MTSP with minmax objective based on ACA with limited cities and cross avoidance

1. Problem Statement
   1. Problem definition and notation

Consider a complete graph  on a set of cities, in which V is set of vertexes and A represents set of arcs, and a cost matrix, who indicates the cost( distance ) associated with each of the arc. The cost matrix can be either symmetric or asymmetric and we are going to partition the vertex set such that , in witch 1 is the depot node and  is the customer nodes. At the very beginning, there are m salesmen located at the depot node 1.

A minmax MTSP problem tries to find tours for all the salesmen such that all the costumer nodes are visited exactly once, and the tour with the maximum cost is minimized.

* 1. Problem formulation

Formulation for MMTSP presented by Soheil Ghafurian and Nikbakhsh Javadian handles fixed destination MTSP with multiple depots. We adopt this formulation and adjust to deal with our objectives.

First, a binary parameter is defined to indicate whether a link is in kth tour



For any salesman,  is number of customer nodes visited till  node, including the  node. We introduce  and  as an interval of number of visited cities, where  is the lower bound and  represents the higher bound. Thus for every customer node , , and for the last customer node in a tour, .



Note that , , and  is the number of salesmen.

In this formulation, constraint ensures that exactly  salesmen depart from depot point 1, constraint (2) makes sure that exactly  salesmen arrive at depot vertex 1. constraint (3) ensures that every costumer node is visited once and constraint (4) ensures route continuity around customer node. Constraint (5) and (6) impose higher limits and lower limits on number of costumer nodes in a route, respectively. In addition, route with just one costumer node is avoided. Constraint (8) eliminates sub-tour within costumer nodes. A sub-tour is a route with no depot node as its starting or ending point. If constraint (8) is abandoned, solution provided by left constraints may contain sub-tours.

1. Solution construction
   1. ACA with limited cities

Initially, there are  artificial ants. For each of the ants, it departs from the depot city and continuously choose one of the allowed cities to visit until it returns to the depot city and completes a tour. Then it will start a new tour until it finishes  tours.

4.1.1 Transition probability

Transition probability by M. Dorgio, V. Maniezzo is adopted here. However, some strategies is imposed on  list, which contains cities that cannot be chosen by ants. Currently, the ant is visiting city , the probability of it visiting the  city can be obtained by following formulation:



Where  and indicates the list of cities that the ant is currently at city  can visit.  and  control the relative importance of amount of trail verse greedy of the algorithm.

To choose next city, following strategies should be considered in order to construct valid solutions:

1. Cases when the ant must return to depot city and finish the tour
2. The ant has visited  ants
3. Number of costumer cities left for per agent falls below  if the ant keeps visiting other cities
4. Cases when the depot city must be in 
5. The ant has not visited  cities
6. The ant is handling its last tour
7. Number of costumer cities left for per agent exceeds  if the ant returns to depot city and finishes the tour.

4.1.2 Trail update

At the end of a cycle, amount of trail on the route should be updated. Let  be the amount of trail on the edge  at the th cycle. Thus formation of updating trail could be,



Where  is the quantity of trail left on the edge  by the th ant between th cycle and  cycle.  can be calculated by



Where  is the objective function of the solution produced by the th ant. That is to say,  is the largest sub-tour produced by the th ant.

4.1.3 The algorithm

The algorithm will run repetitively, mostly  cycles. For each of the cycle,  ants will construct  solutions using the method mentioned above. At the end of the cycle, quantity of the trail will be updated, using the solution produced.

The algorithm will stop after finishing  cycles or no improvement no improvement found in the best solution of current cycle compared to the global best solution.

• Initialize 

• Do:

* 1. Construct  solutions using method mentioned above
  2. Update  using the solutions

• While( This is improve on the best solution and number of cycles if no more than 

• Return the best solution.

* 1. Optimization with cross avoidance within a sub-tour

The following two properties make it available to optimize a sub-tour

Property 1) Local optimization leads to global optimization

As is seen in figure 1, length of the tour can be calculated by following formulation:



where if we can optimize , we can optimize 

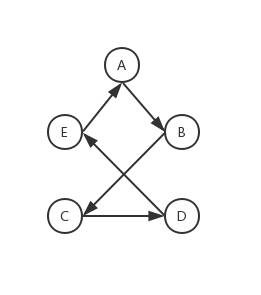


Fig 1. Case of a tour

Property 2) In a quadrilateral, length sum of two opposite edges are shorter than length sum of two diagonal edges.

As is seen in figure 2, we guarantee that flowing formulations are valid. This tells that if there is a cross in a tour, we can adjust the cross to shorten the tour.



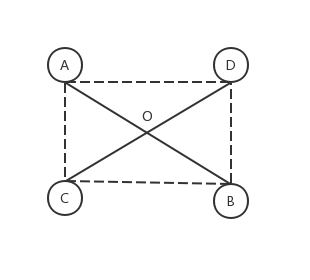


Fig 2. Case of a quadrilateral

* + 1. Cross detection

Name the four nodes whose edges cross with each other , and cross point , as is seen in figure 2. Therefor, the following formulations is guaranteed:



Where,



If the equations have solution, then . Thus we can get the solution by the following formation:



If there is a cross, following conditions should be satisfied



Or,



Or,



* + 1. Cross elimination

As is seen in figure 3, at the very beginning, the solution is . However, after eliminating the cross, it becomes . As is apparently observed, the procedure to eliminate the cross is to reverse the route between node  and , which begins and ends the cross, respectively. Thus we suppose that to reverse the route between the beginning and end point of the cross will work out to eliminate the cross.

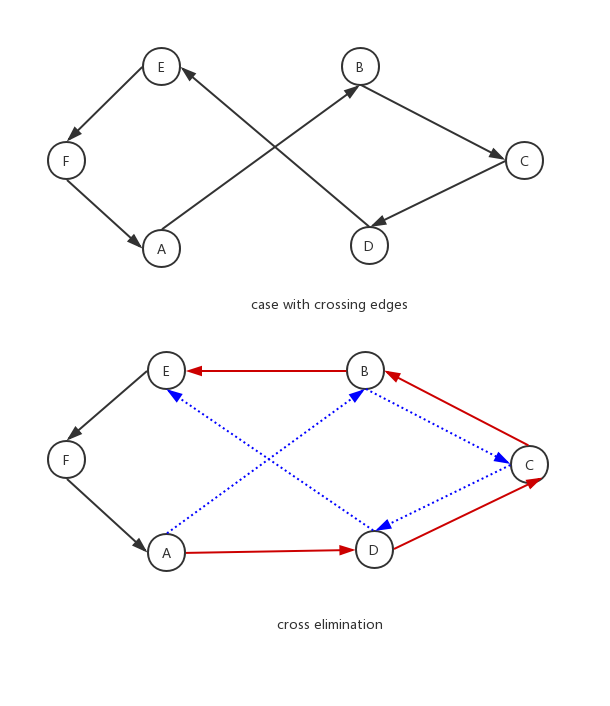


Fig 3. Eliminate cross

* + 1. The algorithm

• while :

1. Check if there is cross
2. If there is cross, eliminate cross
3. If no cross is detected, break loop.
4. Experimental results
   1. Generating problems

There is benchmark data for the Single-Depot Multiple Traveling Salesman Problem (multiple-TSP). They are eil51, berlin52, eil76, and rat99. However, these benchmarks are not always giving globally optimal solutions, which is not very suitable for our criteria to test the performance of our algorithm. To the best knowledge of the author, proportion of the best solution provided by our algorithm to the optimal solution by exact algorithm is proper to test a performance of an algorithm.

We use benchmark data eil51 to generate our problems. In more details, we choose n cities from eil51 randomly. For each , five problems is generated.

* 1. Lingo 17

As a well known effective application for linear, non-linear and integer optimization, Lingo 17 is powerful to use exact algorithms to give global solutions. However, it shares all the downsides of exact algorithms. That is to say, it cannot handle NP-hard problems in reasonable time.

Take one of our problem for example, it takes days to handle a MTSP with 3 salesmen and 20 cities. Thus we make n no more than 16.

Throughout the experimental results section, we use Lingo 17 to calculate the optimal solution of our problems.

* 1. Parameter tuning

Finding the best solution using Ant colony algorithm is very hard job due to parameter setting. That is to say, slight change in parameters will lead to hug difference to the performance of the algorithm. Generally, experimental approximation is used to find the best value of parameters.

As is suggested in Application of MATLAB in math modeling, we set  As is shown in Table 1 for the four problem of , there is a liaison between the time consumption and the number of sub-tours. That is to say, the time consumption increases dramatically along with the number of sub-tours. This mainly results from sharply increment in binary parameter . Thus we have to set k=2 in order to get answers in reasonable time.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| k | Problem 1 | Problem 2 | Problem 3 | Problem 4 | Average time |
| 1 | 0.14 | 0.66 | 0.67 | 0.38 | 0.46 |
| 2 | 1.23 | 10.98 | 15.94 | 7.39 | 8.88 |
| 3 | 9.83 | 98.98 | 140.4 | 114 | 90.80 |
| 4 | 67.91 | 112.28 | 82.98 | 129.84 | 98.25 |

Table 1. CPU times in seconds for problems of n = 10 with different value of k

* + 1. Interval of the number of cities in a sub-tour

As is known in minmax problem, it is trying to balance the workload of every tour. Thus we suppose the number of cities of a sub-tour is distributed around mean value of n. In order to find the liaison between the interval of number of cities of a sub-tour and mean value of n, an implementation with  and different intervals was run 10 times and the best 5 solutions are registered.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| interval | Problem 1 | Problem 2 | Problem 3 | Problem 4 | Average proportion |
| (5, 6) | 0.979 | 0.997 | 0.994 | 0.955 | 0.98125 |
| (4, 7) | 0.979 | 1 | 0.996 | 0.957 | 0.983 |
| (3, 8) | 0.97 | 1 | 0.996 | 0.936 | 0.9755 |

Table 1, average proportion of each interval for each problem for n = 12

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| interval | Problem 1 | Problem 2 | Problem 3 | Problem 4 | Average proportion |
| (6, 7) | 0.995 | 0.968 | 1 | 1 | 0.99075 |
| (5, 8) | 0.984 | 0.965 | 1 | 0.996 | 0.98625 |
| (4, 9) | 0.978 | 0.935 | 1 | 0.982 | 0.97375 |

Table 2, average proportion of each interval for each problem for n = 14

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| interval | Problem 1 | Problem 2 | Problem 3 | Problem 4 | Average proportion |
| (7, 8) | 0.959 | 0.988 | 0.973 | 0.939 | 0.96475 |
| (6, 9) | 1 | 0.983 | 0.978 | 0.964 | 0.98125 |
| (5, 10) | 0.983 | 0.965 | 0.976 | 0.975 | 0.97475 |

Table 3, average proportion of each interval for each problem for n = 16

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| interval | Problem 1 | Problem 2 | Problem 3 | Problem 4 | Average proportion |
| (8, 9) | 0.926 | 1 | 0.983 | 0.93 | 0.95975 |
| (7, 10) | 0.938 | 0.977 | 0.966 | 0.921 | 0.9505 |
| (6, 11) | 0.949 | 0.978 | 0.972 | 0.919 | 0.9545 |

Table 4, average proportion of each interval for each problem for n = 18

As is seen in the tables, we can have a vision of setting the interval for each n, which is recorded in table 5.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| n | 12 | 14 | 16 | 18 |
| interval | (4,7) | (6,7) | (6,9) | (8,9) |

Table 5, best interval for each value of n

* + 1. Number of the ants

In ant colony algorithm, number of the ants is the size of a group. A big group may accelerate to converge to local optimization.While a small amount of ants may leads to a long time to run the program. So how to combine the time consumption and the objective should be considered. Thus we register average proportion of the solution by ant colony algorithm to that by Lingo 11.0 and time consumption in Table 6, and Table 7. In our implementation, ant colony algorithm is run 10 times on each problem of different n and the best 5 solutions are registered. The interval of cities is set according to the result of 5.3.1 and the other parameters are ,

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| antnum | N=12 | N=14 | N=16 | N=18 | Average proportion |
| 1 | 0.91525 | 0.941 | 0.91575 | 0.87725 | 0.9123125 |
| 2 | 0.94425 | 0.97425 | 0.9265 | 0.9065 | 0.937875 |
| 3 | 0.96275 | 0.98 | 0.94125 | 0.93525 | 0.9548125 |
| 4 | 0.97075 | 0.986 | 0.936 | 0.928 | 0.9551875 |
| 5 | 0.969 | 0.98675 | 0.94725 | 0.93525 | 0.9595625 |
| 6 | 0.9725 | 0.97975 | 0.962 | 0.94775 | 0.9655 |
| 7 | 0.96925 | 0.98575 | 0.968 | 0.956 | 0.96975 |
| 8 | 0.95525 | 0.987 | 0.97375 | 0.94975 | 0.9664375 |
| 9 | 0.974 | 0.98575 | 0.95225 | 0.95125 | 0.9658125 |
| 10 | 0.96375 | 0.987 | 0.96725 | 0.93875 | 0.9641875 |
| 11 | 0.97725 | 0.98525 | 0.97225 | 0.95575 | 0.972625 |
| 12 | 0.9755 | 0.986 | 0.96025 | 0.955 | 0.9691875 |
| 13 | 0.9775 | 0.98875 | 0.96725 | 0.9515 | 0.97125 |
| 14 | 0.97375 | 0.987 | 0.97625 | 0.96225 | 0.9748125 |
| 15 | 0.976 | 0.9885 | 0.975 | 0.9565 | 0.974 |
| 16 | 0.98125 | 0.98725 | 0.9695 | 0.9585 | 0.974125 |
| 17 | 0.982 | 0.987 | 0.97525 | 0.95575 | 0.975 |
| 18 | 0.97475 | 0.9875 | 0.98175 | 0.95375 | 0.9744375 |
| 19 | 0.97525 | 0.9865 | 0.97425 | 0.96275 | 0.9746875 |
| 20 | 0.9775 | 0.9885 | 0.9785 | 0.956 | 0.975125 |
| 21 | 0.97275 | 0.98725 | 0.9765 | 0.9575 | 0.9735 |
| 22 | 0.978 | 0.9885 | 0.977 | 0.971 | 0.978625 |
| 23 | 0.977 | 0.98925 | 0.9745 | 0.953 | 0.9734375 |
| 24 | 0.98225 | 0.9885 | 0.98625 | 0.96925 | 0.9815625 |
| 25 | 0.97675 | 0.98925 | 0.9865 | 0.95475 | 0.9768125 |

Table 5, proportion of best solution to optimial solution by with different antnum

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| antnum | N=12 | N=14 | N=16 | N=18 | Average time |
| 1 | 0.057 | 0.05975 | 0.0685 | 0.07975 | 0.06625 |
| 2 | 0.08775 | 0.109 | 0.12475 | 0.144 | 0.116375 |
| 3 | 0.1275 | 0.1585 | 0.18025 | 0.20725 | 0.168375 |
| 4 | 0.1675 | 0.208 | 0.23775 | 0.272 | 0.2213125 |
| 5 | 0.20625 | 0.2575 | 0.2945 | 0.33825 | 0.274125 |
| 6 | 0.246 | 0.3155 | 0.3485 | 0.40275 | 0.3281875 |
| 7 | 0.2975 | 0.3885 | 0.40475 | 0.468 | 0.3896875 |
| 8 | 0.4565 | 0.41875 | 0.461 | 0.5315 | 0.4669375 |
| 9 | 0.52075 | 0.456 | 0.51675 | 0.59725 | 0.5226875 |
| 10 | 0.49775 | 0.53325 | 0.56875 | 0.693 | 0.5731875 |
| 11 | 0.50075 | 0.5675 | 0.6275 | 0.74725 | 0.61075 |
| 12 | 0.62225 | 0.6055 | 0.68125 | 0.84 | 0.68725 |
| 13 | 0.5665 | 0.616 | 0.74125 | 0.91675 | 0.710125 |
| 14 | 0.633 | 0.6645 | 0.7945 | 0.94725 | 0.7598125 |
| 15 | 0.70775 | 0.713 | 0.84725 | 0.98575 | 0.8134375 |
| 16 | 0.75375 | 0.75725 | 0.906 | 1.04625 | 0.8658125 |
| 17 | 0.7515 | 0.83775 | 0.96225 | 1.1545 | 0.9265 |
| 18 | 0.72525 | 0.895 | 1.04075 | 1.47575 | 1.0341875 |
| 19 | 0.763 | 0.91625 | 1.13675 | 1.589 | 1.10125 |
| 20 | 0.8025 | 1.01425 | 1.20175 | 1.61925 | 1.1594375 |
| 21 | 0.843 | 1.16225 | 1.23875 | 1.71625 | 1.2400625 |
| 22 | 0.87225 | 1.116 | 1.29475 | 1.782 | 1.26625 |
| 23 | 0.921 | 1.11325 | 1.3775 | 1.86475 | 1.319125 |
| 24 | 1.12275 | 1.1275 | 1.428 | 2.05575 | 1.4335 |
| 25 | 1.08975 | 1.17625 | 1.4985 | 2.23075 | 1.4988125 |

Table 6, proportion of best solution to optimial solution by with different antnum

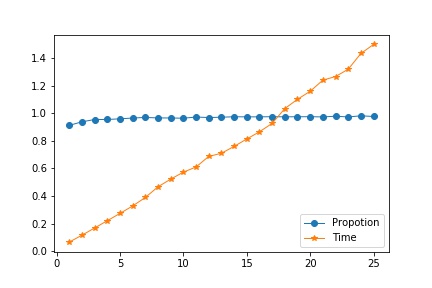


Fig 1. Tend of proportion and time by different antnum

As is seen in figure 1, the major improvement occur before 10 for all the problems. However, the time consumption of 10 is about twice to that of 7 and small improvement is seen between 7 and 10. Thus we spare the improvement after 7 in order to run programs faster. For the later implementations, antnum is 7.

* + 1. Initial amount of trial

It is expected that initial amount of trail will not affect the efficiency of the program due to other situations of ant colony algorithm. However, our case shows a difference. In order to find the best initial amount of trail, all the problems are run 10 times and the best 5 solution is registered. Parameters discussed above are set and . The result is shown in Table 7, table 8 and figure 2.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Initial trail | N=12 | N=14 | N=16 | N=18 | Average proportion |
| 1 | 0.97475 | 0.97725 | 0.95575 | 0.93575 | 0.967125 |
| 10 | 1 | 0.968 | 0.95325 | 0.9995 | 0.965625 |
| 100 | 0.9945 | 1 | 0.973 | 0.964 | 0.963375 |
| 1000 | 0.9185 | 0.997 | 0.952 | 0.885 | 0.9659375 |

Table 7, average proportion by different initial amount of trial

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Initial trail | N=12 | N=14 | N=16 | N=18 | Average proportion |
| 1 | 0.4065 | 0.43575 | 0.5225 | 0.61175 | 0.494125 |
| 10 | 0.32075 | 0.4275 | 0.488 | 0.6295 | 0.4664375 |
| 100 | 0.32425 | 0.4045 | 0.48 | 0.60075 | 0.452375 |
| 1000 | 0.357 | 0.4175 | 0.51625 | 0.59875 | 0.472375 |

Table 8, average time consumption by different initial amount of trial

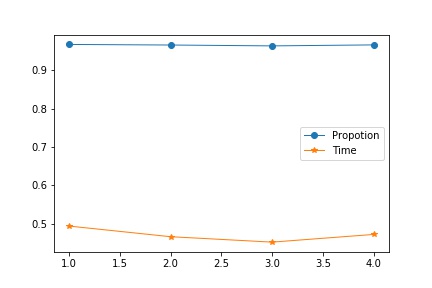


Fig 3. Trend of proportion and time by different initial amount of trail

As is shown in figure 3, it it apparent initial amount of trail will not affect the efficiency of the program, which ensures our assumption. Thus we arbitrarily set the initial amount of trail 1.

* + 1. Evaporation rate

It is expected that a small number of evaporation rate will lead to fast convergence to local optimization and a big number has good performance to find the global optimization. However, a big number of evaporation rate may consume a lot of time to converge. Thus how to combine this time and objective conflict should be considered. To find the best evaporation rate, all the problems are run 10 times by different p ranging from 0.1 to 0.9 increasing by 0.1 and best 5 solutions is registered. The result is shown in Table 9, table 10 and figure 4.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ρ | n12 | n14 | n16 | n18 | average |
| 0.1 | 0.9755 | 0.988 | 0.96525 | 0.95475 | 0.970875 |
| 0.2 | 0.97675 | 0.986 | 0.972 | 0.97 | 0.9761875 |
| 0.3 | 0.9755 | 0.988 | 0.97475 | 0.95775 | 0.974 |
| 0.4 | 0.97325 | 0.987 | 0.9605 | 0.955 | 0.9689375 |
| 0.5 | 0.97625 | 0.98825 | 0.96075 | 0.96 | 0.9713125 |
| 0.6 | 0.97525 | 0.98575 | 0.97125 | 0.941 | 0.9683125 |
| 0.7 | 0.972 | 0.98875 | 0.9645 | 0.95075 | 0.969 |
| 0.8 | 0.9645 | 0.98575 | 0.96325 | 0.953 | 0.966625 |
| 0.9 | 0.96525 | 0.98125 | 0.965 | 0.952 | 0.965875 |

Table 9. average proportion to the optimal solution with different ρ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pho | N=12 | N=14 | N=16 | N=18 | average |
| 0.1 | 0.33025 | 0.41975 | 0.486 | 0.57025 | 0.4515625 |
| 0.2 | 0.3055 | 0.4195 | 0.4895 | 0.57225 | 0.4466875 |
| 0.3 | 0.318 | 0.416 | 0.5055 | 0.61175 | 0.4628125 |
| 0.4 | 0.3315 | 0.407 | 0.5 | 0.597 | 0.458875 |
| 0.5 | 0.3615 | 0.4345 | 0.49 | 0.5825 | 0.467125 |
| 0.6 | 0.366 | 0.4425 | 0.4965 | 0.6685 | 0.493375 |
| 0.7 | 0.36325 | 0.445 | 0.5335 | 0.67775 | 0.504875 |
| 0.8 | 0.3675 | 0.41575 | 0.54325 | 0.6615 | 0.497 |
| 0.9 | 0.346 | 0.438 | 0.5055 | 0.61275 | 0.4755625 |

Table 10. average time in seconds with different ρ

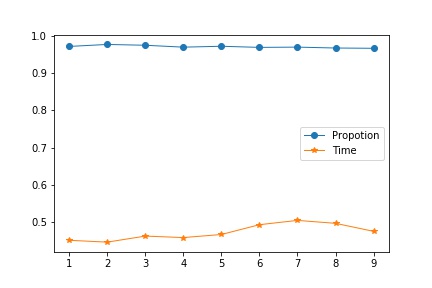


Fig. 4, Trend of average time and proportion

As seen in the figure and tables, it is apparent that our concern is right. As evaporate increase, the time consumption goes higher. Combine the time and objective, we can see ρ= 0.2 is a good choice. Thus for the next test, ρ= 0.2

* + 1. stopping criteria

It is well known that ant colony algorithm will converge at a time and the improvement of the later cycles is vain and will cost a lot of time. Thus how to get a relatively reasonable cyclenum is necessary. To find a stop criteria, we run all the problems 10 times for each and registered the best 5 solutions. Parameters except cyclenum are set according to the discussion above. The result can be seen in table 11, table 12, and figure 5.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| cyclenum | n12 | n14 | n16 | n18 | average |
| 1 | 0.931 | 0.93475 | 0.89125 | 0.88425 | 0.9103125 |
| 2 | 0.94975 | 0.97625 | 0.9305 | 0.88775 | 0.9360625 |
| 3 | 0.9375 | 0.983 | 0.94625 | 0.89675 | 0.940875 |
| 4 | 0.96225 | 0.982 | 0.924 | 0.927 | 0.9488125 |
| 5 | 0.955 | 0.986 | 0.9475 | 0.9155 | 0.951 |
| 6 | 0.96525 | 0.9885 | 0.947 | 0.92175 | 0.955625 |
| 7 | 0.96175 | 0.986 | 0.94625 | 0.91225 | 0.9515625 |
| 8 | 0.96725 | 0.9855 | 0.9495 | 0.92825 | 0.957625 |
| 9 | 0.96725 | 0.98625 | 0.951 | 0.94775 | 0.9630625 |
| 10 | 0.968 | 0.98725 | 0.94325 | 0.939 | 0.959375 |
| 11 | 0.9675 | 0.987 | 0.95175 | 0.935 | 0.9603125 |
| 12 | 0.971 | 0.986 | 0.95075 | 0.93125 | 0.95975 |
| 13 | 0.96775 | 0.98725 | 0.955 | 0.947 | 0.96425 |
| 14 | 0.97075 | 0.987 | 0.9455 | 0.93325 | 0.959125 |
| 15 | 0.967 | 0.986 | 0.9555 | 0.96 | 0.967125 |
| 16 | 0.9745 | 0.987 | 0.9545 | 0.95675 | 0.9681875 |
| 17 | 0.981 | 0.988 | 0.9605 | 0.94475 | 0.9685625 |
| 18 | 0.97 | 0.9885 | 0.95975 | 0.9555 | 0.9684375 |
| 19 | 0.9845 | 0.98825 | 0.96675 | 0.957 | 0.974125 |
| 20 | 0.97525 | 0.98725 | 0.95975 | 0.951 | 0.9683125 |
| 21 | 0.97475 | 0.98825 | 0.96525 | 0.944 | 0.9680625 |
| 22 | 0.9725 | 0.98975 | 0.96675 | 0.9625 | 0.972875 |
| 23 | 0.9805 | 0.988 | 0.96625 | 0.959 | 0.9734375 |
| 24 | 0.9815 | 0.986 | 0.96675 | 0.9655 | 0.9749375 |
| 25 | 0.98375 | 0.9895 | 0.9635 | 0.9615 | 0.9745625 |

Table 10 average proportion to the optimal solution by different cyclenum

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| cyclenum | n12 | n14 | n16 | n18 | average |
| 1 | 0.02325 | 0.031 | 0.03575 | 0.04325 | 0.0333125 |
| 2 | 0.03475 | 0.05175 | 0.06425 | 0.0665 | 0.0543125 |
| 3 | 0.0485 | 0.06925 | 0.08425 | 0.09625 | 0.0745625 |
| 4 | 0.0625 | 0.0945 | 0.117 | 0.12375 | 0.0994375 |
| 5 | 0.077 | 0.1175 | 0.13125 | 0.153 | 0.1196875 |
| 6 | 0.09875 | 0.13275 | 0.1675 | 0.1815 | 0.145125 |
| 7 | 0.12 | 0.15075 | 0.21125 | 0.197 | 0.16975 |
| 8 | 0.13725 | 0.1705 | 0.2495 | 0.21975 | 0.19425 |
| 9 | 0.15575 | 0.19575 | 0.2305 | 0.248 | 0.2075 |
| 10 | 0.166 | 0.219 | 0.24575 | 0.28875 | 0.229875 |
| 11 | 0.1795 | 0.25075 | 0.2795 | 0.29075 | 0.250125 |
| 12 | 0.2005 | 0.246 | 0.2945 | 0.31975 | 0.2651875 |
| 13 | 0.2215 | 0.28175 | 0.337 | 0.4015 | 0.3104375 |
| 14 | 0.23275 | 0.299 | 0.371 | 0.39325 | 0.324 |
| 15 | 0.271 | 0.3105 | 0.3975 | 0.40775 | 0.3466875 |
| 16 | 0.26975 | 0.352 | 0.3755 | 0.4855 | 0.3706875 |
| 17 | 0.311 | 0.3825 | 0.414 | 0.49575 | 0.4008125 |
| 18 | 0.2965 | 0.3775 | 0.43775 | 0.50625 | 0.4045 |
| 19 | 0.32475 | 0.39025 | 0.5115 | 0.517 | 0.435875 |
| 20 | 0.3535 | 0.42 | 0.54125 | 0.55675 | 0.467875 |
| 21 | 0.3585 | 0.44525 | 0.538 | 0.5675 | 0.4773125 |
| 22 | 0.378 | 0.49875 | 0.54825 | 0.61175 | 0.5091875 |
| 23 | 0.412 | 0.53475 | 0.599 | 0.62225 | 0.542 |
| 24 | 0.44025 | 0.513 | 0.64375 | 0.6615 | 0.564625 |
| 25 | 0.45075 | 0.4955 | 0.604 | 0.723 | 0.5683125 |

Table 12, average time by different cyclenum

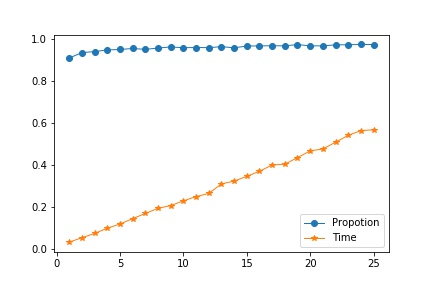


Fig 5, trend of average time and proportion by different cyclenum

As is seen in the tables and figure, we can infer that after 18, there is no significant improvement on the performance. Thus we spare the improvement after 18 and cyclenum =18 is relatively reasonable.

* 1. Comparison to Lingo 11

In order to test the accuracy and efficiency of the ant colony algorithm, comparison to Lingo 11 is done. All the problems is handled by ant colony algorithm 10 times and the best solution is registered and by Lingo 11 one time. The result can be seen in Table 13. The parameters is considered according to parameter tuning sector.

The ant colony algorithm is implemented in Pycharm with a Python programming language on a personal computer with AMD A8-7200P Radeon, 8 cores, 2.4GHz. The Lingo 11 runs on a Aliyun cloud computer with Intel(R) Xeon(R) Platinum 8163, 2.5GHz.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Group of problem | Number of problem | AS answer | Lingo answer | AS proportion to the optional solution | AS times | Lingo times |
| N=12 | 1 | 100.3029027 | 100.3 | 0.999971061 | 0.307776451 | 4.17 |
| 2 | 140.4132922 | 140.41 | 0.999976554 | 0.260736942 | 119.83 |
| 3 | 142.7954759 | 142.79 | 0.999961652 | 0.299647331 | 4.52 |
| 4 | 122.4635091 | 121.32 | 0.99066245 | 0.315632105 | 2.01 |
| N=14 | 1 | 119.3142751 | 116.44 | 0.975910049 | 0.378154993 | 18.52 |
| 2 | 150.5384386 | 145.7 | 0.967859115 | 0.348677158 | 99.35 |
| 3 | 146.107778 | 146.1 | 0.999946765 | 0.386567831 | 7.59 |
| 4 | 133.7847537 | 133.78 | 0.999964468 | 0.373278379 | 142 |
| N=16 | 1 | 128.9788985 | 119.31 | 0.925035036 | 0.503167152 | 12.08 |
| 2 | 131.5255069 | 131.52 | 0.999958131 | 0.500524282 | 45.36 |
| 3 | 147.2566827 | 144.01 | 0.977952222 | 0.489317417 | 623.5 |
| 4 | 173.0698964 | 172.02 | 0.993933686 | 0.443151712 | 629.81 |
| N=18 | 1 | 130.5820008 | 128.33 | 0.982754125 | 0.514460087 | 130.64 |
| 2 | 139.4970817 | 139.49 | 0.999949234 | 0.506672859 | 157.89 |
| 3 | 145.564674 | 145.17 | 0.997288669 | 0.563979626 | 273.16 |
| 4 | 165.478646 | 163.9 | 0.990460123 | 0.607200861 | 1828.01 |

Table 12, AS results compared to optimal solutions

* 1. Discussion of the result

As is seen in Table 12, the ant colony algorithm offers thrilling result compared to the optimal solutions. Most of the time, the ant colony algorithm provides solutions with slight loss compared to the optimal one and reduces amounts of time. Take group n=18 for example, the average loss is around 0.008 while the average time is 0.55 verse 597.42, about 99.9 percent of the average time is saved. Thus the algorithm is efficient and accurate.

1. Conclusion

In this paper, we design an ant colony algorithm to handle multiple salesmen TSP. In order to test the accuracy and efficiency of the algorithm, several problems with different size is generated. What’s more, parameter tuning is done on those problems. At last, the ant colony algorithm results are compared to the Lingo 11.0 solutions and the results tell a good performance.