Route Planning Algorithms Based on Ant Colony Optimization

Course

Data Structures and Algorithms, 332:573

Group Members

Yuhang Zhou, yz853

Wesley Lui, wl409

Submission Date

4/23/2019

Motivation

From an every-day trip to the grocery store via car to having to travel long-distance through trains, traveling is a critical component of everyone's day. Almost everyone would love to be able to get to their destination immediately and save time, if even just minutes, on each trip taken. While the technological growth of the automobile industry is impressive, improving the maximum speed of production cars from around 150mph in the 1950s to close to 280mph today, the primary bottleneck in everyday travel lies not in the physical capabilities of cars but rather the routes that must be taken; after all, even on highways we often have speed limits around 70mph (in the US).

After new breakthrough technology (like teleportation), it follows that optimizing the route taken is the most important factor in minimizing the time spent traveling in the average person's day. Route planning may also save precious minutes when factoring in the need to travel to multiple locations in a specified order. Already, route planning is used in many technologies, including Uber and Google maps. We would like to explore an efficient route planning algorithm based on ant colony optimization (ACO).

Algorithms Employed

Our route-planning algorithm will be based on the ant colony optimization algorithm. One of the main characteristics of ACO algorithms is based on ant colony behavior. In particular, ant colonies efficiently signal paths between their nest and food by communicating through residue pheromones left in the paths of scouting ants. Generally, the ant colony will find the shortest (and safest) path by following the scent of the densest pheromones, as the route with the most amount of this substance indicates the most activity of the scouting ants and offers the highest chance of success.

Process of our algorithm implementation:

1. Create some ants, and put them in random positions as their start.
2. Calculate the probability for one ant to travel from the recent position to one destination.

   The function of the probability

   there are two factors in the possibility: Distance and Pheromones

   for example, this ant is in position O, he has 3 destinations A, B and C to travel through, so that means he has three choices for the next destinations.

   function for the probability

$$\text{Probability(O->A)}= \frac{\text{phe(OA)}^{\alpha} * (1/\text{dis(OA)})^{\beta}}{\sum_{i=A,B,C} \text{phe(Oi)}^{\alpha} * (1/\text{dis(Oi)})^{\beta}}$$

   the α & β are parameters

3. Make a decision for which destination to go in the next travel based on probabilities we get in step 2

   for example, in step 2, we get the probability for the ant travel to A, B or C, they are 0.2, 0.5, 0.3. Then we determine the cumulative sum of these probabilities, shown below:

	O->A	O->B	O->C
Probability	0.2	0.5	0.3
Cumulative sum	0.2	0.7	1.0

then we get a random number between 0 and 1. The interval this random number belongs to will be the decision, for example, the random number is 0.4, and 0.2<0.4<0.5, so that this ant will go to the destination B

4. Every ant does step 2 and 3 repeatedly until every ant finishes traveling to all the destination.

5. Calculate the total distance of every ant traveled, and update the pheromones in the routes this ant passed.

Also, the original pheromones will decrease a little bit because of evaporation. So the function of pheromones will be:

Phe = ρ*phe + Q/total_dis

ρ & Q are parameters

6. Step 1-5 as one iteration, repeat this for some iterations

Experimental Configuration

We would like to experiment with our route planning algorithm on a graph. On the graph, we will set multiple positions to be visited and draw the optimal route with our algorithm. We will calculate the length of the total route and compare it to that generated by other algorithms. Besides correctness, we would also like to compare the complexity of our algorithm to other solutions.

As for inputs, we read line-by-line from a comma-separated value (csv) file. The first line describes the number of dimensions to be read per line; we simulate the ACO algorithm on a 2D graph and thus have only two coordinates: x and y. The subsequent lines, therefore, contain two values each: an x-coordinate to be read and a y-coordinate.

The programming language, associated libraries, and other relevant information regarding the experimental setup are provided below.

Programming language: Python 3.7
Libraries: numpy 1.16.0
matplotlib 3.0.3
Environment: Microsoft Windows 10 17134
Visual Studio Code 1.33.1
Processor: Intel Core i5-7200U @2.50GHz
RAM: 8.00 GB

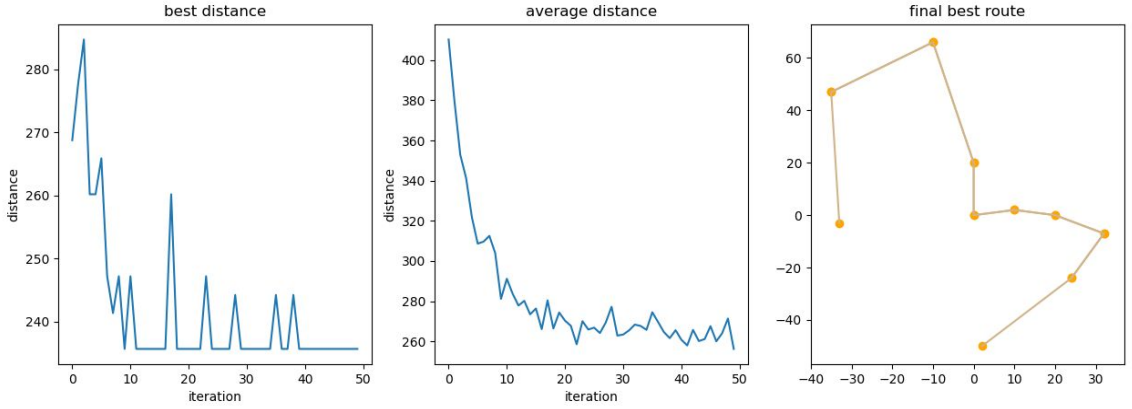
Results and Analysis

We provide the results/analysis of both the correction of the ACO algorithm and our implementation below.

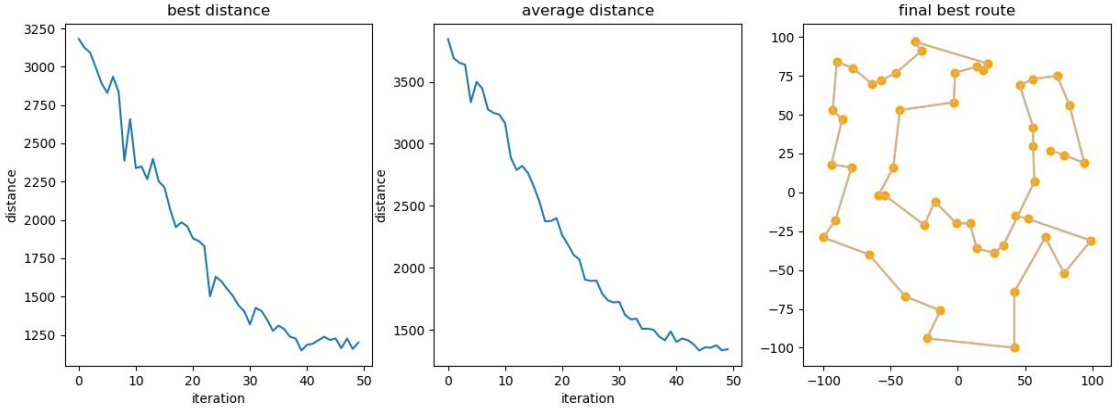
Correction Analysis

We fix the number of ants to 100, fix the number of destinations to increasing values, and compute the best distance and average distance when modifying the number of iterations from 1 to 50. We also provide the final best route to each destination, which is taken from the iteration that finds the smallest best distance.

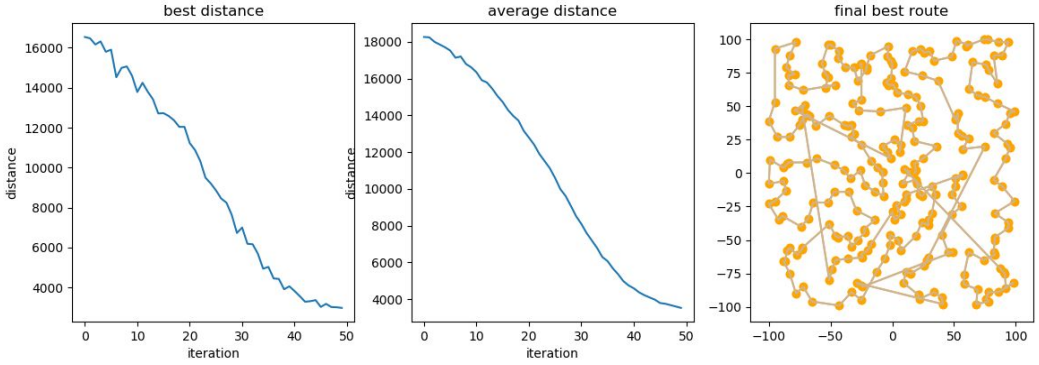
10 destinations:



50 destinations:



256 destinations:



We see that as the number of iterations increases, the values of the best distance between all routes tends to decrease and converge. This is to be expected, and we also predict that when increasing the number of ants, it might take fewer iterations for the algorithm to stabilize given a fixed number of destinations. Furthermore, we may observe that the final best route is often taken from one of the later iterations.

For the rest of the report, we define the following parameters for our analysis:

- **a** = number of ants
- **n** = number of destinations
- **i** = number of iterations
- RWC = random while loop complexity - used to determine which destination to next visit
- Overall time complexity (in big-oh notation) T(n)

Runtime Analysis of Implementation (theoretical calculation)

- Reading input: O(2*n)
- Calculating distances: $O(n^2)$
- Setting up pheromones chart $O(n^2)$
- Setting up traveled chart O(a*n)
- Setting up best route chart O(i*n)
- Best distance and avg distance initialization: 2O(i)
- Complexity for bulk of operations:
  $O(i*(2*a + n*(a* ((2*n/2) + n + (n/2) + RWC)) + a + a^2 + a + a + n^2 + a*n))$
  - O(2*a) the complexity of initializing start_pos and iterating through it to fill it in
  - O(n): iterating the over all destinations each ant must travel
  - O(a): iterating for each ant
  - O(2*n/2) (on avg): we initialize prob and not_traveled arrays, which on average each take O(n/2)
    - O(n) initially, when there are n destinations to visit, O(1) when there's one left
    - Do it twice => multiply by 2
  - O(n): iterating with variable k
  - O(n/2) on average, because k iterates from 0 to n-j and j goes from 0 to n
  - RWC for the while loop (expected to be a small constant)
  - O(a): for initialization of total_dist array
  - $O(a^2)$: filling in the total_dist values
  - O(a): finding minimum distance of unsorted array

    - O(a): calculating mean using built-in function
    - O(n^2): initializing an n x n matrix (phe_delta) with with zeros
    - O(a*n): to calculating each phe_delta value
- 2*O((n-1)*2): initializing 2 plot_route arrays
- O(i): to searching for minimum value in best_dist
- Setting up plots: O(n)
- Add complexity for matlab plot functions (assume it's a constant for simplicity of calculation)
- Total complexity T(n):

  T(n) = O(2*n) + O(n^2) + O(n^2) + O(a*n) + O(i*n) + 2O(i) +
  O(i * (2*a + n*(a * (5n/2 + RWC)) + 3a + a^2 + n^2 + a*n)) +
  2O(n-1)*2) + O(i) + O(n)

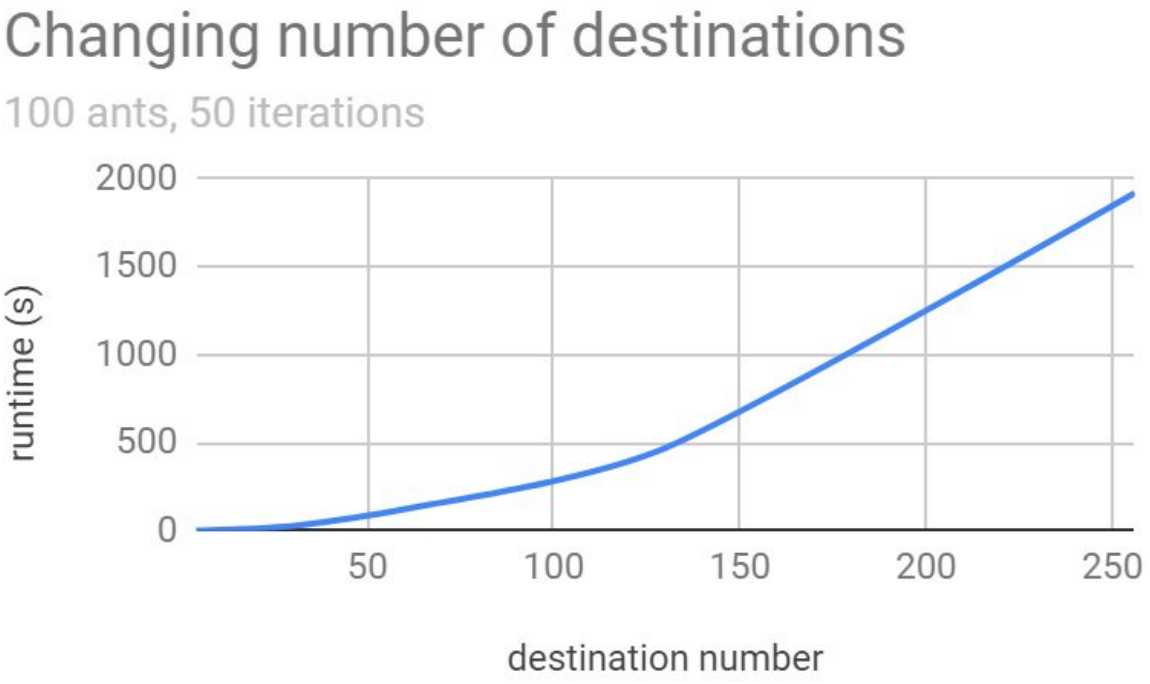
  = 2O(n^2) + O(a*n) + O(i*n) + O(i*(a*n^2 + n^2 + a^2 + a*n)), after dropping out lower order terms like O(n), O(a), O(i).

Predicted/Theoretical Runtime complexity

- T(n) = O(**i*****a*****n**^{2}) if we assume **n** >> **a**
- T(n) = O(**i*****a**^{2}) if **a** >> **n**

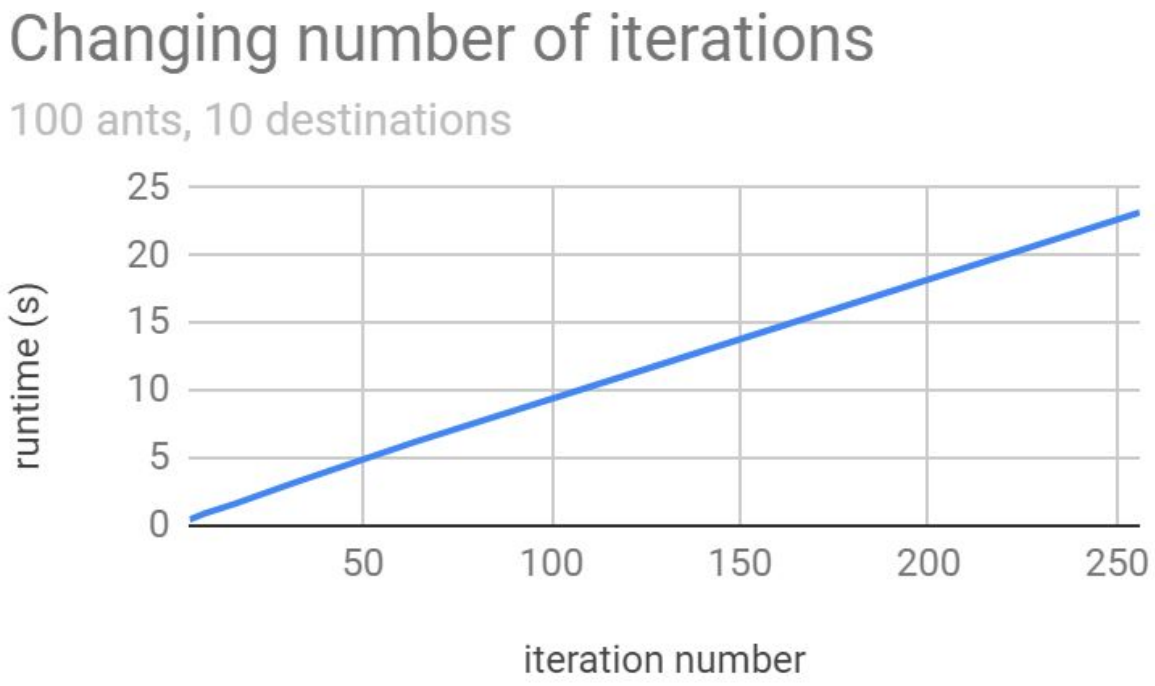
Runtime experimental results:

Destinations, n	Runtime (seconds)	$\log_2$(ratio)
4	1.25	N/A
8	3.29	1.396
16	9.57	1.540
32	32.8	1.778
64	138.33	2.076
128	451.89	1.708
256	1917.35	2.085



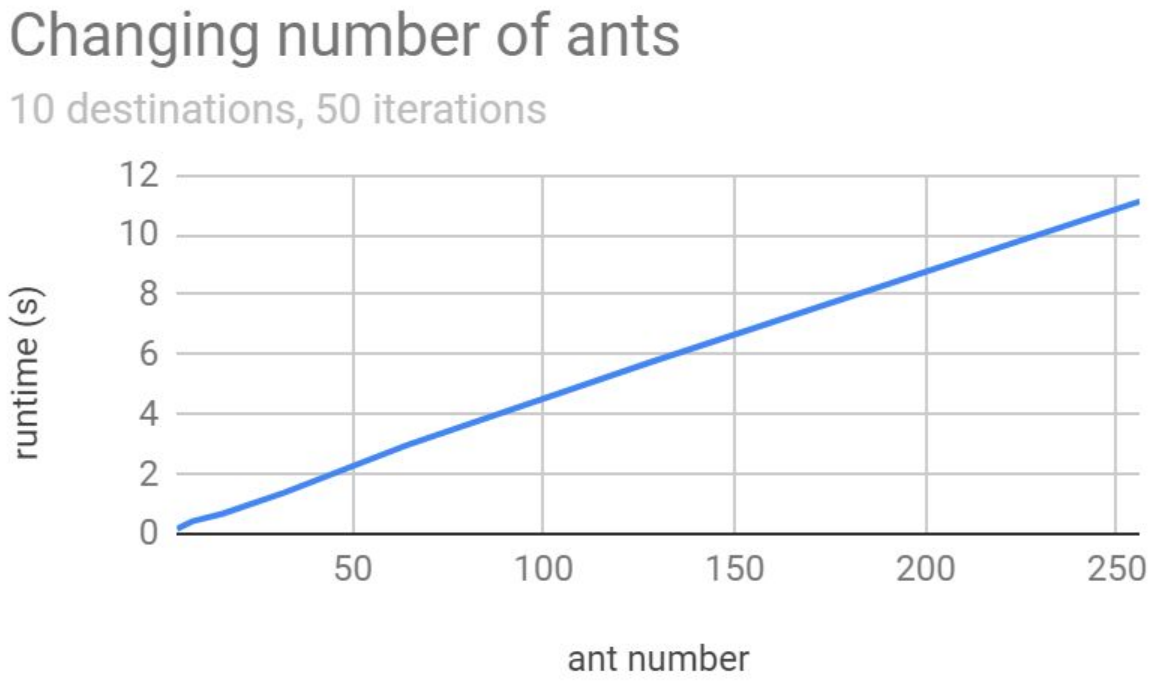
Fixing **i** and **a**, the runtime appears to be proportional to $\mathbf{n}^2$. This is further supported by applying the doubling hypothesis, where the $\log_2$ of the ratio of the current runtime to that of the previous runtime seems to converge to 2.

Iterations, i	Runtime (seconds)	$\log_2$(ratio)
4	0.44	N/A
8	0.89	1.016
16	1.62	0.864
32	3.2	0.982
64	6.22	0.956
128	11.84	0.929
256	23.15	0.967



Fixing **n** and **a**, the runtime appears to be proportional to **i**. This is further supported by applying the doubling hypothesis, where the $\log_2$ of the ratio of the current runtime to that of the previous runtime seems to converge to 1.

Ants, a	Runtime (seconds)	$\log_2$(ratio)
4	0.18	N/A
8	0.43	1.256
16	0.69	0.682
32	1.39	1.010
64	2.97	1.095
128	5.75	0.953
256	11.12	0.952



Fixing **n** and **i**, the runtime appears to be proportional to **a**. This is further supported by applying the doubling hypothesis, where the $\log_2$ of the ratio of the current runtime to that of the previous runtime seems to converge to 1.

We observe that the runtime is linear in changes to parameters **a** and **i**, which were determined by changing each individually and keeping the other two fixed. As for parameter **n**, the runtime is quadratic. Thus, to determine the overall runtime complexity T with respect to all three variables, we multiply the individual runtime complexities and find that T(n) is proportional to $\mathbf{a}*\mathbf{i}*\mathbf{n}^2$.

Discussion/Conclusion

Our analysis of ACO yields important information regarding its correctness and runtime.

From the correction analysis graphs, we observe that increasing the number of destinations requires more iterations in order to minimize the distance to all destinations. Furthermore, an increase in the number of iterations ensures that the average distance converges to the minimized best distance.

Note that the computed best distance may still increase in later iterations. We believe that this results from the randomness of the ant's decision, which is affected by the parameters α, β, and ρ. The uncertainty of ant movement helps explain the instability of the computed best distance to each destination in the early iterations. On the other hand, the average distance is much more stable (constantly decreasing with an increase in iterations) because it's computed from the average result of many ants. Given more destinations to travel, we can guarantee correctness by increasing the number iterations and the number of ants or adjust parameters based on the dataset.

Although increasing the number of iterations and ants seems to be the most intuitive way of guaranteeing the correctness of the final result, the time needed to run the program greatly increases. From the runtime analysis, the total runtime has linear growth with respect to the number of iterations and ants. However, it is quadratic in the number of destinations provided, and increasing this particular parameter causes the runtime to quickly increase. Intuitively, more destinations means that every ant has more choices to begin with and more destinations to travel to in every iteration. Thus, the run time is the product of the number of ants, the number of iterations, and the square of the number of destinations, as calculated.

While we have great insights in ACO from our current work, there are still factors we'd like to consider. One of these include the corrections of ACO; we have found that increasing the number of ants and iterations does help guarantee a degree of stability and confidence in the best route determined for set parameters. However, in the future, we would like to find whether the probability parameters may be chosen such that the stability and minimality of the best route is ensured for a smaller number of ants and iterations. On top of this, it would be of interest to determine formulas for the correctness and

stability of the ACO for a given number of destinations in terms of the number of ants and iterations. Lastly, perhaps our implementation may be improved to handle larger inputs, as already, when just 256 destinations are chosen, the runtime shoots up to around 2000 seconds, or 30 minutes.

In conclusion, the ant colony optimization is a very interesting and unique algorithm inspired by insect behavior. Our implementation of ACO suggests that further study in ant behavior can provide intuitive solutions to other problems outside of route planning.

Roles and Contribution

The contribution is equally distributed among team members.

Yuhang Zhou:

- Algorithm research
- Program implementation and debug
- Slides making

Wesley Lui:

- Experiment design and setup
- Program debug and testing
- Report writing

References:

Ant colony optimization algorithms From Wikipedia, the free encyclopedia
https://en.wikipedia.org/wiki/Ant_colony_optimization_algorithms

Ant colony optimization Marco Dorigo (2007), Scholarpedia, 2(3):1461.
http://www.scholarpedia.org/article/Ant_colony_optimization

Ant Colony Optimization (ACO) by Yarpiz
MATLAB implementation of ACO for Discrete and Combinatorial Optimization Problems
https://www.mathworks.com/matlabcentral/fileexchange/52859-ant-colony-optimization-aco

Ant colony algorithm solves TSP problem by VoidKing
https://www.voidking.com/dev-matlab-aco-tsp/