to solve the optimal version of the vertex cover problem, i.e., we want to find a minimum size vertex cover of a given graph. We call such vertex cover an optimal vertex cover C*.

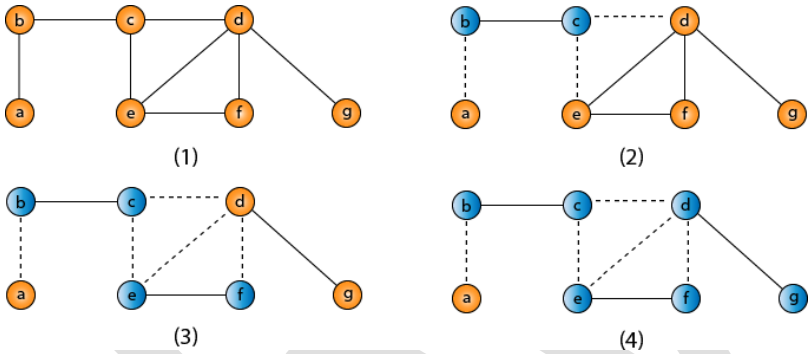
An approximate algorithm for vertex cover:




```
Approx-Vertex-Cover (G = (V, E))
{
   C = empty-set;
   E'= E;
   While E' is not empty do
     {
  Let (u, v) be any edge in E': (*)
```

Add u and v to C;
Remove from E' all edges incident to
u or v;
}
Return C;
}

The idea is to take an edge (u, v) one by one, put both vertices to C, and remove all the edges incident to u or v. We carry on until all edges have been removed. C is a VC. But how good is C?



VC = {b, c, d, e, f, g}

Traveling-salesman Problem

- In the traveling salesman Problem, a salesman must visits n cities. We can say that salesman wishes to make a tour or Hamiltonian cycle, visiting each city exactly once and finishing at the city he starts from. There is a non-negative cost c (i, j) to travel from the city i to city j.
- The goal is to find a tour of minimum cost. We assume that every two cities are connected. Such problems are called Traveling-salesman problem (TSP).
- We can model the cities as a complete graph of n vertices, where each vertex represents a city.It can be shown that TSP is NPC.
- If we assume the cost function c satisfies the triangle inequality, then we can use the following approximate algorithm.

Triangle inequality

Let u, v, w be any three vertices, we have

$$c(u, w) \leq c(u, v) + c(v, w)$$

One important observation to develop an approximate solution is if we remove an edge from H*, then the tour becomes a spanning tree.

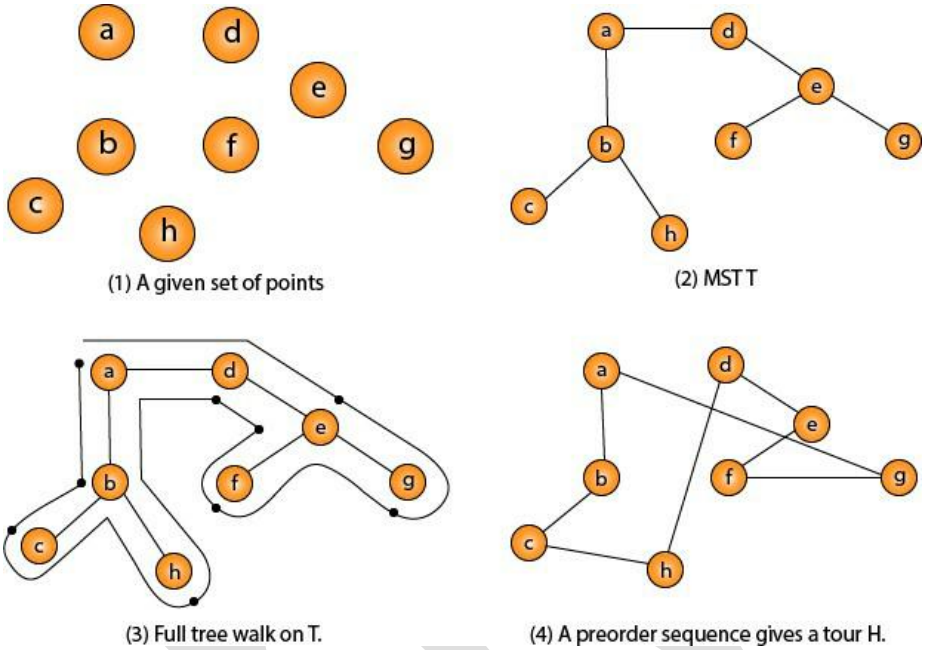
Algorithm:

Approx-TSP (G= (V, E))
{ 1. Compute a MST T of G;
2. Select any vertex r is the root of the tree;
3. Let L be the list of vertices visited in a preorder tree walk of T;

4. Return the Hamiltonian cycle H that visits the vertices in the order L;

}

Traveling-Salesman Problem



Intuitively, Approx-TSP first makes a full walk of MST T, which visits each edge exactly two times. To create a Hamiltonian cycle from the full walk, it bypasses some vertices (which corresponds to making a shortcut)

Q)Briefly discuss about randomized algorithm with an example?

RANDOMIZED ALGORITHM

What is a Randomized Algorithm?

- An algorithm that uses random numbers to decide what to do next anywhere in its logic is called Randomized Algorithm.
- For example, in Randomized Quick Sort, we use random number to pick the next pivot (or we randomly shuffle the array) and in Karger's algorithm, we randomly pick an edge.

How to analyse Randomized Algorithms?

- Some randomized algorithms have deterministic time complexity. For example, this implementation of Karger's algorithm has time complexity as O(E).
- Such algorithms are called Monte Carlo Algorithms and are easier to analyse for worst case.
- On the other hand, time complexity of other randomized algorithms is dependent

on value of random variable. Such Randomized algorithms are called Las Vegas Algorithms.

- These algorithms are typically analysed for expected worst case. To compute expected time taken in worst case, all possible values of the used random variable needs to be considered in worst case and time taken by every possible value needs to be evaluated.

Average of all evaluated times is the expected worst case time complexity. Below facts are generally helpful in analysis of such algorithms. Linearity of expectation expected number of trials until Success.

For example consider below a randomized version of Quick Sort.

A **Central Pivot** is a pivot that divides the array in such a way that one side has at-least 1/4

```
// Sorts an array arr[low..high]randQuickSort(arr[], low, high)

1. If low >= high, then EXIT.
2. While pivot 'x' is not a Central Pivot.
  (i)  Choose uniformly at random a number from [low..high].
       Let the randomly picked numberbex.
  (ii)  Count elements in arr[low..high] that are smaller
thanarr[x]. Let this count be sc.
  (iii)  Count elements in arr[low..high] that are greater
thanarr[x]. Let this count be gc.
(iv)  Letn = (high-low+1). If sc>= n/4 and
gc>= n/4, then x is a central pivot.
3. Partition arr[low..high] around the pivot x.
4. // Recur for smaller
```

elements.

Based on analysis is, the time taken by step 2 is O(n).

How many times while loop runs before finding a central pivot?

The probability that the randomly chosen element is central pivot is 1/2.

- Therefore, expected number of times the while loop runs is 2
- Thus, the expected time complexity of step 2 is O(n)

What is overall Time Complexity in Worst Case?

- In worst case, each partition divides array such that one side has n/4 elements and other side has 3n/4 elements. The worst case height of recursion tree is Log 3/4 n which is O(Log n).

$T(n) < T(n/4) + T(3n/4) + O(n)$

$T(n) < 2T(3n/4) + O(n)$

- Solution of above recurrence is O(n Log n)
- Note that the above randomized algorithm is not the best way to implement randomized Quick Sort. The idea here is to simplify the analysis as it is simple to analyse.
- Typically, randomized Quick Sort is implemented by randomly picking a pivot (no loop) or by shuffling array elements. Expected worst case time complexity of this algorithm is also O(n Log n), but analysis is complex.

Q)Discuss interior point method in detail?

INTERIOR POINT METHOD

Interior-point methods (also referred to as **barrier methods** or **IPMs**) are a certain class of algorithms that solve linear and nonlinear convex optimization problems.

Example: John von Neumann suggested an interior-point method of linear programming, which was neither a polynomial-time method nor an efficient method in practice. In fact, it turned out to be slower than the commonly used simplex method.

- An interior point method discovered by Soviet mathematician I. I. Dikin in 1967 and reinvented in the U.S. in the mid-1980s. In 1984, NarendraKarmarkar developed a method for linear programming called Karmarkar's algorithm, which runs in provably polynomial time and is also very efficient in practice.
- It enabled solutions of linear programming problems that were beyond the capabilities of the simplex method. Contrary to the simplex method, it reaches a best solution by traversing the interior of the feasible region. The method can be generalized to convex programming based on a self- concordant barrier function used to encode the convex set.
- Any convex optimization problem can be transformed into minimizing (or maximizing) a linear function over a convex set by converting to the epigraph form.
- A special class of such barriers that can be used to encode any convex set. They guarantee that the number of iterations of the algorithm is bounded by a polynomial in the dimension and accuracy of the solution.
- Karmarkar's breakthrough revitalized the study of interior-point methods and barrier problems, showing that it was possible to create an algorithm for linear programming characterized by polynomial complexity and, moreover, that

was competitive with the simplex method. Already Khachiyan's ellipsoid method was a polynomial-time algorithm; however, it was too slow to be of practical interest.

- The class of primal-dual path-following interior-point methods is considered the most successful. Mehrotra's predictor–corrector algorithm provides the basis for most implementations of this class of methods.

Primal dual interior Point Method for non-linear Optimization

The primal-dual method's idea is easy to demonstrate for constrained non-linear optimization. For simplicity, consider the all-inequality version of a non-linear optimization problem:

minimize $f(x)$ subject to $c_i(x) \geq 0 \text{ for } i = 1, \ldots, m, \ x \in \mathbb{R}^n,$ where $f : \mathbb{R}^n \to \mathbb{R}, c_i : \mathbb{R}^n \to \mathbb{R}$ (1).

The logarithmic barrier function associated with (1) is

$$B(x, \mu) = f(x) - \mu \sum_{i=1}^{m} \log(c_i(x)). \quad (2)$$

- Here μ is a small positive scalar, sometimes called the "barrier parameter". As μ converges to zero the minimum of B(x,μ) should converge to a solution of (1).

The barrier function gradient is

$$g_b = g - \mu \sum_{i=1}^{m} \frac{1}{c_i(x)} \nabla c_i(x), \quad (3)$$

where g is the gradient of the original function f(x) .

Q) Explain how interior method can be used to achieve optimization?

Interior-point methods (also referred to as barrier methods or IPMs) are a certain class of algorithms that solve linear and nonlinear convex optimization problems. Interior point methods are a type of algorithm that are used in solving both linear and nonlinear convex optimization problems that contain inequalities as constraints. The LP Interior-Point method relies on having a linear programming model with the objective function and all constraints being continuous and twice continuously differentiable. In general, a problem is assumed to be strictly feasible, and

will have a dual optimal that will satisfy Karush-Kuhn-Tucker (KKT) constraints described below. The problem is solved (assuming there IS a solution) either by iteratively solving for KKT conditions or to the original problem with equality instead of inequality constraints, and then applying Newton's method to these conditions.

Interior point methods came about from a desire for algorithms with better theoretical bases than the simplex method. While the two strategies are similar in a few ways, the interior point methods involve relatively expensive (in terms of computing) iterations that quickly close in on a solution, while the simplex method involves usually requires many more inexpensive iterations. From a geometric standpoint, interior point methods approach a solution from the interior or exterior of the feasible region, but are never on the boundary.[1]

There are two important interior point algorithms: the barrier method and primal-dual IP method. The primal-dual method is usually preferred due to its efficiency and accuracy. Major differences between the two methods are as follows. There is only one loop/iteration in primal-dual because there is no distinction between outer and inner iterations as with the barrier method. In primal-dual, the primal and dual iterates do not have to be feasible.

2 types –

Barrier Method

Algorithm

Given strictly feasible $x, t := t^0 > 0, \mu > 1, \epsilon < 0$

repeat

1. Compute $x^*(t)$ by minimizing $tf_0 + \phi$ subject to $Ax = b$, starting at x.
2. Update $x := x^*(t)$.
3. Quit if $\frac{m}{t} \leq \epsilon$, else
4. Increase $t := \mu t$ [3]

Primal-Dual IP Algorithm:

The primal-dual interior-point method can easily be understood by using the simplest NLP problem; one with only inequality constraints. https://optimization.mccormick.northwestern.edu/index.php/Interior-point_method_for_LP

CONCLUSION: The Interior Point method approximates the constraints of a linear programming model as a set of boundaries surrounding a region. These approximations are used

when the problem has constraints that are discontinuous or otherwise troublesome, but can me modified so that a linear solver can handle them. Once the problem is formulated in the correct way, Newton's method is used to iteratively approach more and more optimal solutions within the feasible region. Two practical algorithms exist in IP method: barrier and primal-dual. Primal-dual method is a more promising way to solve larger problems with more efficiency and accuracy. As shown in the figure above, the number of iterations needed for the primal-dual method to solve a problem increases logarithmically with the number of variables, and standard error only increases rapidly when a very large number of dimensions exist.

Q)How linear programming method can be used to improve maximization with an example?

Q)Write the algorithm for Randomized problem?

An algorithm that uses random numbers to decide what to do next anywhere in its logic is called Randomized Algorithm. For example, in Randomized Quick Sort, we use random number to pick the next pivot (or we randomly shuffle the array). Typically, this randomness is used to reduce time complexity or space complexity in other standard algorithms. In common practice, randomized algorithms are approximated using a pseudorandom number generator in place of a true source of random bits; such an implementation may deviate from the expected theoretical behavior.

Examples

Quicksort

Quicksort is a familiar, commonly used algorithm in which randomness can be useful. Many deterministic versions of this algorithm require O(n2) time to sort n numbers for some well-defined class of degenerate inputs (such as an already sorted array), with the specific class of inputs that generate this behavior defined by the protocol for pivot selection. However, if the algorithm selects pivot elements uniformly at random, it has a provably high probability of finishing in O(n log n) time regardless of the characteristics of the input.

Randomized incremental constructions in geometry

In computational geometry, a standard technique to build a structure like a convex hull or Delaunay triangulation is to randomly permute the input points and then insert them one by one into the existing structure. The randomization ensures that the expected number of changes to the structure caused by an insertion is small, and so the expected running time of the algorithm can be bounded from above. This technique is known as randomized incremental construction.[5]

Min cut

Input: A graph G(V,E)

Output: A cut partitioning the vertices into L and R, with the minimum number of edges between L and R.

Recall that the contraction of two nodes, u and v, in a (multi-)graph yields a new node u ' with edges that are the union of the edges incident on either u or v, except from any edge(s) connecting u and v. Figure 1 gives an example of contraction of vertex A and B. After contraction, the resulting graph may have parallel edges, but contains no self loops.

https://en.wikipedia.org/wiki/Randomized_algorithm#:~:text=A%20randomized%20algorithm%20is%20an,as%20part%20of%20its%20logic.&text=In%20common%20practice%2C%20randomized%20algorithms,from%20the%20expected%20theoretical%20behavior