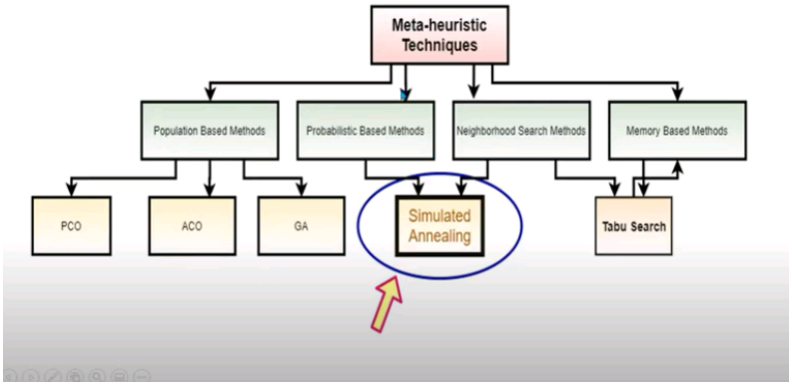
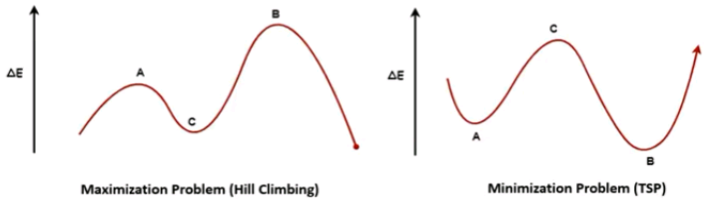


Simulated Annealing

- Its variation of **hill climbing** in which initially some **downhill moves** may be made. The idea is do **enough exploration** of the whole space early on so that final solution is insensitive to the starting state.



Maximization Problem (Hill Climbing)

Minimization Problem (TSP)

Simulated Annealing

- Simulated annealing is a computational process which is derived from **physical process of annealing**.
- In annealing **metals are melted** (raised to high energy levels) and then gradually cooled until some solid state is reached (minimal energy final state).
- Thus physical substances usually moves from high energy configuration to lower one. But there is some probability that a transition to higher energy state will occur. This probability is given by

$$p = e^{-\Delta E/T}$$

Where ΔE is positive change in energy and T is the temperature.

- Probability of large uphill move is lower than small one. Also probability that an uphill move will be made decreases as temperature decreases.

Simulated Annealing

- Simulated Annealing (SA) is an effective and general form of optimization. It is useful in finding global optima in the presence of large numbers of local optima. "Annealing" refers to an analogy with thermodynamics, specifically with the way that metals cool and anneal. Simulated annealing uses the objective function of an optimization problem instead of the energy of a material.

- Implementation of SA is simple. The algorithm is basically hill-climbing except instead of picking the best move, it picks a **random move**. If the selected move improves the solution, then it is always accepted. Otherwise, the algorithm makes the move anyway **with some probability** less than 1. The probability decreases exponentially with the "badness" of the move, which is the amount ΔE by which the solution is worsened (i.e., energy is increased.)

- A **parameter T** is also used to determine this probability. It is analogous to temperature in an annealing system. At higher values of T, uphill moves are more likely to occur. As T tends to zero, they become more and more unlikely, until the algorithm behaves more or less like hill-climbing. In a typical SA optimization, T starts high and is gradually decreased according to an "annealing schedule".

Analogy between Physical System and Optimization Problem

Physical System		Optimization Problem
System State		Solution
Molecular Positions	⟷	Decision Variables
Energy		Objective Function
Minimizing Energy	⟷	Minimizing Cost
Ground State		Global Optima Solution
Quenching		Local Search
Temperature	⟷	Control Parameter, T
Real Annealing		Simulated Annealing

Simulated Annealing Algorithm

1) **Evaluate** the initial state. If it is a goal state, Then, return and quit. Otherwise continue with initial state as the current state.
2) **Initialize** BEST-SO-FAR to current state.
3) **Initialize** T according to the annealing schedule.
4) **Loop** until solution is found or until there are no operators left:
   a) **Select** an operator that has not yet been applied to the current state and apply it to produce a new state.
   b) **Evaluate** the new state. Compute
      ΔE = (Value of current state) – (Value of new state)
      - If the state is a goal state, return it and quit.
      - If it not a goal state, but better than current state, then make it current state (BEST-SO-FAR).
      - If it is **not better than current state**, then make it current state with probability
        $p = e^{-\Delta E/T}$
      This is done by using a **random number** generator by producing a number in the range [0,1]. If the number is less than p, then the move is accepted. Otherwise do nothing.
   c) **Revise T** as necessary according to the annealing schedule.
5) **Return** BEST-SO-FAR as the answer.

Difference between Simulated Annealing and Simple Hill Climbing

1) The **annealing schedule** must be maintained.
2) Moves to the **worse states** may be **accepted**.
3) In addition to current state, **best state found so far is maintained**. Thus, if the final state is worse than the earlier state (because of bad luck in accepting moves to worse state), the earlier state is still available.

Applications of simulated Annealing Algorithm

1) Travelling Salesman Problem
2) Graph Partitioning
3) Quadratic Assignment
4) Scheduling
5) Linear Arrangement
6) VLSI (Placement, Routing, Layout)
7) Image Processing
8) Code Design in Information Theory
9) Matching Problems ....

History of Genetic Algorithms (GA)

- Genetic Algorithms were invented to mimic some of the processes observed in **natural evolution**. Many people, biologists included, are astonished that, life at the level of complexity that we observe, could have **evolved** in the relatively short time suggested by the fossil record.

- The idea with GA is to use this **power of evolution** to solve **optimization problems**.

- The father of the original **Genetic Algorithm** was **John Holland** who invented it in the early 1970.

Why GA?

- It is better than conventional AI in that it is **more robust**. Unlike older AI systems, they **do not break easily** even if the inputs changed slightly, or in the presence of reasonable noise.

- In searching a **large state-space, multi-modal state-space, or n-dimensional surface,** a **genetic algorithm** may offer significant **benefits** over more typical search of optimization techniques. (linear programming, heuristic, depth-first, breath-first)

GA: Overview

- GAs simulates the **survival of the fittest** among individuals over consecutive generation for solving a problem.

- Each generation consists of a **population** of character strings that are analogous to the **chromosome** that we see in our DNA.

- Each **individual** represents a point in a search space and a **possible solution**.

GA: Overview

The individuals in the population are then made to go through a process of **evolution**. GAs are based on an **analogy** with the **genetic structure** and **behavior of chromosomes** within a population of individuals using the following foundations:

- Individuals in a population **compete for resources** and **mates**.
- Those individuals **most successful** in each 'competition' will **produce** more **offspring** than those individuals that perform poorly.
- Genes from `good' individuals propagate throughout the population so that two good parents will sometimes **produce** offspring that are **better than either parent**.

- Thus each successive generation will become more suited to their environment.

GA: State Space

- A population of individuals is maintained within search space for a GA, each representing a possible **solution** to a given problem (**Chromosomes/Genes**).
- Thus a chromosome (solution) is composed of several genes (variables).
- The GA maintains a population of n chromosomes (solutions) with associated **fitness values**.
- Parents are **selected** to mate, on the basis of their **fitness**, producing offspring via a reproductive plan. Consequently highly fit solutions are given more opportunities to reproduce, so that **offspring inherit** characteristics from each parent.
- Individuals in the population **die and are replaced** by the new solutions.
- Each successive generation will contain **more good `partial solutions'** than previous generations.

Implementation of GA

After an initial population is randomly generated, the algorithm evolves the through three operators:

1. **Selection** which equates to survival of the fittest;
2. **Crossover** which represents mating between individuals;
3. **Mutation** which introduces random modifications.

1) Selection Operator

Key idea: Give preference to better individuals, allowing them to pass on their genes to the next generation.

- The goodness of each individual depends on its fitness.
- Fitness may be determined by an objective function or by a subjective judgment.

Implementation of GA

2) Crossover Operator

- Prime **distinguished factor** of GA from other optimization techniques.
- Two individuals are chosen from the population using the **selection** operator.
- A **crossover point** along the bit strings is randomly chosen.
- The values of the two strings are **exchanged** up to this point.
  - If S1=000000 and s2=111111 and the crossover point is 2 then S1'=110000 and s2'=001111.
- The two new offspring created from this mating are put into the **next generation** of the population.
- By recombining portions of good individuals, this process is likely to create even **better individuals**.

Implementation of GA

3) Mutation Operator

- With some **low probability**, a portion of the new individuals will have some of their bits flipped.
- Its purpose is to maintain **diversity** within the population and inhibit premature convergence.
- Mutation alone induces a **random walk** through the search space.
- Mutation and Selection (without crossover) create a **parallel, noise-tolerant**, hill-climbing algorithms.

Genetic Algorithm (GA)

1. Randomly **initialize population(t).**
2. Determine **fitness** of population**(t)**
3. **Repeat**
   1. Select parents from population**(t)**
   2. Perform **crossover** on parents creating population**(t+1)**
   3. Perform **mutation** of population**(t+1)**
   4. Determine **fitness** of population**(t+1)**
4. **Until** best individual is **good** enough.

Representation of Population

- Some of the most **commonly used representations** for **representing population** for genetic algorithms are given below.

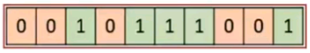
  1) Binary Representation
  2) Real Valued Representation
  3) Integer Representation
  4) Permutation Representation

- However, representation is highly **problem specific**.

Binary Representation

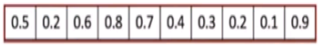
- This is one of the simplest and most widely used representation in GAs. In this type of representation the genotype consists of **bit strings**.
- For some problems when the solution space consists of **Boolean decision variables** – yes or no, the binary representation is natural.
- Example: **Knapsack Problem.**

  If there are n items, we can represent a solution by a binary string of n elements, where the x^{th} element tells whether the item x is picked (1) or not (0).



Real Valued Representation

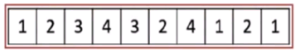
- For problems where we want to define the genes using **continuous** rather than discrete **variables**, the real valued representation is the most natural.



Integer Representation

- For discrete valued genes, **we cannot** always limit the solution space to **binary 'yes' or 'no'**.

- **Example**: If we want to encode the four distances – North, South, East and West, we can encode them as {0,1,2,3}. In such cases, **integer representation** is desirable.



Permutation Representation

- In many problems, the solution is represented by an **order of elements**. In such cases permutation representation is the most suited.
- **Example**: **Travelling Salesman Problem (TSP)**. In this the salesman has to take a tour of all the cities, visiting each city exactly once and come back to the starting city. The total distance of the tour has to be minimized. The solution to this TSP is naturally an ordering or permutation of all the cities and therefore using a permutation representation makes sense for this problem.

1	5	9	8	7	4	2	3	6	0

Population: Important Things

There are several **things to be kept in mind** when dealing with GA population:

1) The **diversity of the population** should be maintained otherwise it might lead to premature convergence.

2) The **population size** should not be kept very **large** as it can cause a GA to **slow down**, while a **smaller** population might **not** be enough for a **good mating pool**. Therefore, an optimal population size needs to be decided by trial and error.

Selection

- To **select** parents from population , **survival of the fittest** criteria is used.
- Some of the selection **techniques** are:
  1) Roulette Wheel Selection
  2) Stochastic Universal Sampling (SUS)
  3) Tournament Selection
  4) Rank Selection
  5) Random Selection

Fitness Proportionate Selection

- Fitness Proportionate Selection is one of the **most popular** ways of parent selection.
- In this every individual can become a parent with a **probability which is proportional to its fitness**.
- Therefore, **fitter individuals** have a **higher chance** of mating and propagating their features to the next generation.
- Therefore, such a selection strategy applies a selection pressure to the more fit individuals in the population, **evolving better individuals** over time.
- Consider a circular wheel. The wheel is divided into **n pies**, where n is the number of individuals in the population. **Each individual** gets a portion of the circle which is **proportional to its fitness value**.
- Two implementations of fitness proportionate selection
  1) **Roulette Wheel Selection**
  2) **Stochastic Universal Sampling**

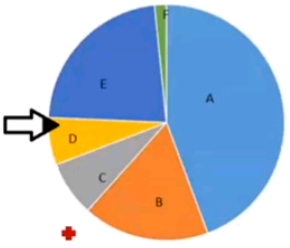
Roulette Wheel Selection

- Here the circular wheel is divided in **n pies (n is #individuals)** as described before. A fixed point is chosen on the wheel circumference as shown and the wheel is rotated. The region of the wheel which comes in front of the fixed point is chosen as the parent. For the second parent, the same process is repeated.
- A **fitter individual** has a greater pie on the wheel and therefore a **greater chance** of landing in front of the fixed point when the wheel is rotated.
- Therefore, **the probability** of choosing an individual **depends** directly on its **fitness**.

Roulette Wheel Selection

To Implementation, we use the following steps –

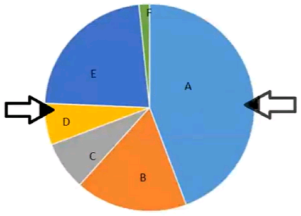
- **Calculate** S = the sum of a finesses.
- **Generate** a random number between 0 and S.
- Starting from the top of the population, **keep adding** the finesses to the partial sum P, till P<S.
- The individual for which P exceeds S is the **chosen individual.**



Chromosome	Fitness Value
A	8.2
B	3.2
C	1.4
D	1.2
E	4.2
F	0.3

Stochastic Universal Sampling (SUS)

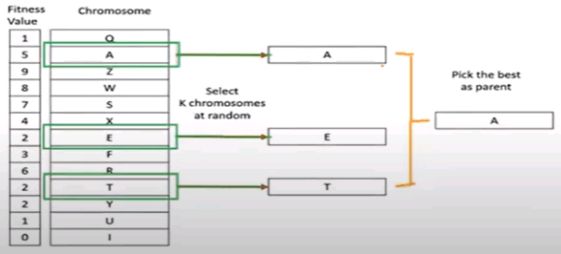
- Here, instead of having just one fixed point, we have **multiple fixed points** as shown in the following Figure. Therefore, all the parents are chosen in just one spin of the wheel. Also, such a setup encourages the highly fit individuals to be chosen at least once.
- It is to be noted that fitness proportionate selection methods **don't work** for cases where the fitness can take a **negative value**.



Chromosome	Fitness Value
A	8.2
B	3.2
C	1.4
D	1.2
E	4.2
F	0.3

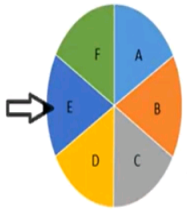
Tournament Selection

- In K-Way tournament selection, we select **K individuals** from the population at **random** and select the best out of these to become a parent. The same process is repeated for selecting the next parent.
- Tournament Selection is also extremely popular in literature as it can even **work with negative fitness values**.



Rank Selection

- Rank Selection also works with **negative fitness values** and is mostly used when the individuals in the population have very close fitness values (this happens usually at the end of the run)
- This leads to each individual having an almost **equal share of the pie** (like in case of fitness proportionate selection) as shown in the following figure and hence each individual no matter how fit relative to each other has an approximately same probability of getting selected as a parent.



Chromosome	Fitness Value
A	8.1
B	8.0
C	8.05
D	7.95
E	8.02
F	7.99

- This in turn leads to **a loss in the selection pressure** towards fitter individuals, making the GA to make poor parent selections in such situations.

Rank Selection

- Here, we remove the concept of a fitness value while selecting a parent. However, every individual in the population is ranked according to their fitness. The selection of the parents depends on the rank of each individual and not the fitness. The higher ranked individuals are preferred more than the lower ranked ones.

Chromosome	Fitness Value	Rank
A	8.1	1
B	8.0	4
C	8.05	2
D	7.95	6
E	8.02	3
F	7.99	5

Random Selection

- In this strategy we **randomly select parents** from the existing population.
- There is **no selection pressure** towards **fitter individuals** and therefore this strategy is usually avoided.

Genetic Algorithms - Crossover

- In this lecture, we will discuss a **Crossover Operator** , their uses and benefits.
- The crossover operator is analogous to **reproduction and biological crossover,** is usually applied with a high probability – p_c.
- In crossover **more than one parent is selected** and one or more **off-springs** are **produced** using the genetic material of the parents.
- These crossover operators are very generic and the GA Designer might choose to implement a **problem-specific crossover operator** as well.

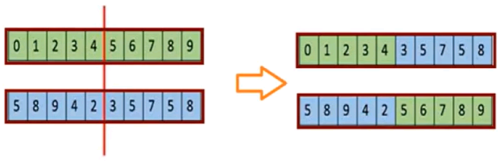
Genetic Algorithms - Crossover

Some of the most popularly used **crossover operators** are:

1) One Point Crossover
2) Multi Point Crossover
3) Uniform Crossover
4) Whole Arithmetic Recombination
5) Devis' Order Crossover

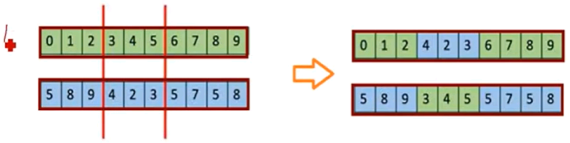
One Point Crossover

In one-point crossover, **a random crossover point** is selected and the **tails of its two parents** are **swapped** to get new off-springs.



Multi Point Crossover

Multi point crossover is a generalization of the one-point crossover wherein **alternating segments are swapped** to get new off-springs.



Uniform Crossover

- In a uniform crossover, we **don't divide the chromosome** into segments, rather we **treat each gene separately**.
- In this, we essentially **flip a coin** for each chromosome to decide **whether or not it'll be included** in the **off-spring**. We can also bias the coin to one parent, to have more genetic material in the child from that parent.

Whole Arithmetic Recombination

- This is commonly used for **integer representations** and works by taking the **weighted average** of the two parents by using the following formulae –

  Child1 = α.x + (1-α).y

  Child2 = α.y + (1-α).x

  Obviously, **if α = 0.5**, then both the children will be **identical** as shown in the following image.

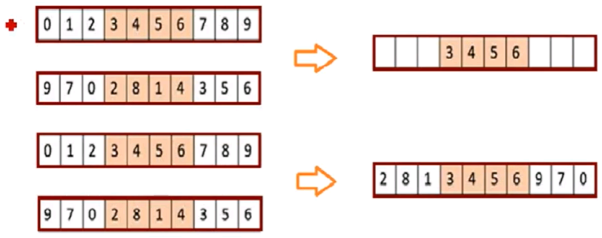
Davis' Order Crossover (OX1)

OX1 is used for **permutation based crossovers** with the intention of transmitting information about relative ordering to the off-springs. It works as follows :

1) **Create** two random crossover points in the parent and **copy** the **segment** between them from the first parent to the first offspring.
2) Now, starting from the second crossover point in the second parent, **copy** the **remaining unused numbers** from the second parent to the first child, wrapping around the list.

Repeat for the **second child** with the parent's role **reversed**.

Davis' Order Crossover (OX1)



Similarly, repeat for second child.

Genetic Algorithms - Mutation

- Mutation may be defined as a **small random tweak** in the **chromosome**, to get a new solution.
- It is used to maintain and introduce **diversity** in the genetic population and is usually applied with a low probability i.e. p_m. If the probability is very high, the GA gets reduced to a random search.
- Mutation is the part of the GA which is related to the "**exploration**" of the search space. It has been observed that **mutation is essential to the convergence of the GA** while crossover is not.

Genetic Algorithms - Mutation

- Some of the most commonly used **mutation operators** are:

  1) Bit Flip Mutation
  2) Random Resetting
  3) Swap Mutation
  4) Scramble Mutation
  5) Inversion Mutation

- The GA designer might find a **combination** of these approaches or a **problem-specific** mutation operator more useful.

Bit Flip Mutation

In bit flip mutation, we select **one or more random bits** and **flip** them.
This is used for **binary encoded** GAs.

Random Resetting

- Random Resetting is an **extension of the bit flip** for the **integer representation**.
- In this, **a random value** from the set of permissible values is assigned to a **randomly chosen gene**.
- Let's say a[i] (an array index/gene) ranges from [1, 6] then random resetting mutation will select one value from [1, 6] and replace a[i]'s value with it.

Swap Mutation

- In swap mutation, we select **two positions** on the chromosome **at random**, and **interchange** the values. This is common in **permutation based encodings**.

Scramble Mutation

- Scramble mutation is also popular with **permutation representations**.
- Here, from the entire chromosome, a **subset of genes** is chosen and their **values are scrambled** or shuffled randomly.

Inversion Mutation

- In inversion mutation, we select a subset of genes like in scramble mutation, but **instead of shuffling** the subset, we merely **invert the entire string** in the **subset**.

Genetic Algorithms - Termination Condition

- The **termination condition** of a Genetic Algorithm is important in determining when a GA run will end.
- It has been observed that **initially**, the GA progresses **very fast** with better solutions coming in every few iterations, but this **tends to saturate** in the **later** stages where the improvements are very small.
- We usually want a **termination condition** such that our **solution is close to the optimal**, at the end of the run.
- Usually, we keep one of the following **termination conditions** –

1) When there has been **no improvement** in the population for X iterations.
2) When we reach an **absolute number of generations**.
3) When the **objective function** value has **reached a certain pre-defined value**.

Genetic Algorithms - Application Areas

1) Optimization
2) Economics
3) Neural Networks
4) Parallelization
5) Image Processing
6) Vehicle Routing
7) Scheduling Algorithms
8) Machine Learning
9) Robot Trajectory Generation
10) Parametric Design of Aircrafts
11) DNA Analysis
12) Multimodal Optimization
13) TSP and its Applications ....

Limitations of GA

- GAs are **not suited** for all problems, especially problems which are simple and for which derivative information is available.
- **Fitness value** is calculated repeatedly which might be **computationally expensive** for some problems.
- Being stochastic, there are **no guarantees on the optimality** or the quality of the solution.
- If not implemented properly, the **GA may not converge** to the optimal solution.

Introduction

- Ant Colony Optimization (ACO) is a derivative of Swarm intelligence (SI).
- The ant colony optimization algorithm (ACO), introduced by Marco Dorigo, in the year 1992.
- It is a paradigm for designing meta heuristic algorithms for optimization problems and is inspired by the foraging behavior of ant colonies.
- Ant Colony Optimization targets discrete optimization problems and can be extended to continuous optimization problems which is useful to find approximate solutions.
- ACO algorithm is the most successful and widely recognized algorithmic based on the ant behavior.

Ant Behavior

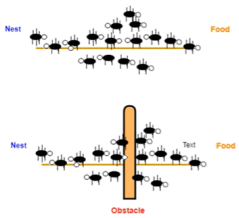
- Ants communicate to one another by laying down pheromones along their trails, so where ants go within and around their ant colony is a stigmergic system.
- Stigmergy (STIG-mər-jee) is a mechanism of indirect coordination, through the environment, between agents or actions. The principle is that the trace left in the environment by an individual action stimulates the performance of a succeeding action (possibly by the same agent, but in many scenarios by a different agent).
- In many ant species, ants walking from or to a food source, deposit on the ground a substance called pheromone.
- Other ants are able to smell this pheromone, and its presence influences the choice of their path, that is, they tend to follow strong pheromone concentrations.

Ant Behavior

- The pheromone deposited on the ground forms a pheromone trail, which allows the ants to find good sources of food that have been previously identified by other ants.
- Using random walks and pheromones within a ground containing one nest and one food source, the ants will leave the nest, find the food and come back to the nest. After some time, the way being used by the ants will converge to the shortest path.

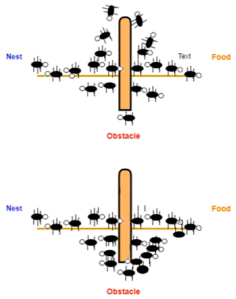
Ant Behavior

Ants in a pheromone trail between nest and food



An obstacle interrupts the trail

Ants find two paths to go around the obstacle

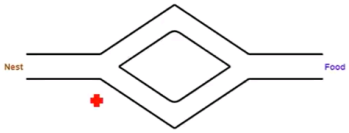


A new pheromone trail is formed along the shorter path

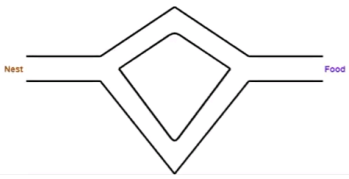
Double Bridge Experiment

- Deneubourg et al. verified the pheromones marking of ants by the experience known as "double bridge experiment".
- In the double bridge experiment, a nest of a colony of Argentine ants is connected to a food source by two bridges.
- The ants can reach the food source and return to the nest using any of the two bridges.
- The aim of the experiment is to observe the resulting behavior of the colony.
- It is observed that if the two bridges have the same length, the ants tend to converge towards the use of one of the two bridges. If this experiment is repeated a number of times, it is observed that each of the two bridges is used in about half of the cases.
- The fact is while moving, ants deposit pheromone on the ground and whenever they have to decide which path should follow, their choice is based higher the pheromone concentration found on a particular path, the higher is the probability to follow that path.

Double Bridge Experiment

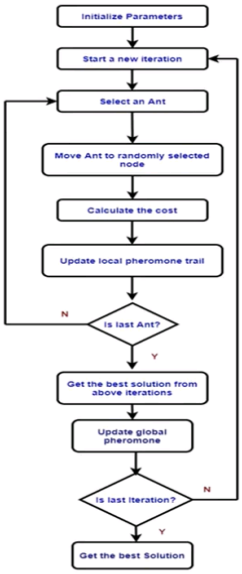


Branches have equal length.



Branches have different length.

ACO Algorithm



Ant Colony Optimization Algorithm

- The ants are driven by a probability rule to choose their solution to the problem, known as tour.
- The probability rule between two nodes *i* and *j*, called Pseudo-Random-Proportional Action Choice Rule, and it depends on two factors: the heuristic and metaheuristic.

$$p_{ij} = \sum_{h \in s} \frac{[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}}{[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}}$$

- Where
  - τ is the pheromone,
  - η is the inverse of the distance between the two nodes.

Ant Colony Optimization Algorithm

- Each ant modifies the environment in two different ways:
  - Local trail updating
  - Global trail updating

Local trail Updating

- As the ant moves between nodes it updates the amount of pheromone on the edge by the following equation

$$\tau_{ij}(t) = (1-\rho)\tau_{ij}(t-1) + \rho\tau_0$$

Where

- ρ is the evaporation constant.
- τ_0 is the initial value of the pheromone trails.
- $\tau_0\ can\ be\ calculated\ as\ \tau_0 = \left(\frac{n}{L_n}\right) - 1$.

  Where
  - n is the number of nodes
  - L_n the total distance covered between the total nodes, produced by one of the construction heuristics.

Global Trail Updating

- When all ants have completed all the nodes that find the shortest path updates the edges in its path using the following equation

$$\tau_{ij}(t) = (1-\rho)\tau_{ij}(t-1) + \rho/L^+$$

Where

- L^+ is the length of the best path generated by one of the ants.

Applications

- Travelling Salesman Problem (TSP)
- Quadratic Assignment Problem
- Vehicle Routing Problem
- Graph Coloring Problem
- Sequential Ordering Problem
- Job Scheduling Problem
- Routing in Telecommunications Networks