Ant Colony Optimization

# Literature Review

Ant colony algorithm, a heuristic simulated algorithm, provides better solutions for non-convex,

non-linear and discontinuous optimization problems. ACO is an evolutionary algorithm, which simulates the ant behaviour of feeding. It was first proposed by Italian scientist Marco Dorigo in 1991 based behavior of biological ants.

Ants search for the shortest path between their colony and a source of food by communicating information between themselves and cooperation. In the movement of the ants, a substance called “pheromone” is laid down on the trail, of which the intensity can be detected and estimated by every ant and lead them to move towards the direction of high pheromone intensity. Using this process, the more ants that pass through a specific edge, the higher the probability other ants would choose this edge in their path. Since more ants are choosing that path, consequently the more pheromone Is being dropped and the shortest path is being chosen. Once a set of ants’ finishes searching, the intensity of the pheromone is updated and this process would be iterated over and over until a majority of the ants are following the same path which would be the optimal path and the searching is then terminated [1].

One of the main uses of ACO is solving Combinatorial Optimization Problems such as the Travelling Salesman Problem (TSP). Starting from Ant System (AS), several improvements of the basic algorithm have been proposed. Typically, these improved algorithms have been tested again on the TSP as TSP has been extensively researched in academic literature and has attracted a considerable amount of research effort. All these improved versions of AS have in common a stronger exploitation of the best solutions found to direct the ants’ search process; they mainly diﬀer in some aspects of the search control. [3]

# TSP/GA Implementation

Simple “pure” GA Implementation as defined by J.-Y. Potvin [1]

1. Create an initial population of P chromosomes (generation 0).
2. Evaluate the fitness of each chromosome.
3. Select P parents from the current population via proportional selection (i.e., the selection probability is proportional to the fitness).
4. Choose at random a pair of parents for mating. Exchange bit strings with the one-point crossover to create two offspring.
5. Process each offspring by the mutation operator and insert the resulting offspring in the new population.
6. Repeat steps 4 and 5 until all parents are selected and mated (P offspring are created).
7. Replace the old population of chromosomes by the new one.
8. Evaluate the fitness of each chromosome in the new population.
9. Go back to step 3 if the number of generations is less than some upper bound. Otherwise, the result is the best chromosome created during the search.

Each chromosome will encode the solution of the problem. In this case the solution would be the path that would be taken through the cities. Since in a TSP context the problem is to minimize the solution, the highest fittest chromosome would the one with the shortest path distance.

As for chromosome encoding, having the encoding as path representation is seen as the most natural way as opposed to ordinal representation. Path representation is also intuitive to use for preforming crossover operation on the chromosomes. [1]

Some possible crossovers for the chromosomes are On-Point crossover, Partially Mapped Crossover (PMX), Cycle Crossover (CX), Order Crossover (OX), Order-Based Crossover (OBX), etc.

In Numerous studies [47, 28, 55] by different people such as Oliver at al. found that order-preserving operators were preforming better than others. In one study, Order Crossover (OX) was found optimum in 25 out of 30 test cases compared to the other crossovers [55]. Therefore, that would be the crossover operator I would choose to implement in my GA.

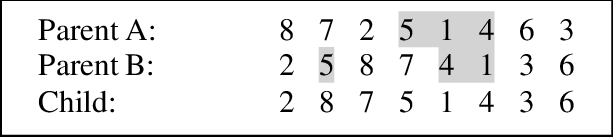


Illustration of the Order Crossover (OX)

The NN algorithm is used initially for the TSP in order to generate the initial population. NN routes are found for each city as starting city. These NN routes are stored and analyzed for their fitness values. The better routes from this NN algorithm are introduced along with randomly generated solutions for the genetic algorithms. The percentage of filling the initial population with top routes found by NN is set to 10%. The remaining 90% of the population are generated randomly similar to a pure GA. This helps the GA evade local optima and search for more optimal routes than the NN while still having diversity in the chromosomes.[2]

For every crossover operation, two chromosomes are randomly selected using roulette wheel selection. In roulette wheel selection, the individuals are given a probability Pi of being selected that is directly proportionate to their fitness therefore the chromosomes with higher fitness stand a better chance for getting selected for crossover [3]. This selection of the chromosome with higher fitness for the crossover produces a better next generation with higher fitness values. The crossover operation continues until the specified crossover rate in met. The crossover rate for binary chromosomes is as high as 80–90%, whereas the crossover rate used here is 50% due to the decimal chromosomes.[2]



Once all the crossover of the previous generation is complete and a new generation of children are generated, mutation is a carried out on a percentage of the children. The mutation operator enhances the ability of the GA to find a near optimal solution to a given problem by maintaining a sufficient level of genetic variety in the population, which is needed to make sure that the entire solution space is used in the search for the best solution. Reverse Sequence Mutation (RSM) was chosen for this implementation. In the reverse sequence mutation operator, we take a sequence S limited by two positions i and j randomly chosen, such that i< j. Then the gene order will be reversed [4]. In decimal chromosomes, the mutation rate goes up to of the order of 85 %. This is due to the fact that, mutation produces the better offspring in the evolution and is the primary genetic operation in this case [5].

After the mutation of the children is complete the children are then added to the population. Elitism is also applied to this GA, so the fitness must be calculated of all the chromosomes. A GA for TSP with the population size of 100, the elitism rate is set to 50%. Thus, the top 50 chromosomes in every generation are passed over to the next generation. The elitism rate directly depends on the size of the population and the rate of elitism should be decreased when the population size is increased. [2]

Repeat these steps for every generation. After all the generation are finished and we have the final population, the fittest chromosome would have the solution for the shortest path distance. All the parameters for this GA would have to be custom for every TSP since it’s virtually impossible to find paraments that would universally return the best path.

# ACO Implementation

A 2D Matrix was initialised that held the distances between the cities so to not calculate them on the fly.

distance = Math.*sqrt*((xb - xa) \* (xb - xa) + (yb - ya) \* (yb - ya));

A 2D matrix for pheromone levels is also initialised, which is of the same size as the distance matrix, however instead stores the pheromone level between every 2 cities.

**Ant Model**

Each Ant stores

* Current City
* Path Distance
* Visited Cities (List of Cities)
* Path (List of Cities)

[1]<https://sci-hub.tw/https://link.springer.com/article/10.1007%2FBF02125403>

[2] <https://ieeexplore-ieee-org.ejournals.um.edu.mt/document/4531202?part=1>

[3] <https://arxiv.org/ftp/arxiv/papers/1203/1203.3099.pdf>

[4] <https://arxiv.org/ftp/arxiv/papers/1203/1203.3097.pdf>

[15]<https://www.researchgate.net/publication/2771967_ACO_Algorithms_for_the_Traveling_Salesman_Problem>