Unit V Approximation and Randomized Algorithms, Natural Algorithms

PARTs 1-2-3-4

Unit V Approximation and Randomized Algorithms, Natural Algorithms

PART 1-Approximation algorithms, Solving TSP by approximation algorithm, approximating Max Clique

PART2-Concept of randomized algorithms, randomized quicksort algorithms,

PART3-Natural Algorithms–Evolutionary Computing and Evolutionary Algorithms, Simulated Annealing

PART4-Introduction to Genetic Algorithm

Approximation Algorithm

- Called **heuristic** algorithms also
- It is a way of **dealing with NP-completeness** for an optimization problem.
- This technique does **not** guarantee the **best solution**
- Goal : come **as close as possible** to the **optimal solution** in **polynomial time**

Features of Approximation Algorithm

- Guarantees to **run in polynomial time** though it does not guarantee the most effective solution.
- **Guarantees to seek out high accuracy and top quality solution** (say within 1% of optimum)
- Thus, used to get an answer near the (optimal) solution of an optimization problem in polynomial time

Performance Ratios for approximation algorithms **(1)**

Suppose that we are working on an optimization problem in which each **potential solution has a cost**, and we wish to find a **near-optimal solution**

- Depending on the problem, optimal solution as one with maximum possible cost /minimum possible cost
  - i.e, can be Maximization / Minimization problem
- We say that an algorithm for a problem has an appropriate ratio of P(n) if, for any input size n, the **cost C** of the solution produced by the algorithm is within a factor of P(n) of the **cost C* of an optimal solution** as follows.
- max(C/C*, C*/C) <= P(n)

Performance Ratios for approximation algorithms **(2)**

If an algorithm reaches an approximation ratio of P(n), then we call it a P(n)-approximation algorithm.

- For a maximization problem, 0< C < C*, and the ratio of C*/C gives the factor by which the cost of an optimal solution is larger than the cost of the approximate algorithm.
- For a minimization problem, 0< C* < C, and the ratio of C/C* gives the factor by which the cost of an approximate solution is larger than the cost of an optimal solution.

Some examples of the Approximation algorithm :

- **The Vertex Cover Problem –**
  In the vertex cover problem, the optimization problem is to find the vertex cover with the **fewest vertices**, and the approximation problem is to find the vertex cover with **few vertices**.

- **Travelling Salesman Problem –**
  In the traveling salesperson problem, the optimization problem is to find the **shortest cycle**, and the approximation problem is to find a **short cycle**.

- **The Set Covering Problem –**
  This is an optimization problem that models many problems that require resources to be allocated. Here, a logarithmic approximation ratio is used.

- **The Subset Sum Problem –**
  In the Subset sum problem, the optimization problem is to find a subset of {x1,×2,×3…xn} whose sum is as large as possible but not larger than the target value t.

Approximation algorithms for TSP

Nearest neighbour (NN) algorithm (a greedy algorithm)

- It lets the salesman choose the nearest unvisited city as his next move. This algorithm quickly yields an effectively short route. For N cities randomly distributed on a plane, the algorithm on average yields a path 25% longer than the shortest possible path
- However, there exist many specially arranged city distributions which make the NN algorithm give the worst route

Approximation algorithms for TSP

- Modern methods can find solutions for extremely large problems (millions of cities) within a reasonable time which are with a high probability just 2–3% away from the optimal solution

Approximation algorithms for TSP

- **Multi-fragment (MF) algorithm : “Greedy Algorithm” for TSP :** The algorithm builds a tour for the traveling salesman one edge at a time and thus maintains multiple tour fragments, each of which is a simple path in the complete graph of cities. At each stage, the algorithm selects the edge of minimal cost that either creates a new fragment, extends one of the existing paths or creates a cycle of length equal to the number of cities.

TSP : Using Approximation Algorithm

Approximation algorithm named
2 approximation algorithm,
that uses **Minimum Spanning Tree**
in order to obtain an approximate path

An Approximate Algorithm for TSP

- If the problem instance satisfies **Triangle-Inequality., i.e.** **“***The least distant path to reach a vertex j from i is always to reach j directly from i, rather than through some other vertex k (or vertices)“*

  i.e.,dis(i, j) <= dis(i, k) + dist(k, j); where **dis(a,b)** = diatance between a & b, i.e. the edge weight
- The Triangle-Inequality holds in many practical situations.

TSP problem is approximated as we have tweaked the cost function/condition to triangle inequality.

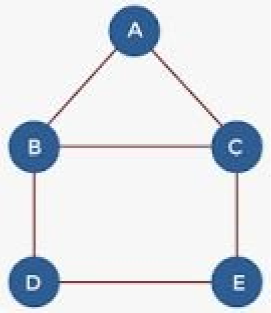
2 approximation algorithm for TSP ?

- When the cost function satisfies the triangle inequality, we may design an approximate algorithm for the TSP that returns a tour whose cost is **never more than twice the cost of an optimal tour**.
  - The idea is to use **Minimum Spanning Tree (MST)**

The 2 Approximation Algorithm :

- Let 0 be the starting and ending point for salesman.
- (1) Construct Minimum Spanning Tree from with 0 as root using **Prim's Algorithm**.
- (2) List vertices visited in preorder walk/Depth First Search of the constructed MST and add source node at the end

Why 2 approximate ?



Following are some important points that maybe taken into account,

- The cost of best possible Travelling Salesman tour is **never less than the cost of MST**. (The definition of MST says, it is a **minimum cost tree** that connects all vertices).
- The total cost of full walk is at most twice the cost of MST (Every edge of MST is visited at-most twice)
- The output of the above algorithm is less than the cost of full walk.

Step(1) MST - Prim's Algorithm (in Brief):.

- Create a set **mstSet** that keeps track of vertices already included in MST
- Assign a key value to all vertices in the input graph.
- Initialize all key values as **INFINITE**.
- Assign key value as 0 for the first vertex so that it is picked first.
- **[The Loop]** While mstSet doesn’t include all vertices
  - Pick a vertex u which is not there in mstSet and has minimum key value.(**minimum_key()**)
  - Include u to mstSet.
  - Update key value of all adjacent vertices of u. To update the key values, iterate through all adjacent vertices. For every adjacent vertex v, if weight of edge u-v is less than the previous key value of v, update the key value as weight of u-v.

Step - 2 - Getting the preorder walk/ Depth first search walk:

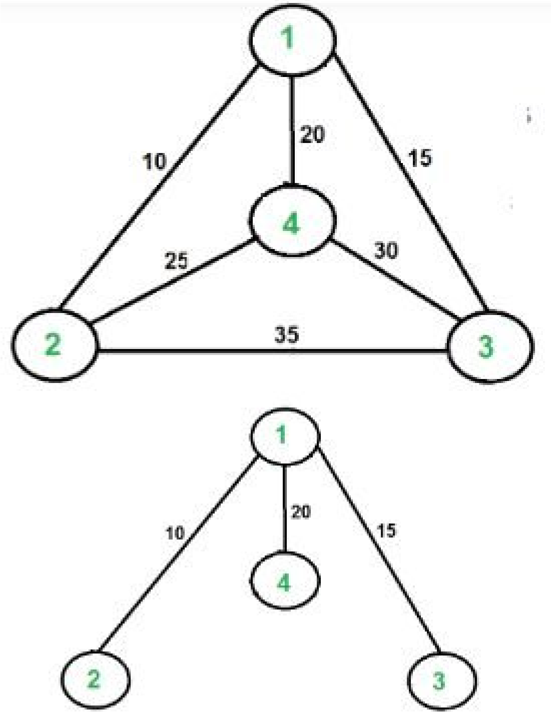
- We have two ways to perform the second step,
  1 - Constructing a **generic tree** on the basic of output received from the step -1
  2 - Constructing an **adjacency matrix** where graph[i][j] = 1 means both i & j are having a direct edge and included in the MST.

Depth First Search Algorithm:

- Push the starting_vertex to the final_ans vector.
- Checking up the visited node status for the same node.
- Iterating over the adjacency matrix (depth finding) and adding all the child nodes to the final_ans.
- Calling recursion to repeat the same.

Full walk

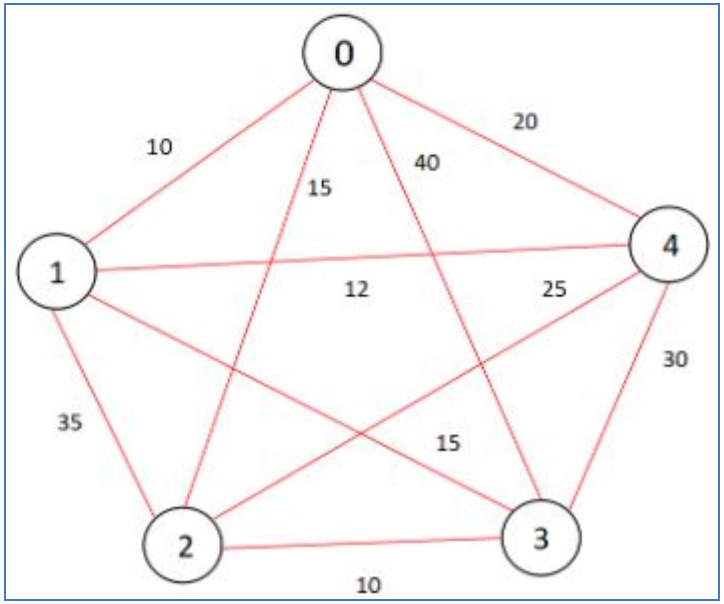
- A full walk lists all vertices when they are first visited in preorder, it also list vertices when they are returned after a subtree is visited in preorder. The full walk of tree below would be 1-2-1-4-1-3-1.



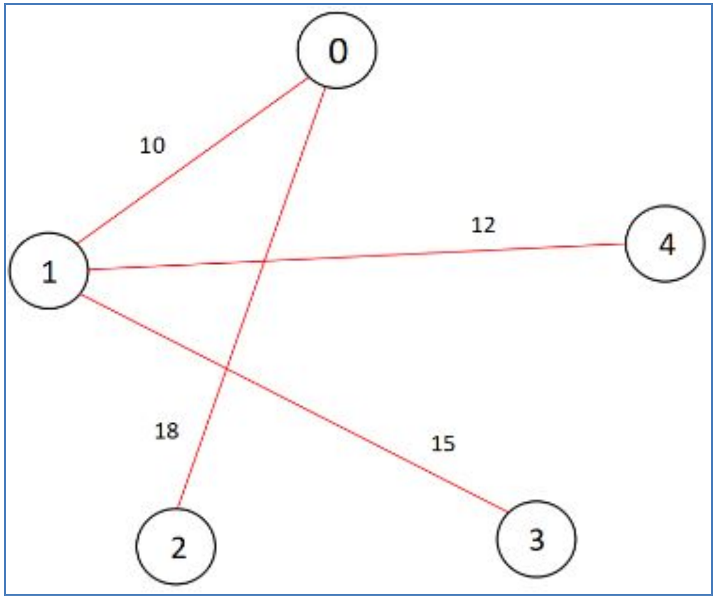
Minimum Spanning Tree of the graph with 1 as root. The Preorder Traversal of MST is 1-2-4-3. So the output is 1-2-4-3-1

Following are some important facts that prove the 2-approximateness.

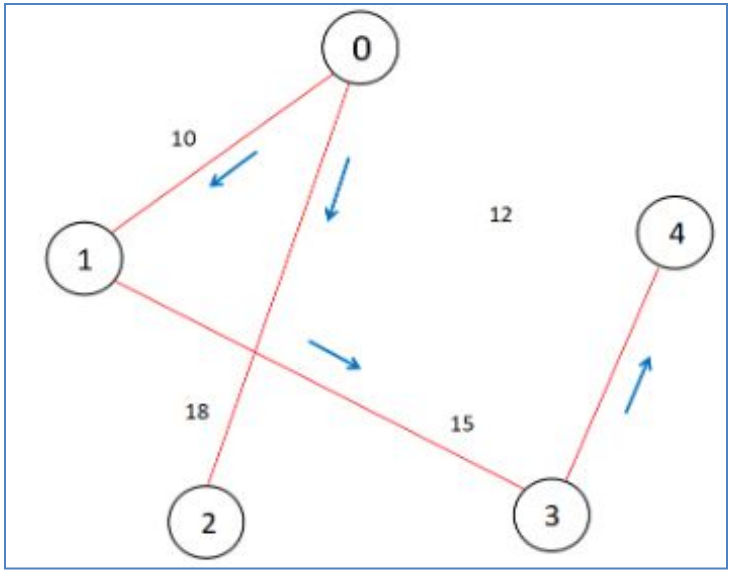
- The cost of best possible Travelling Salesman tour is never less than the cost of MST. (The definition of MST says, it is a minimum cost tree that connects all vertices).
- The total cost of full walk is at most twice the cost of MST (Every edge of MST is visited at-most twice)
- The output of the above algorithm is less than the cost of full walk. In above algorithm, we print preorder walk as output. In preorder walk, two or more edges of full walk are replaced with a single edge. For example, 2-1 and 1-4 are replaced by 1 edge 2-4. So if the graph follows triangle inequality, then this is always true.
- From the above three statements, we can conclude that the cost of output produced by the approximate algorithm is never more than twice the cost of best possible solution.



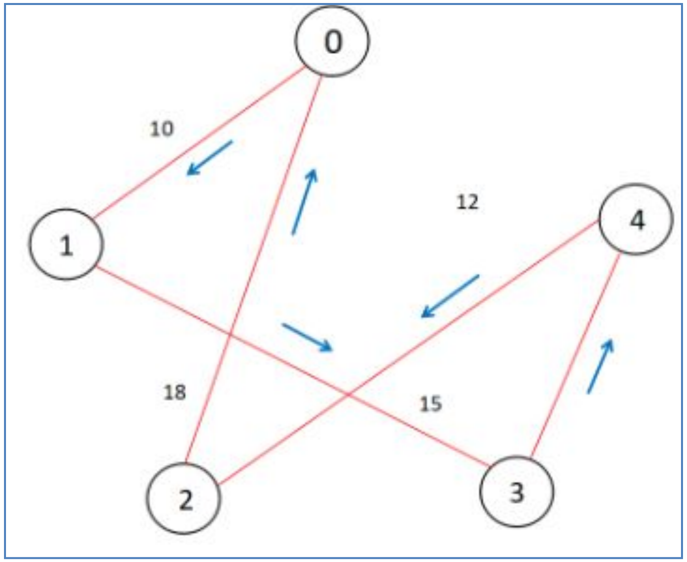
TSP problem



Step1: Build MST with start node as root



Step2: Perform Depth-First search (0,1,3,4,2)



Add start node 0 to DFS order nodes) (0,1,3,4,2,0)

The final_ans vector will contain the answer path.

3/2 Approximation (TSP)

Christofides algorithm

Concepts used

- Triangle Inequality, MST
- Perfect Matching
- Handshaking Lemma
- Eulerian tour

Christofides algorithm

Let *G = (V,w) be an instance of the travelling salesman problem. That is, G is a **complete graph** on the set V of vertices, and the function w assigns a nonnegative real weight to every edge of G.*

- *According to the triangle inequality, for every three vertices u, v, and x, it should be the case that w) ≥ w(ux). Then the algorithm can be described in pseudocode as follows.*
- *Create a minimum spanning (uv) + w(vxtree T of G.* (O ((V + E) l o g V)*)*
- *Let O be the set of vertices with **odd degree** in T. By the handshaking lemma, O has a even number of vertices. (**handshaking lemma** is the statement that, in every finite undirected graph, the **number of vertices that touch an odd number of edges even**.ie in a party of people who shake hands, the number of people who shake an odd number of other people's hands is even)*
- *Find a **minimum-weight perfect matching** M in the induced subgraph given by the vertices from O.*
- *Combine the edges of M and T to form a **connected multigraph** H in which each vertex has even degree.*
- *Form an Eulerian circuit in H.*
- *Make the circuit found in previous step into a Hamiltonian circuit by skipping repeate vertices (shortcutting).*

The steps 5 and 6 do not necessarily yield only one result. As such the heuristic can give several different paths.

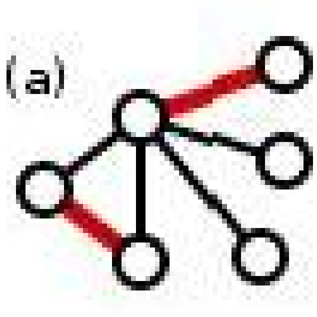
*The cost of the solution produced by the algorithm is within **3/2 of the optimum**.*

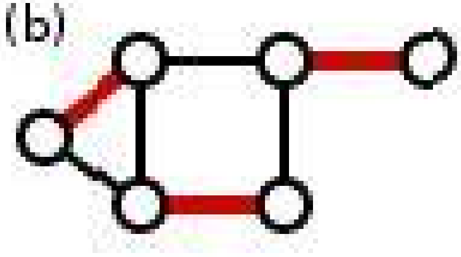
July 2020 Improvement to 1.5 Approximate

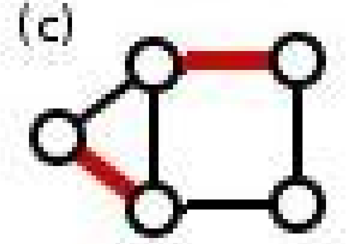
- In July 2020 however, Karlin, Klein, and Gharan released a preprint in which they introduced a novel approximation algorithm and claimed that its approximation ratio is $1.5 - 10^{-36}$.
  - Their method follows similar principles to Christofides' algorithm, but uses a randomly chosen tree from a carefully chosen random distribution in place of the minimum spanning tree.
  - The paper was published at STOC'21 where it received a best paper award.

More Graph Theory Concepts

- **Matching** (i.e. Independent edge set) in an undirected graph is a set of edges **with no common vertices**
- **Perfect Matching** =matching that **covers every vertex**
- **(b)** has a perfect matching, (a), (c) do not



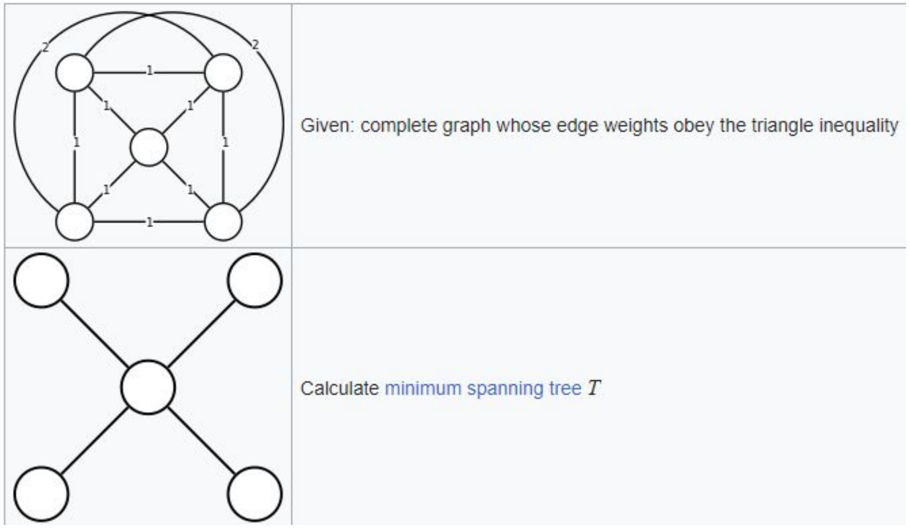




More Graph Theory Concepts

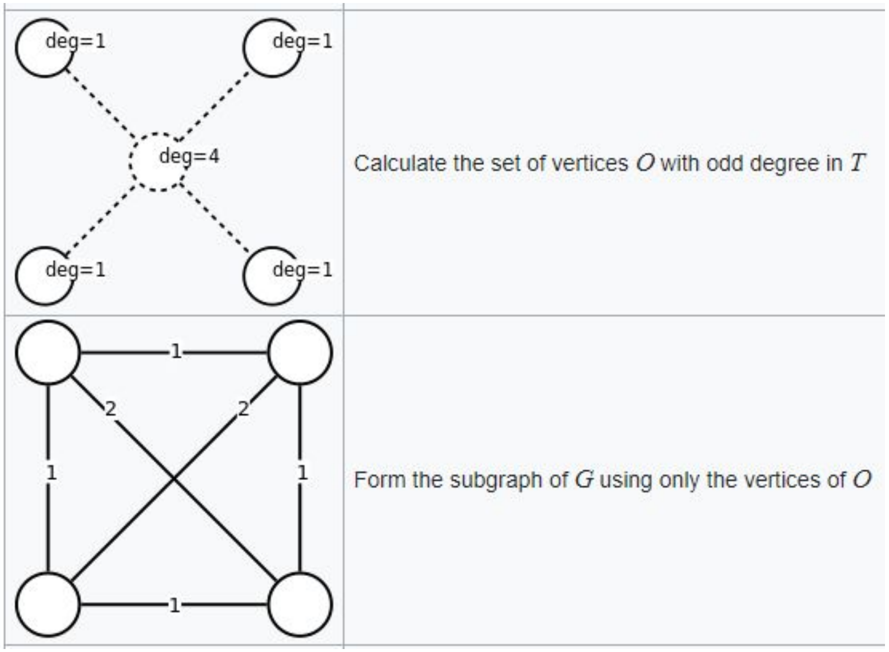
- **Eulerian trail** (or **Eulerian path/tour**) is a trail in a finite graph that **visits every edge** exactly once (allowing for revisiting vertices)
- **Eulerian circuit** or **Eulerian cycle** is an Eulerian trail that starts and ends on the same **vertex**

Christofides algorithm



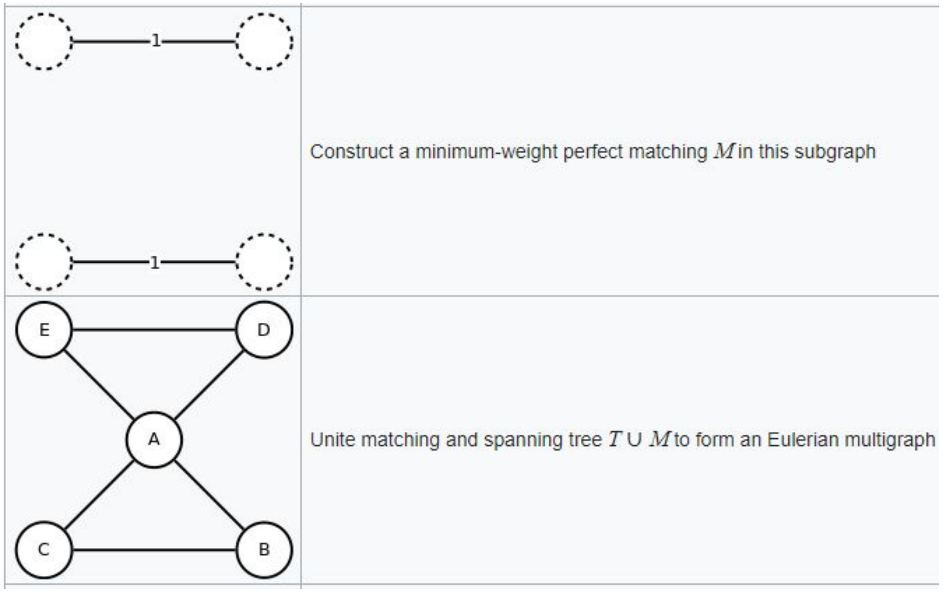
Given: complete graph whose edge weights obey the triangle inequality

Calculate minimum spanning tree T



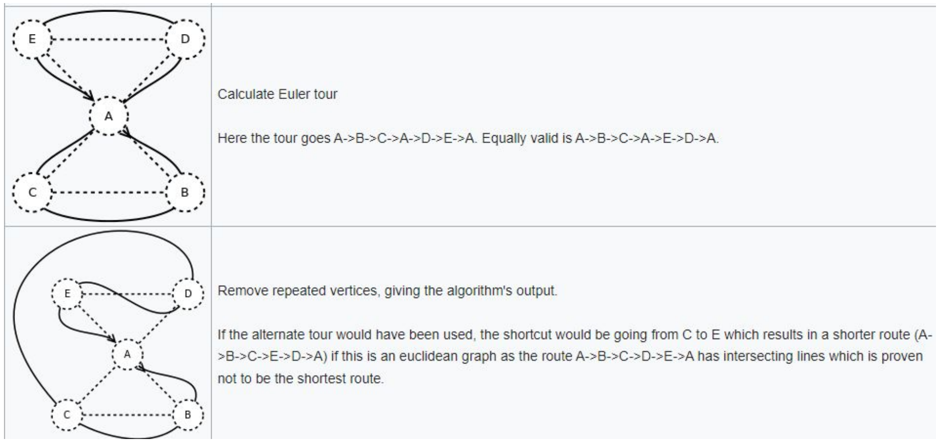
Calculate the set of vertices O with odd degree in T

Form the subgraph of G using only the vertices of O



Construct a minimum-weight perfect matching M in this subgraph

Unite matching and spanning tree $T \cup M$ to form an Eulerian multigraph



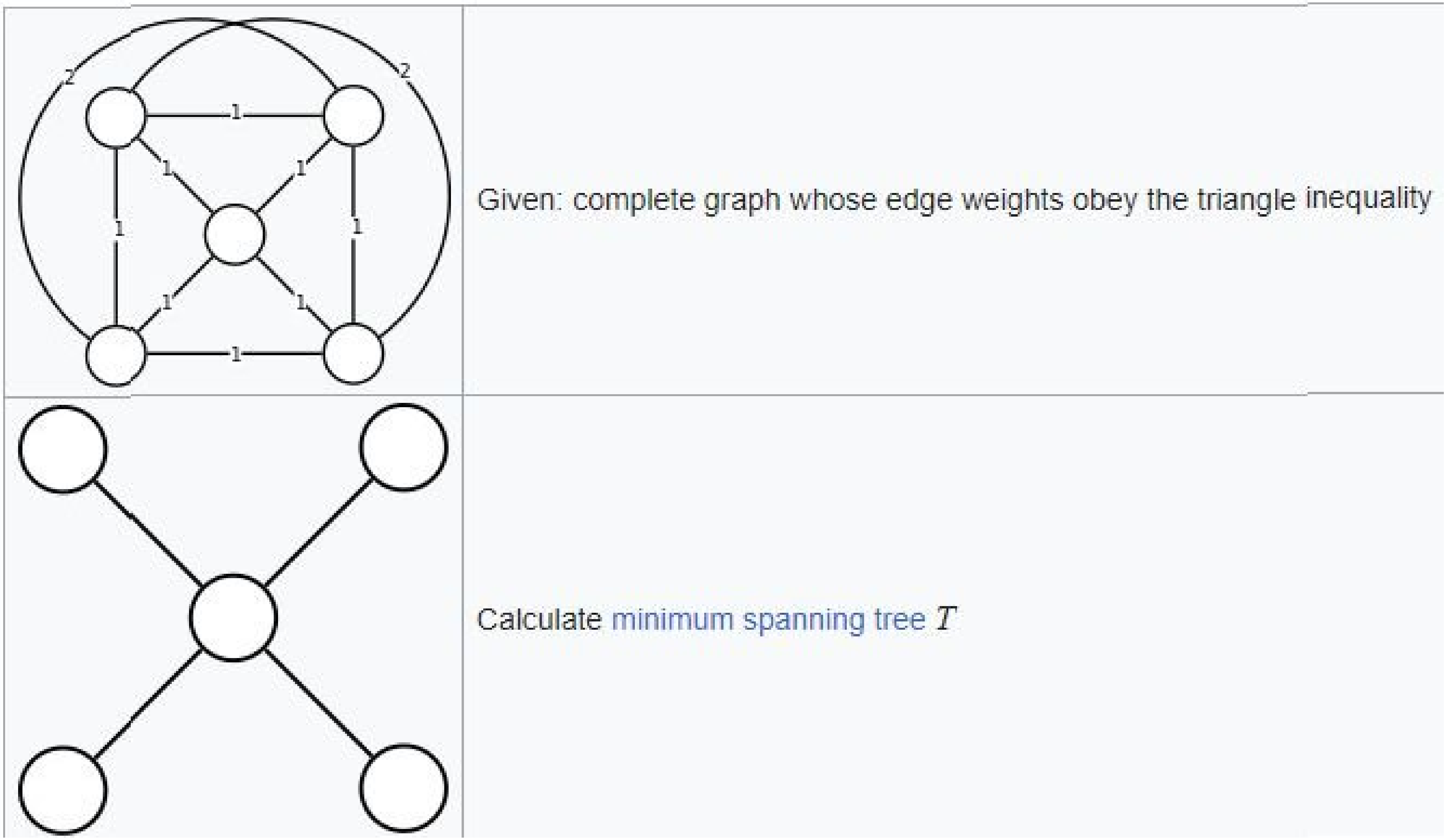
Calculate Euler tour

Here the tour goes A->B->C->A->D->E->A. Equally valid is A->B->C->A->E->D->A.

Remove repeated vertices, giving the algorithm's output.

If the alternate tour would have been used, the shortcut would be going from C to E which results in a shorter route (A->B->C->E->D->A) if this is an euclidean graph as the route A->B->C->D->E->A has intersecting lines which is proven not to be the shortest route.

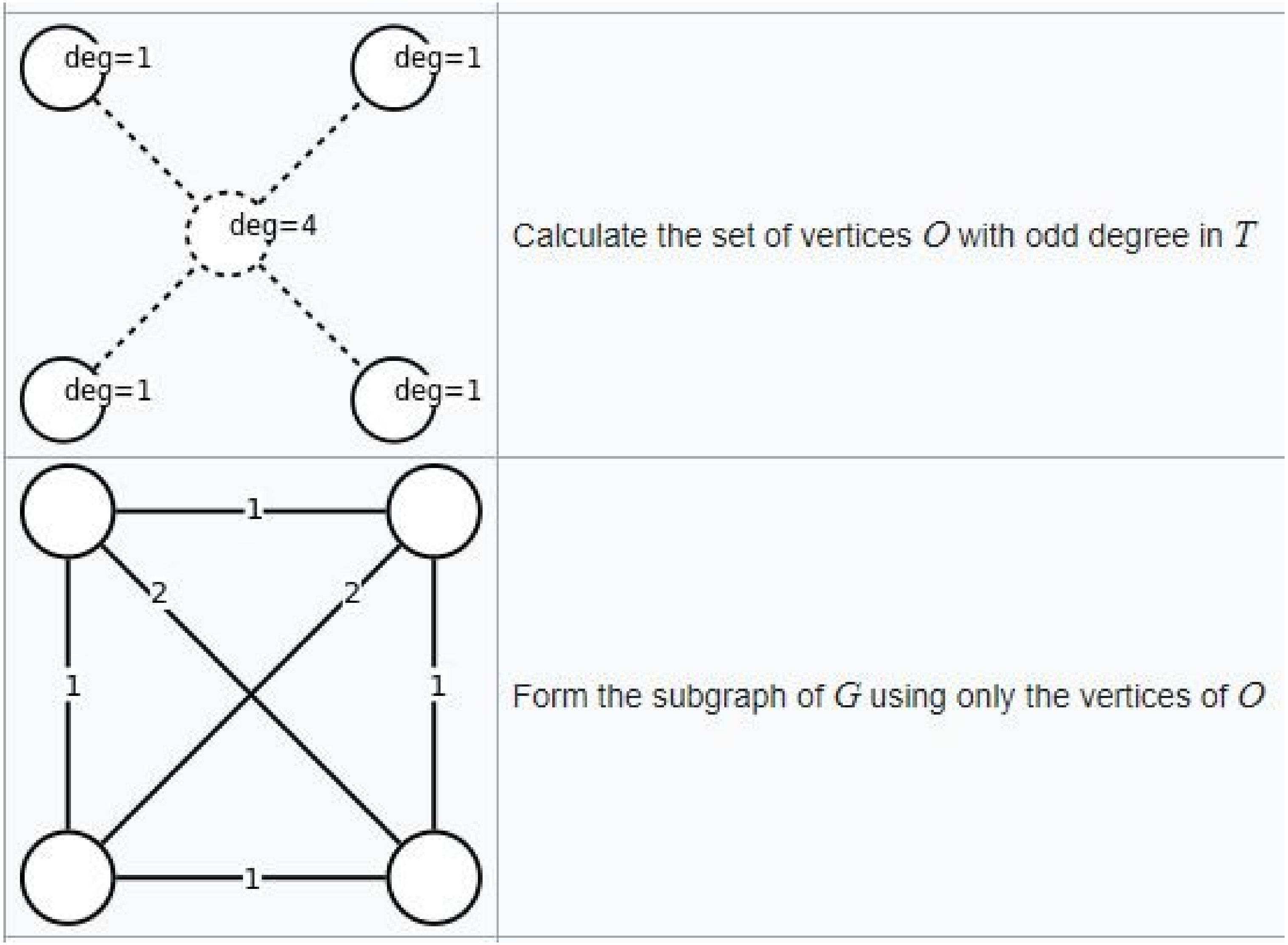
Christofides algorithm



Given: complete graph whose edge weights obey the triangle inequality

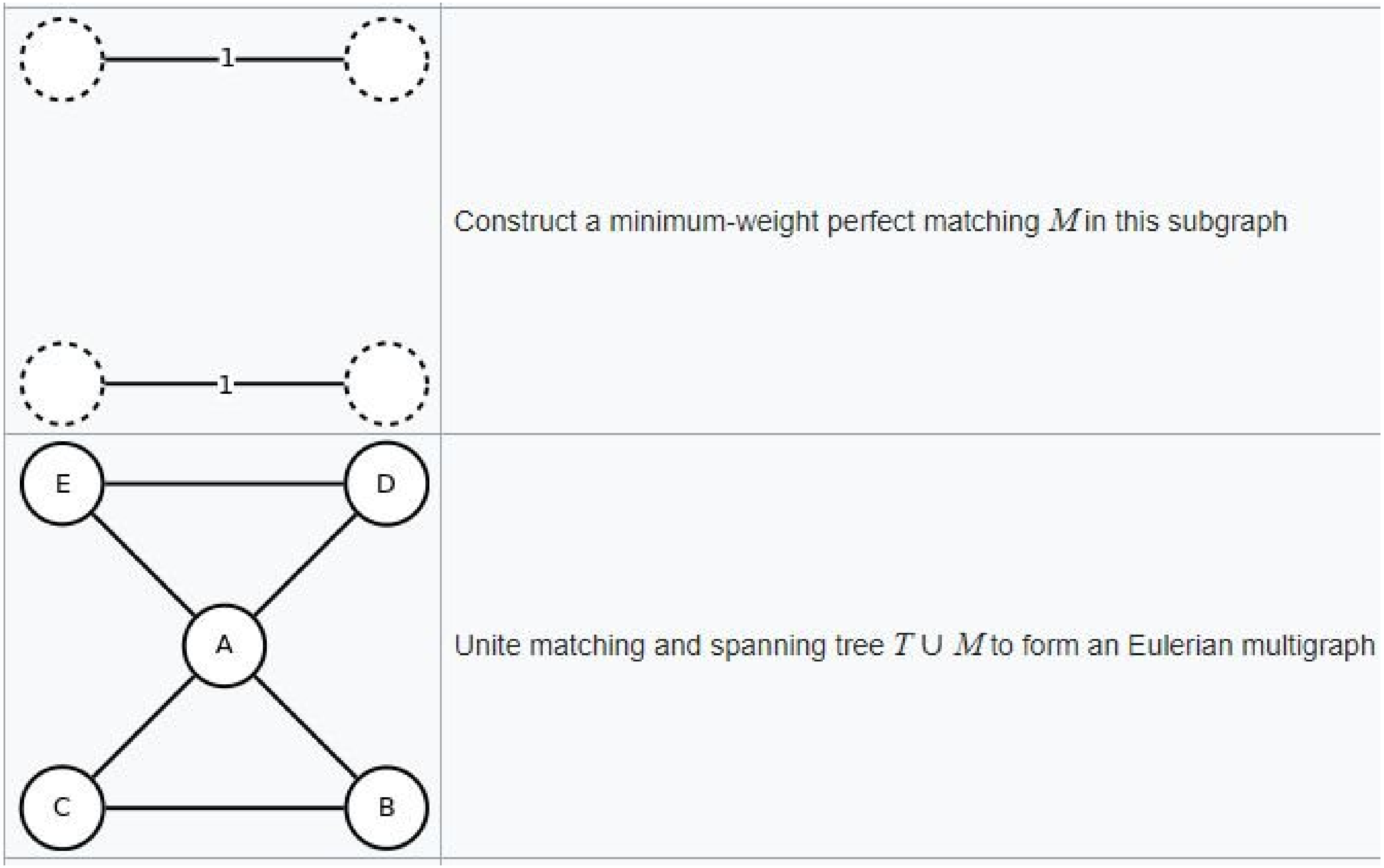
Calculate minimum spanning tree T

..contd..Christofides algorithm



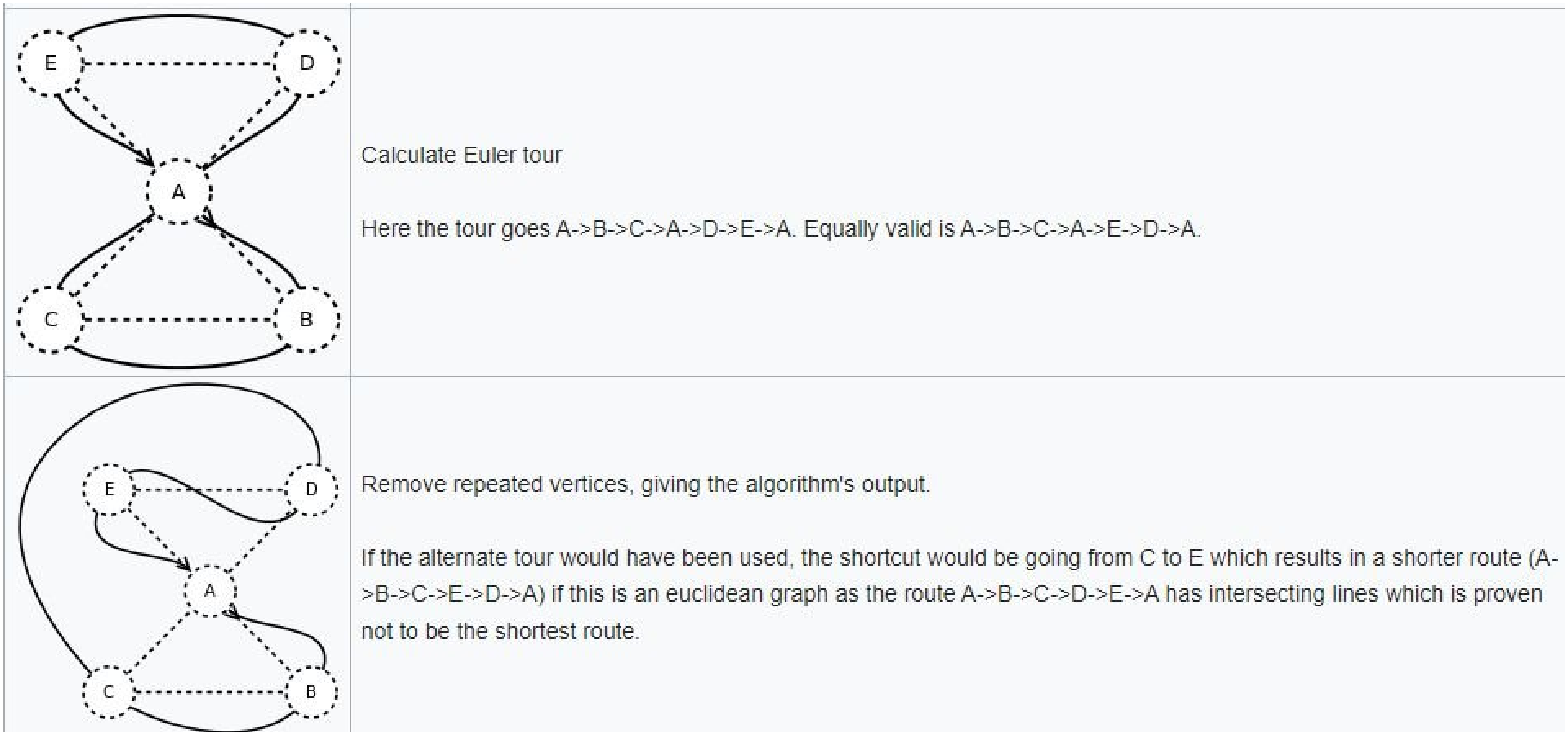
	Calculate the set of vertices O with odd degree in T
	Form the subgraph of G using only the vertices of O

..contd..Christofides algorithm



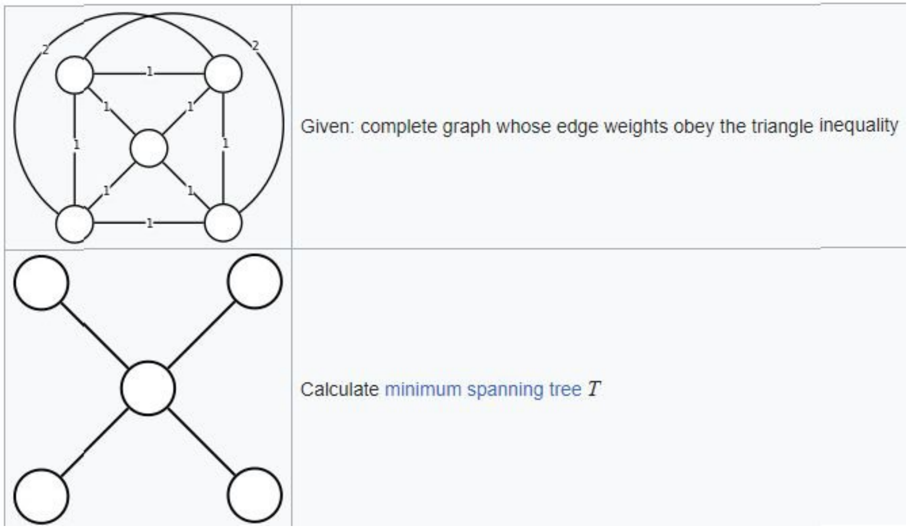
	Construct a minimum-weight perfect matching M in this subgraph
	Unite matching and spanning tree $T \cup M$ to form an Eulerian multigraph

..contd..Christofides algorithm



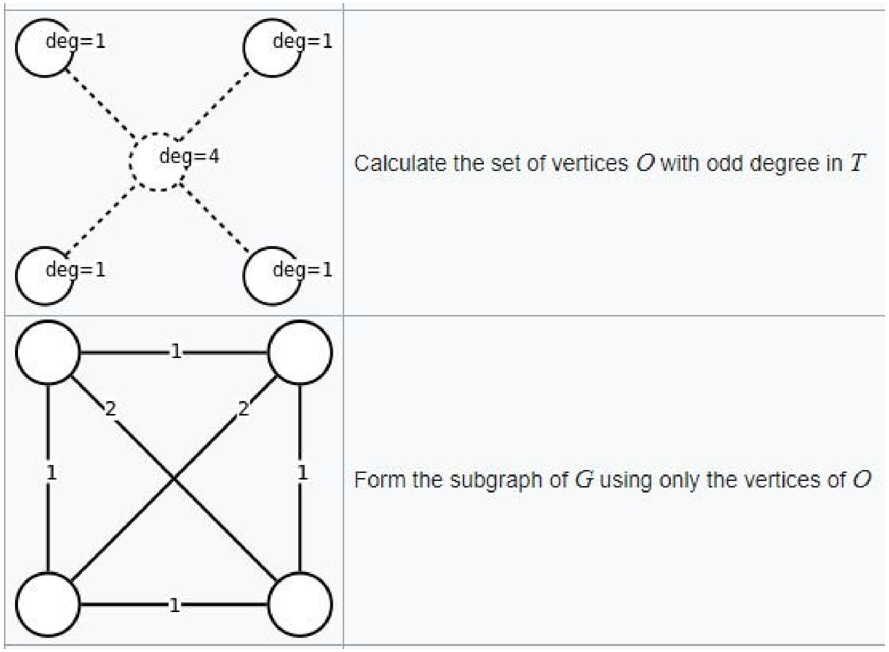
	Calculate Euler tour Here the tour goes A->B->C->A->D->E->A. Equally valid is A->B->C->A->E->D->A.
	Remove repeated vertices, giving the algorithm's output. If the alternate tour would have been used, the shortcut would be going from C to E which results in a shorter route (A->B->C->E->D->A) if this is an euclidean graph as the route A->B->C->D->E->A has intersecting lines which is proven not to be the shortest route.

..contd..Christofides algorithm



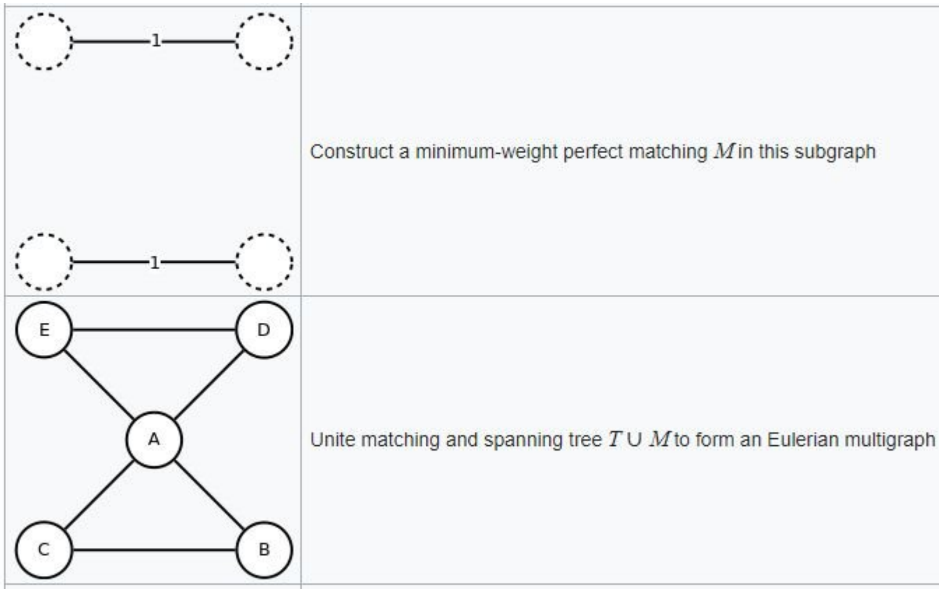
Given: complete graph whose edge weights obey the triangle inequality

Calculate minimum spanning tree T



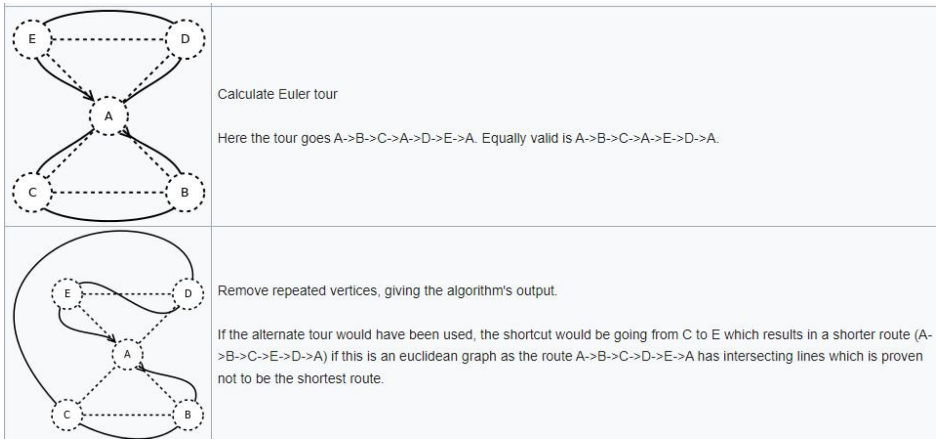
Calculate the set of vertices O with odd degree in T

Form the subgraph of G using only the vertices of O



Construct a minimum-weight perfect matching M in this subgraph

Unite matching and spanning tree $T \cup M$ to form an Eulerian multigraph



Calculate Euler tour

Here the tour goes A->B->C->A->D->E->A. Equally valid is A->B->C->A->E->D->A.

Remove repeated vertices, giving the algorithm's output.

If the alternate tour would have been used, the shortcut would be going from C to E which results in a shorter route (A->B->C->E->D->A) if this is an euclidean graph as the route A->B->C->D->E->A has intersecting lines which is proven not to be the shortest route.

Christofides algorithm

Given: complete graph whose edge weights obey the triangle inequality

- Calculate minimum spanning tree *T (mst)*
- Calculate the set of vertices *O* with odd degree in *T*
- Form the subgraph of *G* using only the vertices of *O*
- Construct a minimum-weight perfect matching *M* in this subgraph
- Unite matching and spanning tree $T \cup M$ to form an Eulerian multigraph
- Calculate Euler tour
- Remove repeated vertices, giving the algorithm's output.

Approximating Max Clique

Max Clique

- A clique is a sub graph in which all pairs of vertices are mutually adjacent.
- A maximum clique is a set of objects which are mutually related in some specified criterion.
- **A complete graph is often called a clique**.
- The size of the largest clique that can be made up of edges and vertices of G is called the clique number of G.
- The clique decision problem is NP-complete

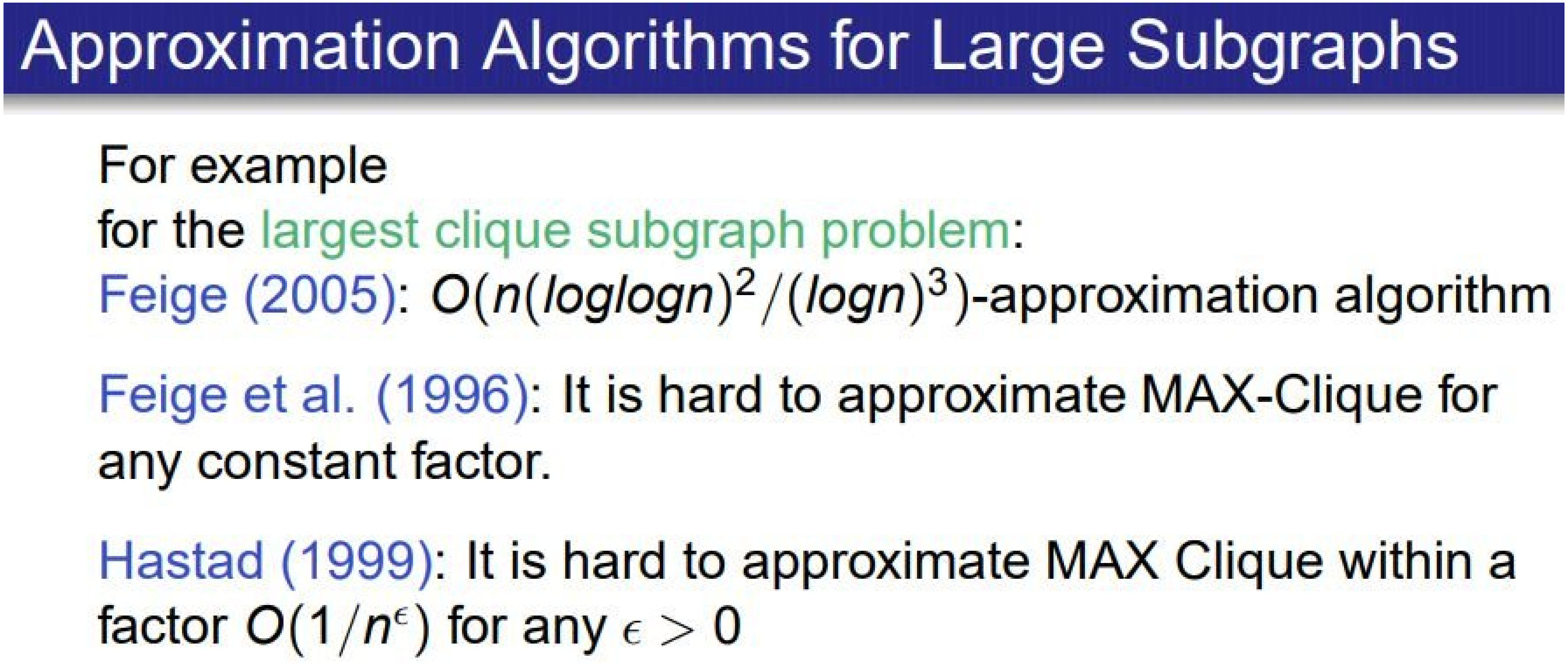
Max Clique

- The maximum clique problem is a well-known NP-hard problem in computer science, which means that **finding an exact solution** to this problem can be computationally expensive for large graphs.
- However, **several approximation techniques** can be used to find near-optimal solutions to this problem efficiently.
- Some soft computing techniques can be used alone or in combination, to find near-optimal solutions to the maximum clique problem efficiently.
  - The choice of the technique depends on the specific characteristics of the problem and the available computing resources.

Real Life Clique problem

- Consider a social network, where the graph's vertices represent people, and the graph's edges represent mutual acquaintance.
- Then a clique represents a subset of people who all know each other, and algorithms for finding cliques can be used to discover these groups of mutual friends.
- Along with its **applications in social networks**, the clique problem also has many applications in **bioinformatics**, and **computational chemistry**

Approximation for Max-Clique



Approximation Algorithms for Large Subgraphs

For example
for the largest clique subgraph problem:
Feige (2005): $O(n(loglogn)^2/(logn)^3)$-approximation algorithm

Feige et al. (1996): It is hard to approximate MAX-Clique for any constant factor.

Hastad (1999): It is hard to approximate MAX Clique within a factor $O(1/n^{\epsilon})$ for any $\epsilon > 0$

Some popular approximation techniques for the maximum clique problem

- The **greedy algorithm** is a simple and effective technique for finding a maximal clique in a graph. The algorithm starts with a single node and iteratively adds nodes that are adjacent to all the nodes in the current clique until no more nodes can be added. While this algorithm does not guarantee a maximum clique, it can find a near-optimal solution quickly in many cases.
- The Bron-Kerbosch algorithm is a **recursive algorithm** that finds all maximal cliques in a graph efficiently.
  - The algorithm maintains three sets of nodes: R, which contains the nodes in the current clique, P, which contains the nodes that can be added to the current clique, and X, which contains the nodes that cannot be added to the current clique. The algorithm recursively searches for all maximal cliques by adding nodes from P to R and updating P and X accordingly.

Approximation techniques for the maximum clique problem

- Several authors have considered approximation algorithms that attempt to find a clique or independent set that, although not maximum, has size as close to the maximum as can be found in polynomial time. Although much of this work has focused on independent sets in sparse graphs, a case that does not make sense for the complementary clique problem, there has also been work on approximation algorithms that do not use such sparsity assumptions.
- Feige (2004) describes a polynomial time algorithm that finds a clique of size $\Omega((\log n/\log\log n)^2)$ in any graph that has clique number $\Omega(n/\log^k n)$ for any constant k. By using this algorithm when the clique number of a given input graph is between $n/\log n$ and $n/\log^3 n$, switching to a different algorithm of Boppana & Halldórsson (1992) for graphs with higher clique numbers, and choosing a two-vertex clique if both algorithms fail to find anything, Feige provides an approximation algorithm that finds a clique with a number of vertices within a factor of $O(n(\log\log n)^2/\log^3 n)$ of the maximum.
- Although the approximation ratio of this algorithm is weak, it is the best known to date. The results on hardness of approximation described below suggest that there can be no approximation algorithm with an approximation ratio significantly less than linear.

Unit V Approximation and Randomized Algorithms, Natural Algorithms

Approximation algorithms,

Solving TSP by approximation algorithm,

approximating Max Clique

Concept of randomized algorithms,

randomized quicksort algorithms,

Natural Algorithms–Evolutionary Computing and Evolutionary Algorithms,

Introduction to Genetic Algorithm,

Simulated Annealing

References

- https://nptel.ac.in/courses/112105235
- https://www.geeksforgeeks.org/approximation-algorithms/

Unit V Approximation and Randomized Algorithms, Natural Algorithms

PART 1-Approximation algorithms, Solving TSP by approximation algorithm, approximating Max Clique

PART 2-Concept of randomized algorithms, randomized quicksort algorithms,

PART3-Natural Algorithms–Evolutionary Computing and Evolutionary Algorithms, Simulated Annealing

PART4-Introduction to Genetic Algorithm

Randomized Algorithms

Randomized Algorithms: Basics

- A **randomized algorithm** is a technique that uses a source of randomness as part of its logic. It is typically used to reduce either the running time, or time complexity; or the memory used, or space complexity, in a standard algorithm. The algorithm works by generating a random number, *r*, within a specified range of numbers, and making decisions based on *r*'s value.
- A randomized algorithm could help in a situation of doubt by flipping a coin or a drawing a card from a deck in order to make a decision. Similarly, this kind of algorithm could help speed up a brute force process by randomly sampling the input in order to obtain a solution that may not be totally optimal, but will be good enough for the specified purposes.

...contd..Randomized Algorithms: Basics

- A superintendent is attempting to score a high school based on several metrics, and she wants to do so from information gathered by confidentially interviewing students. However, the superintendent has to do this with all the schools in the district, so interviewing every single student would take a time she cannot afford. What should she do?
- The superintendent should employ a randomized algorithm, where, without knowing any of the kids, she'd select a few at random and interview them, hoping that she gets a wide variety of students.
  - This technique is more commonly known as random sampling, which is a kind of randomized algorithm.
    - There are diminishing returns from each additional interview, and should stop when the quantity of data collected measures what she was trying to measure to an **acceptable degree of accuracy**.
  - The way that the superintendent is determining the score of the school can be thought of as a randomized algorithm

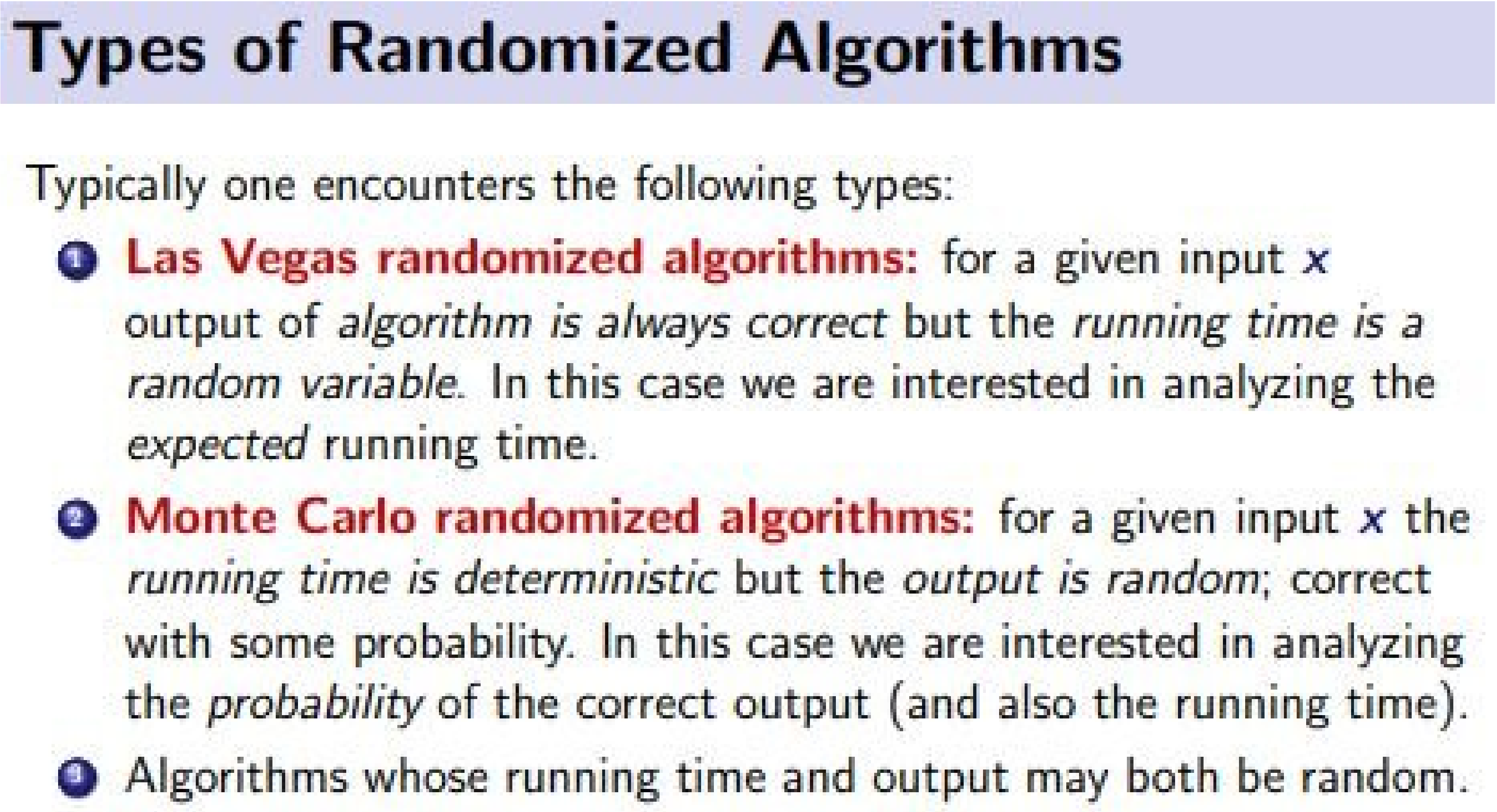
…contd.. Randomized Algorithms: Basics

- They are used when presented with a time or memory constraint, and an average case solution is an acceptable output.
- Due to the potential erroneous output of the algorithm, **an algorithm known as amplification** is used in order to boost the probability of correctness by sacrificing runtime.
  - Amplification works by repeating the randomized algorithm several times with different random subsamples of the input, and comparing their results.
  - It is common for randomized algorithms to amplify just parts of the process, as too much amplification may increase the running time beyond the given constraints.

...contd.. Randomized Algorithms: Basics

- Practically speaking,
  - Computers cannot generate completely random numbers, so randomized algorithms in computer science are approximated using a **pseudorandom number generator** in place of a true source of random number, such as the drawing of a card

Common FORMS of Randomized



Types of Randomized Algorithms

Typically one encounters the following types:

1. **Las Vegas randomized algorithms:** for a given input x output of *algorithm is always correct* but the *running time is a random variable*. In this case we are interested in analyzing the *expected* running time.
2. **Monte Carlo randomized algorithms:** for a given input x the *running time is deterministic* but the *output is random*; correct with some probability. In this case we are interested in analyzing the *probability* of the correct output (and also the running time).
3. Algorithms whose running time and output may both be random.

Las Vegas Algorithm

- A Las Vegas algorithm runs **within a specified amount of time.**
- If it finds a solution within that timeframe, the solution will be exactly correct; however, it is possible that it runs out of time and does not find any solutions.
  - It does, however, guarantee an upper bound in the worst-case scenario

Understanding Las Vegas Algorithm

Sample

- Las Vegas algorithms occur almost every time people look something up.
- Think of a Las Vegas algorithm as a task that when the solution is found, it has full confidence that it is the correct solution, yet the path to get there can be murky.

An Extremely Generalized and Simplified Example

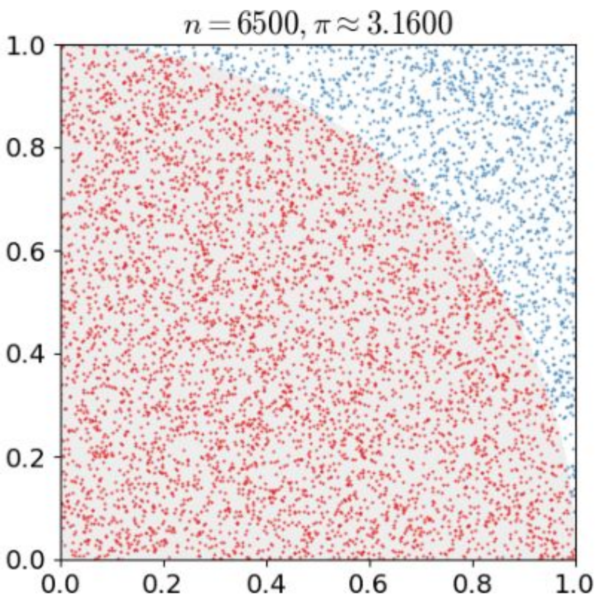
- Las Vegas algorithm could be thought of as the strategy used by a user who searches for something online.
- Since searching every single website online is extremely inefficient, the user will generally use a search engine to get started.
- The user will then surf the web until a website is found which contains exactly what the user is looking for.
- Since clicking through links is a decently randomized process, assuming the user does not know exactly what's contained on the website at the other end of the, the time complexity ranges from getting lucky and reaching the target website on the first link, to being unlucky and spending countless hours to no avail.
- What makes this a Las Vegas algorithm is that **the user knows exactly what she is looking for**, so once the website is found, there is no probability of error.
- Similarly, if **the user's allotted time to surf the web is exceeded**, she will terminate the process knowing that the solution was not found.

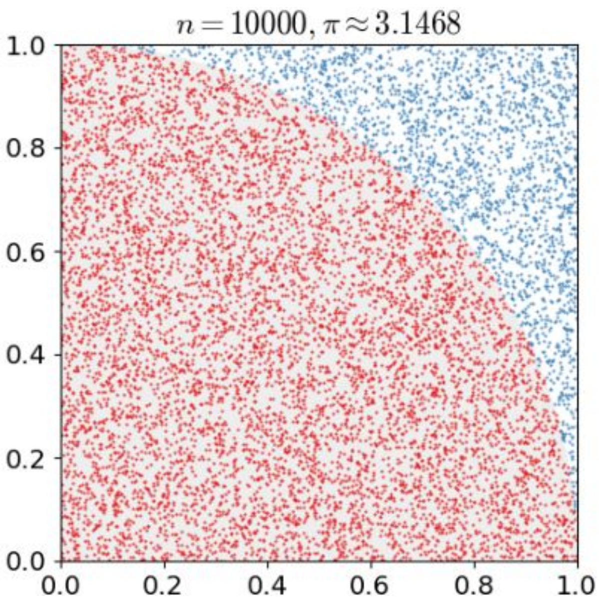
Monte Carlo algorithm

- Monte Carlo algorithm is a probabilistic algorithm which, depending on the input, has a slight probability of producing an incorrect result or failing to produce a result altogether.

Approximating pi : Monte Carlo

- Start with a Cartesian plane (x,y coordinates) with an x-axis from −1 to 1, and a y-axis from −1 to 1. This will be a 2 × 2 box.
- Then generate many random points on this grid.
- Count the number of points, *C*, that fall within a distance of 1 from the origin (0,0), and the number of points, *T*, that don't.
  - For instance, (0.5,0) is less than 1 from the origin, but (−0.9,0.9) is √(1.62) from the origin, by the pythagorean theorem.
- We can now approximate pi by knowing that the ratio of the area of a circle to the area of the rectangular bounds should be equal to the ratio of points inside the circle to those outside the circle; namely,
- $\pi r^2 / 4r^2 \approx (C/C+T)$





Randomized Quicksort

Randomized quick sort algorithm

- It picks a **random number x** from the set of numbers given as input,
- Partitions the input numbers with x as pivot, and recursively sorts the set of numbers less than x and the set of numbers greater than x.

Randomized Quicksort

- It follows the divide-and-conquer strategy, with improvements.
- The Pivot element, partitions the array around the pivot, and recursively sorts the sub-arrays. However, in the case of Quicksort, the choice of the pivot element can greatly impact the performance of the algorithm.
- Randomized Quicksort selects a random element from the array as the pivot element, ensuring that the algorithm's performance is not significantly affected by the input data's initial order.
- Steps involved in the Randomized Quicksort algorithm are as follows:
  - Choose a random element from the array as the pivot element.
  - Partition the array around the pivot element, so that all the elements smaller than the pivot are on the left, and all the elements greater than the pivot are on the right.
  - Recursively apply steps 1 and 2 to the sub-arrays on the left and right of the pivot element.

The performance of Randomized Quicksort

- It is measured using the average-case time complexity, which is O(nlogn) for most inputs.
- The worst-case time complexity can be O(n^2) in the presence of an adversarial input that causes the pivot selection to always result in unbalanced partitions
- Randomized Quicksort is Expected O(n log n)

Proof : Randomized Quicksort

- For any given array input array I of n elements, the expected time of this algorithm is E[T(I)]= O(n log n)
- This is called a Worst-case Expected-Time bound.
- Note: It is better than an average-case bound because we are no longer assuming any special properties of the input
- Uses : Probabilistic Analysis

Important: Proof of time complexity of Randomized Quicksort

https://www.cs.cmu.edu/afs/cs/academic/class/15451-s07/www/lecture_notes/lect0123.pdf

Unit V Approximation and Randomized Algorithms, Natural Algorithms

PART 1 - Approximation algorithms, Solving TSP by approximation algorithm, approximating Max Clique

PART 2 - Concept of randomized algorithms, randomized quicksort algorithms,

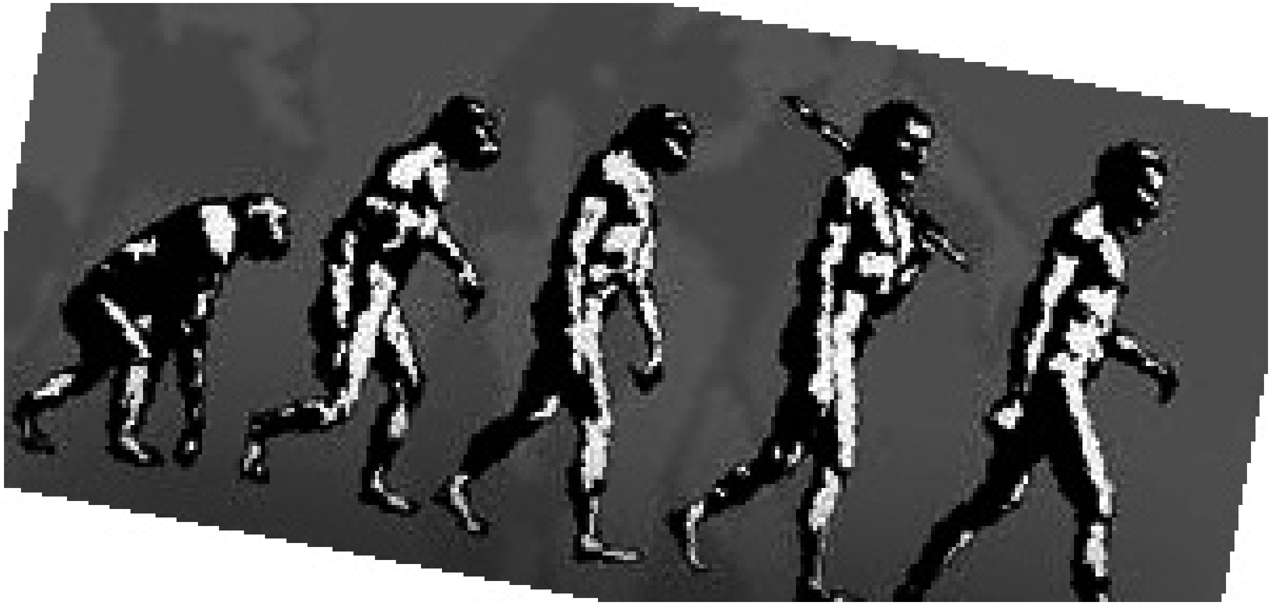
PART 3 - Natural Algorithms–Evolutionary Computing and Evolutionary Algorithms, Simulated Annealing

PART 4 - Introduction to Genetic Algorithm

Evolutionary Computation and Evolutionary Algorithms

Evolution

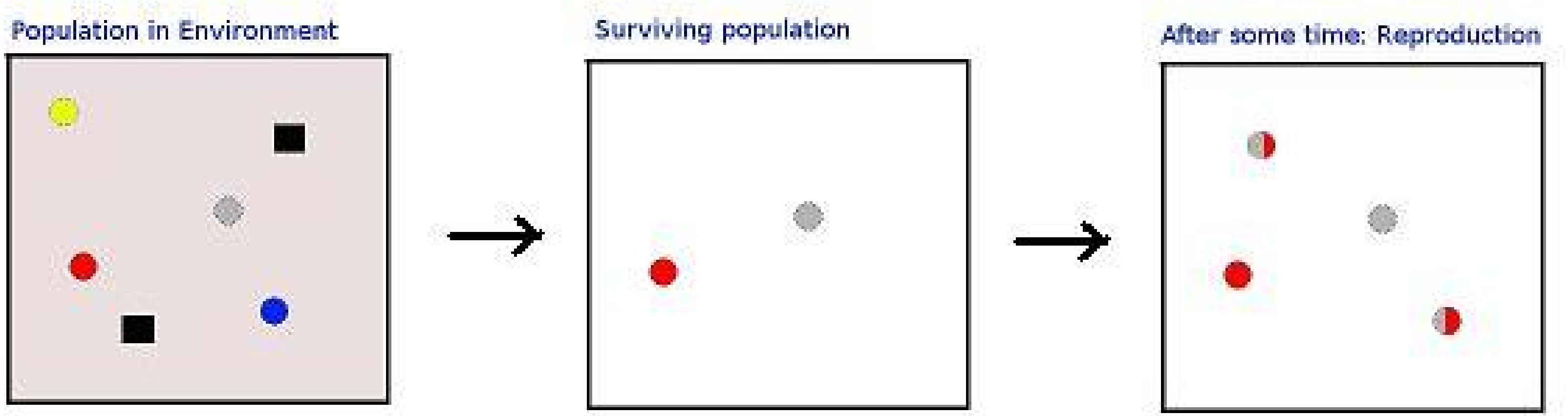
- Evolution is the change in the inherited traits of a population from one generation to the next.



- Natural selection leading to better and better species

Evolution – Fundamental Laws

- Survival of the fittest.
- Change in species is due to change in genes over reproduction or/and due to mutation.



An Example showing the concept of survival of the fittest and reproduction over generations.

Evolutionary Computation

- Evolutionary Computation (EC) refers to computer-based problem solving systems that use computational models of evolutionary process.
- Terminology:
  - *Chromosome* – It is an individual representing a candidate solution of the optimization problem.
  - *Population* – A set of chromosomes.
  - *gene* – It is the fundamental building block of the chromosome, each gene in a chromosome represents each variable to be optimized. It is the smallest unit of information.
- Objective: To find a best possible chromosome to a given optimization problem.

Evolutionary Algorithm: A meta-heuristic

```
Let $t = 0$ be the generation counter;
create and initialize a population $P(0)$;

repeat
    Evaluate the fitness, $f(\mathbf{x}_i)$, for all $\mathbf{x}_i$ belonging to $P(t)$;
    Perform cross-over to produce offspring;
    Perform mutation on offspring;
    Select population $P(t+1)$ of new generation;
    Advance to the new generation, i.e. $t = t+1$;
until stopping condition is true;
```

Evolutionary Algorithms

- Evolutionary Algorithms are heavily used in the search of solution spaces in many NP-Complete problems
- **Genetic Algorithms** : NP-Complete problems like Network Routing, TSP and even problems like Sorting are optimized by the use of Genetic Algorithms as they can rapidly locate good solutions, even for difficult search space
- **Evolutionary Programming** : Emphasizes the development of behavioural models rather than genetic models
- **Evolutionary Strategies** : In this not only the solution but also the evolutionary process itself evolves with generations (evolution of evolution)
- **Differential Programming** : Arithmetic cross-over operators are used instead of geometric operators like cut and exchange.

Brief Introduction to various types of Evolutionary Algorithms

GA (IN THE NEXT PART)

EVOLUTIONARY PROGRAMMING

EVOLUTIONARY STRATEGIES

DIFFERENTIAL PROGRAMMING

Unit V Approximation and Randomized Algorithms, Natural Algorithms

PART 1 -Approximation algorithms, Solving TSP by approximation algorithm, approximating Max Clique

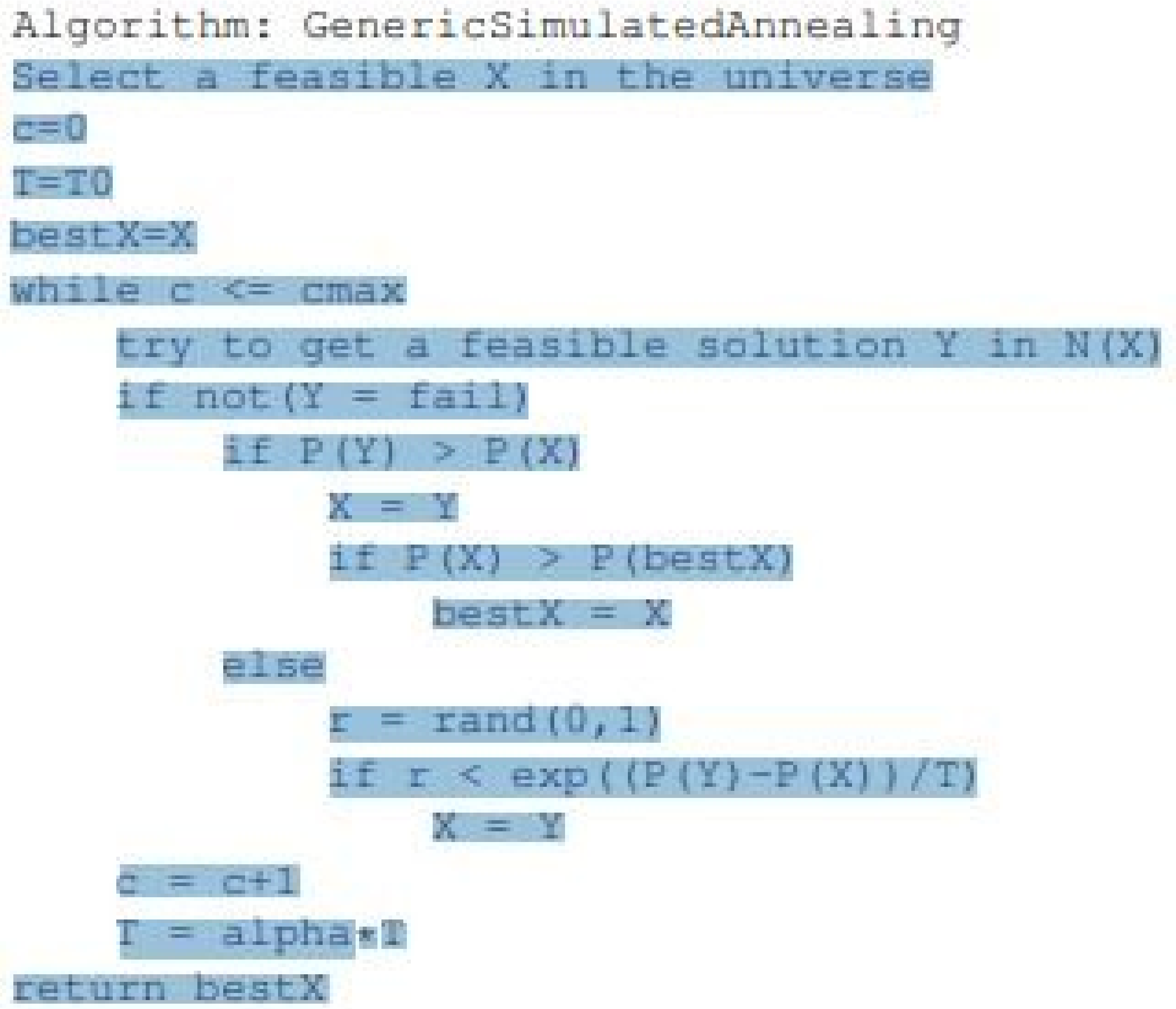
PART 2 -Concept of randomized algorithms, randomized quicksort algorithms,

PART 3 - Natural Algorithms–Evolutionary Computing and Evolutionary Algorithms, Simulated Annealing

PART 4 - Introduction to Genetic Algorithm

Simulated Annealing

General Technique



```
Algorithm: GenericSimulatedAnnealing
Select a feasible X in the universe
c=0
T=T0
bestX=X
while c <= cmax
    try to get a feasible solution Y in N(X)
    if not(Y = fail)
        if P(Y) > P(X)
            X = Y
            if P(X) > P(bestX)
                bestX = X
        else
            r = rand(0,1)
            if r < exp((P(Y)-P(X))/T)
                X = Y
    c = c+1
    T = alpha*T
return bestX
```

General Technique

```
Select a feasible X in the universe
 c=0
T=T0
bestX=X
while c <= cmax
    try to get a feasible solution Y in N(X)
    if not(Y = fail)
      if P(Y) > P(X)
            X = Y
            if P(X) > P(bestX)
                  bestX = X
      else
             r = rand(0,1)
             if r < exp((P(Y)-P(X))/T)
                  X = Y
    c = c+1
    T = alpha*T
return bestX
```

Using Simulated Annealing

- The cooling schedule can be chosen based on the problem and the characteristics of the dataset.
- A typical choice is to decrease the temperature exponentially or linearly with time.
- The acceptance probability is based on the Metropolis criterion, where the probability of accepting a worse solution decreases as the temperature decreases.
- Simulated Annealing can be a powerful algorithm to solve combinatorial optimization problems such as the 0/1 knapsack problem.
- The performance of the algorithm depends on the choice of the cooling schedule and the initial solution.
- In practice, multiple runs of the algorithm with different initial solutions and cooling schedules can be performed to increase the chances of finding a good solution.

Simulated Annealing Model (Quick Review)

- Simulated annealing involves a cooling schedule determined by β and a stationary probability
  $\pi_{\beta}(\xi) = 1/(\ Z(\beta))^{*}\ e{-}^{\beta V\ (\xi)}$
- If we cool slowly enough, we can avoid getting trapped in local optima
- We determine the probability
  $P = \min\{1, e^{(\beta\Delta)}\}$ where $\Delta = V_1' - V_1$; where V_1', V_1 are the new and the old solutions
- β is determined more or less by trial and error

0/1 Knapsack: Simulated Annealing

Solving 0/1 Knapsack

- To apply simulated annealing to the 0/1 knapsack problem, we can define a fitness function that evaluates the value of a particular solution (i.e., selection of items) while ensuring that it does not exceed the knapsack's capacity.
- We can then generate neighboring solutions by randomly flipping the selection of one or more items and evaluate their fitness.
- By using a carefully designed cooling schedule, simulated annealing can effectively explore the solution space of the 0/1 knapsack problem and avoid getting trapped in local optima.
  - In particular, the cooling schedule should start with a high temperature that allows for more exploration of the search space, and gradually decrease the temperature to focus more on exploiting promising solutions.
  - Additionally, the rate of temperature decrease should be carefully chosen to balance exploration and exploitation, with a slower rate promoting more exploration and a faster rate promoting more exploitation.

Simulated Annealing Steps to Solve Solving 0/1 Knapsack

1- Initialize the solution with a random set of items.

2- Set the temperature T to a high value.

3- While the temperature T is greater than zero, repeat steps 4-9.

4- Generate a new solution by randomly selecting an item to add or remove from the current solution.

5- Calculate the cost of the new solution by computing the total value of the items in the solution and checking if it exceeds the weight limit.

6- If the new solution is feasible, set it as the current solution. If it is not feasible, accept it with probability exp[(cost_current - cost_new)/T].

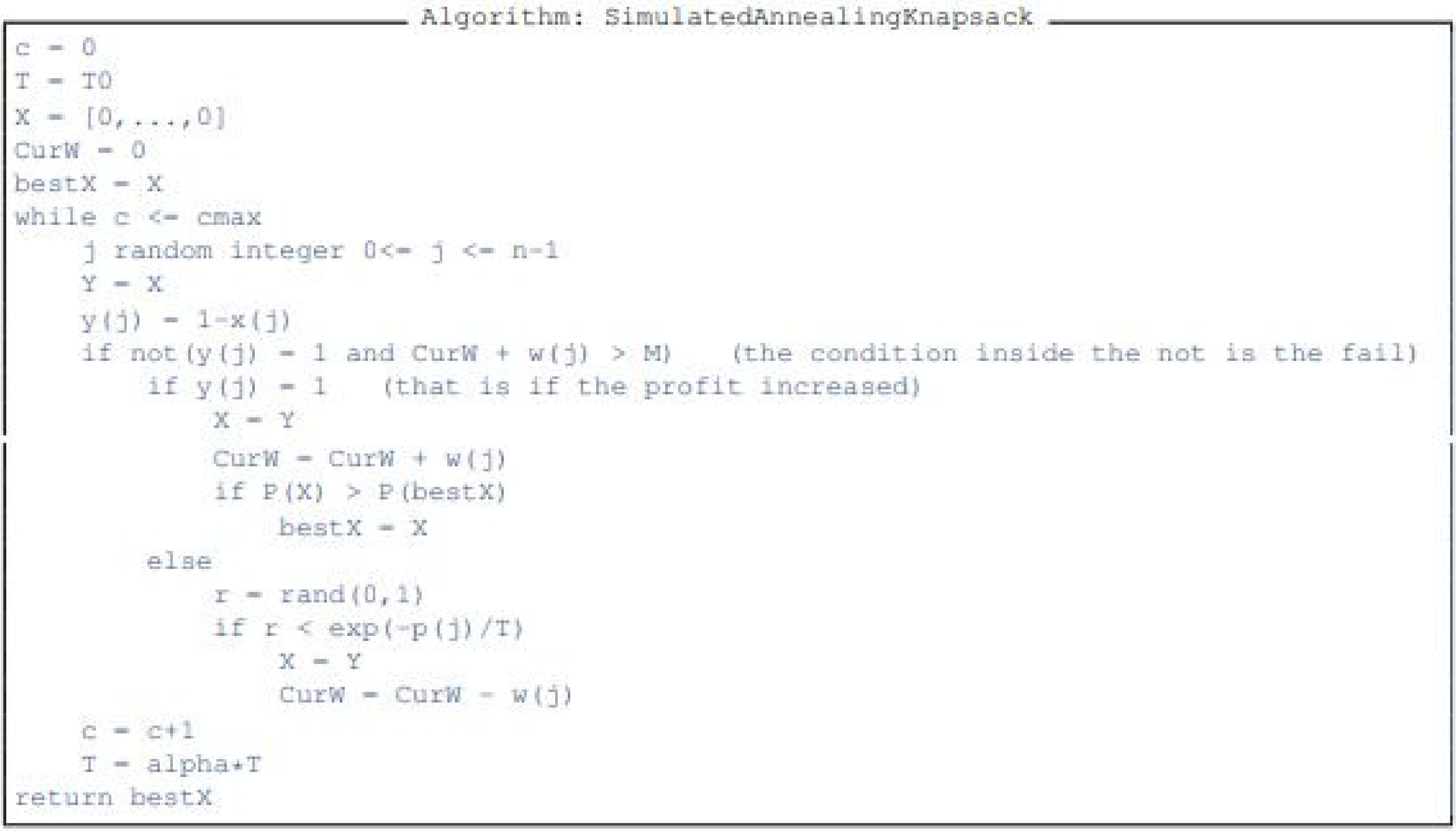
7- Decrease the temperature T using a cooling schedule.

8- Check if the stopping criterion has been met.

This could be based on a fixed number of iterations or a threshold value for the temperature.

9- If the stopping criterion has not been met, go back to step 4.

10- Return the best solution found during the iterations.



```
Algorithm: SimulatedAnnealingKnapsack
c = 0
T = T0
X = [0,...,0]
CurW = 0
bestX = X
while c <= cmax
    j random integer 0<= j <= n-1
    Y = X
    y(j) = 1-x(j)
    if not(y(j) = 1 and CurW + w(j) > M)    (the condition inside the not is the fail)
        if y(j) = 1    (that is if the profit increased)
            X = Y
            CurW = CurW + w(j)
            if P(X) > P(bestX)
                bestX = X
        else
            r = rand(0,1)
            if r < exp(-p(j)/T)
                X = Y
                CurW = CurW - w(j)
    c = c+1
    T = alpha*T
return bestX
```

Sample problem and Solution

- value=[50 40 30 50 30 24 36];
- weight=[5 4 6 3 2 6 7];
- TotalWeight=20;
- beta=0:.01:1;
- n=1000;

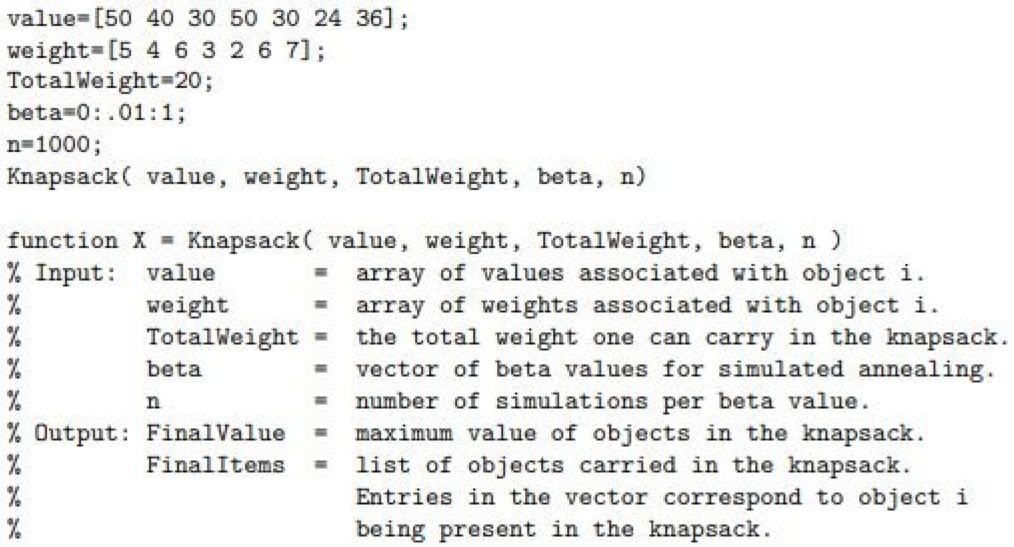
Knapsack(value, weight, TotalWeight, beta, n)

Solution: The final solution is profit value 200 with objects 1, 2, 3, 4, 5.

The simulated annealing code solves it in most of the trials

(some other models sometimes get stuck at value 176 with objects 1, 2, 4, 7)

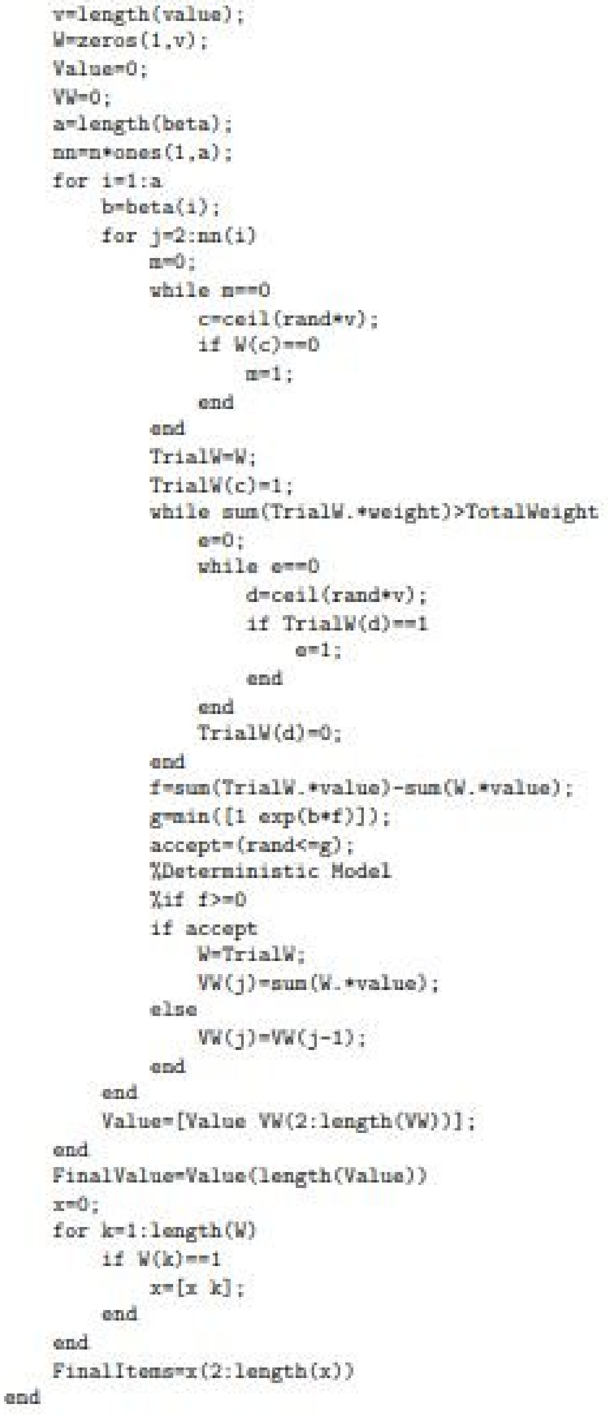
…contd… Sample problem and Solution



```
value=[50 40 30 50 30 24 36];
weight=[5 4 6 3 2 6 7];
TotalWeight=20;
beta=0:.01:1;
n=1000;
Knapsack( value, weight, TotalWeight, beta, n)

function X = Knapsack( value, weight, TotalWeight, beta, n )
% Input:  value       =  array of values associated with object i.
%         weight      =  array of weights associated with object i.
%         TotalWeight =  the total weight one can carry in the knapsack.
%         beta        =  vector of beta values for simulated annealing.
%         n           =  number of simulations per beta value.
% Output: FinalValue  =  maximum value of objects in the knapsack.
%         FinalItems  =  list of objects carried in the knapsack.
%                        Entries in the vector correspond to object i
%                        being present in the knapsack.
```

…contd… Sample problem and Solution



```
    v=length(value);
    W=zeros(1,v);
    Value=0;
    VW=0;
    a=length(beta);
    nn=n*ones(1,a);
    for i=1:a
        b=beta(i);
        for j=2:nn(i)
            m=0;
            while m==0
                c=ceil(rand*v);
                if W(c)==0
                    m=1;
                end
            end
            TrialW=W;
            TrialW(c)=1;
            while sum(TrialW.*weight)>TotalWeight
                e=0;
                while e==0
                    d=ceil(rand*v);
                    if TrialW(d)==1
                        e=1;
                    end
                end
                TrialW(d)=0;
            end
            f=sum(TrialW.*value)-sum(W.*value);
            g=min([1 exp(b*f)]);
            accept=(rand<=g);
            %Deterministic Model
            %if f>=0
            if accept
                W=TrialW;
                VW(j)=sum(W.*value);
            else
                VW(j)=VW(j-1);
            end
        end
        Value=[Value VW(2:length(VW))];
    end
    FinalValue=Value(length(Value))
    x=0;
    for k=1:length(W)
        if W(k)==1
            x=[x k];
        end
    end
    FinalItems=x(2:length(x))
end
```

UNIT-5 (Part4)

Genetic Algorithm (GA)

Genetic Algorithm (GA)

Books :

BOOK1: B1:S. Rajsekaran & G.A. Vijaya Lakshmi Pai, “Neural Networks, Fuzzy Logic and Genetic Algorithm: Synthesis and Applications” Prentice Hall of India.

BOOK2: B2: N.P.Padhy,”Artificial Intelligence and Intelligent Systems” Oxford University Press.

BOOK3: B3: Principles of soft computing –by Sivandudam and Deepa, John Mikey India.

NPTEL Course : Soft Computing, Debasis Samanta

Topics

Genetic Algorithm

Basic concepts, working principle, procedures of GA, flow chart of GA,

Genetic representations, (encoding) Initialization and selection,

Genetic operators, Mutation, Generational Cycle,

TSP using GA

Basic Concepts of **Genetic Algorithm**

Optimization Problems

Background of Genetic Algorithm

Type of Problems Solved with GA

Optimization Problems

Concept of optimization problem

Optimization : Optimum value that is either minimum or maximum value.

Sample: **What is NOT an optimization problem**

$y = f(x)$

Example: $2x- 6y = 11$ or $\mathbf{y = (2x- 11) / 6}$

Can we determine an **optimum value** for y?

Similarly, in the following case $\mathbf{3x + 4y >=56}$

These are really not related to optimization problem!

Defining an optimization problem

Formal Definition:

Maximize (or Minimize)

yi = fi (x1, x2, ..,xn) where i = 1, 2,..,k, k>=1

Subject to

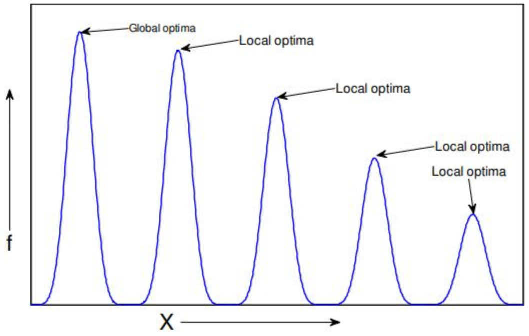
gi (x1, x2,..,xn) **relopi** ci, where i = 1, 2,..,j, j>=0

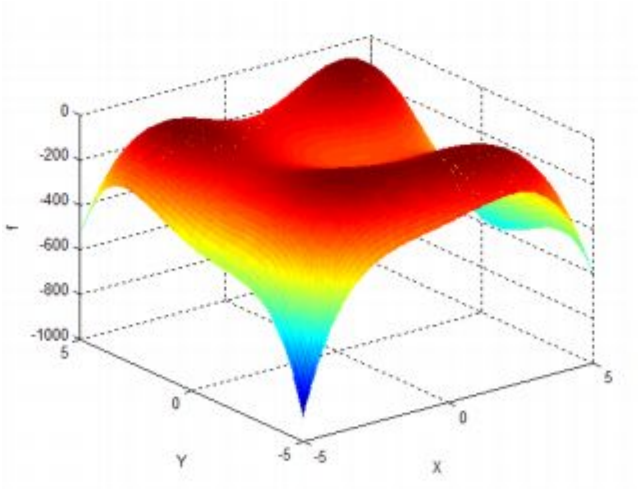
relopi denotes some **relational operator** and **ci** = 1,2, .., j are some constants.

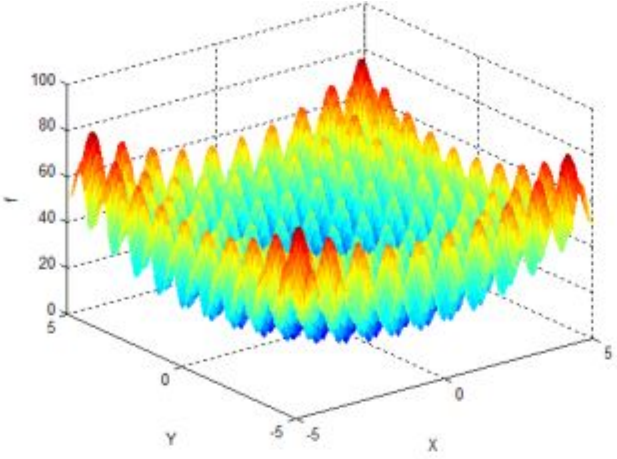
And **xi ROP di** , for all i=1,2...n (n >=1)

Here, **xi** denotes a design parameter and **di** is some constant.

Optimization Process



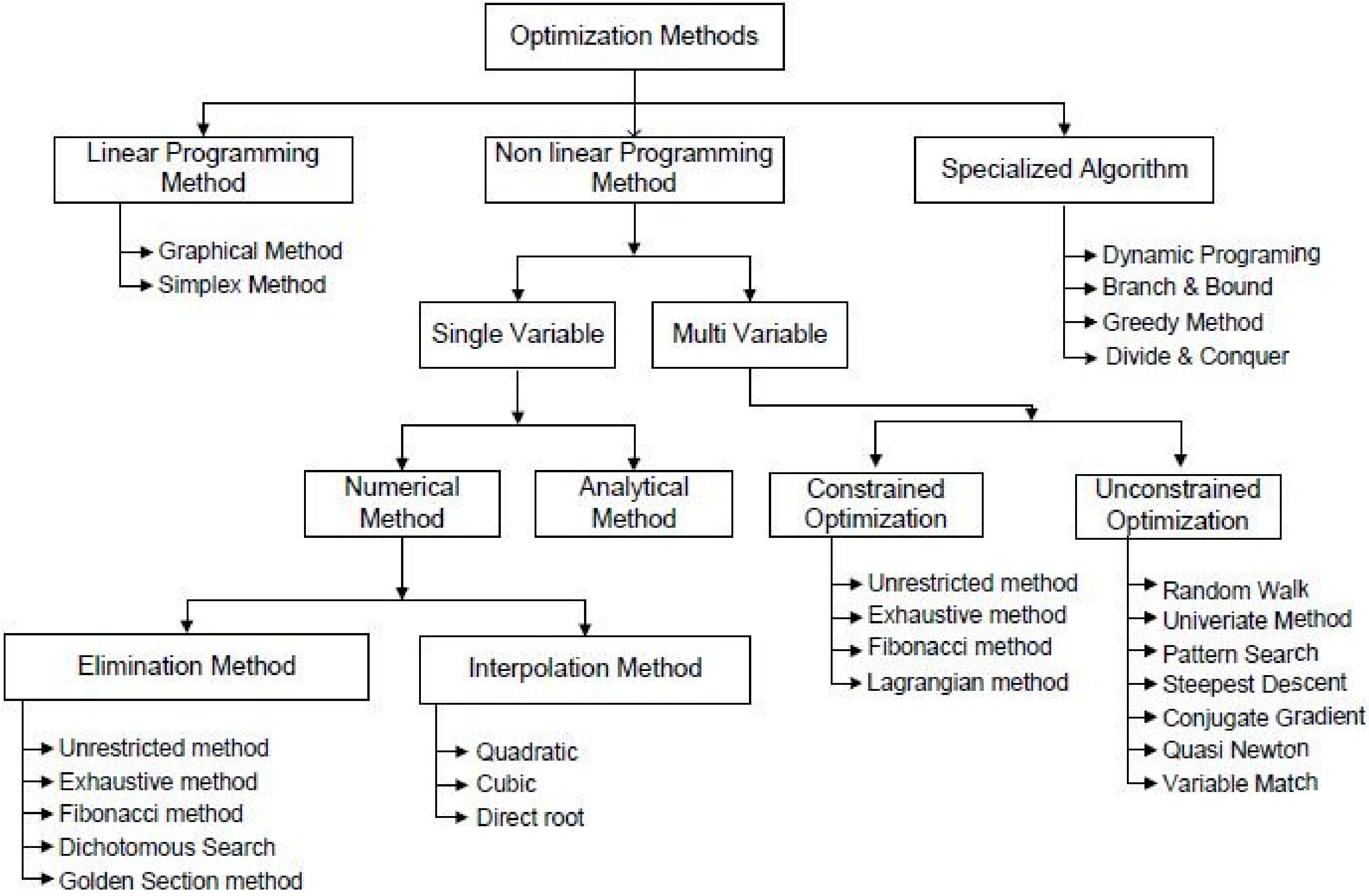




Sample Optimization Problems

- Traveling Salesman Problem
- Knapsack Problem
- Graph Coloring Problem
- Job Machine Assignment Problem
- Coin Change Problem
- Binary search tree construction problem

Traditional approaches to solve optimization problems



Optimization Methods

Linear Programming Method
- Graphical Method
- Simplex Method

Non linear Programming Method
- Single Variable
  - Numerical Method
    - Elimination Method
      - Unrestricted method
      - Exhaustive method
      - Fibonacci method
      - Dichotomous Search
      - Golden Section method
    - Interpolation Method
      - Quadratic
      - Cubic
      - Direct root
  - Analytical Method
- Multi Variable
  - Constrained Optimization
    - Unrestricted method
    - Exhaustive method
    - Fibonacci method
    - Lagrangian method
  - Unconstrained Optimization
    - Random Walk
    - Univeriate Method
    - Pattern Search
    - Steepest Descent
    - Conjugate Gradient
    - Quasi Newton
    - Variable Match

Specialized Algorithm
- Dynamic Programing
- Branch & Bound
- Greedy Method
- Divide & Conquer

Limitations of the traditional optimization approach

- Computationally expensive.
- For a discontinuous objective function, methods may fail.
- Method may not be suitable for parallel computing.
- Discrete (integer) variables are difficult to handle.
- Methods may not necessarily adaptive.

Soft Computing techniques have been evolved to address the above mentioned limitations of solving optimization problem with traditional approaches.

Evolutionary Algorithms

The algorithms, which follow some biological and physical behaviors:

Biologic behaviors:

- Genetics and Evolution –> Genetic Algorithms (GA)
- Behavior of ant colony –> Ant Colony Optimization (ACO)
- Human nervous system –> Artificial Neural Network (ANN)

In addition to that there are some algorithms inspired by some physical behaviors:

Physical behaviors:

- Annealing process –> Simulated Annealing (SA)
- Swarming of particle –> Particle Swarming Optimization (PSO)
- Learning –> Fuzzy Logic (FL)

Background of Genetic Algorithm

Evolutionary Algorithms

The algorithms, which follow some biological and physical behaviors:

Biologic behaviors:

- Genetics and Evolution –> Genetic Algorithms (GA)
- Behavior of ant colony –> Ant Colony Optimization (ACO)
- Human nervous system –> Artificial Neural Network (ANN)

In addition to that there are some algorithms inspired by some physical behaviors:

Physical behaviors:

- Annealing process –> Simulated Annealing (SA)
- Swarming of particle –> Particle Swarming Optimization (PSO)
- Learning –> Fuzzy Logic (FL)

Genetic Algorithm

It is a subset of evolutionary algorithm:

- Ant Colony optimization
- Swarm Particle Optimization

Models biological processes:

- Genetics
- Evolution

To optimize highly complex objective functions:

- Very difficult to model mathematically
- **NP-Hard** (also called combinatorial optimization) problems (which are computationally very expensive)
- Involves large number of parameters (discrete and/or continuous)

Background of GA

- First time introduced by Prof**. John Holland** (of Michigan University, USA, **1965**).
- But, the first article on GA was published in **1975**.

Basic Principles of GA

Principles of GA based on two fundamental biological processes:

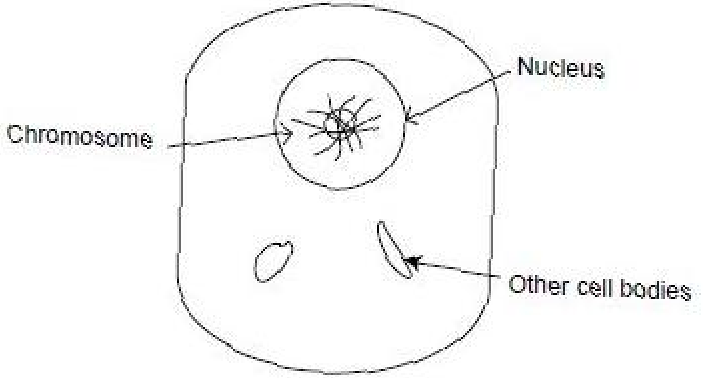
- **Genetics: Gregor Johan Mendel (1865)**
- **Evolution: Charles Darwin (1875)**

A Brief Account on Genetics and Evolution

A Brief Account on Genetics

Cell and Chromosomes

- Cells are the basic building blocks in living bodies. Each cell carries the **basic unit of heredity**, called **gene**



For a particular specie,

number of chromosomes is fixed.

Examples

- Mosquito: 6
- Frogs: 26
- Human: 46 (23 pairs)
- Goldfish: 94

Genetic code

Spiral helix of protein substance is called DNA.

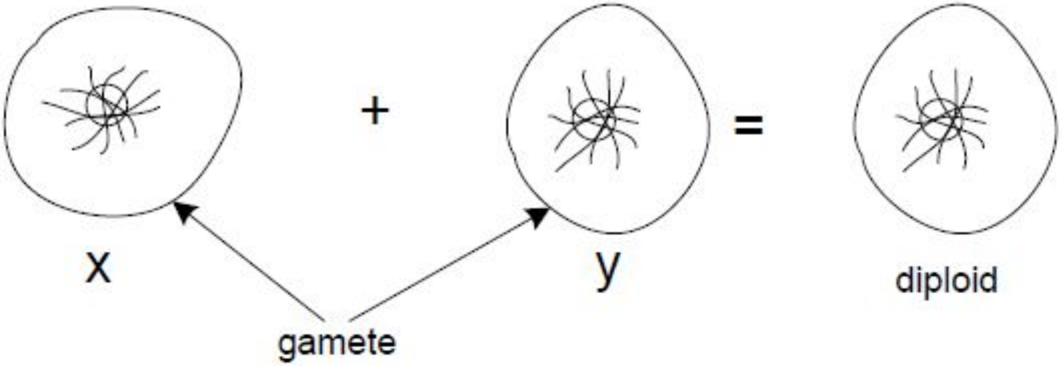
For a specie,

DNA code is unique, that is, vary uniquely from one to other.

DNA code (inherits some characteristics from one generation to next generation)

is used as biometric trait.

Reproduction



Organism's cell : Cell division

haploid | (Reproductive cell has half the number of chromosomes)

diploid | Each chromosome from both haploids are combined to have full numbers

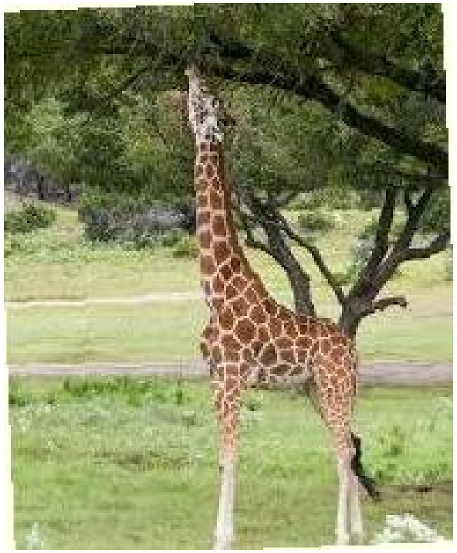
A Brief Account on Evolution

Evolution : Natural Selection

Four primary premises:

- **Information propagation: An offspring has many of its** characteristics of its parents (i.e. information passes from parent to its offspring). **[Heredity]**
- **Population diversity: Variation in characteristics in the next** generation. **[Diversity]**
- **Survival for existence: Only a small percentage of the offspring** produced survive to adulthood. **[Selection]**
- **Survival of the best: Offspring survived depends on their** inherited characteristics. **[Ranking]**

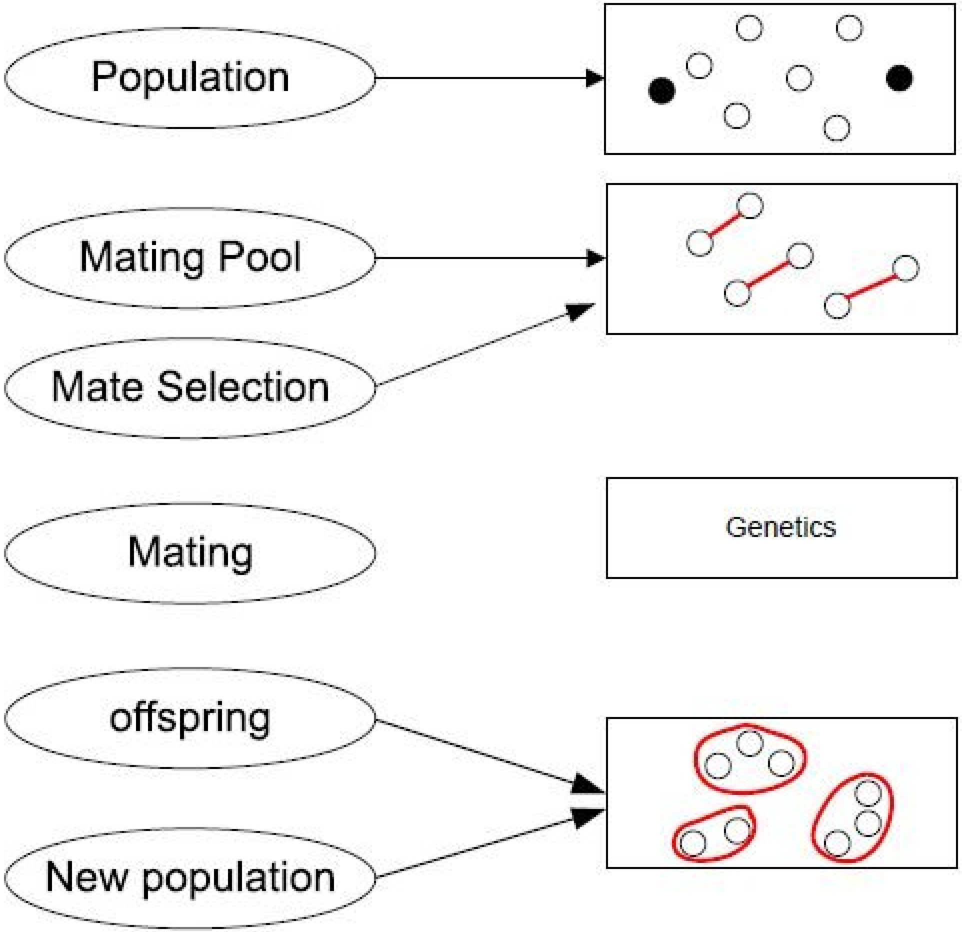
Evolution Example : Giraffes have long necks



- Giraffes with slightly longer necks could feed on leaves of higher branches when all lower ones had been eaten off.
- They had a better chance of survival.
- Favorable characteristic propagated through generations of giraffes.
- Now, evolved species has long necks
- This longer necks may have due to **the effect of mutation** initially. However as it was favorable, this was propagated over the generations.

(**Mutation:** To make the process forcefully dynamic when variations in population going to stable)

Biological process : A quick overview



Introduction to Genetic Algorithms (GAs)

What is a GA?

- A **genetic algorithm** is a search heuristic that is inspired by Charles Darwin’s theory of natural evolution
- This algorithm reflects the process of **natural selection** where the **fittest individuals** are **selected for reproduction** in order to **produce offspring of the next generation**

GAs for problem solving

- Genetic Algorithms are the **heuristic search** and **optimization techniques** that **mimic** the process of **natural evolution**

- Definition: Genetic algorithm is a **population-based probabilistic search** and **optimization techniques**, which works based on the mechanisms of **natural genetics** and **natural evolution**

Which of the following is not true for Genetic algorithms?

(a) It is a probabilistic search algorithm.

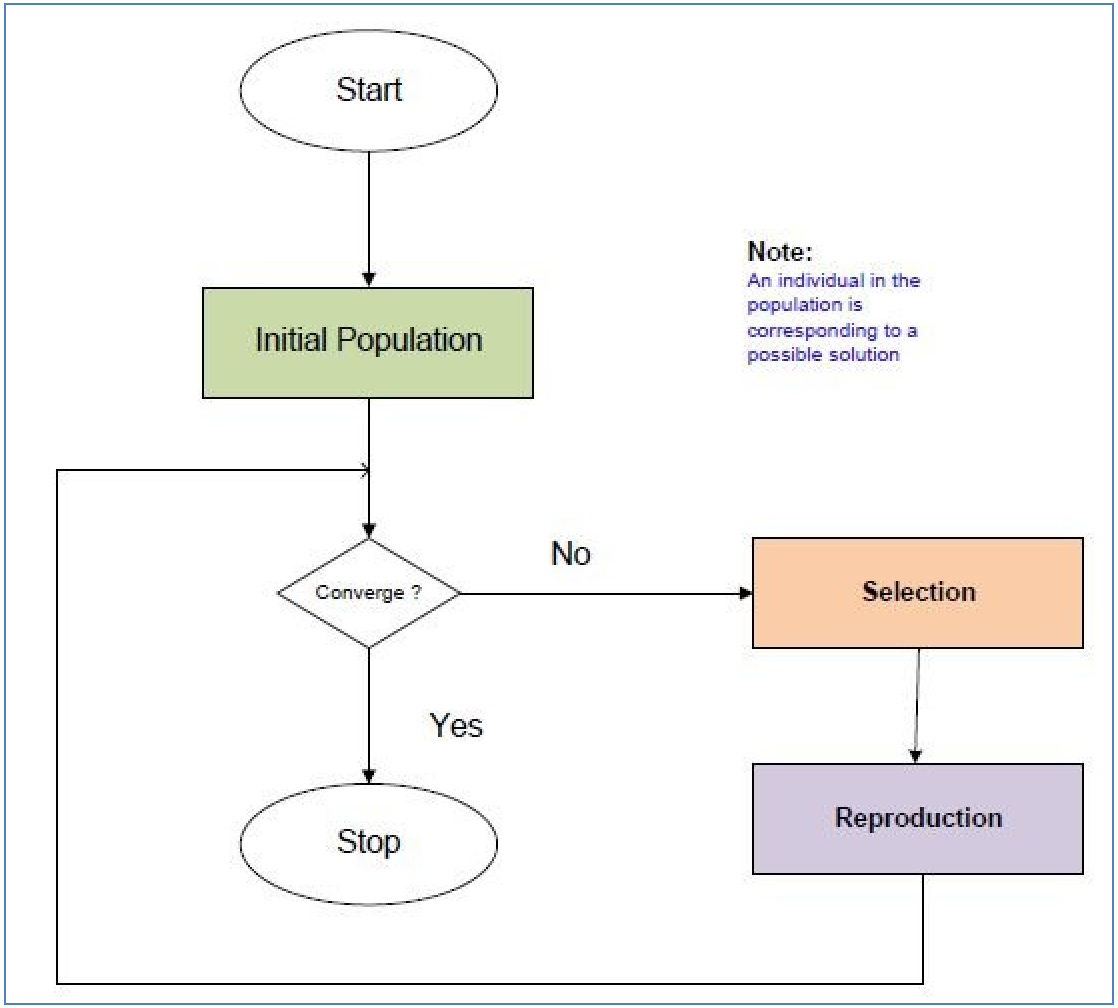
(b) *It is guaranteed to give global optimum solutions*.

(c) If an optimization problem has more than one solution, then it will return all the solutions.

(d) It is an iterative process suitable for parallel programming.

Working principle of GA

Framework of GA

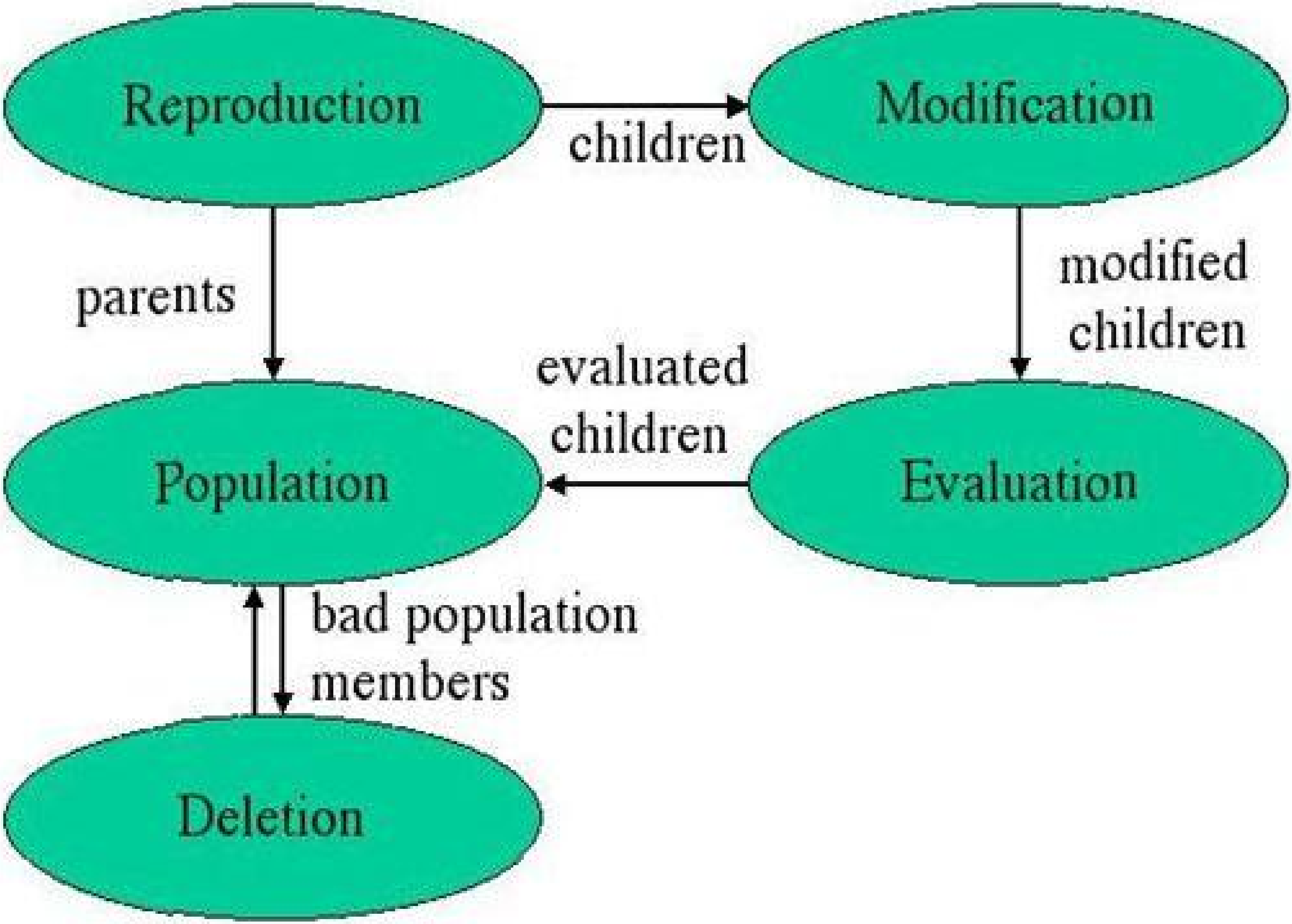


Note:
An individual in the population is corresponding to a possible solution

Note:

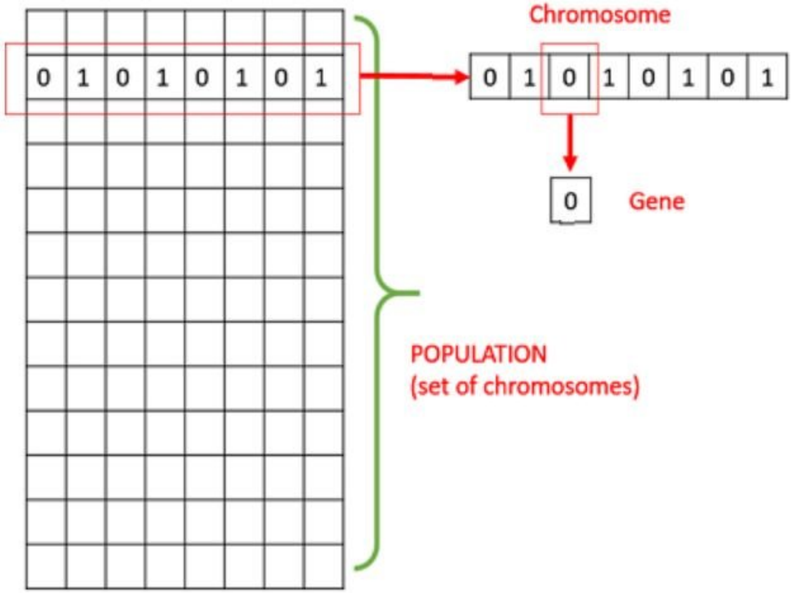
1 GA is an iterative process.
2 It is a searching technique.
3 Working cycle with / without convergence.
4 Solution is not necessarily guaranteed. Usually, terminated with a local optima.

Anatomy of GA



Basic elements of GAs

- **populations** of chromosomes,



- **selection** according to fitness,
- **crossover** to produce new offspring,
- and random **mutation** of new offspring

Population of Chromosomes

- The chromosomes in GAs represent the space of **candidate solutions**
- Possible chromosomes **encodings** **are binary**, **permutation**, **value**, and **tree encodings**.
    - For the **Knapsack problem**, we use binary encoding, where every chromosome is a string of bits, 0 or 1.

0	0	1	0	1	1	1	0	0	1

Value: Real / integer

0.5	0.2	0.6	0.8	0.7	0.4	0.3	0.2	0.1	0.9

1	2	3	4	3	2	4	1	2	1

TSP : Permutations

1	5	9	8	7	4	2	3	6	0

Fitness function

- GAs require a **fitness function** which allocates a score to each chromosome in the current population.
- Thus, it can calculate how well the solutions are coded and how well they solve the problem

Selection

- The selection process is based on **fitness.**
- Chromosomes that are evaluated with higher values (fitter) will most likely be selected to reproduce, whereas, those with low values will be discarded.
- The fittest chromosomes may be selected several times, however, the number of chromosomes selected to reproduce is equal to the population size, therefore, keeping the size constant for every generation.
- This phase has an **element of randomness** just like the survival of organisms in nature.
- The most used selection methods, are **roulette-wheel, rank selection, steady-state selection, and some others**.
- Moreover, to increase the performance of GAs, the selection methods are enhanced by eiltism.
  - Elitism is a method, which first copies a few of the top scored chromosomes to the new population and then continues generating the rest of the population. Thus, it prevents loosing the few best found solutions.

Q) The purpose of the fitness evaluation operation is

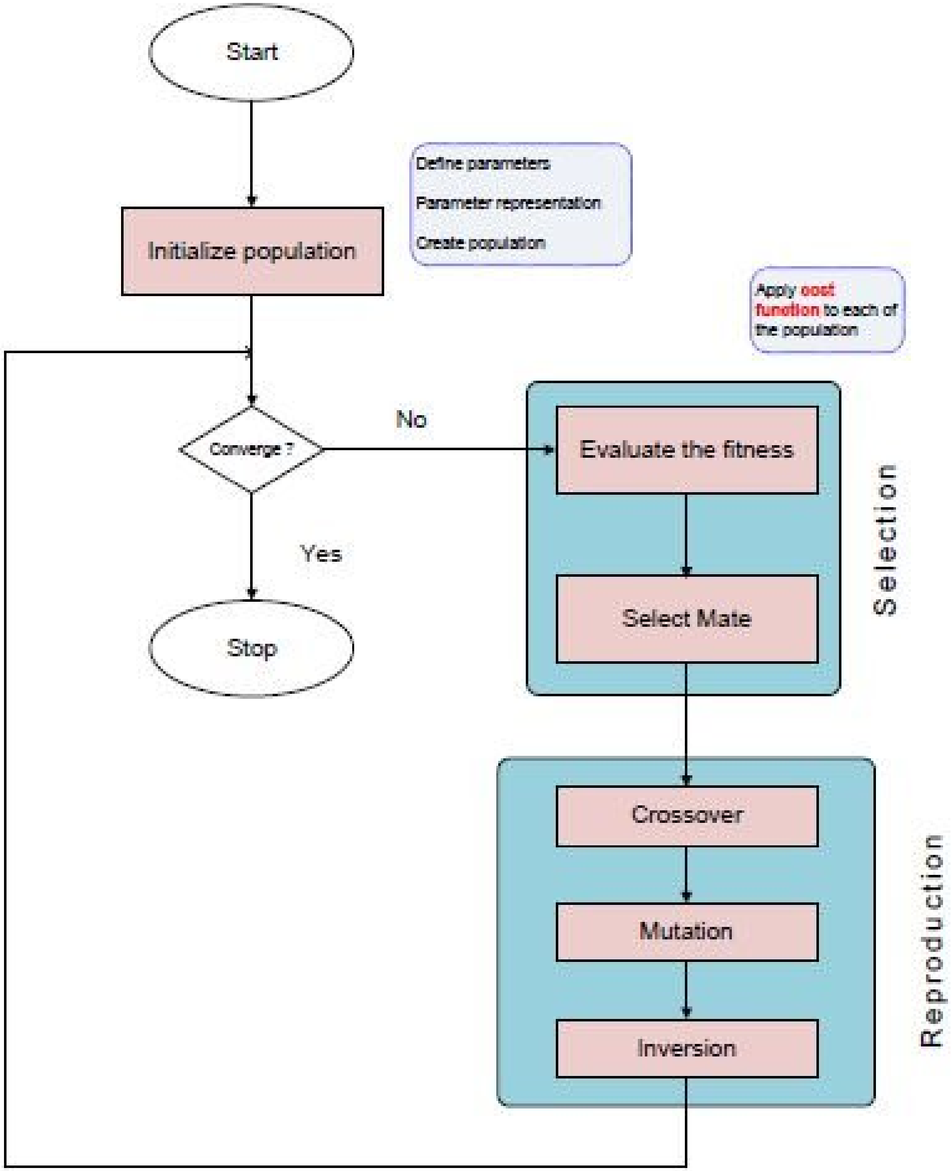
(a) To check whether all individual satisfies the constraints given in the problem.

(b) To decide the termination point.

(c) To select the best individuals.

(d) To identify the individual with worst cost function.

Framework of GA: A detail view



Crossover

- Crossover is the process of combining the bits of one chromosome with those of another.
- This is to create an offspring for the next generation that inherits traits of both parents.
- Crossover randomly chooses a locus and exchanges the subsequences before and after that locus between two chromosomes to create two offspring

Example : Crossover

- For example, consider the following parents and a **crossover point** at position 3:

Parent 1	1 0 0 | 0 1 1 1
Parent 2	1 1 1 | 1 0 0 0
Offspring 1	1 0 0 1 0 0 0
Offspring 2	1 1 1 0 1 1 1

P1: 10010100
P2: 01010101

O1: 1001 0101
O2: 0101 0100

O1: 1001010 1
Q2: 0101010 0

Mutation

- Mutation is performed after crossover
- It prevents falling of, all solutions in the population, into a local optimum of solved problem
- Mutation changes the new offspring by **flipping bits from 1 to 0** or **from 0 to 1**.
- Mutation can occur at each bit position in the string with some probability, usually very small (e.g. 0.001)

Mutation

- For example, consider the following chromosome with mutation point at position 2:

Not mutated chromosome: 1 *0* 0 0 1 1 1

Mutated: 1 *1* 0 0 1 1 1

- The 0 at position 2 flips to 1 after mutation.

O1: 1001 0*1*01

Mutated : 10010*0*01

Knapsack Problem

Suppose we have a knapsack that has a capacity of 13 kg and 3 potential items (labeled ‘A,’ ‘B,’ ‘C’) of different weights and different benefits.

We want to include in the knapsack only these items that will have the greatest total benefit within the constraint of the knapsack’s capacity. The weights and benefits are as follows:

Item #	A	B	C
Benefit	4	3	5
Weight	6	7	8

Possible subsets of items

A B C	Wt of the set(13 max)	Benefit of the set
000	0	0
001	8	5
010	7	3
0 1 1	15	–
1 00	6	4
1 0 1	14	-
1 1 0	13	**7**
1 1 1	21	-

Optimization problem solving with GA

For the optimization problem, identify the following:

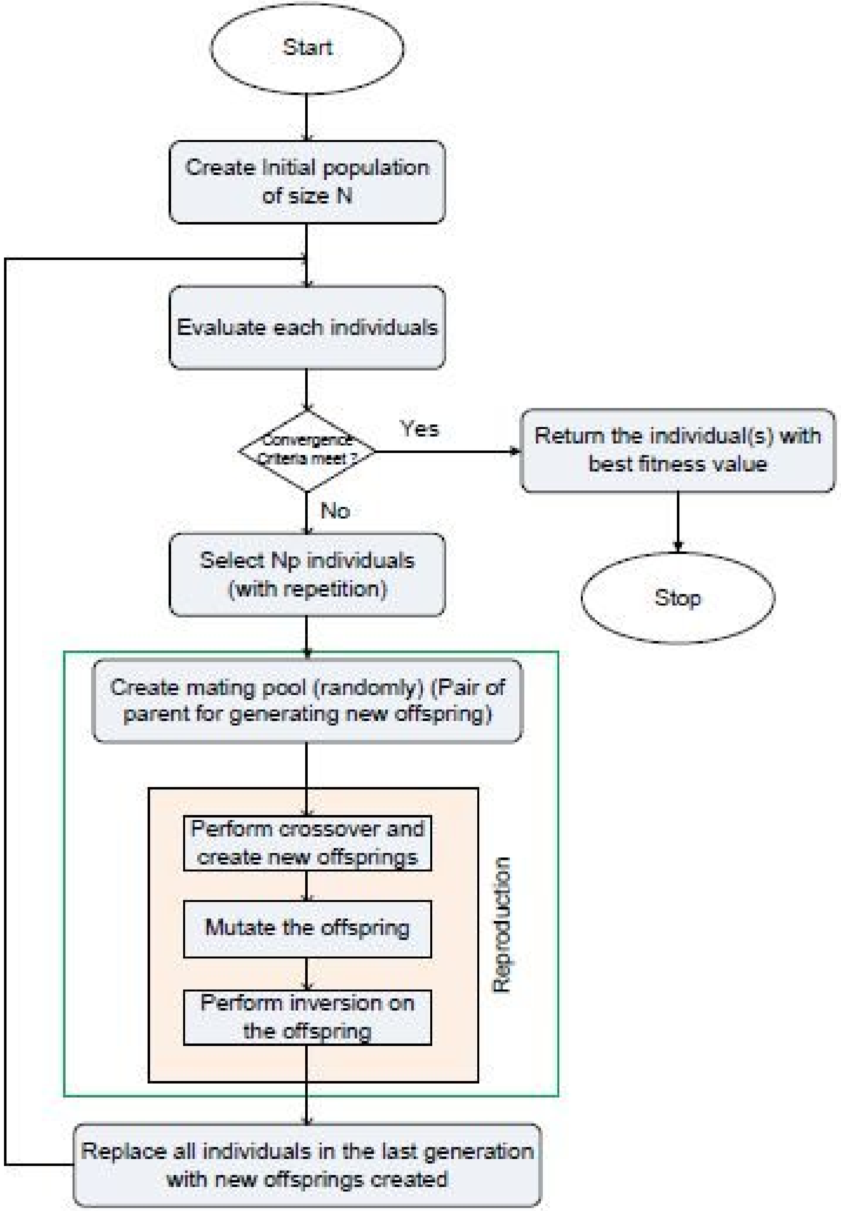
- **Objective function(s)**
- **Constraint(s)**
- **Input parameters**
- **Fitness evaluation (it may be algorithm or mathematical formula)**
- **Encoding**
- **Decoding**

GA Operators

GA implementation involves realization of the following operations

1. **Encoding:** How to represent a solution to fit with GA framework.
2. **Convergence:** How to decide the termination criterion.
3. **Mating pool:** How to generate next solutions.
4. **Fitness Evaluation:** How to evaluate a solution.
5. **Crossover:** How to make the diverse set of next solutions.
6. **Mutation:** To explore other solution(s).
7. **Inversion:** To move from one optima to other.

Simple GA (SGA)


Start
Create Initial population of size N
Evaluate each individuals
Convergence Criteria meet ?
Yes
Return the individual(s) with best fitness value
Stop
No
Select Np individuals (with repetition)
Create mating pool (randomly) (Pair of parent for generating new offspring)
Perform crossover and create new offsprings
Mutate the offspring
Perform inversion on the offspring
Reproduction
Replace all individuals in the last generation with new offsprings created

SGA Parameters

Initial population size : N

Size of mating pool, Np : Np = p % of N

Convergence threshold δ

Mutation μ

Inversion η

Crossover ρ

Simple GA features:

- Have overlapping generation (Only fraction of individuals are replaced).
- Computationally expensive.
- Good when initial population size is large.
- In general, gives better results.
- Selection is biased toward more highly fit individuals; Hence, the average fitness (of overall population) is expected to increase in succession.
- The best individual may appear in any iteration.

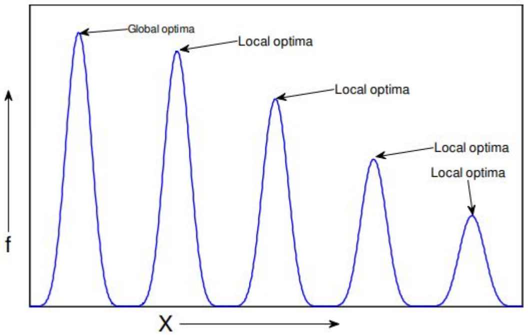
Genetic Algorithms (GAs)

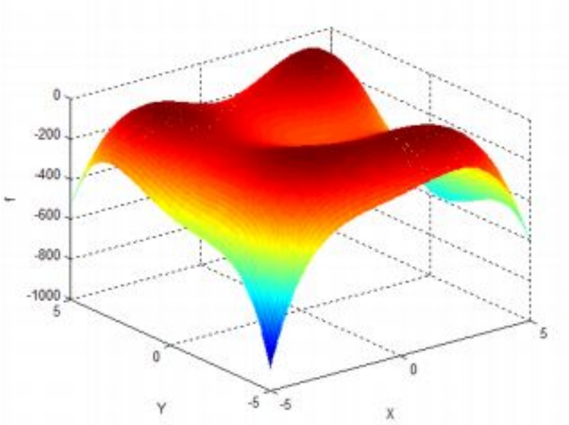
- Type of optimization algorithm
- Inspired by the principles of
  - Evolution
  - Natural selection
- **Particularly well-suited for :**
  - Solving problems that involve finding an **optimal** or **near-optimal solution** within a **large, complex search space**

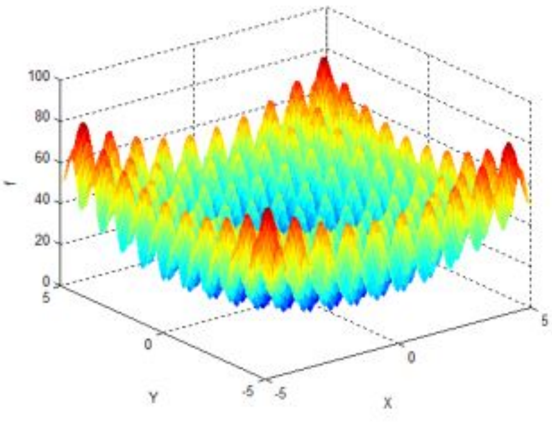
Some Computational Problems for which Genetic Algorithms (GAs) have been shown to give very good results

Function optimization

- Genetic algorithms can be used to find the **minimum or maximum of a function,** even when the function **is complex, noisy, or has multiple peaks**







Sample Function optimization problems

- **Portfolio Optimization**: In finance, portfolio optimization is a classic example of an optimization problem that involves **finding the minimum or maximum value of a complex function**. The objective is to find the portfolio that maximizes return while minimizing risk. The function involves multiple assets with varying returns and risks, and often has noisy data due to market fluctuations.

- **Neural Network Training**: In machine learning, the process of training a neural network involves finding the optimal weights and biases that minimize the error between the predicted and actual output.
  - The **loss function** used in training the neural network is typically complex, noisy, and has multiple local minima.

- **Design Optimization**: In engineering, the design of complex systems involves finding the optimal set of design parameters that maximize or minimize certain objectives, such as minimizing weight, maximizing strength, or minimizing cost. These problems often **involve complex, non-linear functions** with **multiple peaks.**

- **Drug Discovery**: In pharmaceuticals, drug discovery involves identifying compounds that are effective at treating a particular disease. This process involves finding the optimal chemical structure that maximizes efficacy while minimizing toxicity. The **function used to evaluate the efficacy and toxicity of a compound** is often complex, noisy, and has multiple peaks.

- **Traffic Optimization**: In transportation, traffic optimization involves finding the optimal traffic flow that minimizes congestion and travel time. This problem involves **multiple variables**, such as traffic volume, road capacity, and traffic signal timings, and often has a complex, non-linear function with multiple peaks.

Travelling salesman problem (TSP)

- TSP= find shortest possible route that visits a set of cities and returns to the starting point
- GAs very effective at finding good solutions for this problem

Machine learning with GAs

- Optimize the **parameters** of NN / Decision Trees to improve their accuracy and performance

Resource allocation

Optimize allocation of resources, - personnel, equipment, or funding, to **maximize efficiency and minimize waste**

Scheduling

- Like - employee scheduling or production scheduling,
  - to minimize idle time and maximize productivity

Image recognition

- GAs to optimize the parameters of image recognition algorithms, such as CNNs to improve their accuracy and performance
-
- It's worth noting that genetic algorithms are not always the best or most efficient approach for these problems, and their effectiveness can depend on factors such as the size and complexity of the search space, the quality of the fitness function, and the chosen parameters of the algorithm.

Game playing

- GAs can be used to **develop strategies for playing games**, (Chess / Go), to compete with or even surpass human performance

Note:

- Genetic algorithms are not always the best or most efficient approach for these problems, and their effectiveness **can depend on** factors such
  - Size and complexity of the search space
  - Quality of the fitness function
  - Chosen parameters of the algorithm

GA OPERATORS: DETAILS

Details

GA Operators

Following are the GA operators in Genetic Algorithms.

1. Encoding
2. Convergence test
3. Mating pool
4. Fitness
5. Evaluation
6. Crossover
7. Mutation

Inversion

Encoding Operation

1. **Encoding**
2. Convergence test
3. Mating pool
4. Fitness
5. Evaluation
6. Crossover
7. Mutation

Inversion

Different Encoding Schemes

- **Different GAs**
  - Simple Genetic Algorithm (SGA)
  - Steady State Genetic Algorithm (SSGA)
  - Messy Genetic Algorithm (MGA)

- **Encoding Schemes**
  - Binary encoding
  - Real value encoding
  - Order encoding
  - Tree encoding

Different Encoding Schemes

Often, GAs are specified according to the encoding scheme it follows.

For example:

- Encoding Scheme
- Binary encoding –> Binary Coded GA or simply **Binary GA**
- Real value encoding –> Real Coded GA or simply **Real**
- **GA** Order encoding –> **Order GA** (also called as
- **Permuted GA**) Tree encoding

Encoding Schemes in GA

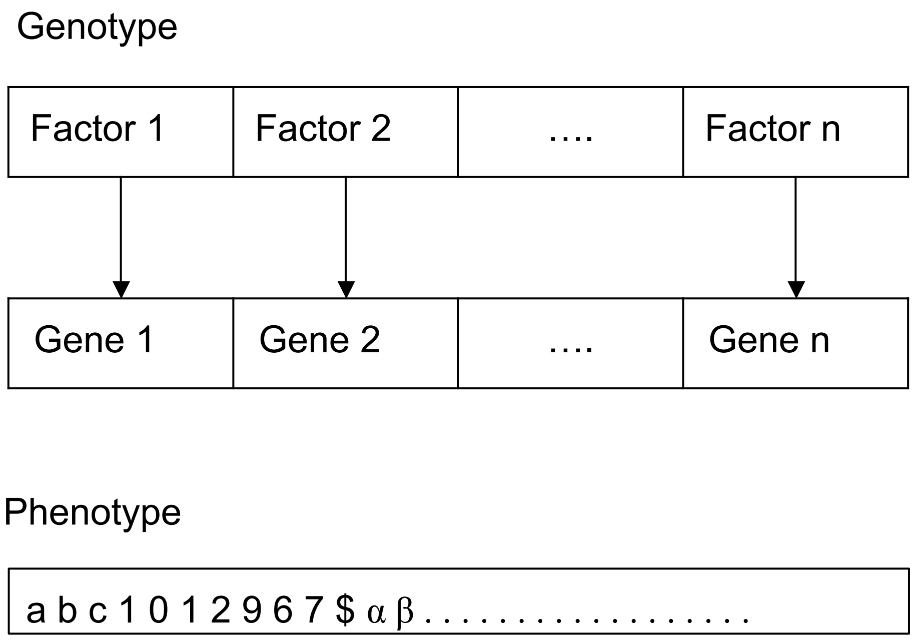
Genetic Algorithm uses metaphor consisting of two distinct elements :

1. Individual
2. Population

An individual is a single solution while a population is a set of individuals at an instant of searching process.

Individual Representation :Phenotype and Genotype

- An individual is defined by a chromosome. A chromosome stores genetic information (called phenotype) for an individual.
- Here, a chromosome is expressed in terms of factors defining a problem.



Genotype

Factor 1	Factor 2	….	Factor n
Gene 1	Gene 2	….	Gene n

Phenotype

a b c 1 0 1 2 9 6 7 $ α β . . . .

Chromosome

Individual Representation :Phenotype and Genotype

Note :

- A gene is the GA’s representation of a single factor (i.e. a design parameter), which has a domain of values (continuous, discontinuous, discrete etc.) symbol, numbering etc.
- In GA, there is a mapping from genotype to phenotype. This eventually decideds the performance (namely speed and accuracy) of the problem solving.

Encoding techniques

There are many ways of encoding:

1. **Binary encoding:** Representing a gene in terms of bits (0s and 1s).
2. **Real value encoding:** Representing a gene in terms of values or symbols or string.
3. **Permutation (or Order) encoding:** Representing a sequence of elements)
4. **Tree encoding:** Representing in the form of a tree of objects.

Binary Encoding

In this encoding scheme, a gene or chromosome is represented by a string (fixed or variable length) of binary bits (0’s and 1’s)

A : 0 1 1 0 0 1 0 1 0 1 0 1 0 1 1 1 1 0 Individual 1

B : 0 0 1 0 1 0 1 1 1 0 1 0 1 0 1 0 0 0 Individual 2

Example: 0-1 Knapsack problem

- There are *n* items, each item has its own cost (c_i) and weight (w_i).
- There is a knapsack of total capacity *w* .
- The problem is to take as much items as possible but not exceeding the capacity of the knapsack.

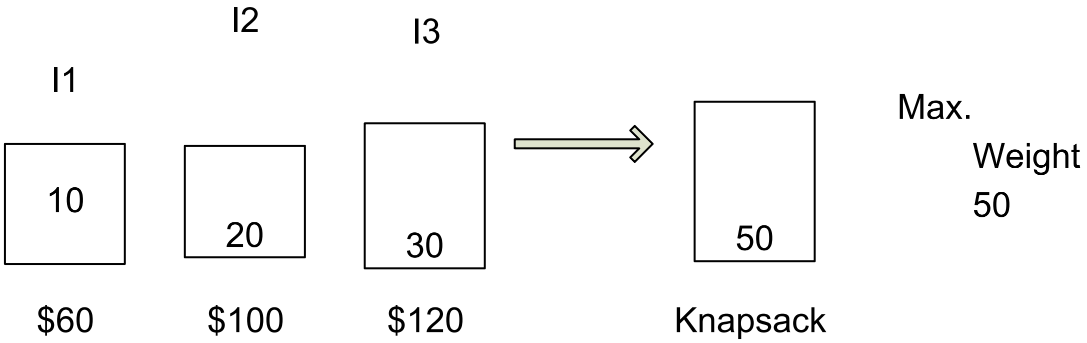
This is an optimization problem and can be better described as follows.

Maximize : $\sum_i c_i * w_i * x_i$

Subject to $\sum x_i * w_i \leq W$ where $x_i \in [0 \cdots 1]$

Example: 0-1 Knapsack problem

Consider the fallowing, an instance of the 0-1 Knapsack problem.



I1 10 $60

I2 20 $100

I3 30 $120

50 Knapsack

Max. Weight 50

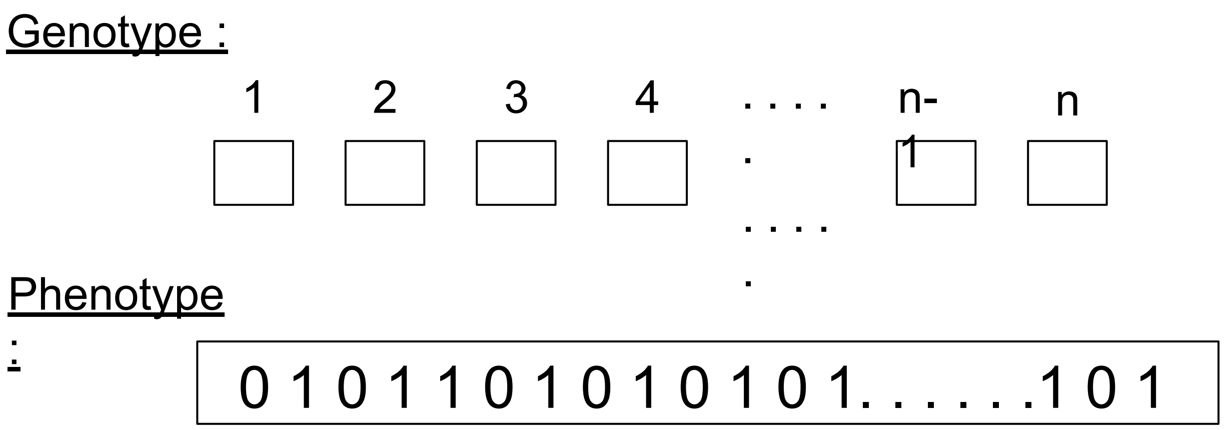
Brute force approach to solve the above can be stated as follows:

- Select at least one item
- [10], [20], [30], [10, 20], [10, 30], [, 20, 30], [10, 20, 30]
- So, for n-items, are there are $2^n -$ 1 trials.

0 - means item not included and 1 - means item included

Example: 0-1 Knapsack problem

The encoding for the 0-1 Knapsack, problem, in general, for *n* items set would look as follows.



Genotype :

Phenotype :

0 1 0 1 1 0 1 0 1 0 1 0 1. . . . . . .1 0 1

A binary string of n-bits

Few more examples

- **Example 1 :**

Minimize :

$$f(x) = \frac{x^2}{2} + \frac{125}{x}$$

where $0 \leq x \leq 15$ and x is any discrete integer value.

Genotype :

x

Phenotype :

0 1 1 0 1

A binary string of 5-bits

Few more examples

- **Example 2 :**

Maximize :

$$f(x, y) = x^3 - x^2 y + x y^2 + y$$

3

Subject to :

and

$$x + y \leq 10$$

$$1 \leq x \leq 10$$

$$-10 \leq y \leq$$

10

Genotype :

x	y

Phenotype :

0 1 1 0 1	1 1 0 0 1

Two binary string of 5-bits each

Pros and cons of Binary encoding scheme

- **Limitations:**
  1. Needs an effort to convert into binary from
  2. Accuarcy depends on the binary

     reprresentation
- **Advantages**
  :
  1. Since operations with binary represntation is faster, it provide a faster implementations of all GA operators and hence the execution of GAs.
  2. Any optimization problem has it binary-coded GA implementation

Real value encoding

- The real-coded GA is most suitable for optimization in a continuous search space.
- Uses the direct representations of the design paparmeters.
- Thus, avoids any intermediate encoding and decoding steps.

Genotype :

x	y

Phenotype :

5.28	-475.36

Real-value representation

Real value encoding with binary codes

Methodology: Step 1 [Deciding the precision]

For any continuous design variable *x* such that $X_L \leq x \leq X_U$, and if ε is the precision required, then string length *n* should be equal to

$$n = \log_2 \left(\frac{X_U - X_L}{\varepsilon} \right)$$

where $X_L \leq x \leq X_U$

Equivalently,

$$\varepsilon = \left(\frac{X_U - X_L}{2^n} \right)$$

In general, $\varepsilon = [0 \cdots 1]$. It is also called, *Obtaianable accuracy*

Note: If $\varepsilon = 0.5$, then 4.05 or $4.49 \equiv 4$ and 4.50 or $4.99 \equiv 4.5$ and so on.

Real value encoding: Illustration 1

1. **Example 1:**

   $1 \leq x \leq 16$, $n = 6$. What is the accuracy?

   $$\varepsilon = \frac{16-1}{2^6} = \frac{15}{64} = 0.249 \approx 0.25$$

2. **Example 2:**

   What is the obtainable accuracy, for the binary representation for a variable X in the range range $20.1 \leq X \leq 45.6$ with 8-bits?

3. **Example 3:**

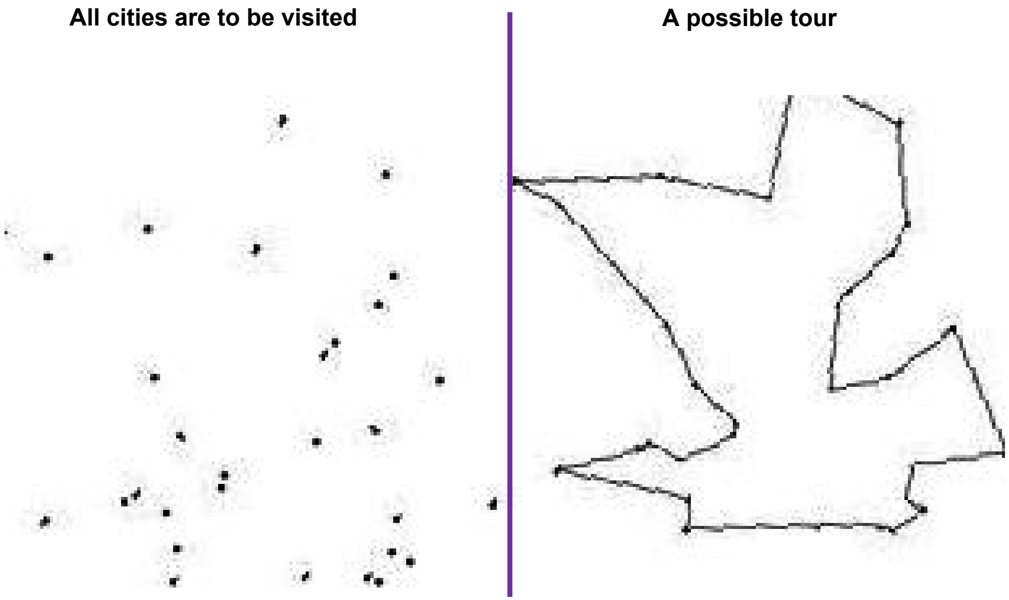
   In the above case, what is the binary representation of $X = 34.35$?

Order Encoding

Let us have a look into the following instance of the Traveling Salesman Problem (TSP).

TSP

- Visit all the cities
- One city once only
- Starting and ending city is the same



All cities are to be visited

A possible tour

How we can formally define the TSP?

Order Encoding for TSP

Understanding the TSP:

There is a cost of visiting a city from another city and hence the total cost of visiting all the cities but exactly once (except the starting city).

Objective function: To find a tour (i.e. a simple cycle covering all the cities) with a minimum cost involved.

Constraints:

- All cities must be visited.
- There will be only one occurrence of each city (except the starting city).

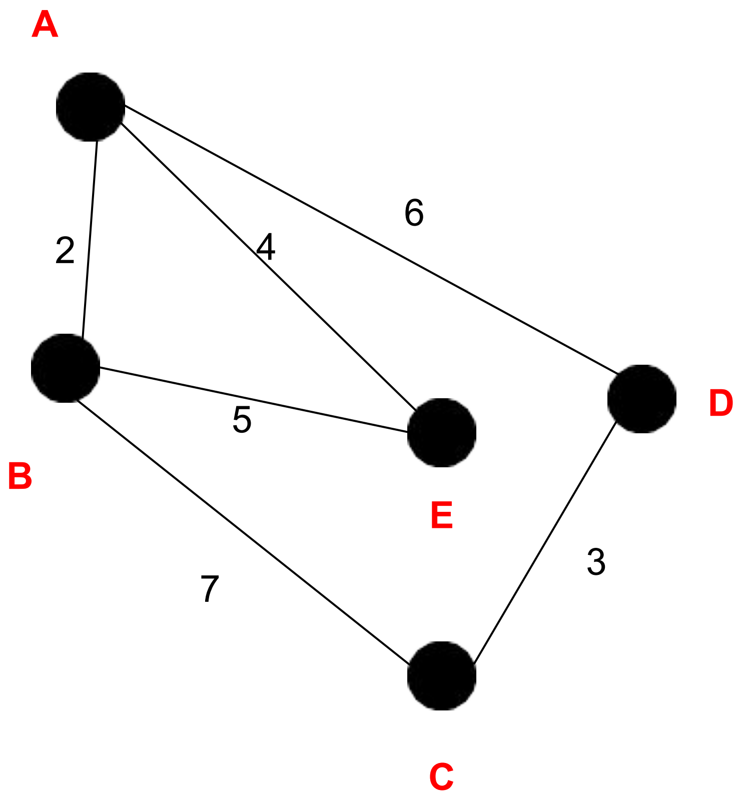
Design parameters:

- Euclidean distance may be taken as the measurement of the cost, otherwise, if it is specified explicitly.
- The above stated information are the design variables in this case.

We are to search for the best path out of *n*! possible paths.

d	A	B	C	D	E
A	0	2	∝	6	4
B	2	0	7	∝	5
C	∝	7	0	3	1
D	6	∝	3	0	∝
E	4	5	1	∝	0

d= Distance matrix



Connectivity among cities

Defining the TSP

Minimizing

$$\text{cost} = \sum_{i=0}^{n-2} d(c_i, c_{i+1}) + d(c_{n-1}, c_0)$$

Subject to

$P = [c_0, c_1, c_2, \cdots, c_{n-1}, c_0]$ where $c_i \in X$;

Here, P is an ordered collection of cities and $c_i \neq c_j$ such that $\forall i, j = 0, 1, \cdots, n-1$

Note: P represents a possible tour with the starting cities as c_0.

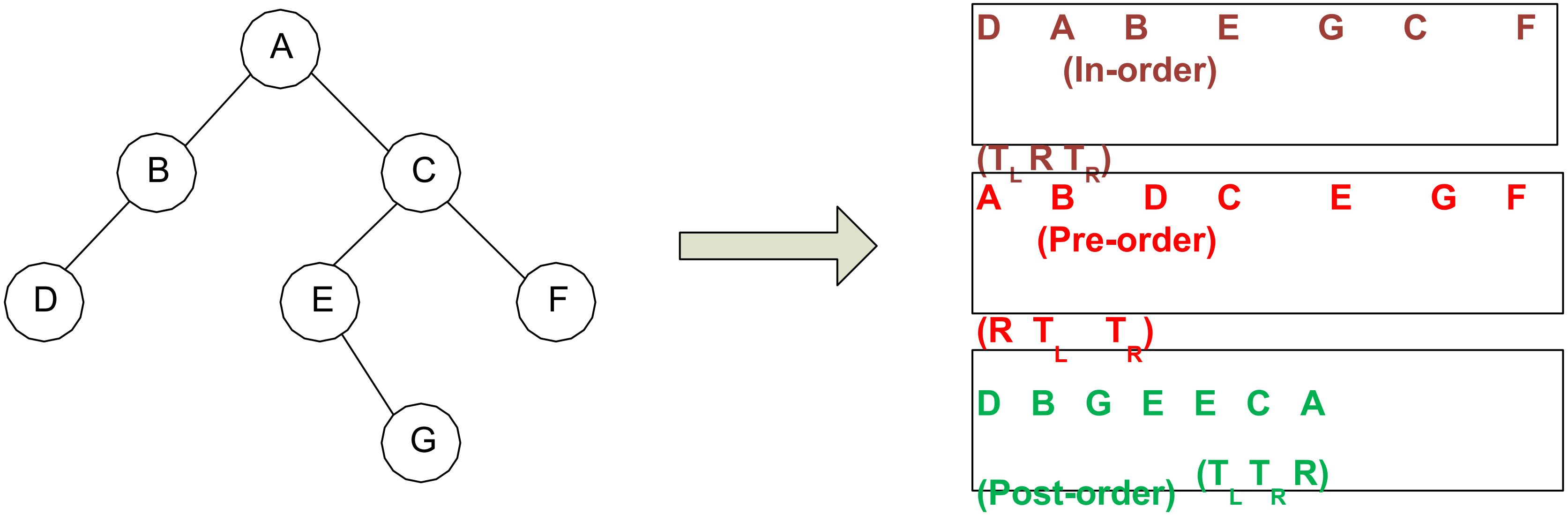
and

$X = x_1, x_2, \cdots, x_n$, set of n number of cities,

$d(x_i, x_j)$ is the distance between any two cities x_i and x_j.

Tree encoding

In this encoding scheme, a solution is encoded in the form of a binary tree.



D A B E G C F
(In-order)

$(T_L\ R\ T_R)$

A B D C E G F
(Pre-order)

$(R\ T_L\ T_R)$

D B G E E C A

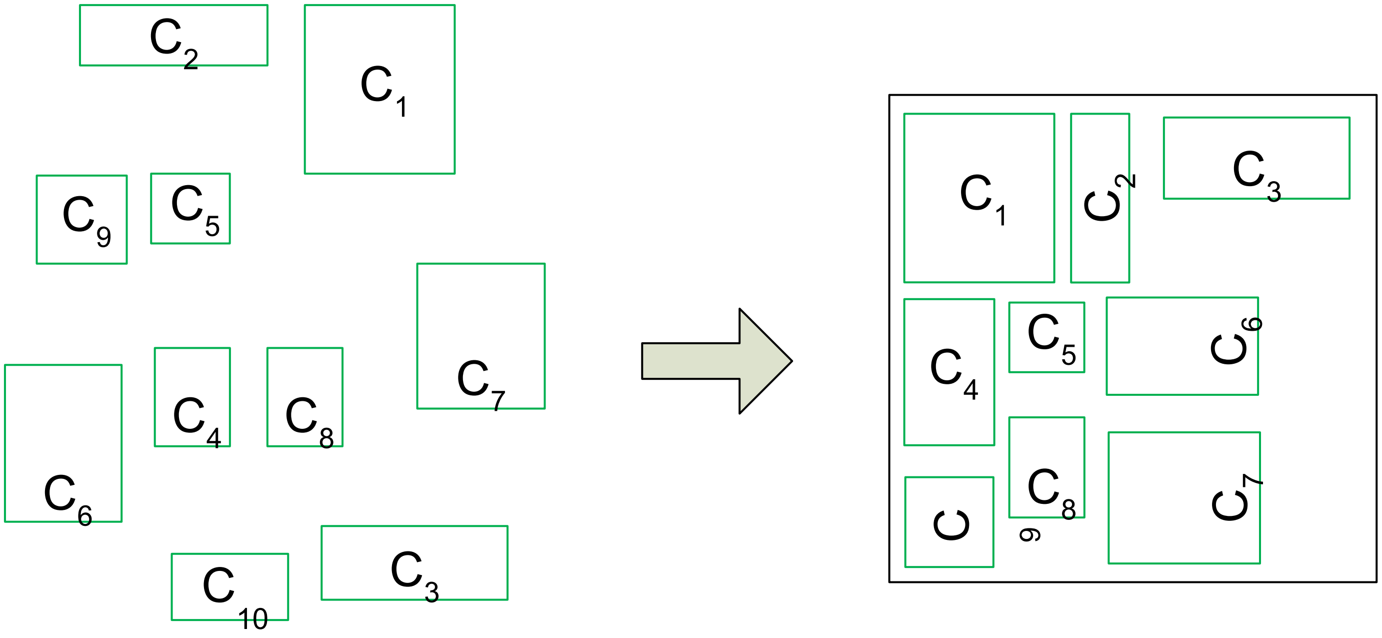
(Post-order) $(T_L\ T_R\ R)$

A binary tree

Three compact representation

Floor Planning : An example of tree encoding

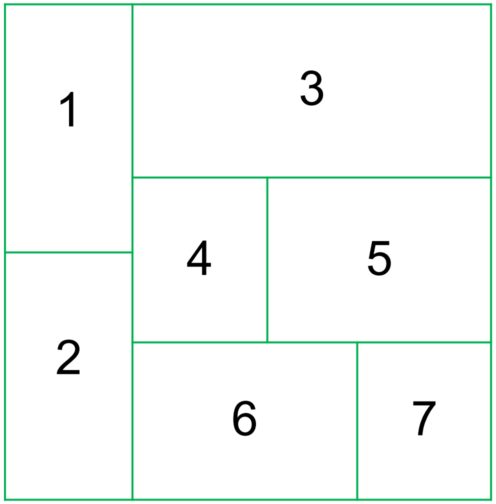
Floor planning is a standard problem in VLSI design. Here, given *n* circuits of different area requirements, we are to arrange them into a floor of chip layout, so that all circuits are placed in a minimum layout possible.



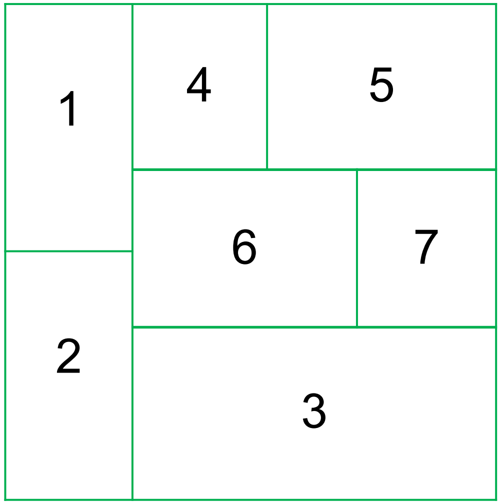
10 circuits

A floor plan

Tree encoding for Floor planning problem



Floor Plan I



Floor Plan II

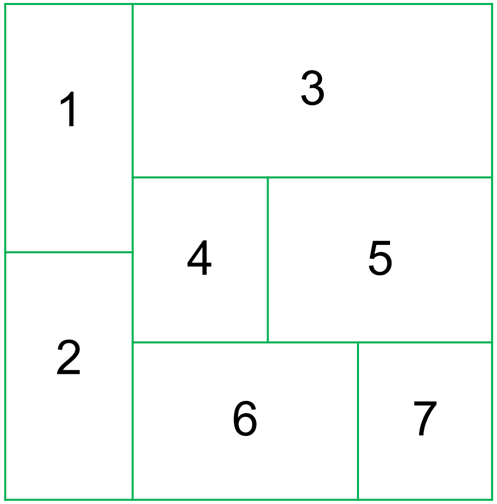
1. How many floor plans are possible?
2. Can we find a binary tree representation of a floor plan??

Binary tree representation for floor planning

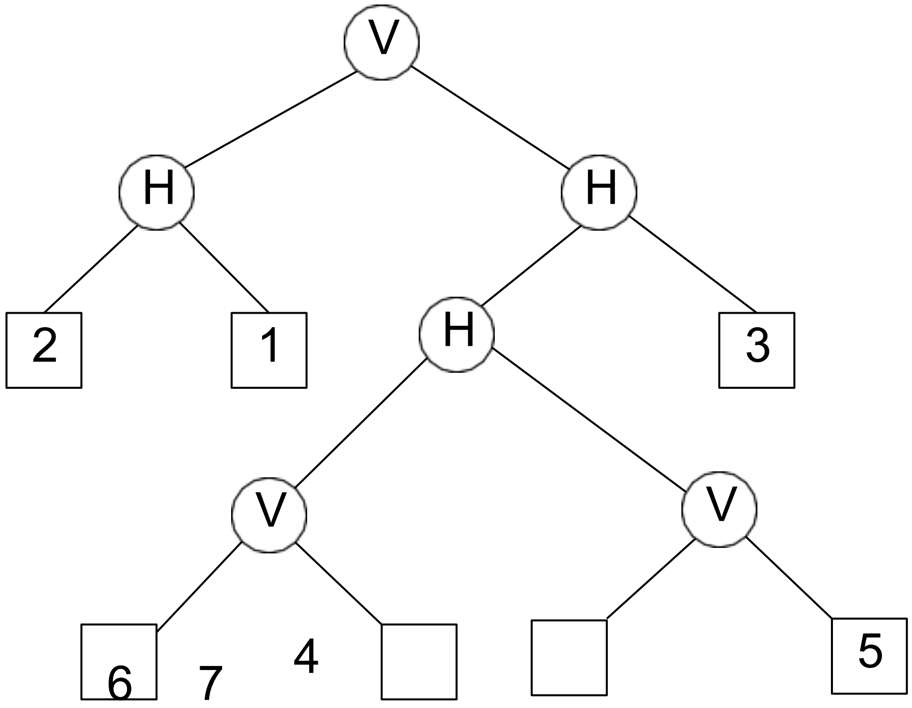
A floor plan can be modeled by a binary tree with n leaves and $n-1$ nodes where

- each node represents a vertical cut-line or horizontal cut-line, and Letter V and H refer to vertical and horizontal cut-operators.
- each leaf node represents a rectangle blocks.

Example : Floor plane I



Floor Plan
I



Binary tree representation of the floor plan I

Example : Floor plane I

Note 1:

The operators H and V expressed in polish notation carry the following meanings:

$ijH \rightarrow$ Block b_j is on top of the block b_i .

$ijV \rightarrow$ Block b_i is on the left of block b_j .

Note 2: A tree can be represented in a compact form using Polish notation

Note 3: Polish notation

$a + b \div c = a + (b \div c) = abc \div +$

$a + b - c = ab + c-$

Using GA
(A. Initial Population,
B. Fitness Value
C. Selection
D. Crossover
E. Mutation)
(Say, for TSP)

A. Initial Population

- Initial population of chromosomes is created randomly by using unique random number generator
- The initial population created is shown below, for a 15 city travelling salesman problem (TSP)
- The population consist of **ten chromosomes**, where each chromosome denotes the sequence in which cities have to be traversed and each gene represent the number assigned to a city.

- Chromosome1   1 4 13 3 8 2 5 15 7 10 14 12 6 9 11
- Chromosome 2   3 14 13 2 9 10 5 7 1 15 6 12 8 11 4
- Chromosome 3   1 15 3 7 14 11 9 2 13 5 12 4 8 10 6
- Chromosome 4   4 12 14 13 5 9 11 8 1 3 10 2 6 7 15
- Chromosome 5   11 2 9 5 13 14 3 12 8 1 15 6 4 10 7
- Chromosome 6   3 10 7 13 11 2 9 4 15 12 6 5 14 1 8
  so on……………………….
- Chromosome 10   3 7 10 13 11 2 4 9 15 12 6 5 14 1 8

B. Fitness Value

- The Purpose of the fitness function is to decide if a chromosome is good then **how good it is**?
- In the travelling salesman problem (TSP), the criteria for good chromosome is its length.
- Calculation takes place during the creation of the chromosomes as given in equation .
- Each chromosome is created and then its fitness function is calculated.
- $$fitness_{chrosome} = \sum_{i=1}^{towncount} t_i$$

towncount = total number of cities

ti = distance

C. Selection

Selection is used to also select the chromosome whose fitness value is small

Selection Techniques

GA Selection

- After deciding an encoding scheme, the second important things is how to perform selection from a set of population, that is, how to choose the individuals in the population that will create offspring for the next generation and how many offspring each will create.
-

  The purpose of selection is, of course, to emphasize **fittest** individuals in the population in hopes that their offspring will in turn have even higher fitness.

Selection operation in GAs

Selection is the process for creating the population for next generation from the current generation

To generate new population: Breeding in GA

- Create a mating
- pool Select a pair
- Reproduce

Fitness evaluation

- **In GA, there is a need to create next generation**
  - The next generation should be such that it is toward the (global) optimum solution
  - Random population generation may not be a wiser
  - strategy Better strategy follows the biological process: **Selection**

- **Selection involves:**
  - Survival of the fittest
  - Struggle for the existence

- **Fitness evaluation is to evaluate the survivability of each individual in the current population**

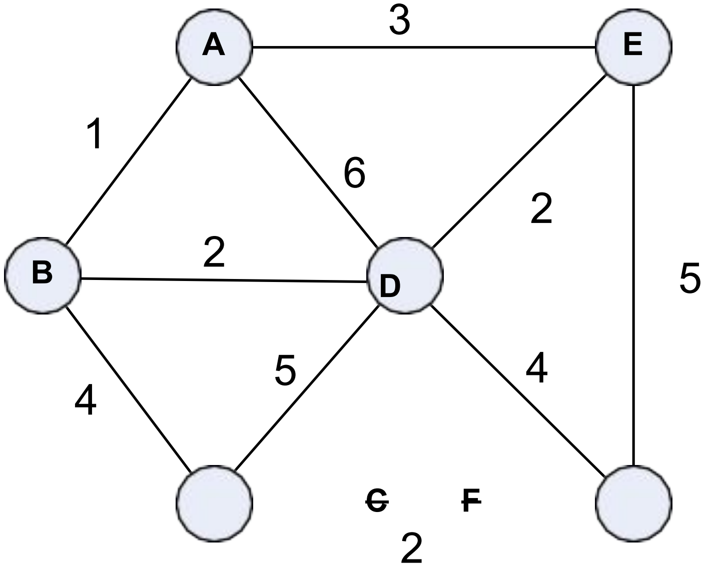
Fitness evaluation

How to evaluate the fitness of an individual?

- A simplest strategy could be to take the confidence of the value(s) of the objective function(s)
  - Simple, if there is a single objective function
  - But, needs a different treatment if there are two or more objective functions
    - They may be in different scales
    - All of them may not be same significant level in the fitness calculation

. . . etc.

For a 6 city TSP, and 5 chromosomes



P1:	C B A D F E	11
P2:	A B D C E F	19
P3:	A C B F E D	16
P4:	F C D B E A	12
P5:	C F D A B E	10

Selection Schemes in GAs

Different strategies are known for the selection:

- **Canonical selection** (also called proportionate-based selection)
- **Roulette Wheel selection** (also called proportionate-based selection)
- **Rank-based selection** (also called as ordinal-based selection)
- **Tournament selection**
- **Steady-state**
- **selection** Boltzman

selection

Canonical selection

- In this techniques, fitness is defined for the $i - th$ individual as follows.

$$fitness(i) = \frac{f_i}{E}$$

where f_i is the evaluation associated with the $i - th$ individual in the population.

- $\bar{F}$ is the average evaluation of all individuals in the population size N and is defined as follows. $\bar{F} = \frac{\sum_{i=1}^{N} f_i}{N}$

Canonical selection

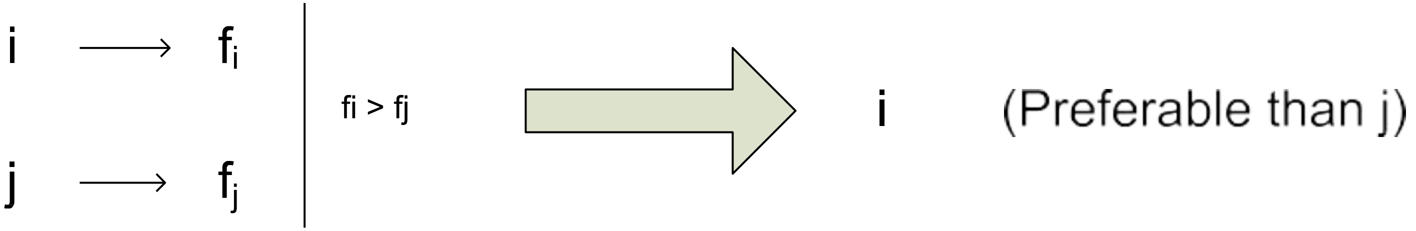
- In an iteration, we calculate $\frac{f_i}{E}$ for all individuals in the current population.
- In Canonical selection, the probability that individuals in the current population are copied and placed in the mating pool is proportional to their fitness.

Note :

- Here, the size of the mating pool is $p\% \times N$, for some
- p. Convergence rate depends on p.

Roulette-Wheel selection

- In this scheme, the probability for an individual being selected in the mating pool is considered to be proportional to its fitness.
- It is implemented with the help of a wheel as shown.

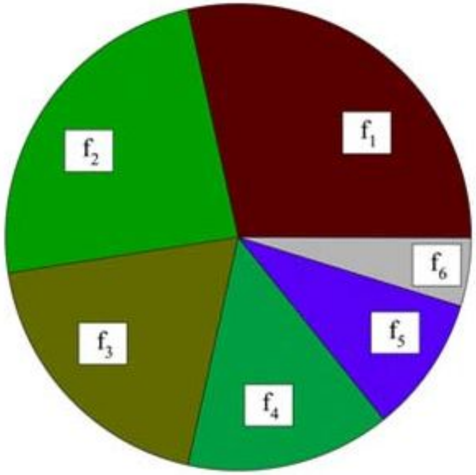


$i \longrightarrow f_i$

$j \longrightarrow f_j$

fi > fj

i (Preferable than j)



Wheel Game
(Rotate and See
the Pointer)

Roulette-Wheel selection mechanism

- The top surface area of the wheel is divided into *N* parts in proportion to the fitness values $f_1, f_2, f_3 \cdot \cdot \cdot f_N$.
- The wheel is rotated in a particular direction (either clockwise or anticlockwise) and a fixed pointer is used to indicate the winning area, when it stops rotation.
- A particular sub-area representing a GA-Solution is selected to be winner probabilistically and the probability that the $i-th$ area will be declared as

$$p_i = \frac{f_i}{\Sigma_{i=1}^{N} f_i}$$

- In other words, the individual having higher fitness value is likely to be selected more.

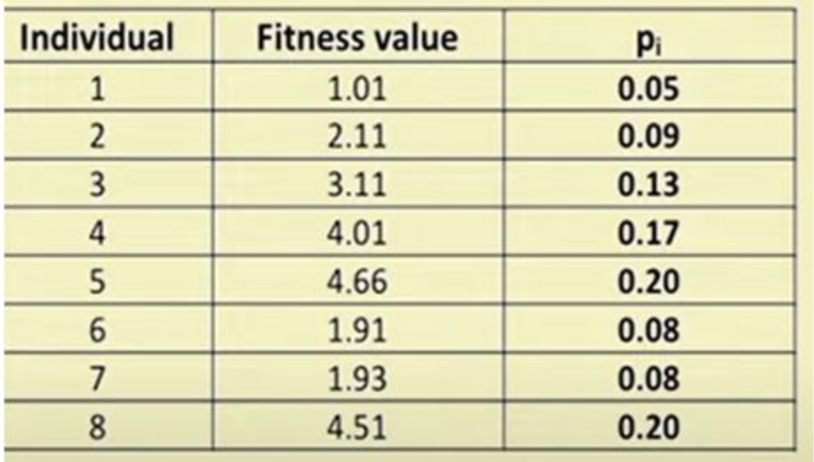
Roulette-Wheel selection mechanism

The wheel is rotated for N_p times (where $N_p = p\%N$, for some p) and each time, only one area is identified by the pointer to be the winner.

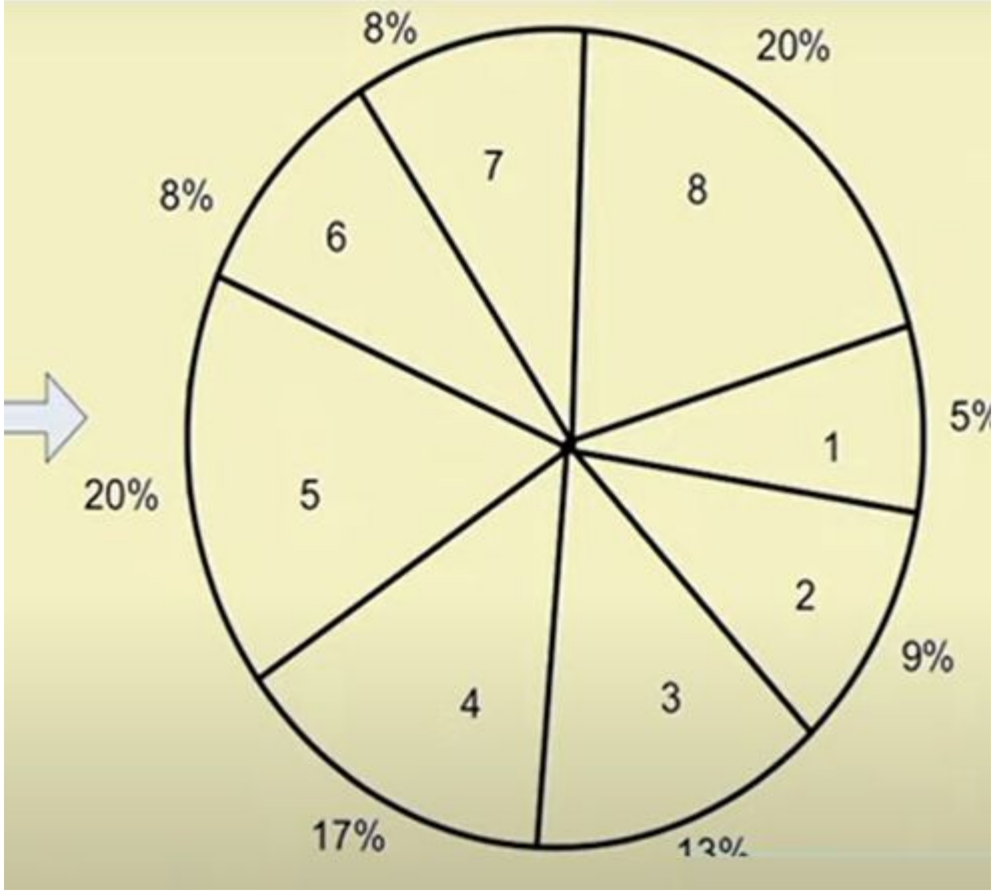
Note :

- Here, an individual may be selected more than once.
- Convergence rate is fast.

Roulette-Wheel selection mechanism: Example



Individual	Fitness value	p_i
1	1.01	**0.05**
2	2.11	**0.09**
3	3.11	**0.13**
4	4.01	**0.17**
5	4.66	**0.20**
6	1.91	**0.08**
7	1.93	**0.08**
8	4.51	**0.20**



Roulette-Wheel selection : Implementation

Input: A Population of size N with their fitness values
Output: A mating pool of size N_p

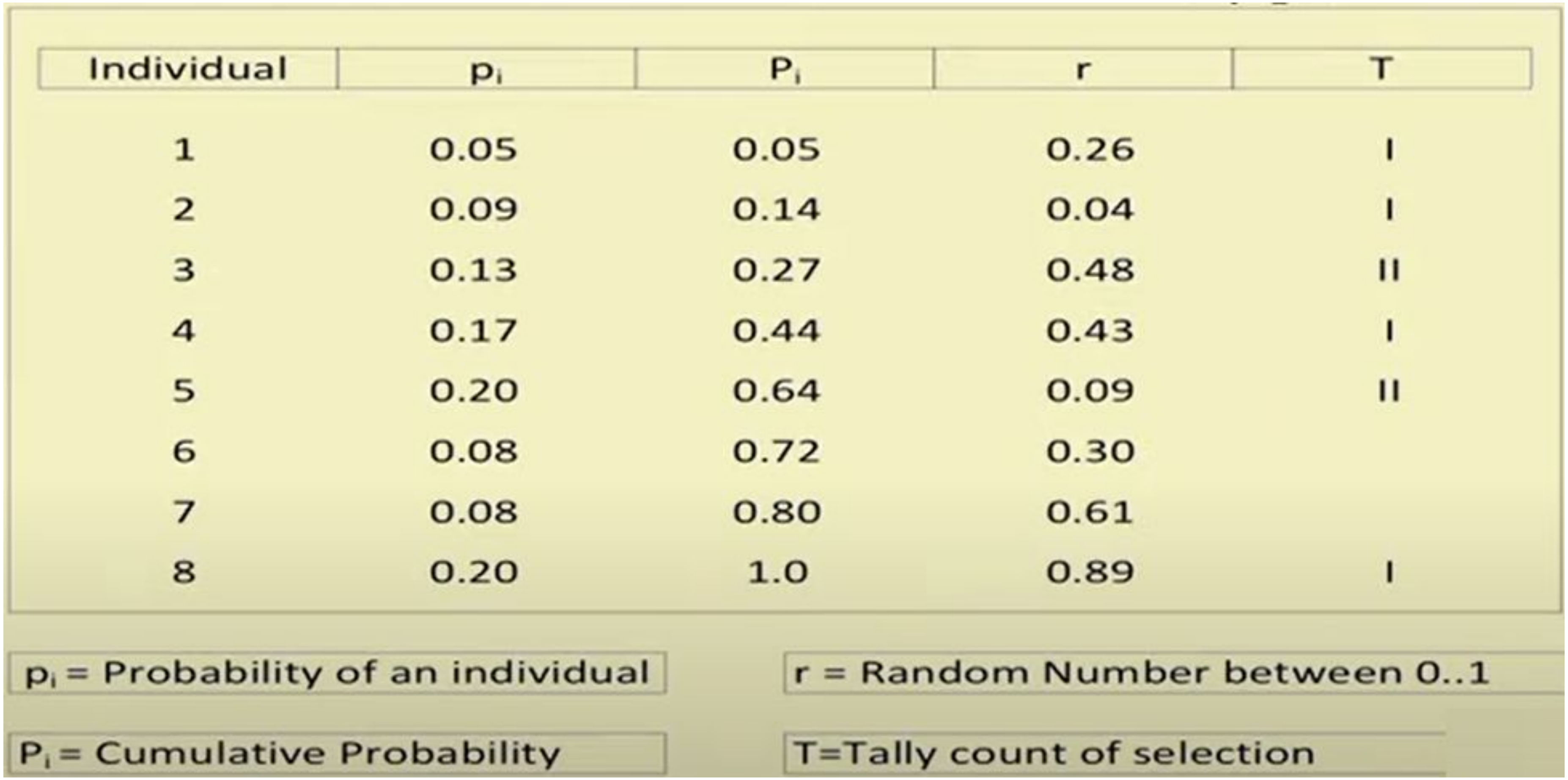
Steps:

1. Compute $p_i = \frac{f_i}{\sum_{i=1}^{N} f_i}, \forall i = 1, 2 \cdots N$
2. Calculate the cumulative probability for each of the individual starting from the top of the list, that is $P_i = \sum_{j=1}^{i} p_j$, for all $j = 1, 2 \cdots N$
3. Generate a random number say r between 0 and 1.
4. Select the j-th individual such that $P_{j-1} < r \leq P_j$
5. Repeat Step 3-4 to select N_p individuals.
6. End

Roulette-Wheel selection: Example

The probability that i-th individual will be pointed is

$$p_i = \frac{f_i}{\sum_{i=1}^{N} f_i}$$



Individual	p_i	P_i	r	T
1	0.05	0.05	0.26	I
2	0.09	0.14	0.04	I
3	0.13	0.27	0.48	II
4	0.17	0.44	0.43	I
5	0.20	0.64	0.09	II
6	0.08	0.72	0.30	
7	0.08	0.80	0.61	
8	0.20	1.0	0.89	I

p_i = Probability of an individual

r = Random Number between 0..1

P_i = Cumulative Probability

T=Tally count of selection

T= Total tally of individuals selected

Drawback in Roulette-Wheel selection

- Suppose, there are only four binary string in a population, whose fitness values are f_1, f_2, f_3 and f_4.
- Their values 80%, 10%, 6% and 4%, respectively.

What is the expected count of selecting f_3, f_4, f_2 or f_1?

Rank-based selection

- To overcome the problem with Roulette-Wheel selection, a rank-based selection scheme has been proposed.

- The process of ranking selection consists of two steps.

  1. Individuals are arranged in an ascending order of their fitness values. The individual, which has the lowest value of fitness is assigned rank 1, and other individuals are ranked accordingly.
  2. The proportionate based selection scheme is then followed based on the assigned rank.

Note:

- The % area to be occupied by a particular individual *i*, is given by

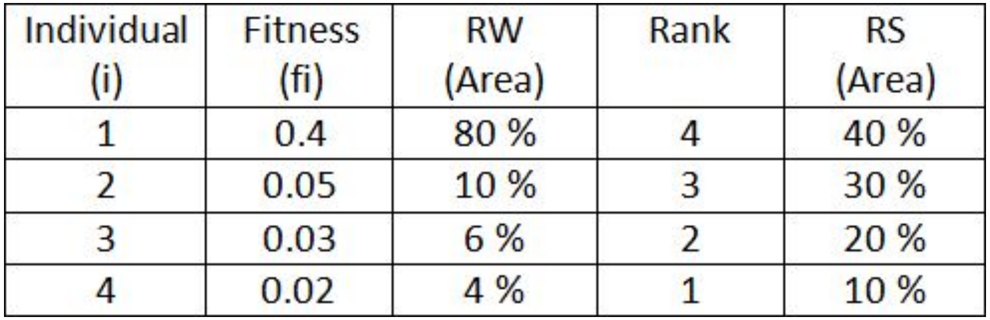
$$\frac{r_i}{\sum_{i=1}^{N} r_i} \times 100$$

where r_i indicates the rank of $i-th$ individual.

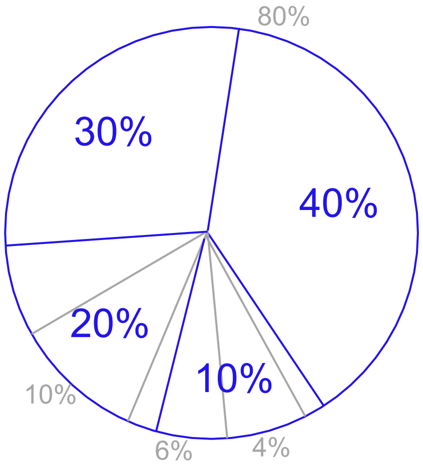
- Two or more individuals with the same fitness values should have

Rank-based selection: Example

- Continuing with the population of 4 individuals with fitness values:
  $f_1 = 0.40$, $f_2 = 0.05$, $f_3 = 0.03$ and $f_4 = 0.02$.
- Their proportionate area on the wheel are: 80%, 10%, 6% and 4% Their ranks are shown in the following figure.



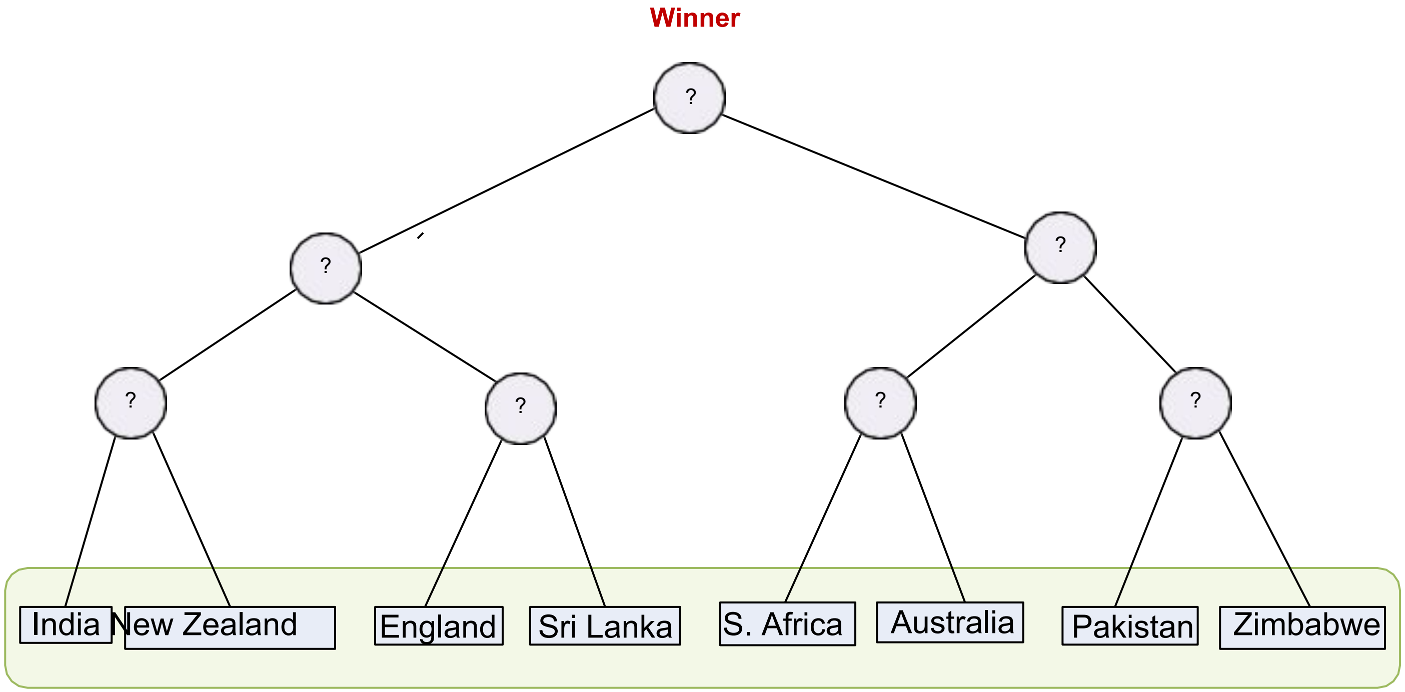
Individual (i)	Fitness (fi)	RW (Area)	Rank	RS (Area)
1	0.4	80 %	4	40 %
2	0.05	10 %	3	30 %
3	0.03	6 %	2	20 %
4	0.02	4 %	1	10 %



It is evident that expectation counts have been improved compared to Routlette-Wheel selection.

Basic concept of tournament selection

Who will win the match in this tournament?



Tournament selection

1. In this scheme, we select the tournament size *n* (say 2 or 3) at random.
2. We pick *n* individuals from the population, at random and determine the best one in terms of their fitness values.
3. The best individual is copied into the mating pool.
4. Thus, in this scheme only one individual is selected per tournament and N_p tournaments are to be played to make the size of mating pool equals to N_p .

Note :

- Here, there is a chance for a good individual to be copied into the mating pool more than once.
- This techniques founds to be computationally more faster than both Roulette-Wheel and Rank-based selection scheme.

Tournament selection : Example

$$N = 8, N_U = 2, N_p = 8$$

Input :

Individual	1	2	3	4	5	6	7	8
Fintess	1.0	2.1	3.1	4.0	4.6	1.9	1.8	4.5

Output :

Trial	Individuals	Selected
1	2, 4	4
2	3, 8	8
3	1, 3	3
4	4, 5	5
5	1, 6	6
6	1, 2	2
7	4, 2	4
8	8, 3	8

If the fitness values of two individuals are same, than there is a tie in the match!! So, what to do????

Mating Pool Prior to Crossover

Mating Pool: Prior to crossover operation

- A mating pair (each pair consists of two strings) are selected at random. Thus, if the size of mating pool is N, then $\frac{N}{2}$ mating pairs are formed.[Random Mating]

- The pairs are checked, whether they will participate in reproduction or not by tossing a coin, whose probability being p_c

  If p_c is head, then the parent will participate in reproduction. Otherwise, they will remain intact in the population.

Note :

Generally, $p_c = 1.0$, so that almost all the parents can participate in production.

D. Crossover

Crossover operation

Once, a pool of mating pair are selected, they undergo through crossover operations.

1. In crossover, there is an exchange of properties between two parents and as a result of which **two** offspring solutions are produced.
2. The crossover point(s) (also called k-point(s)) **is(are)** decided using a random number generator generating integer(s) in between 1 and *L*, where *L* is the length of the chromosome.
3. Then we perform exchange of gene values with respect to the k-point(s)

There are many exchange mechanisms and hence crossover strategies.

Crossover Techniques in Binary Coded GA

Crossover operations in Binary-coded GAs

- There exists a large number of crossover schemes, few important of them are listed in the following.

1. Single point
2. crossover Two-point
3. crossover
4. Multi-point crossover (also called n-point
5. crossover) Uniform crossover (UX)
6. Half-uniform crossover
7. (HUX) Shuffle crossover
8. Matrix crossover (Tow-dimensional

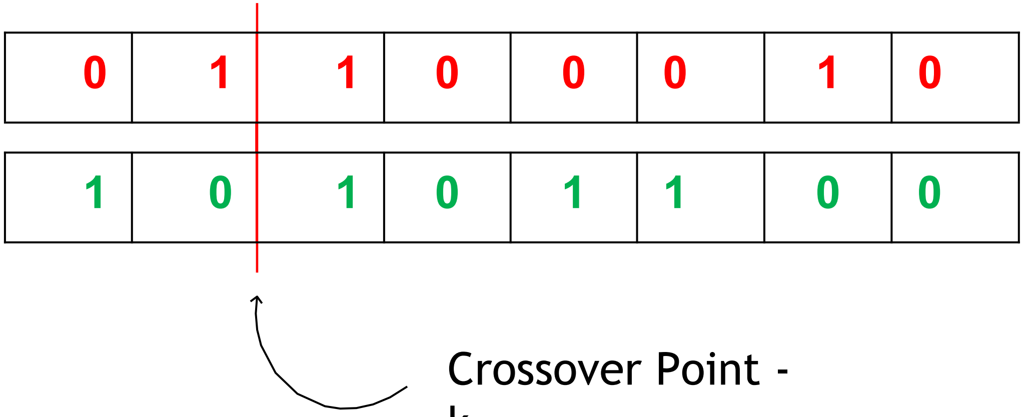
crossover) Three parent crossover

Single point crossover

1. Here, we select the K-point lying between 1 and *L*. Let it be *k* .
2. A single crossover point at *k* on both parent’s strings is selected.
3. All data beyond that point in either string is swapped between the two parents.
4. The resulting strings are the chromosomes of the offsprings produced.

Single point crossover: Illustration

Before Crossover



Parent 1 :	0	1	1	0	0	0	1	0
Parent 2 :	1	0	1	0	1	1	0	0

Two diploid from a mating pair

Crossover Point - k

Select crossover points randomly

Offspring 1:	0	1	1	0	1	1	0	0
Offspring 2:	1	0	1	0	0	0	1	0

Two diploid for two new offspring is produced

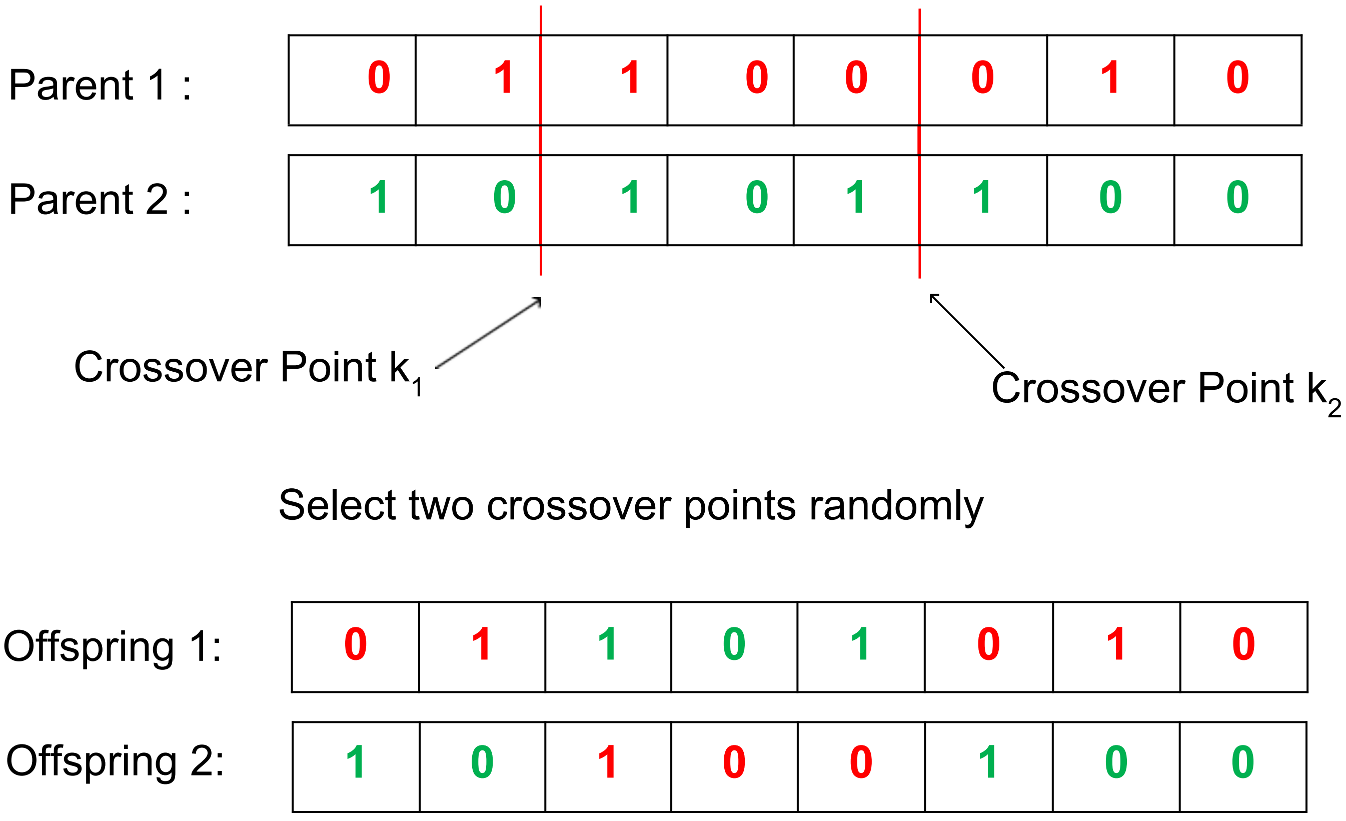
After Crossver

Two-point crossover

1. In this scheme, we select two different crossover points k_1 and k_2 lying between 1 and L at random such that k_1 ƒ= k_2.
2. The middle parts are swapped between the two strings.
3. Alternatively, left and right parts also can be swapped.

Two-point crossover: Illustration

Before Crossover



Parent 1 :	0	1	1	0	0	0	1	0
Parent 2 :	1	0	1	0	1	1	0	0

Crossover Point k_1

Crossover Point k_2

Select two crossover points randomly

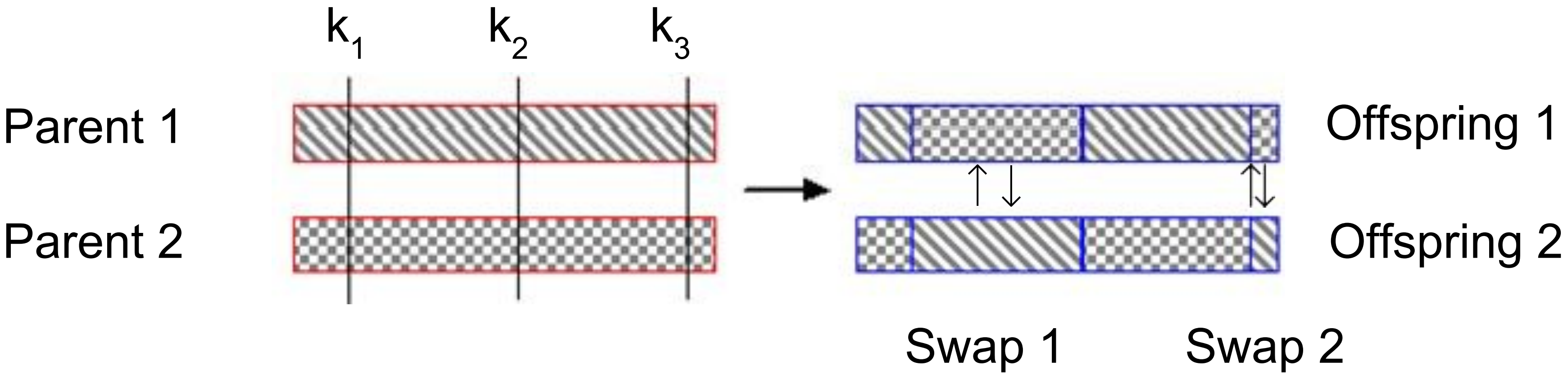
Offspring 1:	0	1	1	0	1	0	1	0
Offspring 2:	1	0	1	0	0	1	0	0

After Crossver

Multi-point crossover

1. In case of multi-point crossover, a number of crossover points are selected along the length of the string, at random.

2. The bits lying between alternate pairs of sites are then swapped.



Uniform Crossover (UX)

- Uniform crossover is a more general version of the multi-point crossover.
- In this scheme, at each bit position of the parent string, we toss a coin (with a certain probability p_s) to determine whether there will be swap of the bits or not.
- The two bits are then swapped or remain unaltered, accordingly.

Uniform crossover (UX): Illustration

Before crossover

Parent 1 :	1	1	0	0	0	1	0	1	1	0	0	1
Parent 2 :	0	1	1	0	0	1	1	1	0	1	0	1
Coin tossing:	1	0	0	1	1	1	0	1	1	0	0	1

After crossover

Offspring 1:	1	1	1	0	0	1	1	1	1	1	0	1
Offspring 2:	0	1	0	0	0	1	0	1	0	0	0	1

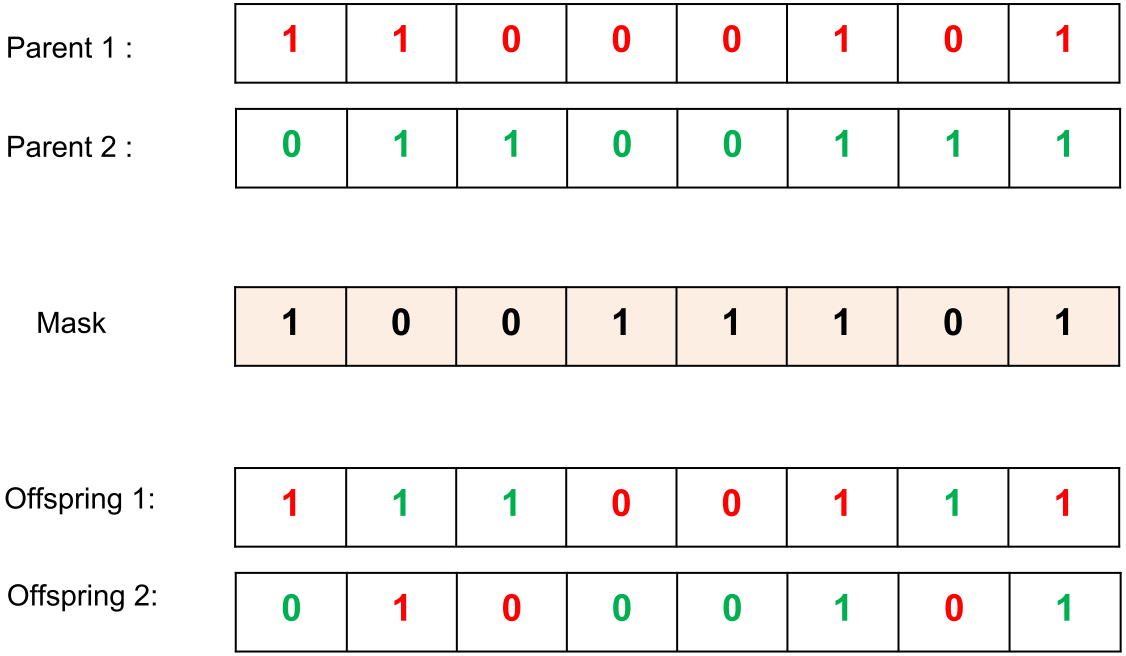
Rule: If the toss is 0 than swap the bits between P1 and P2

Uniform crossover with crossover mask

- Here, each gene is created in the offspring by copying the corresponding gene from one or the other parent chosen according to a random generated binary crossover mask of the same length as the chromosome.
- Where there is a 1 in the mask, the gene is copied from the first parent
- Where there is a 0 in the mask, the gene is copied from the second parent.
- The reverse is followed to create another offsprings.

Uniform crossover with crossover mask: Illustration

Before Crossover



Parent 1 :	1	1	0	0	0	1	0	1
Parent 2 :	0	1	1	0	0	1	1	1
Mask	1	0	0	1	1	1	0	1
Offspring 1:	1	1	1	0	0	1	1	1
Offspring 2:	0	1	0	0	0	1	0	1

When there is a 1 in the mask, the gene is copied from Parent 1 else from Parent 2.

When there is a 1 in the mask, the gene is copied from Parent 2 else from Parent 1.

After Crossver

Crossover Techniques in Order GAs

Crossover techniques in order GA

Some important crossover techniques in Order-coded GAs are:

1. Single-point order
2. crossover Two-point order
3. crossover
4. Partially mapped crossover
5. (PMX) Position based crossover
6. Precedence-preservation crossover

(PPX) Edge recombination crossover

Assumptions: For all crossover techniques, we assume the following:

- Let L be the length of the chromosome.
- P_1 and P_2 are two parents (are selected from the mating pool).
- C_1 and C_2 denote offspring (initially empty).

Single point order crossover

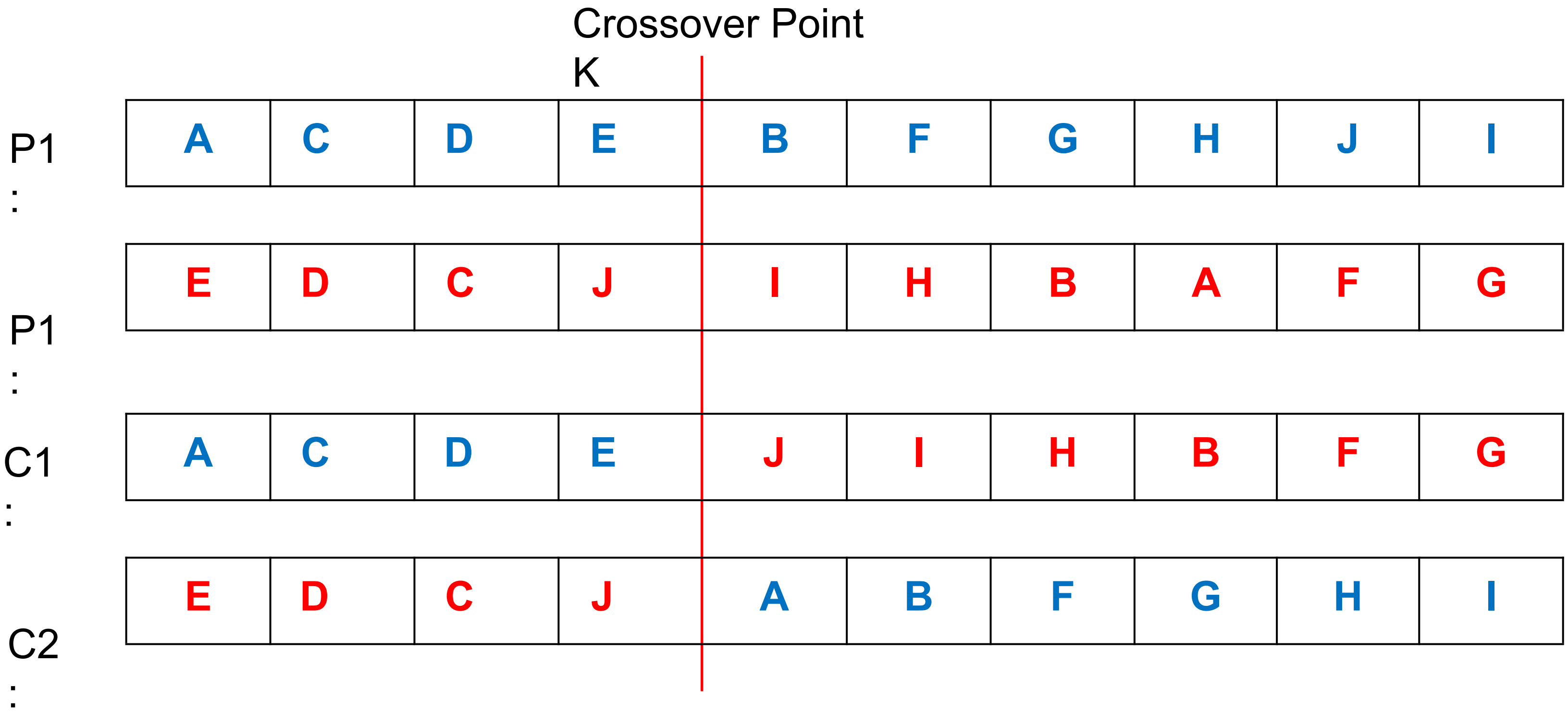
Given two parents P_1 and P_2 with chromosome length, say L.

Steps:

1. Randomly generate a crossover point K such that $(1 < K < L)$.
2. Copy the left schema of P_1 into C_1 (initially empty) and left schema of P_2 into C_2 (also initially empty).
3. For the schema in the right side of C_1, copy the gene value from P_2 in the same order as they appear but not already present in the left schema.
4. Repeat the same procedure to complete C_2 from P_1.

Single point order crossover: Illustration

Example :



Crossover Point K

P1 :	A	C	D	E	B	F	G	H	J	I
P1 :	E	D	C	J	I	H	B	A	F	G
C1 :	A	C	D	E	J	I	H	B	F	G
C2 :	E	D	C	J	A	B	F	G	H	I

Two-point order crossover

It is similar to the single-point order crossover, but with two $k-$points.

Steps:

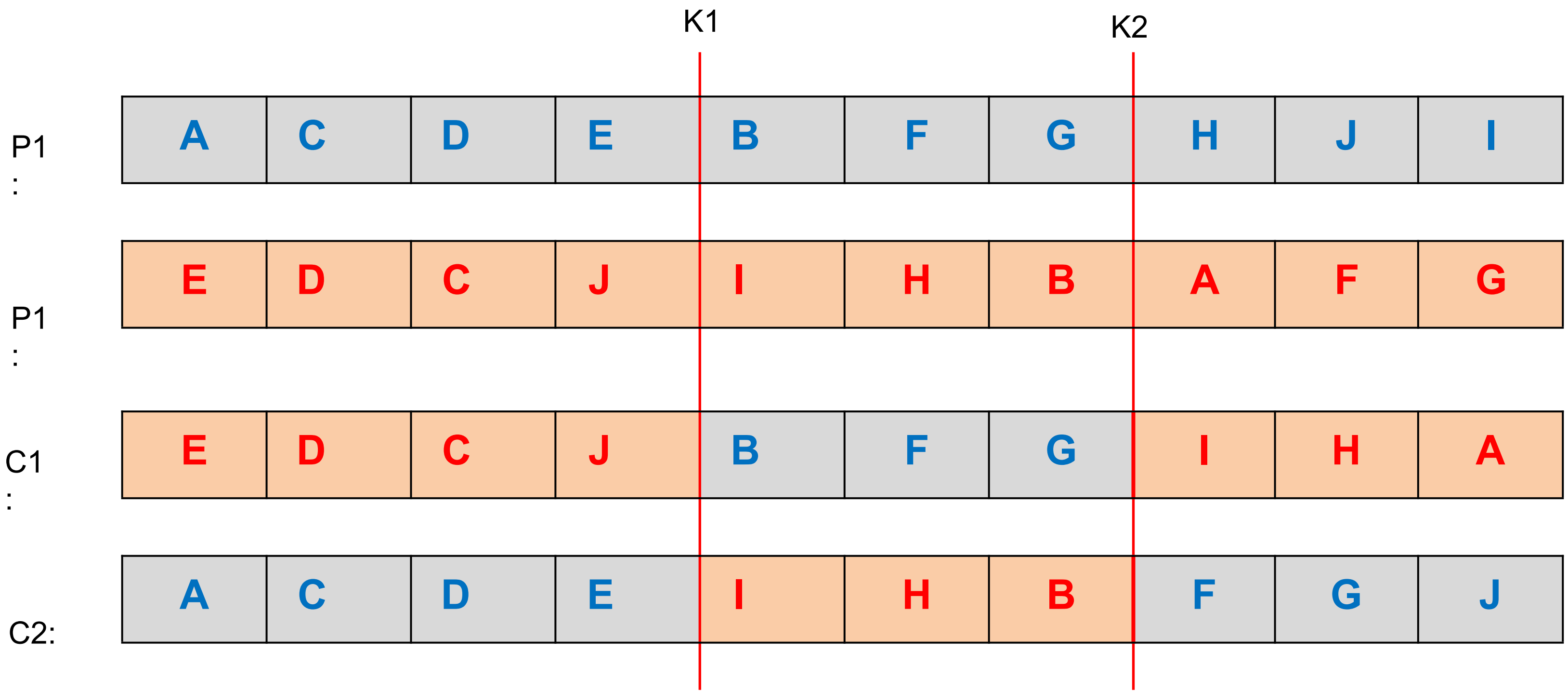
1. Randomly generate two crossover points K_1 and K_2. $1 < K_1, K_2 < L$

2. The schema in middle of P_1 and P_2 are copied into C_1 and C_2 (initially both are empty), respectively in the same location as well as in the same order.

3. The remaining schema in C_1 and C_2 are copied from P_2 and P_1 respectively, so that an element already selected in child solution does not appear again.

Two-point order crossover: Illustration

Example :



P1 :	A	C	D	E	B	F	G	H	J	I
P1 :	E	D	C	J	I	H	B	A	F	G
C1 :	E	D	C	J	B	F	G	I	H	A
C2:	A	C	D	E	I	H	B	F	G	J

Precedence preservation order crossover

Let the parent chromosomes be P_1 and P_2 and the length of chromosomes be L.

Steps:

(a) Create a vector V of length L randomly filled with elements from the set $\{1, 2\}$.

(b) This vector defines the order in which genes are successfully drawn from P_1 and P_2 as follows.

1. We scan the vector V from left to right.
2. Let the current position in the vector V be i (where $i = 1, 2, \cdots, L$).
3. Let j (where $j = 1, 2, \cdots, L$) and k (where $k = 1, 2, \cdots, L$) denotes the j^{th} and k^{th} gene of P_1 and P_2, respectively. Initially $j = k = 1$.

Precedence preservation order crossover

4. If i^{th} value is 1 then

   Delete j^{th} gene value from P_1 and as well as from P_2 and append it to the offspring (which is initially empty).

5. Else

   Delete k^{th} gene value from P_2 and as well as from P_1 and append it to the offspring.

6. Repeat Step 2 until both P_1 and P_2 are empty and the offspring contains all gene values.

Precedence preservation order crossover : Example

Example :

Random Vector σ	2	1	1	2	1	1	2	2	1	2
P1 :	A	C	D	E	B	F	G	H	J	I
P2 :	E	D	C	J	I	H	B	A	F	G
C1 :	E	C	D	J	B	F	H	A	I	G
C2:	?									

Note : We can create another offspring following the alternative rule for 1 and 2.

Position-based order crossover

Steps :

1. Choose n crossover points $K_1, K_2 \cdot\cdot\cdot K_n$ such that n L, the length of chromosome.
2. The gene values at $K_1^{th}, K_2^{th} \cdot\cdot\cdot K_n^{th}$ positions in P_1 are directly copied into offspring C_1 (Keeping their position information intact).
3. The remaining gene values in C_1 will be obtained from P_2 in the same order as they appear there except they are already not copied from P_1.
4. We can reverse the role of P_1 and P_2 to get another offspring C_2.

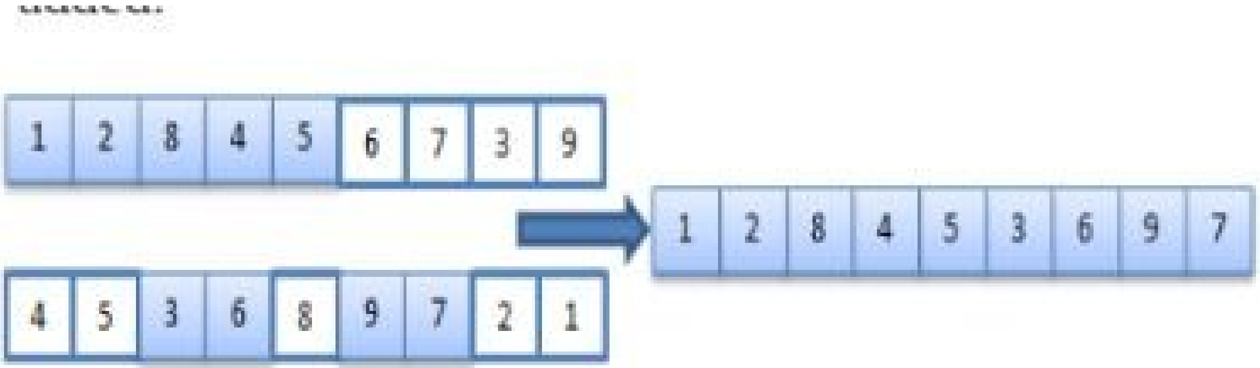
Position-based order crossover : Example

Let su consider three $k-$points namely K_1, K_2 and k_3 in this example.

			K1 ↓		K2 ↓			K3 ↓		
P1 :	A	C	D	E	B	F	G	H	J	I
P1 :	E	D	C	J	I	H	B	A	F	G
C1 :	E	C	D	J	B	I	A	H	F	G
C2:	D	E	C	B	I	F	G	A	H	J

D. Crossover

- 2-point crossover is applied to the pair of chromosomes so that new chromosomes will be generated which might have better fitness value.
- In 2-point crossover, randomly two positions in the chromosomes are chosen and then replace the gene with each other in both chromosomes.

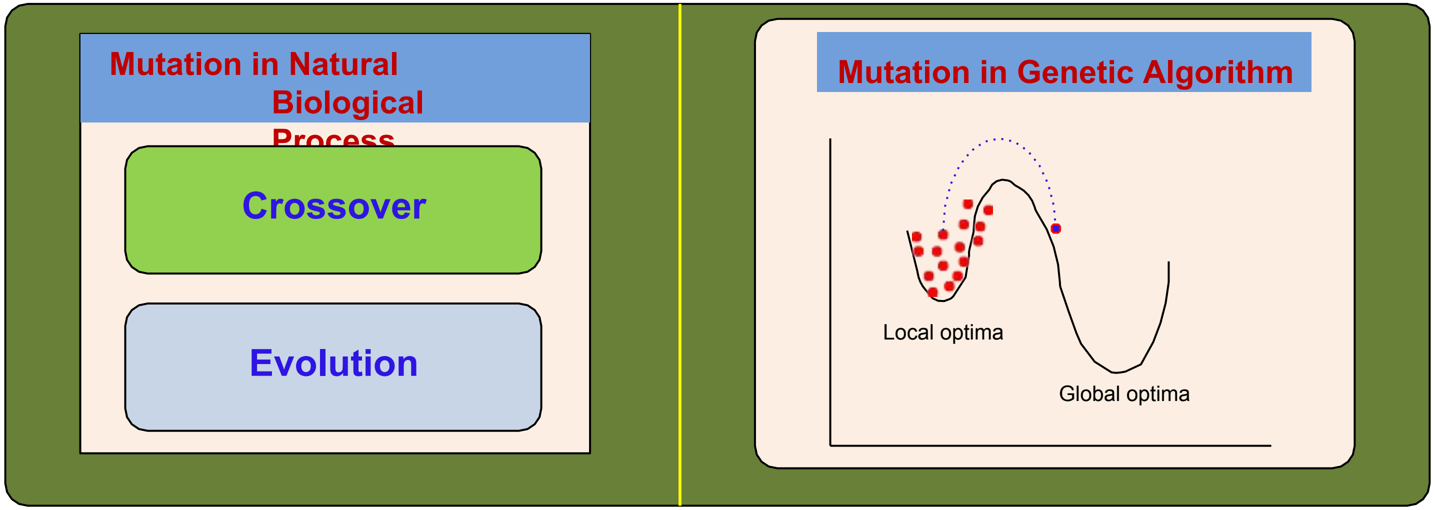


1. Encoding
2. Fitness Evaluation and Selection
3. Mating pool
4. **Reproduction**
   - Crossover
   - Mutation
   - Inversion
5. Convergence test

1. Encoding
2. Fitness evaluation and
3. Selection Mating pool
4. Crossover
5. **Mutation**
6. **Inversion**
7. **Convergence**
8. **test Fitness**

   scaling

- In genetic algorithm, the mutation is a genetic operator used to maintain genetic diversity from one generation of a population (of chromosomes) to the next.
- It is analogues to biological mutation.
- In GA, the concept of biological mutation is modeled artificially to bring a local change over the current solutions.



Mutation in Natural Biological Process

Crossover

Evolution

Mutation in Genetic Algorithm

Like different crossover techniques in different GAs there are many variations in mutation operations.

- **Binary Coded GA :**

  Flipping
  Interchangin
  g Reversing

- **Real Coded GA :**

  Random mutation
  Polynomial
  mutation

-

- **Order GA :**

  Tree-encoded GA :

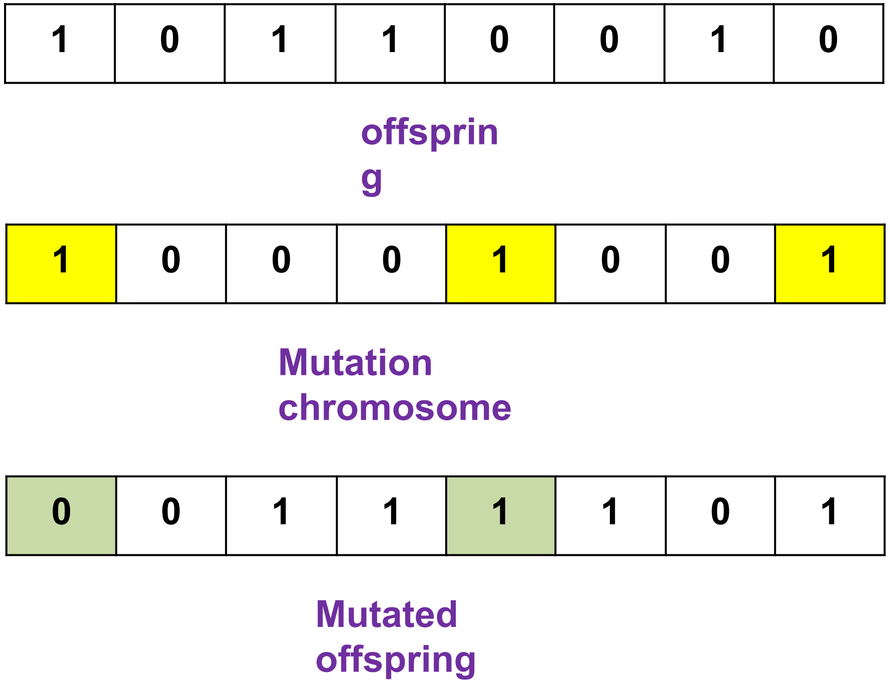
Mutation operation in Binary coded GA

- In binary-coded GA, the mutation operator is simple and straight forward.
- In this case, one (or a few) 1(s) is(are) to be converted to 0(s) and vice-versa.
- A common method of implementing the mutation operator involves generating a random variable calledmutation probability **(**μ_p**)** for each bit in a sequence.
- This mutation probability tells us whether or not a particular bit will be mutated (i.e. modified).

Note :

- To avoid large deflection, μ_p is generally kept to a low value.
- It is varied generally in the range of $\frac{0.1}{L}$ to $\frac{1.0}{L}$, where L is the string length.

- Here, amutation chromosomeof the same length as the individual's chromosome is created with a probability p_{μ} of $1^{j}s$ in the bit.
- For a 1 in mutation chromosome, the corresponding bit in the parent chromosome is flipped (0 to 1 or 1 to 0) and mutated chromosome is produced.



1	0	1	1	0	0	1	0

offsprin
g

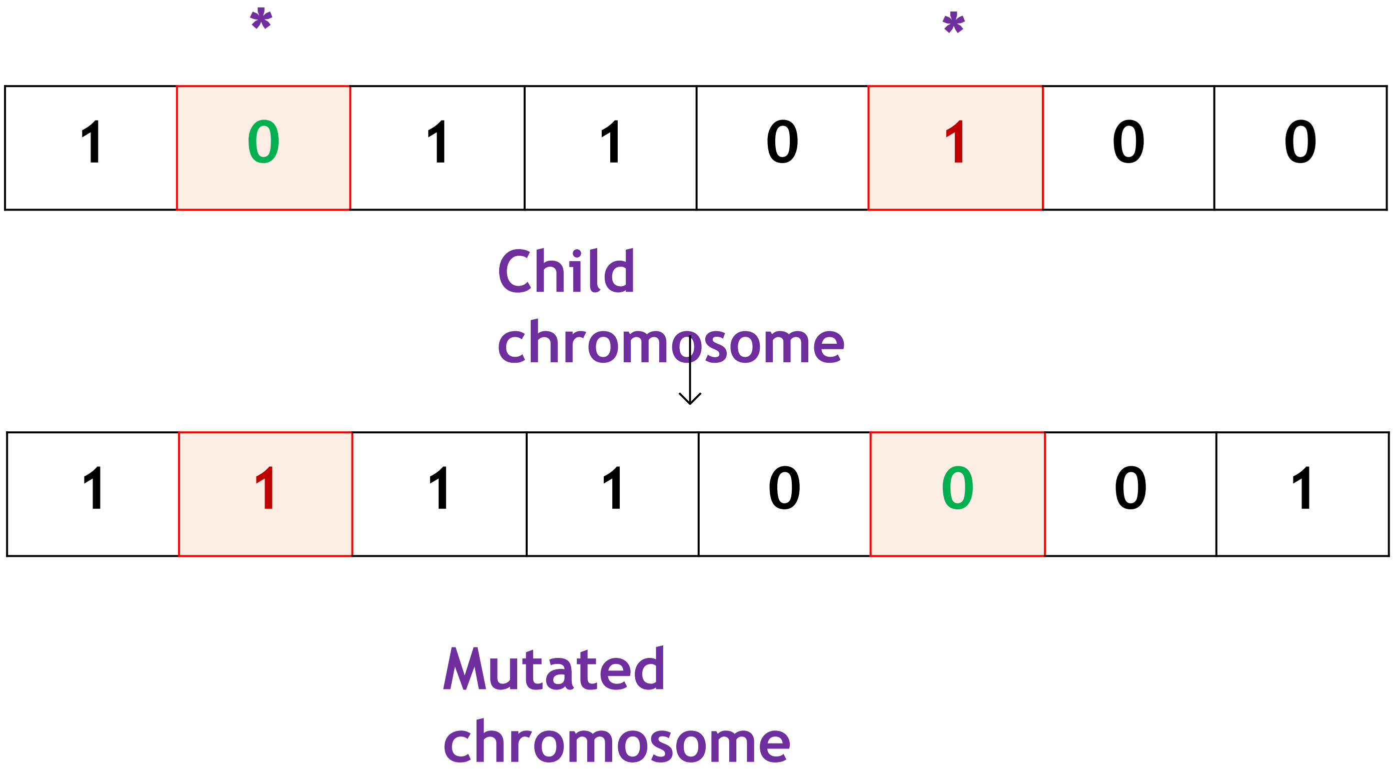
1	0	0	0	1	0	0	1

Mutation
chromosome

0	0	1	1	1	1	0	1

Mutated
offspring

- Two positions of a child’s chromosome are chosen randomly and the bits corresponding to those position are interchanged.



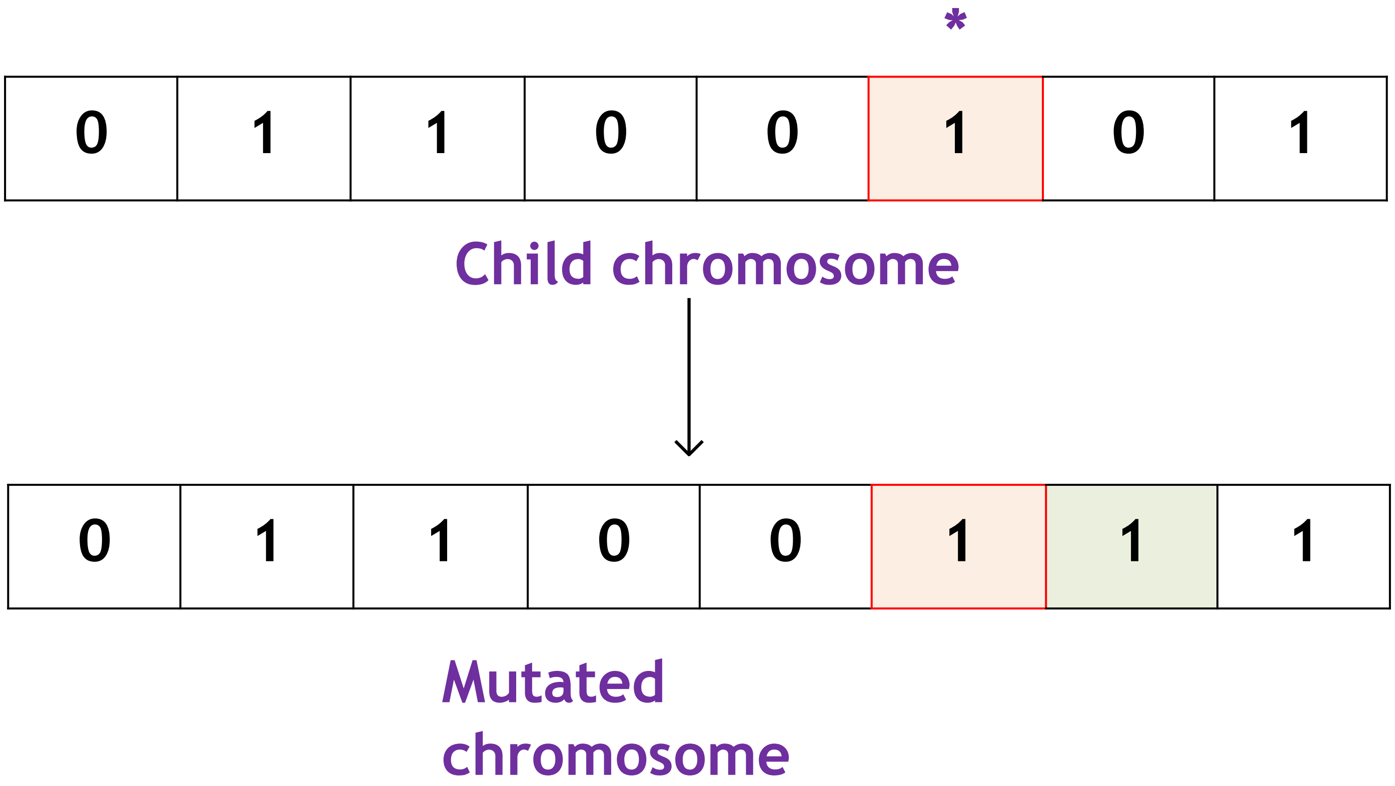
	*				*		
1	0	1	1	0	1	0	0

Child chromosome

1	1	1	1	0	0	0	1

Mutated chromosome

- A positions is chosen at random and the bit next to that position is reversed and mutated child is produced.



					*		
0	1	1	0	0	1	0	1

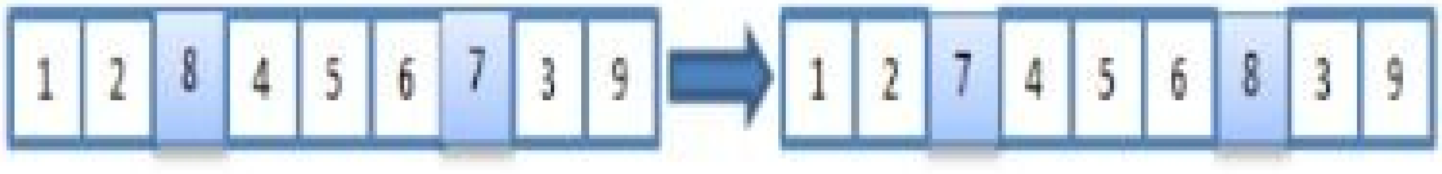
Child chromosome

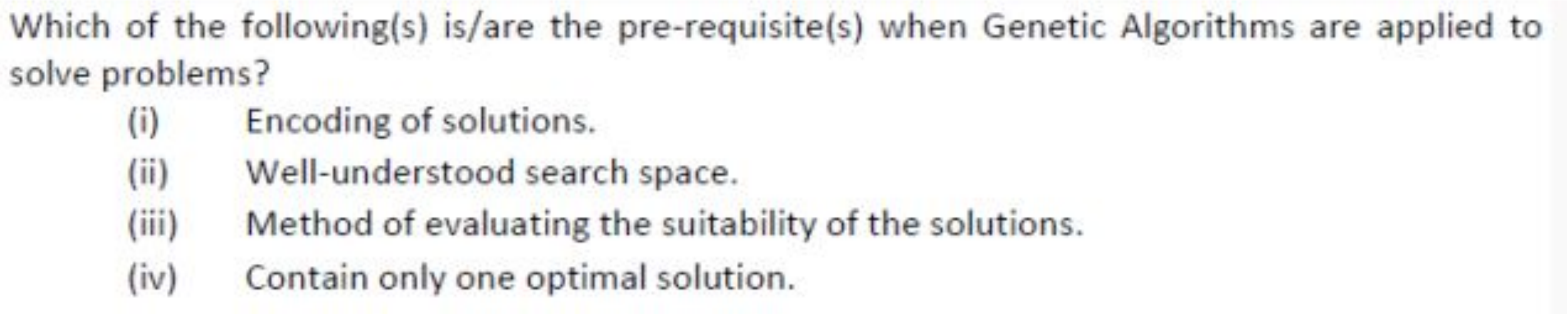
0	1	1	0	0	1	1	1

Mutated chromosome

E. Mutation

- Mutation is applied to form a new generation. We apply interchange mutation. In interchange mutation, randomly select two genes from a chromosome and then swap them.





Which of the following(s) is/are the pre-requisite(s) when Genetic Algorithms are applied to solve problems?

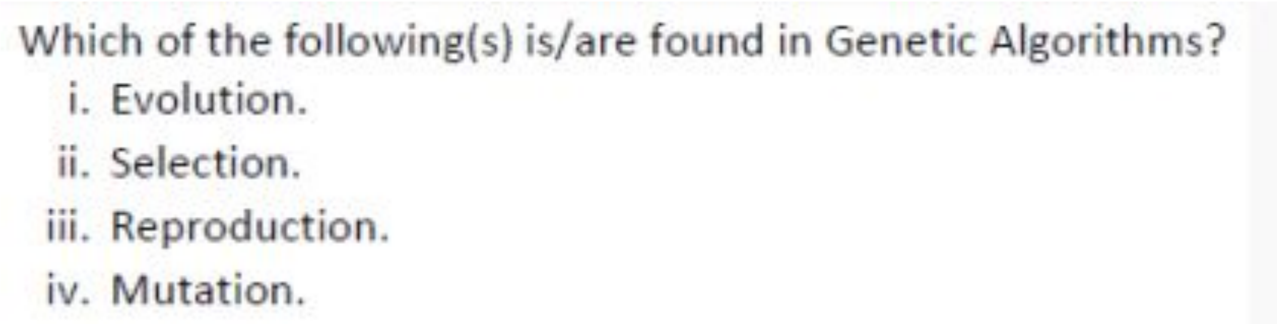
(i) Encoding of solutions.

(ii) Well-understood search space.

(iii) Method of evaluating the suitability of the solutions.

(iv) Contain only one optimal solution.

- (i) and (iii)



Which of the following(s) is/are found in Genetic Algorithms?

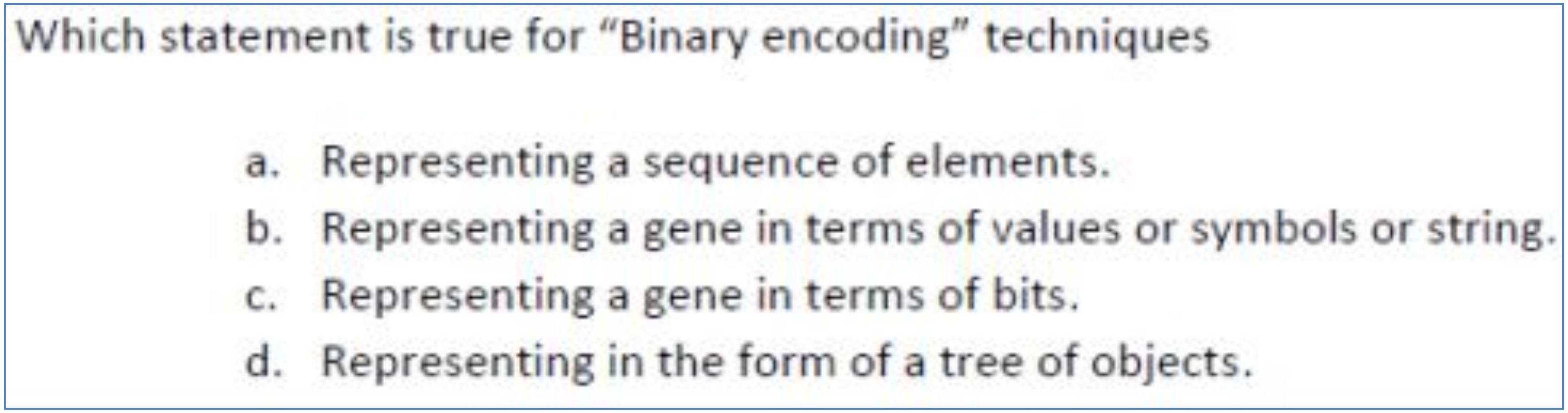
i. Evolution.

ii. Selection.

iii. Reproduction.

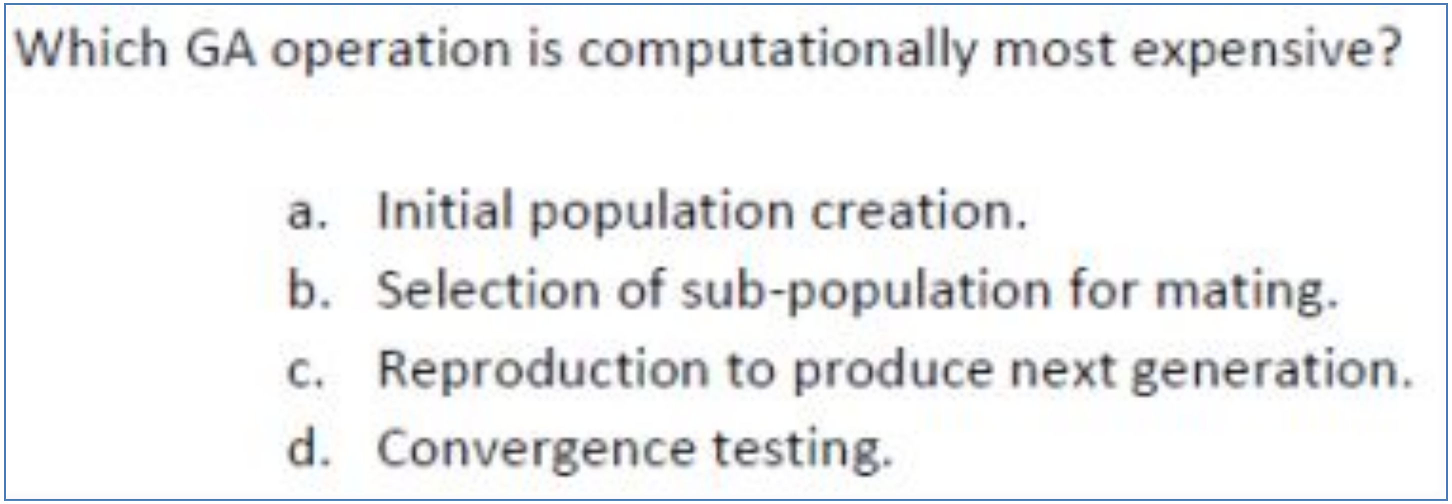
iv. Mutation.

- All



Which statement is true for “Binary encoding” techniques

a. Representing a sequence of elements.
b. Representing a gene in terms of values or symbols or string.
c. Representing a gene in terms of bits.
d. Representing in the form of a tree of objects.



Which GA operation is computationally most expensive?

a. Initial population creation.
b. Selection of sub-population for mating.
c. Reproduction to produce next generation.
d. Convergence testing.

Example

- Implement travelling sales person problem (TSP) using genetic algorithms.

Sample Steps for TSP using GA

- Encoding – Order Encoding
- Fitness Function - Path cost
- Selection - Roulette Wheel
- Crossover / Mutation – Single point
- New population Formation
- Convergence criteria – (Reasonable number of iterations / 2 MST)

Thank You