**Analysis of Papers**

**Paper 1 : Ant Colony Optimization-Artificial Ants as a Computational Intelligence Technique**

Ant Colony Optimization (ACO) is a metaheuristic algorithm inspired by the foraging behavior of ants. The algorithm was introduced by Marco Dorigo in the early 1990s.It has been successfully applied to various combinatorial optimization problems, including the Traveling Salesman Problem (TSP) where the goal is to find the shortest possible tour that visits a set of cities and returns to the starting city.

Traditional methods such as Brute Force and Dynamic Programming often have advantages in terms of finding optimal or near-optimal solutions for small to medium-sized instances. However, they may struggle with scalability for larger instances, where metaheuristic approaches like ACO can be more effective.

This paper provides a comprehensive overview of Ant Colony Optimization for solving the Traveling Salesman Problem (TSP)

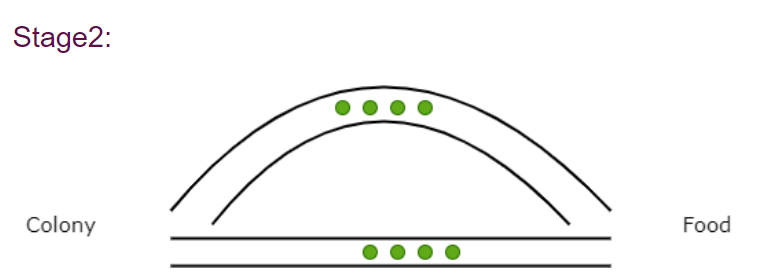
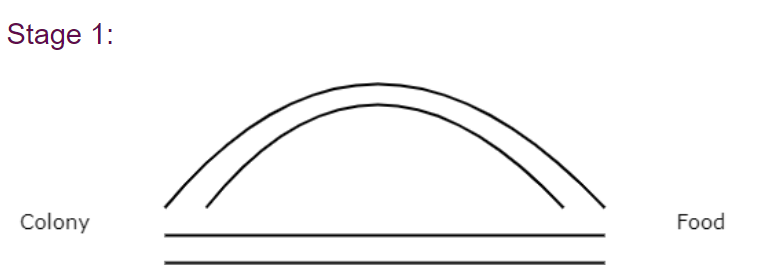
**Biological Inspiration of Ant Colony Optimisation:-**

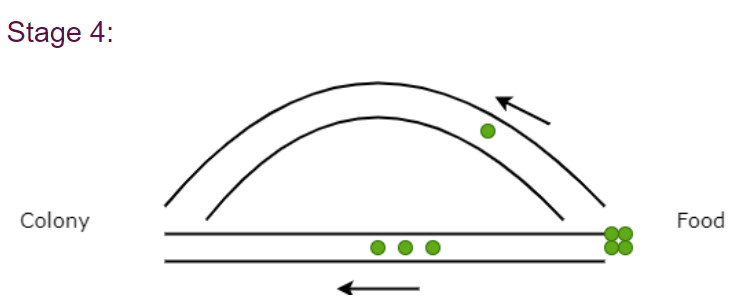
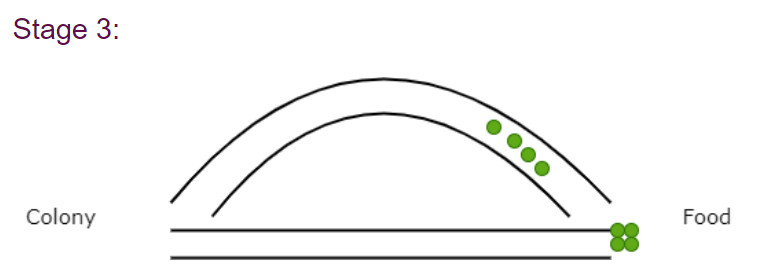
Ant Colony Optimization (ACO) is inspired by the foraging behavior of real ants. The biological inspiration comes from the observation of how ant colonies efficiently find the shortest path between their nest and a food source.

**How Ants Communicate?**

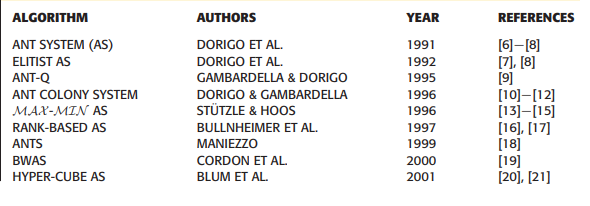
Ants indirectly communicate with each other using Pheromones. It is a Chemical that ants deposit on the ground in order to mark some favourable path that should followed by other members of the colony.

**Principle of Ant Colony Optimization:-**



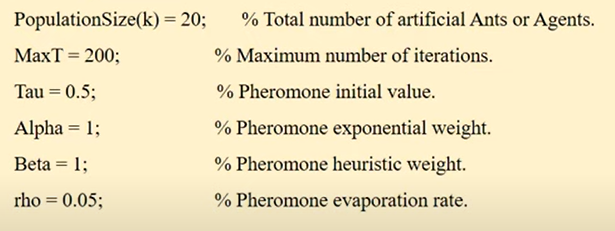


**Successful Ant Colony Ant Optimization Algorithms:-**

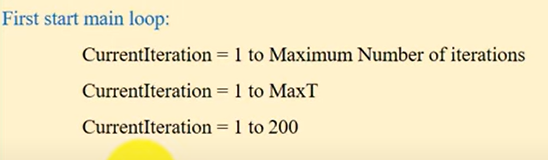


***ACO(Ant Colony Optimization) Algorthim step-by-step:-***

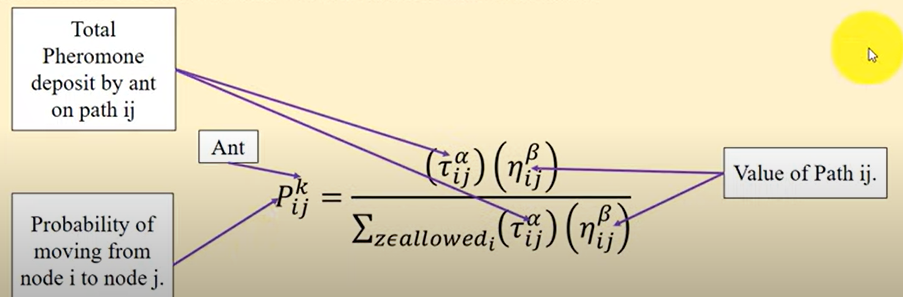
***Step 1. Inititalize ACO Parameters***

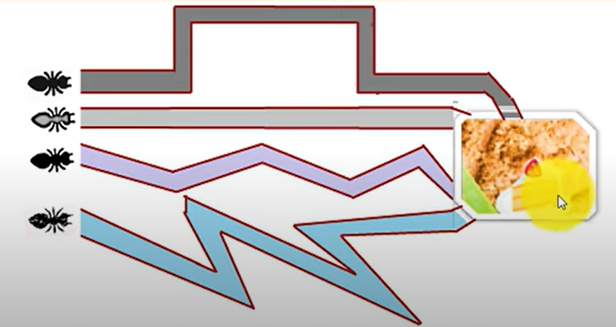
******

***Step 2 : Construct Ant Solution***

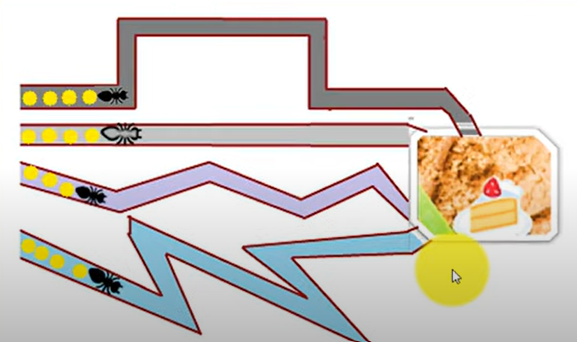
******

***Step 3 :- Position each ant in the Starting Node:-***

******

***Here kth ant move from node I to node j with probability. ***

***Step 4 :- Each ant will select next node by applying state transition rule.***

******

***Step 5:- Repeat until ant build the best solution , then compute the fitness value.***

***Step 6:- Update the best solution***

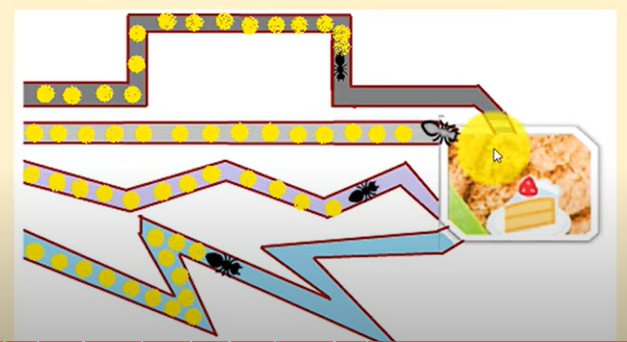
**Compare the best solution with easy ant solution**

If (ant(k) solution < Best Solution)

{

Consider Ant(k) Best Solution

}

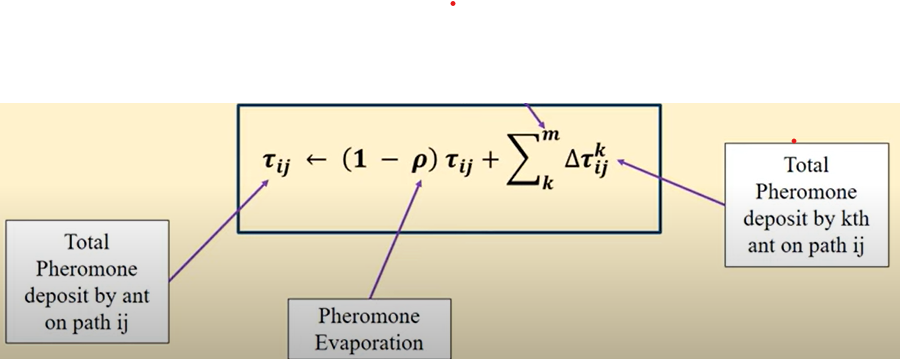
***Step 7 :-Apply offline Pheromone update***

(Note- Pheromone trail are updated when all ants complete their cost/solution)

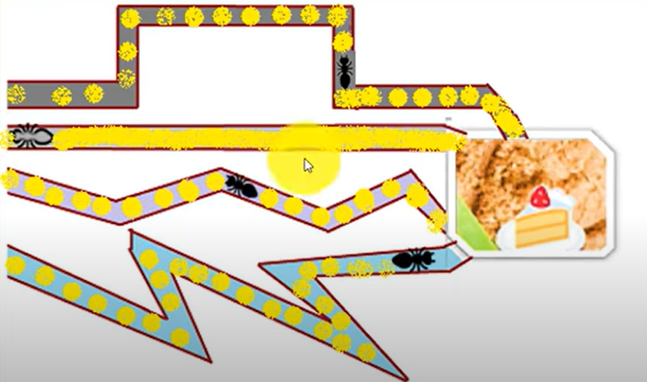
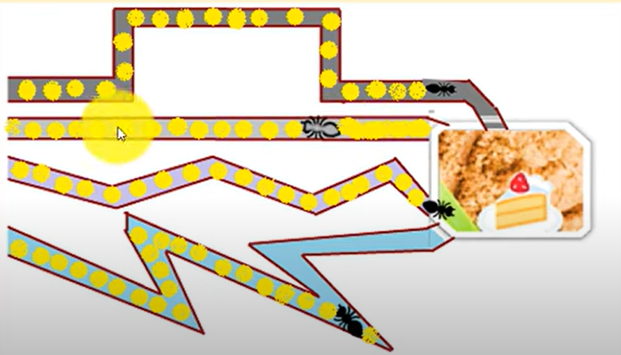
Case 1 – For best solution Increase the level of Pheromone Trials

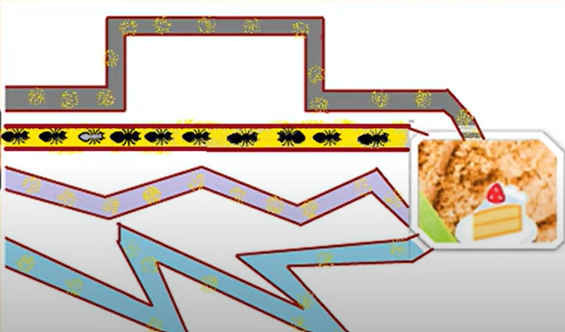
Case 2 - For worst solution Decrease the level of Pheromone Trials

**Pheromone Level Updating Equation:-**



***Step 8 – Display the Best Solution***





**The ACO algorithm can be formalized as follows:**  
Input: A complete undirected graph G with n vertices and edge weights d(u, v)  
Output: An approximate tour T of the TSP  
  
Initialize a set of initial solutions S  
while the termination criterion is not met  
    for each ant in S  
        Construct a solution T' by probabilistically selecting the next city to visit  
        Update the pheromone levels on the edges in T'  
        Apply local search to T'  
    end for  
    Replace the worst solution in S with the best solution constructed in the current iteration  
end while  
Output the best solution in S

**Problems ACO Has Solved Effectively**

* Traveling salesman problem and vehicle routing
* Scheduling and project planning optimization
* Telecommunications network routing
* Protein folding configurations
* Machine learning classification and data mining

The overall time complexity of Ant Colony Optimization (ACO) for solving the Traveling Salesman Problem (TSP) is O(n^2 \* m), where n is the number of vertices in the graph and m is the number of ants.

**Paper 2 :- On the Nearest Neighbor Algorithms for the Traveling Salesman Problem**

In this Paper it proposes a hybrid construction algorithm for solving TSP based on the nearest neighbor algorithm and the greedy algorithm. These algorithms are commonly used in TSP problem-solving due to their simplicity and efficiency. Combining them in a hybrid approach aims to leverage their individual strengths for better results.

As in this Research Paper it is mentioned, that there are four main classes of algorithms used to solve the TSP:

**The algorithms for solving TSP can be divided into four classes:**

1. **Exact algorithms:** These guarantee finding the optimal solution but can be computationally demanding.
2. **Heuristic algorithms:** These aim to find good solutions but don't guarantee optimality. They are generally more computationally efficient than exact algorithms.
3. **Approximate algorithms:** These provide solutions that are close to optimal but may not be the exact optimal solution.
4. **Metaheuristic algorithms:** These are higher-level strategies that guide other heuristics to explore the solution space more efficiently.

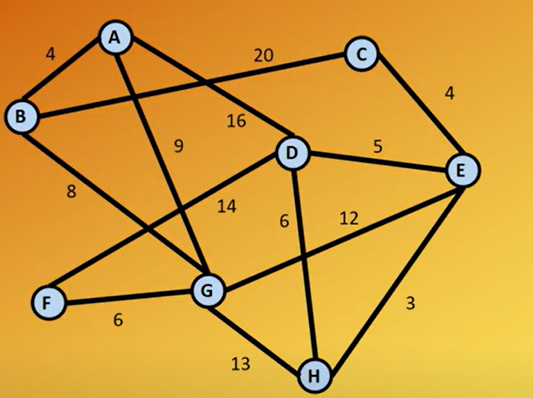
Within heuristic algorithms, there are three main classes:

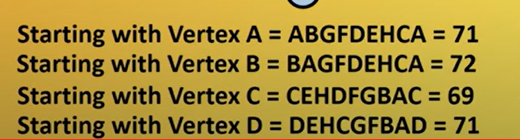
* **Tour construction algorithms:** These gradually build a tour by adding a new city at each step.
* **Tour improvement algorithms:** These enhance an existing tour by performing various exchanges.
* **Hybrid algorithms:** These combine both construction and improvement heuristics simultaneously.

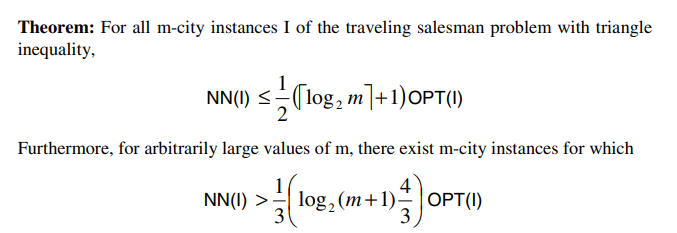
**1.The Nearest Neighbor Algorithm (NN):-**

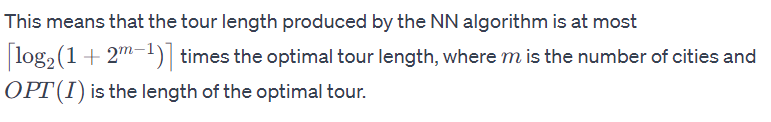
The nearest neighbor (NN) algorithm for determining a traveling salesman tour is as follows. The salesman starts at a city, then visits the city nearest to the starting city. Afterwards, he visits the nearest unvisited city, and repeats this process until he has visited all the cities, in the end, he returns to the starting city.

**The steps of the algorithm are as following:-**







***Limitations :-***The NN algorithm does not guarantee good approximation ratios in general, especially as the number of cities increases

**2) The Nearest Neighbor Algorithm (NND) from Both End-Points**

The Nearest Neighbor Algorithm from Both End-Points (NND) is a modification of the classic Nearest Neighbor (NN) algorithm for solving the Traveling Salesman Problem (TSP). The NND algorithm introduces a variation in the way it selects the next city to visit, considering two end vertices instead of one. NND algorithm is a heuristic Algorithm.

**Here are the steps of the NND algorithm:**

1. **Choose an Arbitrary Vertex:**
   * Start by selecting an arbitrary vertex in the graph as the starting point.
2. **Visit the Nearest Unvisited Vertex:**
   * Move to the nearest unvisited vertex from the current vertex.
3. **Visit the Nearest Unvisited Vertex to Both End Vertices:**
   * Add a vertex to the tour that has not been visited before and is the nearest to both the current vertex and the starting vertex. Update the end vertices to include this newly visited vertex.
4. **Check for Unvisited Vertices:**
   * If there are still unvisited vertices, go back to step 3 and repeat the process.
5. **Return to the Starting Vertex from the Other End Vertex:**
   * Once all vertices are visited, return to the starting vertex from the other end vertex to complete the tour.

In essence, the NND algorithm extends the idea of the NN algorithm by maintaining two end vertices during the construction of the tour. It selects the next city to visit based on its proximity to both of these end vertices.

**3) Greedy Algorithm**

The Greedy Algorithm described in the provided text is a heuristic approach for solving the Traveling Salesman Problem (TSP). The algorithm constructs a tour by iteratively selecting the shortest edge and adding it to the tour, subject to certain constraints. The goal is to gradually build a tour that visits all cities while avoiding cycles with fewer than N edges and limiting the increase in the degree of any node by more than 2.

**Here are the steps of the Greedy Algorithm:**

1. **Sort Edges by Increasing Lengths:**
   * Arrange the edges of the graph in ascending order based on their lengths (distances).
2. **Select the Shortest Edge and Add to Tour:**
   * Choose the shortest edge from the sorted list.
   * Add the selected edge to the tour if adding it doesn't violate the specified constraints:
     + It doesn't create a cycle with fewer than N edges.
     + It doesn't increase the degree of any node by more than 2.
     + The same edge is not added twice.
3. **Check for the Number of Edges in the Tour:**
   * Verify whether the number of edges in the tour is equal to N (the total number of cities).
   * If not, go back to step 2 and repeat the process.

**Limitation :** While the Greedy Algorithm is a simple and intuitive heuristic, it does not guarantee an optimal solution for the TSP. It may produce suboptimal results, and the quality of the solution depends on the specific characteristics of the TSP instance.

**4) Hybrid algorithm NNDG**

The Hybrid algorithm described, named NNDG (a hybrid of NND and Greedy algorithms), combines aspects of the Nearest Neighbor from Both End-Points (NND) algorithm and the Greedy algorithm. It aims to benefit from the strengths of both algorithms to find a solution to the Traveling Salesman Problem (TSP). Below are the steps of the NNDG algorithm:

1. **Find the Vertex with the Smallest Row-Sum:**
   * Compute the row-sum for each vertex by summing the entries in its row in the adjacency matrix.
   * Identify the vertex with the smallest row-sum.
2. **Apply NND Algorithm Starting from the Smallest Row-Sum Vertex:**
   * Start the NND algorithm from the vertex with the smallest row-sum.
   * Continue the NND algorithm until reaching a vertex that has been added to the tour before.
3. **Continue Applying NND from the Last Vertex in the Tour:**
   * Repeat the NND algorithm starting from the last vertex in the tour until reaching a vertex that has been added to the tour before.
4. **Calculate [*nt/k]***
   * Calculate **[*nt/k]*** where *n* is the number of vertices, *t* is the number of edges taken from each solution found by NND, and *k* is the number of implementations of NND.
5. **Take First *t* Edges from Each NND Solution:**
   * Take the first *t* edges respectively from each solution found by NND.
   * Add the edges that do not form subtours to the solution.
6. **Apply Greedy Algorithm to the Remaining Edges:**
   * Apply the Greedy algorithm to the remaining edges not included in the solution.

**Conclusion:** The NNDG algorithm aims to leverage the NND algorithm for its ability to explore diverse paths while avoiding cycles, and then combines it with the Greedy algorithm to handle the remaining edges. This hybrid approach is an attempt to balance the exploration capabilities of NND with the efficiency of Greedy in handling the remaining edges.

**Results:-**

* Proposed NND algorithm improves on basic Nearest Neighbor, reduces tour lengths
* NNDG hybrid further enhances quality of solutions
* Starting tour construction from smaller row-sum vertices gives better results
* Tested on TSPLIB benchmark instances, tour lengths closer to optimum

**Time Complexity**

* Nearest Neighbor algorithms - O(n^2)
* Greedy algorithm - O(n^2 log n)
* NNDG hybrid - O(n^2 log n)

**PAPER-3 An effective heuristic for the traveling salesman problem" by Lin and Kernighan**

The Lin-Kernighan (LK) heuristic is a greedy algorithm for improving a given tour of the traveling salesman problem (TSP). It works by iteratively swapping pairs of edges to reduce the total length of the tour. The LK heuristic is considered one of the most effective heuristics for the TSP, especially for large instances.

**Mathematical Approach:**

The LK heuristic is based on the concept of 2-opt moves. A 2-opt move is a pair of edges (u, v) and (w, x) in a tour that can be swapped to produce a shorter tour. The LK heuristic starts with an initial tour and then repeatedly applies 2-opt moves to improve the tour. The heuristic stops when no further 2-opt moves can be found that reduce the tour length.

**Algorithmic Steps:**

1. **Initialize:** Start with an initial tour of the TSP.
2. **Identify 2-opt Moves:** Identify all possible 2-opt moves that can be applied to the current tour.
3. **Perform 2-opt Move:** Select the 2-opt move that results in the largest reduction in tour length and apply it.
4. **Repeat:** Repeat steps 2 and 3 until no further 2-opt moves can be found.

**Mathematical Formulation:**

Let G = (V, E) be a complete undirected graph with n vertices, where V = {v1, v2, ..., vn} is the set of vertices and E is the set of edges. Let d(u, v) be the distance between vertices u and v. The TSP is to find the shortest Hamiltonian cycle in G, which is a cycle that visits each vertex exactly once.

**The LK heuristic can be formalized as follows:**

Input: A tour T of the TSP  
Output: An improved tour T'  
  
T' = T  
while there exists a 2-opt move (u, v), (w, x) in T' such that  
    d(u, v) + d(w, x) > d(u, w) + d(v, x)  
    T' = swap((u, v), (w, x)) in T'  
end while  
  
return T'

**Time Complexity:**

The time complexity of the LK heuristic is O(n^3), where n is the number of vertices in the graph. This means that the algorithm's running time grows cubically with the number of vertices.

**Practical Applications:**

The LK heuristic is used in a variety of applications, including:

* Route planning for delivery services
* Scheduling for manufacturing processes
* Circuit design for electronics
* Sequencing for genome assembly