L-Università ta' Malta

ICS2207

Machine Learning

Zakkarija Micallef - *0466500L*
B.Sc. in IT(Hons.) (Artificial Intelligence)

Contents

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].

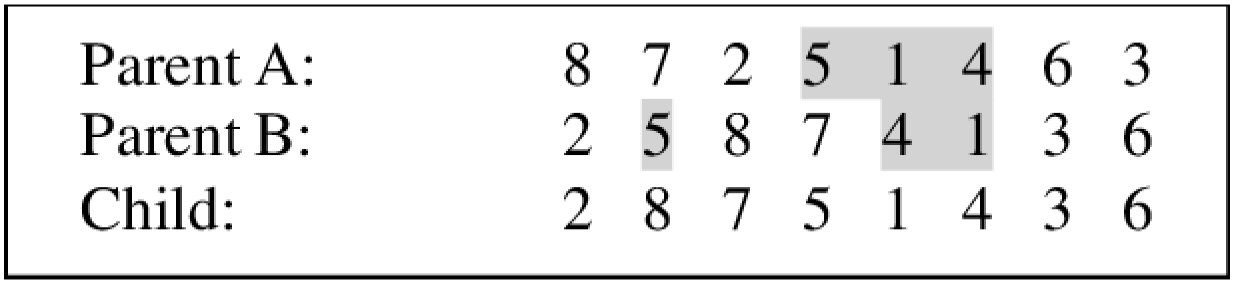
One of the main uses of ACO is solving Combinatorial Optimization Problems such as the Travelling Salesman Problem (TSP). Starting from Ant System (AS), several improvements of the basic algorithm have been proposed. Typically, these improved algorithms have been tested again on the TSP as TSP has been 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 [9] 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 [9]. Therefore, that would be the crossover operator I would choose to implement in my GA.



Parent A:	8	7	2	5	1	4	6	3
Parent B:	2	5	8	7	4	1	3	6
Child:	2	8	7	5	1	4	3	6

Illustration of the Order Crossover (OX) 1

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]

$$\frac{1}{N-1} \times \left(1 - \frac{f_i}{\sum_{j \in \text{Population}} f_j}\right)$$

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

The aim of this application was to use the Ant Colony Optimization algorithm on the Travelling Salesman Problem. A folder of TSP problems was obtained from TSPLIB [7] website that included the various cities with their respective location (x-axis and y-axis).
These files were put into a folder called "dataset". The Application goes through each file in the folder that has a *"tsp"* extension and passes it to the **FileInput** class. The **FileInput** class traverses through the TSP files and tokenizes it. It creates and ArrayList of cities that include the City's ID along with its x-coordinate and y-coordinate.

A matrix (2D Array) is initialized that stores the distance between each and every city in the TSP. Initially, all the elements of this matrix are calculated using the below formula.

$$\text{distance} = \sqrt{(x_b - x_a)^2 + (y_b - y_a)^2}$$

A 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.

The Ant Model

Each Ant stores

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

A slightly modified algorithm by Hui Yu [5]

- Step 1: Parameters are Initialized (quantity of ant M, the maximum iteration number nMax, the importance of pheromone *alpha* , the importance of heuristic information *beta* (trigger importance), the evaporation rate of local pheromone *delta*, the evaporation rate of global pheromone *ro*, pheromone matrix, heuristic information (distance) matrix, iteration number.
- Step 2 ： Put M ants in N cities randomly and store these cities in the ant's visited list.
- Step 3: If the iteration number is smaller than the maximum iteration number nMax, continue the next step, otherwise jump to Step 10.
- Step 4: Ant k chooses next city j according to the city-select strategy, meanwhile put city j in the visited list.
- Step 5: Update the pheromone of the path between the cities I and j according to the local pheromone update.
- Step 6: Once M ants have visited N cities, get the current local best ant.

- Step 7: Compare it with the global optimal solution (best global ant), if it is smaller, the local best ant becomes the global best ant, iterate until the best global ant is found.
- Step 8: Preform Global pheromone update.
- Step 9: Increment iteration number and jump to Step 3.
- Step 10: Output the optimal solution (the shortest path and path length).

ACO Parameters:

- Number of ants that is generated is determined to be equal to the number of cities in the TSP as it was found to be optimal according to [5]. Except when the number of cities is larger than 75, the number of ants is set to 10 so that it would compute in a reasonable amount of time.
- q_0 is the pre-set probability trigger when choosing the ant's next city. When the number of cities is less than 50 q_0 is set to 0.01 else it's set to 0.1
- *Alpha* is the importance that is given to the amount of pheromone on an edge. Beta is the importance given to the heuristic information. *Alpha* and *Beta* perform better when they are increased when the number of cities increase and when *alpha* is less than *beta*. *Alpha* was set to 2 and *Beta* was set to 3 as the preloaded TSP's are on the smaller side.
- *Delta* is the global evaporation rate while *rho* is the local pheromone evaporation rate. *Delta* and *rho* were both set to 0.1.

Ants are generated with a random initial city. Every ant is then moved to a new location until all the ants have moved one city. Then every ant again moves one city until they all have covered the entire map.

The next city that an ant goes on is determined by two things, pheromone and distance. Once an ant moves from one city to another, pheromone is left in that edge so that other wants would sense it and follow that path. In order to choose the next city a prior probability q_0 is set and a random number q is generated. When the ant needs to go to the next city q is compared with q_0, if it is bigger than q_0 then the next city is selected by probability using the below formula (2), otherwise its selected according to the formula (3). Unless *i* was equal to *j* in that case the probability is set to 0. These are taken from [5]

$$P_{ij}^{k} = \tau_{ij}^{\alpha}\eta_{ij}^{\beta}$$

$$P_{ij}^{k} = \frac{\tau_{ij}^{\alpha}\eta_{ij}^{\beta}}{\sum \tau_{ij}^{\alpha}(\tau)\eta_{ij}^{\beta}}$$

After the next city has been decided, the pheromone matrix is updated using the local pheromone update strategy. In the local pheromone update the path that ant has just travelled is updated with a new pheromone level. The new pheromone level is calculated using the below formula (4). Delta is the local evaporation rate (0 < delta < 1), N is the number of cities and L_{nn} is calculated by using the Nearest neighbour algorithm. The same is done to the path between the first city and the last city since this is a symmetric graph.

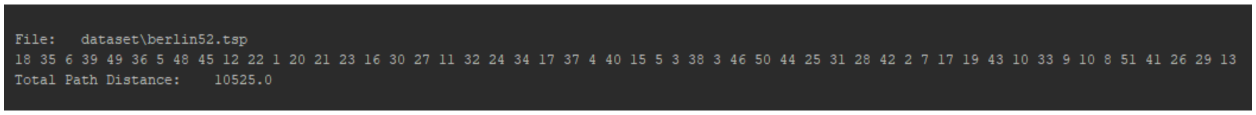
$$\tau_{ij} = (1-\delta)\,\tau_{ij} + \frac{\delta}{N \cdot L_{nn}}$$

Nearest neighbour algorithm: A random city is initially selected, the city with the shortest distance from the current city is selected as the next city. The total distance after visiting all the cities is the Lnn.

Once the local pheromone update is finished, the path distance of every ant is calculated, and the global pheromone update is performed. The ant with the shortest distance is found and has its path’s pheromone increased so that the ant colony would merge to one path.

$$\tau_{ij} = (1-\rho) \cdot \tau_{ij} + \frac{\rho}{L_{\text{ant}}}$$

Finally, these steps are iterated for how many times was defined, which by default is 100. Once a final list of ants is calculated, the best ant is found, and it path, and path distance is printed as it has the shortest distance.



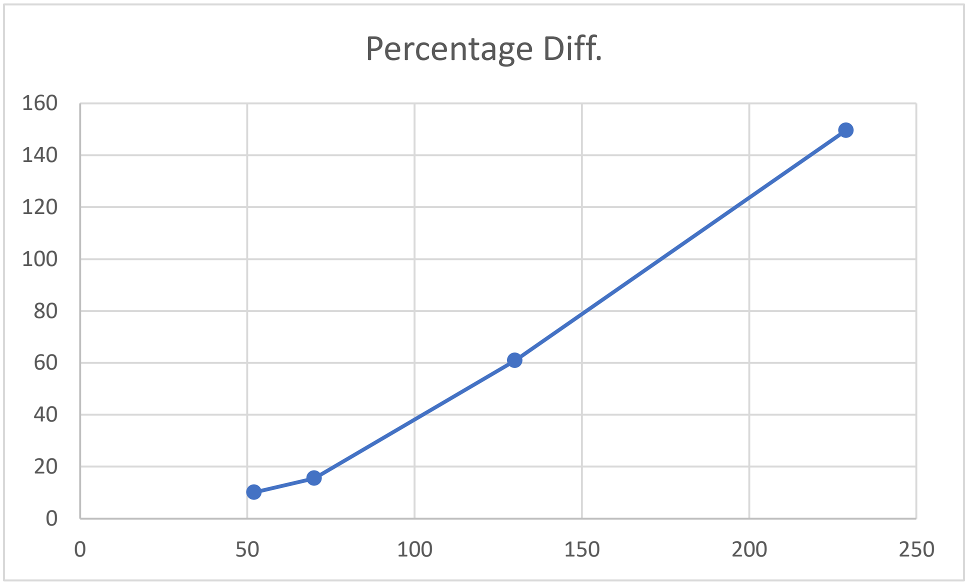
```
File:   dataset\berlin52.tsp
18 35 6 39 49 36 5 48 45 12 22 1 20 21 23 16 30 27 11 32 24 34 17 37 4 40 15 5 3 38 3 46 50 44 25 31 28 42 2 7 17 19 43 10 33 9 10 8 51 41 26 29 13
Total Path Distance:    10525.0
```

Evaluation and Statistics

The TSP that were tested were chosen randomly as though not to influence the outcome of the results. Each TSP was run 5 times each time recording the Best (Optimal), worst and the Deviance of the ACO path distance and put into the table below. The known optimal output was taken from [6][7][8]

Name	**Optimal Solution**	**Worst**	**TSPLIB Optimal**	**Deviation**	**Percentage Diff.**
Berlin52	8341	8665	7542	127	10.1
st70	789	821	675	11	15.5
ch130	12250	13759	6528	523	60.9
pr229	334365	338023	48191	2766	149.6

The deviation in Table 1 presents the discrepancy when comparing the best and worst results using this ACO. The Last column shows us the percentage difference between the known optimal solution and the best solution of this ACO implantation.



As can clearly be seen, the highest the number of cities in the TSP, the worse this ACO is preforming. This could be due to the fact that the ants are merging into the same path too quickly meaning that we would need more diversity in terms of path and ants choosing different ways. Though when *q* was Set to a higher value leading to more random ant choices over those led by pheromones it didn't always lead to better results.

One major improvement that could easily be done is that for every TSP, custom parameters are set. Simply by changing the alpha and beta values better results were seen, although they were not drastic.

Future Improvements

During the global pheromone updated, when the current best solution is preserved for certain iterations, better solutions can be found by rewarding the pheromone on current better path to accelerate convergence of ant colony, but this can lead to trapping the solution to a local optimum. This can be avoided using a penalty on the current optimal path to prevent from all the ants converging to a local optimum and hence finding a better overall solution. Instead of running all the TSP using one set of parameters, for each TSP the optimal parameters such as alpha and beta could be found in order to get a more optimal solution.

To Run the Application

Any TSP file which complies with the format of the *“tsp”* extension may be run by this application. All that must be done is that it must be placed in the “dataset” folder. By clicking the *ACO.bat* file, the application will open on the CMD and run all the TSP in the dataset folder and will also print the path and path distance for every TSP provided.

This application along with the documentations should also be available on GitHub using this link should something not work:
https://github.com/ZikoMic/TSPusingACO

References

[1] Potvin, JY. Ann Oper Res (1996) 63: 337. https://doi.org/10.1007/BF02125403

[2] D. Kaur and M. M. Murugappan, "Performance enhancement in solving Traveling Salesman Problem using hybrid genetic algorithm," NAFIPS 2008 - 2008 Annual Meeting of the North American Fuzzy Information Processing Society, New York City, NY, 2008, pp. 1-6.
URL:http://ieeexplore.ieee.org.ejournals.um.edu.mt/stamp/stamp.jsp?tp=&arnumber=4531202&isnumber=4531194

[3] O. ABDOUN, J. ABOUCHABAKA and C. TAJANI, "Analyzing the Performance of Mutation Operators to Solve the Travelling Salesman Problem", LaRIT Laboratory, Faculty of sciences, Ibn Tofail University, Kenitra, Morocco.

[4] A. Otman and A. Jaafar, "A Comparative Study of Adaptive Crossover Operators for Genetic Algorithms to Resolve the Traveling Salesman Problem", *International Journal of Computer Applications (0975 – 8887)*, vol. 31, no. 11, pp. 49-57, 2011. [Accessed 3 January 2020].

[5] Yu, Hui. (2014). Optimized Ant Colony Algorithm by Local Pheromone Update. TELKOMNIKA Indonesian Journal of Electrical Engineering. 12. 10.11591/telkomnika.v12i2.4211.

[6] St, Thomas & Dorigo, Marco. (1999). ACO Algorithms for the Traveling Salesman Problem.

[7] A novel two-stage hybrid swarm intelligence optimization algorithm and application - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/The-comparisons-for-att48_tbl1_257432450 [accessed 3 Jan 2020]

[8] TSPLIB: http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsplib.html

[9] T. Starkweather, S. McDaniel, K. Mathias, D. Whitley and C. Whitley, A comparison of genetic sequencing operators, in: Proc. 4th Int. Conf. on Genetic Algorithms (ICGA '91), University of California at San Diego, San Diego, CA (1991) pp. 69–76.

[10] Hybridizing PSM and RSM Operator for Solving NP-Complete Problems: Application to Travelling Salesman Problem - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/The-optimal-solution-of-Berlin52_fig2_221901574 [accessed 3 Jan 2020]