Solving Traveling Salesman Problem Using Ant Colony Optimization

SHEN Shuyang

Waseda University
shua@akane.waseda.jp

Abstract. Travelling salesman problem is one of the most famous combinatorial optimization problems. Which is usually optimized by Ant Colony Optimization (ACO) technique. ACO has very good search capability for optimization problems as well as effective variations like $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System, Elitist Ant System and Ant-Q. $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System and Ant-Q are implemented and compared in this report. Several improvements is applied to enhance the performances of the two variations. I finally achieved stable global optima with Ant-Q.

Keywords: parameter · optimization · local search

1 Introduction

Ant Colony Optimization (ACO) is a meta-heuristic and successful technique in the field of swarm intelligence. This technique is used for many applications especially problems that belong to the combinatorial optimization. In this report, the variations of ACO are applied to solve traveling salesman problem (TSP). Although ACO has a powerful capacity to find out solutions to combinatorial optimization problems, it has the problems of stagnation, premature convergence and the convergence speed of ACO is always slow. I developed a general understanding of ACO optimization methods and techniques based on the survey written by Stützle et al [3]. In which some papers proposed effective mechanism [1, 2, 4, 5] to address these problems. I implemented these algorithm and made some novel improvements to improve the efficiency and effectiveness of ACO. Finally achieving stable global optima with Ant-Q.

2 Problem Formulation and Methodology

This section introduces two ACO algorithm ($\mathcal{MAX}$-$\mathcal{MIN}$ Ant System and Ant-Q) I implemented. Despite the differences between these two algorithm, they share some parameters with same meaning. And thus I make the definitions on the common parameters of ACO for TSP here. Given a set of n cities, and for each pair of cities d_{ij} is the distance from city i to j. TSP is stated as the problem of finding a minimal length closed tour that visits each city once. An instance of the TSP is given by a graph (N, E), where N, $|N| = n$, is the set of cities and

E is the set of edges between cities (a fully connected graph in the Euclidean TSP). In each iteration of ACO algorithm, m ants are generated to optimize TSP till the result converges or the number of iteration reaches the limitation. The following part is specific explanation of the two ACO variations.

Full code have been uploaded to Github[1].

2.1 MMAS

Ants lay down pheromones directing each other to resources while exploring their environment. The simulated 'ants' similarly record their positions and the quality of their solutions, so that in later simulation iterations more ants locate better solutions. Pheromone matrix P record the pheromone intensity of $e \in E$ where $\tau_{ij} = \tau_{ji}$ is the pheromone intensity of the edge between city i and j. In ACO, pheromone intensity would evaporate over time, here the evaporating facor is ρ. $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System uses a rather simple mechanism for limiting the strengths of the pheromone trails to effectively avoids premature convergence of the search. Let τ_{max} and τ_{min} be the upper and lower bound of pheromone intensity, the complete problem formulation of MMAS can be stated as follows.

- Initialize pheromone matrix.
- Start iteration. Let m ants walk through the random chosen tour and record the length of the tour.
- Update pheromone matrix after all the ants finished the tour.
- Repeat from step 2 till the result converges or the number of iteration reaches the limitation

Now there are still two issues remaining in the steps above: 1) How to choose the tour path to converge fast and good. 2) How to appropriately update pheromone matrix.

For the former issue, heuristic factor η is introduced to inspire ant to choose city node in N, where $\eta_{ij} = \eta_{ji}$ is usually a function of d_{ij}. Here I use the function $\eta_{ij} = \frac{1}{d_{ij}}$. And thus a transfer matrix can be generated with heuristic matrix and pheromone matrix as **Equation 1**, where α and β are the emphasises on pheromone intensity and heuristic factor respectively.

$$t_{ij} = \tau_{ij}^{\alpha} \eta_{ij}^{\beta} \tag{1}$$

Every time ant tries to walk from city i to j where $j \in N_{allowed}$ and $N_{allowed}$ is a set of the available cities (have not been visited) for city i to transfer to. Random selection satisfies transition probability in **Equation 2**.

$$p_{ij} = \frac{t_{ij}}{\sum_{z \in N_{allowed}} t_{iz}} \tag{2}$$

In code, *numpy.random.choice*() is utilized to perform random selection.

[1] `https://github.com/Humbornjo/ACO-for-TSP`

The latter is much more complicated though. Stützle et al [3,4] explored various pheromone update methods. The classic pheromone update method exploits the information of every ant in the iteration. The tour traveled by ant $k \in [1, m]$ in one iteration contributes $\Delta\tau^k$ as is showed in **Equation 3**. Where Q is a constant quantity for influencing the trail strength and L_k is the length.

$$\Delta\tau_{ij}^k = \frac{Q}{L_k} \tag{3}$$

And thus the trail strengths are updated according to the formula

$$\tau_{ij}^{new} = (1-\rho)\tau_{ij}^{old} + \sum_{k=1}^{m} \Delta\tau_{ij}^k \tag{4}$$

Besides, I used a bonus global to enlarge the weight assigned to global best. Thus the final update formula is showed by **Equation 5**

$$\tau_{ij}^{new} = (1-\rho)\tau_{ij}^{old} + \sum_{k=1}^{m} \Delta\tau_{ij}^k + l\frac{Q}{L_{best}} \tag{5}$$

Where L_{best} is the global best tour length and l is a function of n ($r = \lfloor \frac{n}{20} \rfloor$ in the code).

I adopt the classic pheromone update method. And the other methods will be briefly introduced here. Elitist strategy was proposed that is essentially based on additionally reinforcing the arcs belonging to the best tour found in the run of the algorithm. Which only This is achieved by adding a quantity $x\frac{Q}{L_{best}}$ to all arcs belonging to the best-so-far tour whenever the trails are updated. x is the number of elitist ants and L_{best} is the length of the best found tour. Similarly, L_{best} can be explained as the the best found tour in the current iteration.

2.2 Ant-Q

Different from the traditional ACO algorithm like MMAS, Ant-Q introduces local updated rule inspired by Q-learning and the pheromone notion is dropped to be replaced by Ant-Q values μ instead. The goal of Ant-Q is to learn Ant-Q values such that the discovery of good solutions is favored in probability. Ant-Q values express how useful it is to move to node j from the current node i. The complete problem formulation of Ant-Q can be stated as follows.

- Initialize pheromone matrix.
- Start iteration. Let m ants walk through the random chosen tour and update Ant-Q value as an ant finished the tour (Local update).
- Execute global update after all the ants finished the tour.
- Repeat from step 2 till the result converges or the number of iteration reaches the limitation

Now there are still two issues remaining in the steps above as is mentioned in **Section 2.1**: 1) How to choose the tour path to converge fast and good. 2) How to appropriately update pheromone matrix.

For the former issue, heuristic factor η is introduced to inspire ant to choose city node in N, where $\eta_{ij} = \eta_{ji}$ is usually a function of d_{ij}. Similarly, function $\eta_{ij} = \frac{1}{d_{ij}}$ is applied. And thus a transfer matrix can be generated with heuristic matrix and pheromone matrix as **Equation 6**, where α and β are the emphasises on Ant-Q value and heuristic factor respectively.

$$t_{ij} = \mu_{ij}^{\alpha} \eta_{ij}^{\beta} \tag{6}$$

Every time ant tries to walk from city i to j where $j \in N_{allowed}$ and $N_{allowed}$ is a set of the available cities (have not been visited) for city i to transfer to. Random selection satisfies transition probability in **Equation 7**. (Ant-Q proposed in the paper used another casting transition rule, I didn't use it because of the bad result)

$$p_{ij} = \frac{t_{ij}}{\sum_{z \in N_{allowed}} t_{iz}} \tag{7}$$

In code, *numpy.random.choice*() is utilized to perform random selection.

The latter issue can be defined as one formula for both local and global update in **Equation 8**.

$$\mu_{ij}^{new} = (1 - \rho)\mu_{ij}^{old} + \rho(\Delta\mu_{ij} + \gamma \max_{u \in \mathcal{N}_j^k} \{\mu_{ju}^{old}\}) \tag{8}$$

Where $\mathcal{N}_j^k$ is the neighbor cities of city j in the trail of ant k. In local update, $\Delta\mu_{ij}$ is always 0 while $\Delta\mu_{ij} = \frac{Q}{L_{best}}$ is applied in global update. L_{best} is either the iteration-best or global-best and Q is a constant quantity for influencing the trail strength.

If we want the elite ant to lead the rest of the ants, we should use small γ and large Q so that large weight can be assigned to the best trail. γ and Q are determined with a lot of experiments. ($Q = 100$ and $\gamma = 0.04$ is finally determined)

3 Local Search

Local search is a heuristic method for solving computationally hard optimization problems. Local search can be used on problems that can be formulated as finding a solution maximizing a criterion among a number of candidate solutions. Local search algorithms move from solution to solution in the space of candidate solutions (the search space) by applying local changes, until a solution deemed optimal is found or a time bound is elapsed.

In optimization, 2-opt is a simple local search algorithm for solving the traveling salesman problem [2–5]. It exchange every possible edge pairs in a tour and to iteratively find a better solution. In my code, 2-opt is executed after all the

ants in the iteration finished their tours. And 2-opt do iterative optimization to the iteration best tour in the end of every iteration. 2-opt costs but makes ACO converges very fast so that it can balance the total cost with some convergence detecting mechanisms.

It should be noted that 3-opt normally performs better than 2-opt. However, its complexity becomes $O(n^3)$ correspondingly. In my implementation, a trade-off was made by utilizing 2-opt considering the scale of problem.

4 Other Improvements

This section introduced some improvement on the two ACO algorithm. (Aim to improve the efficiency)

4.1 Dumb for MMAS

dumb is used to depicts how good the latest global best might be. Every time the global best is updated, dumb will be reset to 1. And if the current iteration can not find a new global best, dumb would increase to assign less weight to the global best because we have reasons to believe that the current best trail has a large chance to be a bad local optima other than a better solution towards global optima. The weight should decrease slowly at first, and will decrease faster and faster over time. Thus $\sqrt{\frac{1}{dumb}}$ is finally adopted. Which makes the update strategy

$$\tau_{ij}^{new} = (1-\rho)\tau_{ij}^{old} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k} + l\sqrt{\frac{1}{dumb}}\frac{Q}{L_{best}} \tag{9}$$

4.2 Improved Transition Rule for Ant-Q

Improved Transition Rule use a hyperparameter r_0 and a casted value r. The next city node is fixed to the city with the highest transfer probability as long as a certain comparative relationship is satisfied between r_0 and r.

Time cost is reduced with the utilization of Improved Transition Rule because you don't need to cast and select a next node for some circumstances. But the r_0 selected by $U(0,1)$ can really cause worse result. (Maybe I didn't implemented the algorithm appropriately)

Normal distribution is used instead in my code. Value r is generated by the distribution $N(p,1)$ where p is the corresponding value in transfer matrix. Which means a transfer with high probability is easier to be fixed. (when $r > r_0$, fix the city with the highest transfer probability as the next city to be visited)

4.3 Convergence Mechanism for Both

Let I_{last} be the number of the latest iteration where the global best is updated, $I_{current}$ be the number of the current iteration and c be a counter.

When $\frac{I_{last}}{I_{current}} < \frac{1}{2}$, c starts counting by add one in each following iteration as long as I_{last} doesn't change (c would be reset to 1 if I_{last} changes).

Iterating would stop if $c > c_0$, where c_0 is a function of the initially set iteration number g. ($c_0 = \lfloor \frac{g}{10} \rfloor$ is applied in my code).

$\frac{I_{last}}{I_{current}} < \frac{1}{2}$ prevents algorithm from stopping looking for the global optimum when there might be a chance. While $c > c_0$ prevents algorithm from stopping looking for the global optimum when the iteration number is too small. (*e.g.* For a tsp problem, set the iteration number 200. And it finds global optima in iteration 3. Which means it will stop at iteration 26=3+3+20)

5 Results

Every Box-plot in **Result** runs the code for 10 times. The unit of time is seconds. As for dataset, berlin52.tsp is used.

5.1 Parameters

Figure 1 and **Figure 2** shows how I determine alpha and beta in Ant-Q and MMAS. Here only the results of Ant-Q is showed. And I choose $\alpha = 1,\ \beta = 4$ as the final configuration of Ant-Q. (Local search is not used to reveal the effect of alpha and beta as clear as possible)

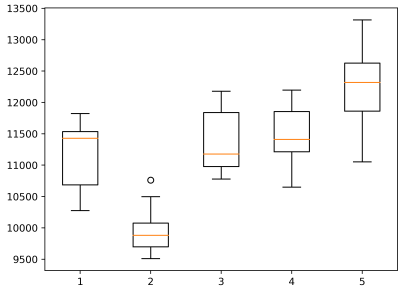

Fig. 1. $\beta = 1,\ \alpha \in [1,\ 5]$

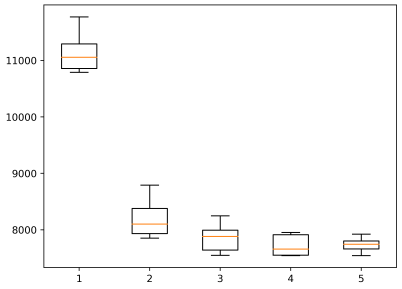

Fig. 2. $\alpha = 1,\ \beta \in [1,\ 5]$

5.2 2-opt

In the paper which proposed Ant-Q [1], iteration-best update is proved to be a better to find global optima. I confirmed it with that in **Figure 3**.

However, after applying 2-opt, global-best update becomes much better as showed in **Figure 4**. It is obvious that Ant-Q with 2-opt can always find the global optima without a miss.

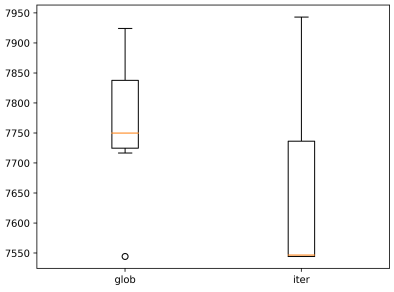

Fig. 3. Ant-Q with different update strategy (without 2-opt)

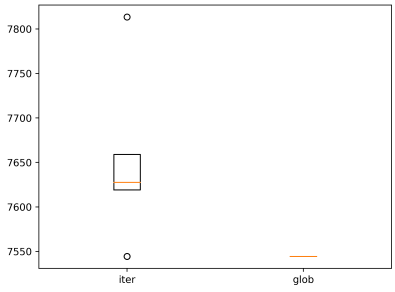

Fig. 4. Ant-Q with different update strategy (with 2-opt)

5.3 Time cost

Ant-Q costs less time compared with MMAS as is showed in **Figure 5**. It should be noted that the number of ant in Ant-Q and MMAS doesn't equal. Ant-Q generates less ant in one iteration. But MMAS still can not beat Ant-Q by result. Which indicates that Ant-Q should cost less time to obtain the global optima.

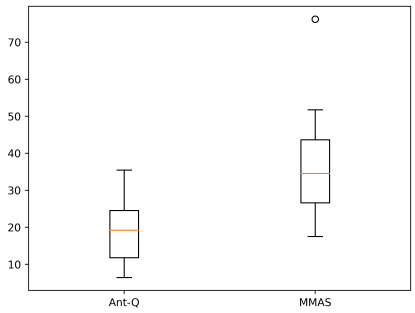

Fig. 5. Time cost of Ant-Q and MMAS

5.4 Evaluation on Improvements

Dumb for MMAS

The utilization of *dumb* slightly reduces the time cost but not apparent enough. Maybe the problem with large scale could reveal the effectiveness of

this improvement clearly. I didn't proceed further experiment on this improvement in this report.

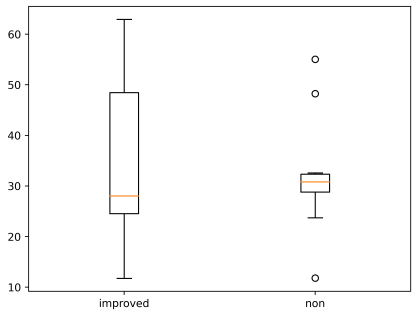

Fig. 6. Time cost with *dumb* in MMAS

Improved Transition Rule for Ant-Q

Fix the city to be visited could considerably reduces the time cost according to **Figure 7**. But the result will become unstable, and it will not find the global optima every time as the original Ant-Q .

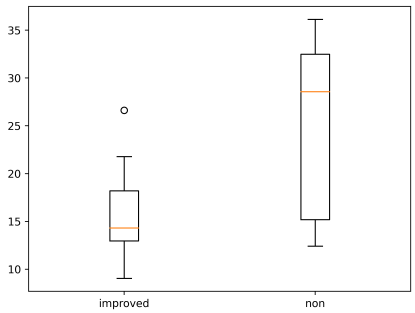

Fig. 7. Time cost with Improved Transition Rule in Ant-Q

Convergence Mechanism for Both

Convergence mechanism could reduce the time cost while keeping the result as good as possible. In the left figure in **Figure 8**, Ant-Q stops iterating before

iteration 40 and MMAS stops iterating before iteration 100, which saves a lot of invalid calculations.

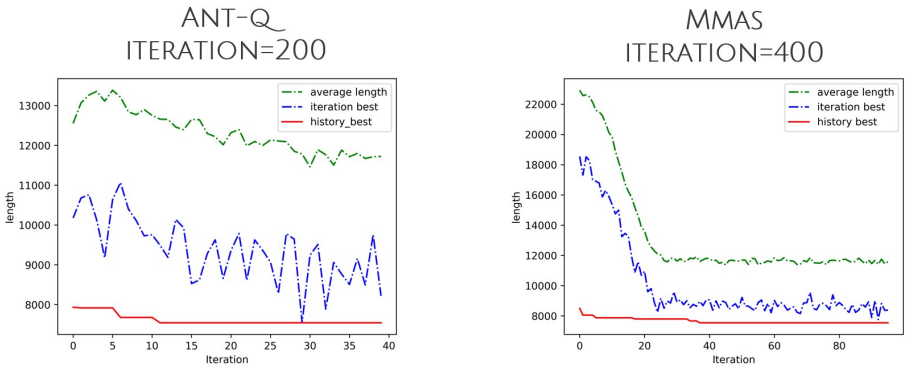

Fig. 8. Convergence mechanism in Ant-Q and MMAS

5.5 Solution

In the code, Improved Transition Rule for Ant-Q is not used. *dumb* is used in MMAS, and convergence mechanism is applied in the results from **Figure 3** to **Figure 8**. $\rho = 0.1$ is set for the results in this report.

$Q = 100$, $\gamma = 0.04$ and $m = n$ is determined in the result of **Figure 9**. Which depicts the output optimized by Ant-Q, the relatively better ACO algorithm.

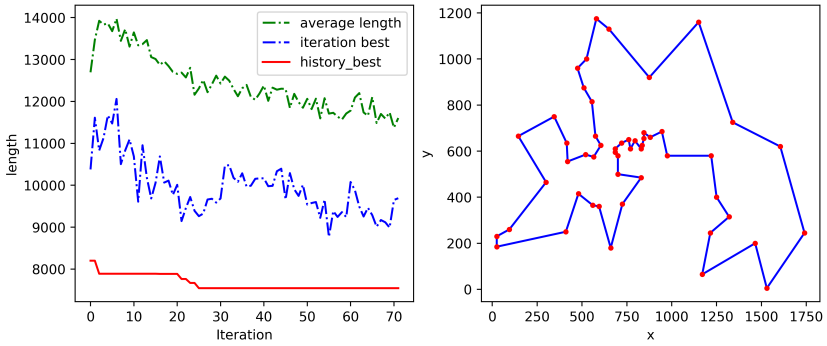

Fig. 9. Ant-Q

6 Conclusion

Local search can improve both the output and the time cost of ACO algorithm. But the specific method need to be carefully chosen according to the scale of TSP and the computational resources.

Based on my experiment, Ant-Q is apparently superior to MMAS both on time cost and result. So I chose Ant-Q with the improvements as the algorithm to optimize TSP.

For further research, I think the optimization of local search should be a good direction. since there are considerably amount of regions in local search which do not seem to have the chance to improve the solution. If we can reduce these invalid calculation, the efficiency of ACO for TSP is bound to be enhanced correspondingly.

References

1. Luca M Gambardella and Marco Dorigo. Ant-q: A reinforcement learning approach to the traveling salesman problem. In *Machine learning proceedings 1995*, pages 252–260. Elsevier, 1995.
2. Shen Lin. Computer solutions of the traveling salesman problem. *Bell System Technical Journal*, 44(10):2245–2269, 1965.
3. Thomas Stützle, Marco Dorigo, et al. Aco algorithms for the traveling salesman problem. *Evolutionary algorithms in engineering and computer science*, 4:163–183, 1999.
4. Thomas Stützle and Holger H Hoos. Improving the ant system: A detailed report on the max–min ant system. *FG Intellektik, FB Informatik, TU Darmstadt, Germany, Tech. Rep. AIDA–96–12*, 1996.
5. Thomas Stützle and Holger H Hoos. Max–min ant system. *Future generation computer systems*, 16(8):889–914, 2000.