A New Algorithm for TSP Bases on Greedy and Kruskal Algorithm

Wu Tianyang 2204210460 Class 7

**Abstract** Path planning is a common problem in our life. Travelling Salesman Problem (TSP) is an abstraction of those problem. Our algorithm (G&M algorithm) improves the speed and also has a better accuracy than normal greedy algorithm. G&M algorithm bases on greedy algorithm and Kruskal algorithm. We tested many data and compared with other famous algorithms. Through testing and calculation, we have concluded that G&M algorithm has a better perform than other random algorithms in large data. Therefore, we suggest a new direction to the research of the TSP problem.

**Key words** graph theory, TSP, algorithm

1. Introduction

Travelling Salesman Problem (TSP) is one of the most famous problems in mathematics. The problem is there are more than one city and there has a cost between two cities, a salesman wants to start at one city, visit every city exactly once, and return to the city which he starts at. Find the minimum total cost for the salesman to complete his trip. Because it’s useful and difficult, people use many algorithms trying to solve it, but there is not a good result. We found a new way to solve TSP, it may provide another way of thinking about TSP.

Because TSP has been proven to be a NP complete problem (Christos H, 1977), so we can’t use traditional optimization algorithms. Greedy algorithm has been shown to be fast in solving TSP but there is much room for improvement (Lai, 2017). Ant colony algorithm has been shown to be accurately in solving TSP but it needs multiple iterations (Siemiński Andrzej et al., 2019). And many similar random algorithms, like simulated annealing algorithm, genetic algorithm and so on, they both require tuning and have general performance.

Although ant colony algorithm is recognized as the optimal solution of TSP. But we thought outside the box, come up with an optimization algorithm against ant colony algorithm, it could give an approximate optimal solution in a short time and have a better performance than greedy algorithm.

In this paper we address the details of the algorithm:

1). The principle of the algorithm.

2). The application of practical examples.

3). The comparison of different algorithm for complex data.

1. Method

IDE: VScode-1.56.0; Program language: C++14;

Painter: MATLAB-R2020b, Photoshop CS6.

Data sample came from [TSPLIB](http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp/).

* 1. Nomenclature

|  |  |
| --- | --- |
| Symbol | Description (All the points are in two dimensions.) |
|  | : graph, : node set, : edge set |
|  | The number of nodes in the graph |
|  | Two nodes in the graph |
|  | The x coordinate of |
|  | The y coordinate of |
|  | The undirected edge form by  *and* |
|  | The Euclidean metric of |
|  | The -th closed subring of |

Undirected edge: An undirected edge means that the salesman could pass from to , he could also pass from to . It means that, there’s no block between the two points. In TSP problem, each city could pass to any other one. So, there are undirected edges connect every two cities.

Length: means the Euclidean metric of . The Euclidean metric in two dimensions:

Closed subring: , . There is only one way that each node in passing all the others node in and back to itself through the edges in . For example, in Figure 1, there are four closed subrings in the graph. They are green, orange, purple and bule closed subrings. (Notice that two points can also form a closed subring.) Our target was to make a closed ring that included all the nodes in the graph.

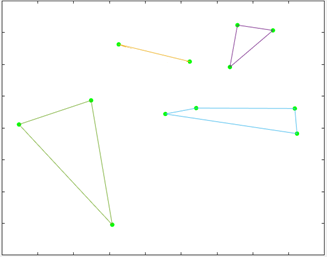


Figure 1

* 1. Principle

This new algorithm is based on greedy algorithm and minimal spanning tree algorithm also called Kruskal algorithm (Joseph B. Kruskal, 1956). The minimal spanning tree algorithm is used to merge the graph. So, we called this new algorithm “Greed&Merge” algorithm. (Here we use G&M algorithm for short.) In **Appendix**, we build the block diagram of the G&M algorithm. The whole algorithm was divided into two main parts: greed and shrink dots.

* 1. Greed

In this part, we defined a tag state for each node in . And a function for the tag:

We defined which contained all the untagged nodes.

If , we started to find the minimum length edge in and called the two endpoints of this edge and .

We selected one of this two points as the starting node, for example . Then we started from and found the node which is the closest to . And the rest may be deduced by analogy, found the closest node which also ensured that . Then we got a node sequence . If we found , we stopped at and cut out all the nodes behind it. At last, added to the node sequence’s end. So, we got a closed subring node sequence . We used an edge connecting two nodes that are adjacent in the sequence and tagged all the nodes in this sequence.

It was observed that, every greedy algorithm could tag at least two nodes. So, greedy algorithm ran at most times. After all the greedy algorithms, we could rebuild the graph (That means to create a new graph to replace the original one).

For example, Figure 2 was the result of the rebuilding. We could easily find that the graph had been divided into several closed subrings.

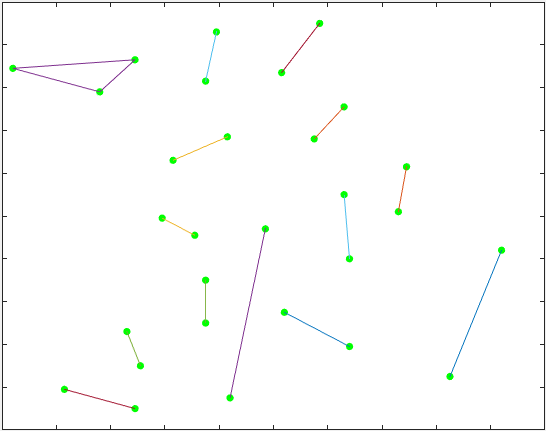


Figure 3

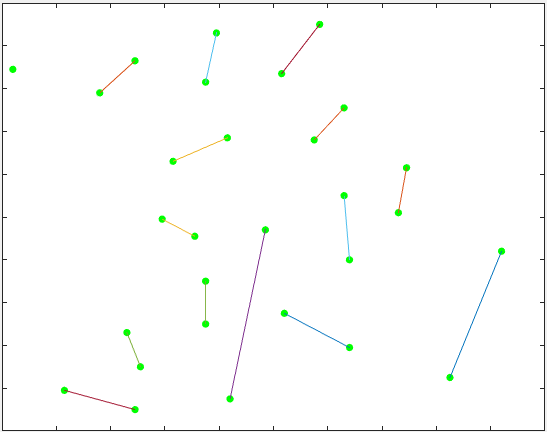


Figure 2

*2.3.1. Deal with Single Dot.* But there was a single dot in the top left corner of Figure 2 and it wasn’t a closed subring. To make sure all the nodes were in a closed subring, we should find the closest subring and connect this dot with this subring, just like Figure 3. Through this operation, all the single dots could be involved in one closed subring.

* 1. Shrink dots

After greedy algorithm, the graph had been divided into several closed subrings, we called them . We regarded each closed subring as a shrink node. Those shrink nodes were defined as a node set .

*2.4.1. Build edge.* We considered to use shortest edge to connect every two nodes in . For each of the two shrink nodes . We defined as the edge set of shrink node . For each of the edge , we define as two endpoints of edge . For shrink node and we defined as:

We recorded and that made this equation true. Next, we used an undirected edge with length to connect shrink node and . For each of the two shrink node built edges according to this operation, then we got a new graph composed of shrink nodes.

*2.4.2 Kruskal.* We considered to use Kruskal algorithm (Joseph B. Kruskal, 1956) to construct a spinning subtree of minimum length in . We added a new operation “Link and Cut” in connection operation in Kruskal algorithm. In this new operation, we need to define a cutting state for each edge in . And a function for the tag:

If Kruskal algorithm connected node and node . Then, we checked whether and were both false. If they were, we started “Link and Cut” operation.

*2.4.3 Link and Cut.* We defined and . This operator was easy to understand by Figure 4.

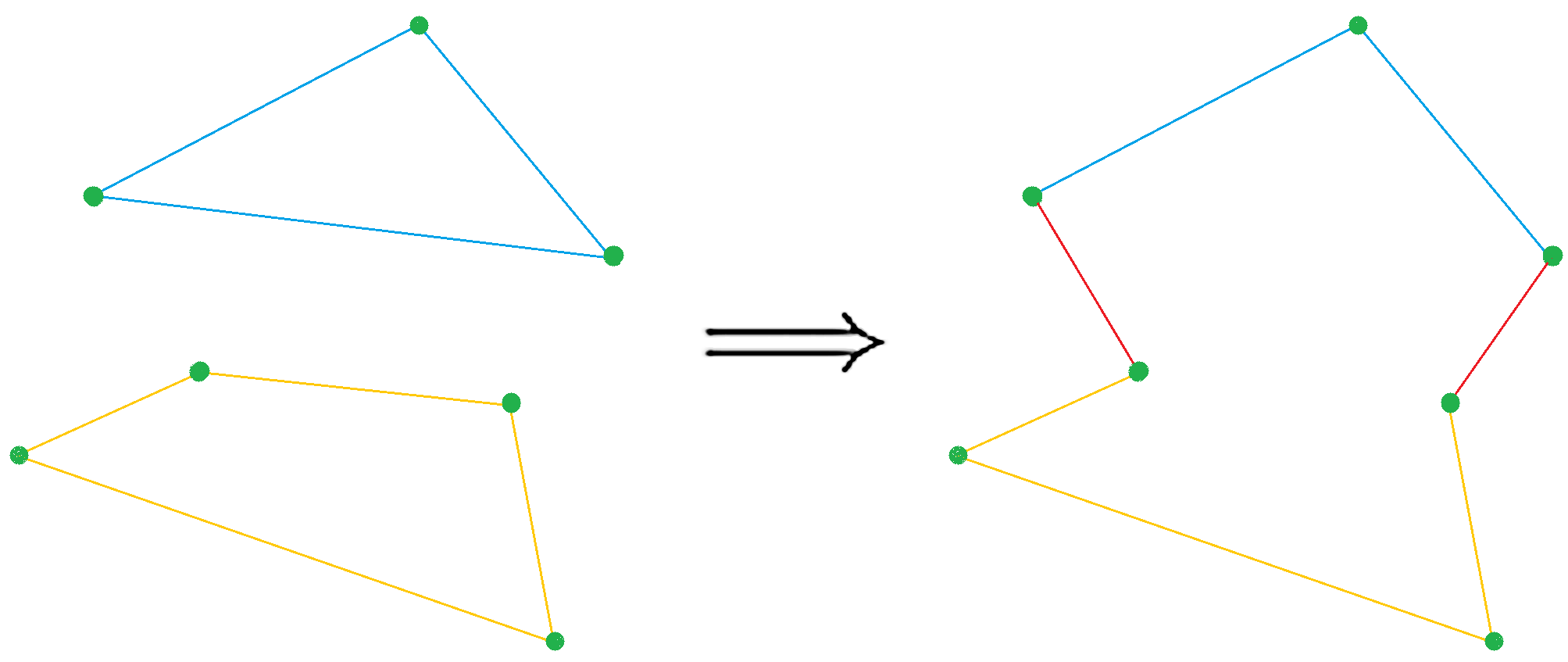
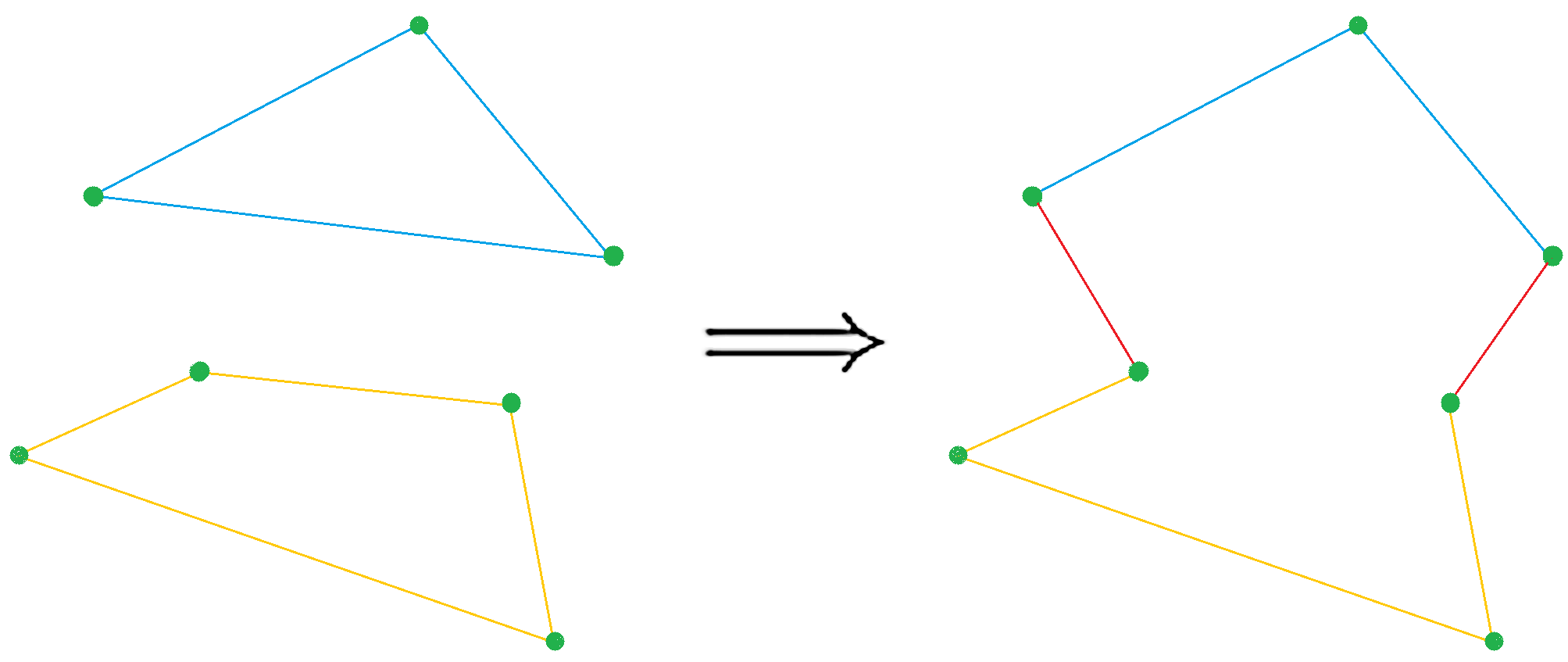
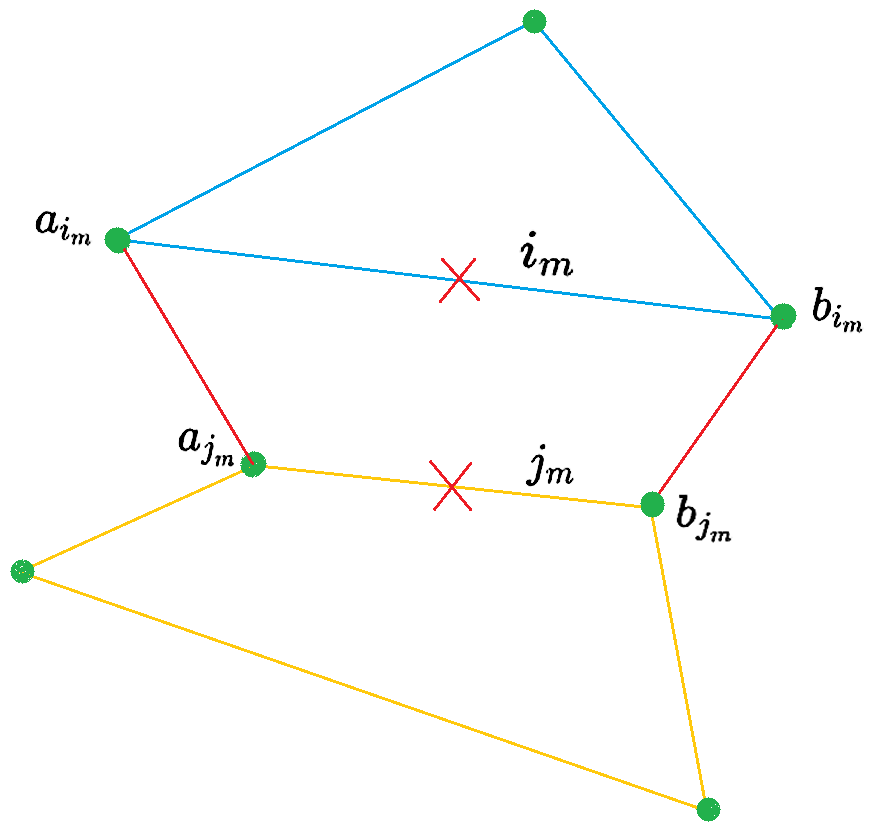


Figure 4.1

Figure 4.2

Figure 4.3

In Figure 4.1, there were two closed subrings, by the same time, they were two shrink nodes in graph . And Kruskal algorithm was connecting them. Because and were both false, which could make sure that edge and edge are both existed in graph . First, we cut edge and edge in graph and changed and . Second, we connected the minimal sum length between and . In Figure 4.2, we connected and but not and . Because and we chose the minimal combination to connect. And Figure 4.3 was the final state after “Link and Cut” operation.

It was observed that each Shrink dots operation at least merge two closed subrings into one closed subring. So, the shrink dots operation at most ran times. We showed an sample in Figure 5.

Figure 5.1

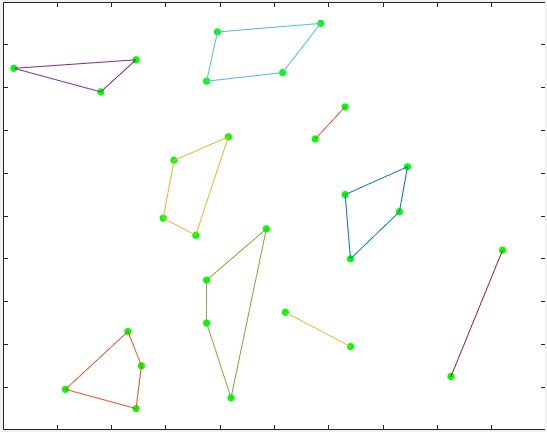
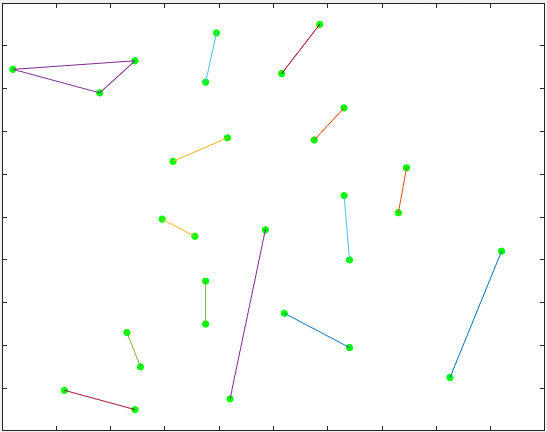


Figure 5.2

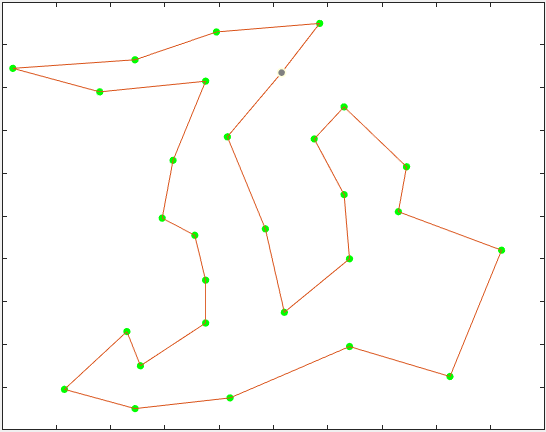
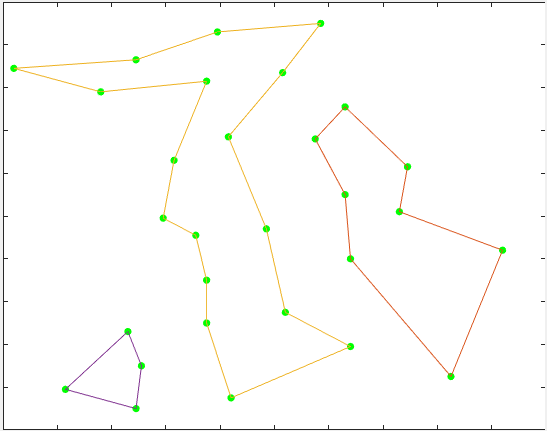


Figure 5.3

Figure 5.4

* 1. Greedy algorithm optimization

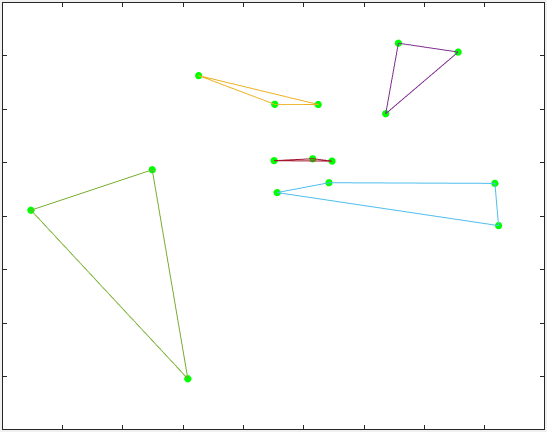
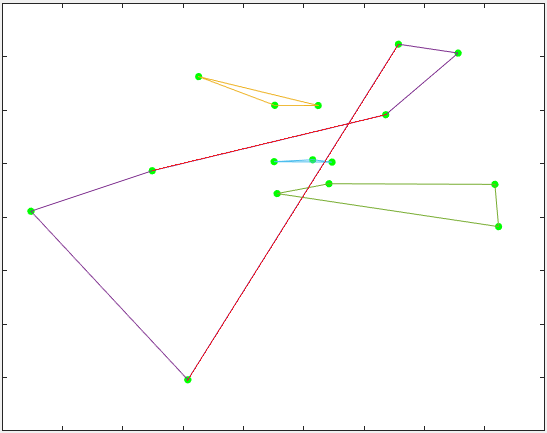


Figure 6.1

Figure 6.2

Using the greedy algorithm in 2.3. might got the result in Figure 6.1. We didn’t want to see the red edges in Figure 6.1, because those edges crossed two closed subrings: the blue one and the green one. To solve this problem, we added a new rule when we were building an edge . The nodes and could form a rectangle. If there was any other node in this rectangle, we shouldn’t build the edge . By this way, we could change Figure 6.1 into Figure 6.2 which was more in line with our vision. Although, this optimization might result in more single dots. But the operation of *2.3.1. Deal with Single Dot* could solve this problem.

* 1. Time complexity

Most random algorithm didn’t have a clearly time complexity. But G&M algorithm was based on greedy algorithm and Kruskal algorithm, they both have clearly time complexity.

G&M’s time complexity equaled “Greedy algorithm” plus “Deal with single dot” plus “Shrink operations” plus “Print graph” .

G&M was faster than normal random algorithm for example: anti colony algorithm, SA algorithm, genetic algorithm. It could perform much better in big data.

1. Result

We compared with three algorithms: Greedy algorithm, G&M algorithm and Ant colony algorithm (ACO) (A. Colorni, M. Dorigo al et., 1991). We used data “att48.tsp”, “d198.tsp”, “kroB150.tsp” and “a280.tsp” from [TSPLIB](http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp/).

We plotted the shortest distance and the number of iterations of the ACO. Because the greedy algorithm and G&M algorithm calculated the result by one time, therefore, their lines were straight lines. We also recorded the time taken when the ACO’s result was stable, the time is measured in (milliseconds), (seconds), (hours). The number in the text’s name represented the number of cities in this graph. For example, “att48.tsp” means that there are 48 cities in this graph.

Figure 7.1



Table 1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | att48 | | | d198 | | |
| Algorithm | distance | Error | time | distance | error | time |
| Greedy | 37880 | 12.90% | 0 ms | 17662 | 11.93% | 142 ms |
| G&M | 35107.7 | 4.64% | 8 ms | 17032 | 7.93% | 16 ms |
| ACO | 33600.6 | 0.14% | 168 ms | 16211.7 | 2.74% | 173.694 s |
| True | 33552 | 0.00% |  | 15780 | 0.00% |  |
|  |

Table 2



Figure 7.2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | pcb442 | | | pr1002 | | |
| Algorithm | distance | Error | time | distance | error | time |
| Greedy | 58896 | 15.99% | 633 ms | 311987 | 20.44% | 7.6 s |
| G&M | 53636.1 | 5.63% | 52 ms | 292501 | 12.92% | 394 ms |
| ACO | 53844.6 | 6.04% | 19.74 min | 293178 | 13.18% | 5.52 h |
| True | 50778 | 0.00% |  | 259045 | 0.00% |  |
|  |

Test equipment: CPU: AMD Ryzen 7 4800U, RAM: 16.0 GB

We calculated error by this formula:

Compared G&M algorithm with ACO. From Figure 7.1 and Table 1, we could draw a conclusion that ACO had better accuracy than G&M algorithm. But ACO became slower when the data got larger. We could clearly find that from Table 2. ACO calculated more than 5 hours to get the same result as G&M algorithm.

Compared G&M algorithm with Greedy algorithm. For each case, we could make sure that G&M algorithm was better than the greedy algorithm. In this way, G&M algorithm was an improvement on the Greedy algorithm.

Through Figure 7.1&7.2 and Table 1&2, G&M algorithm was fast and had a better accuracy than other algorithms when data volume exceeded 200 cities. When data volume didn’t exceed 100 cities, ACO was better than G&M algorithm. But TSP problem tended to be very complex in our daily lives. G&M algorithm would perform better in such condition.

1. Discussion

The purpose of this new algorithm was to improve the running speed to solve TSP problem in a large size of data. We made G&M algorithm base on greedy algorithm to divide the graph into small closed subrings and then Kruskal algorithm to merge two closed subrings into one closed subring, through repeated recursion we got a large closed subring which covered all cities. G&M has stable time complexity to solve the problem, but it also has some disadvantages. First, because it is based on greedy algorithm, the optimal distance is determined by one calculation. This result that it can’t update the optimal distance by double calculation. Second, solving the problem in the whole graph would greatly reduce the algorithm’s accuracy.

The next step includes development of G&M algorithm with other random algorithm to update the optimal distance instead of getting the result by one calculation. Also, it is important to divide a large graph in to some small graph and calculate them separately (Haim Kaplan et al., 2005). We recommend a new direction based on greedy algorithm to the computer researchers.

**References**

[A. Colorni, M. Dorigo, V. Maniezzo et al. Distributed Optimization by Ant Colonies. European conference on artificial life, Paris, France, Elsevier Publishing 1991: 134-142.](http://faculty.washington.edu/paymana/swarm/colorni92-ecal.pdf)

[Haim Kaplan, Moshe Lewenstein, Nira Shafrir & Maxim Sviridenko, 2005. Approximation algorithms for asymmetric TSP by decomposing directed regular multigraphs. *Journal of the ACM (JACM)*(4),. doi:10.1145/1082036.1082041.](http://www.cs.tau.ac.il/~haimk/papers/journal6.pdf)

[Joseph B. Kruskal. (1956). On the shortest spanning subtree of a graph and the traveling salesman problem. proc, 7(1), pp. 48-50.](https://www.ams.org/journals/proc/1956-007-01/S0002-9939-1956-0078686-7/home.html)

[来学伟.贪心算法在TSP问题中的应用[J].*许昌学院学报*,2017,036(002):41-44](https://www.zhangqiaokeyan.com/academic-journal-cn_journal-xuchang-university_thesis/0201233540231.html).

[Papadimitriou, Christos H. (1977). The Euclidean traveling salesman problem is NP-complete,](https://www.sciencedirect.com/science/article/pii/0304397577900123)*[Theoretical Computer Science](https://www.sciencedirect.com/science/article/pii/0304397577900123)*[,](https://www.sciencedirect.com/science/article/pii/0304397577900123) **[4](https://www.sciencedirect.com/science/article/pii/0304397577900123)**[(3): 237–244, doi:10.1016/0304-3975(77)90012-3, MR 0455550](https://www.sciencedirect.com/science/article/pii/0304397577900123).

[Siemiński Andrzej et al. Solving dynamic TSP by parallel and adaptive ant colony communities[J]. Journal of Intelligent & Fuzzy Systems, 2019, 37(6): 7607-7618.](https://schlr.cnki.net/en/Detail/index/WWMERGEJLAST/SIJDB4D962A97CCE39BA322A7A41D054528F)

**Appendix**

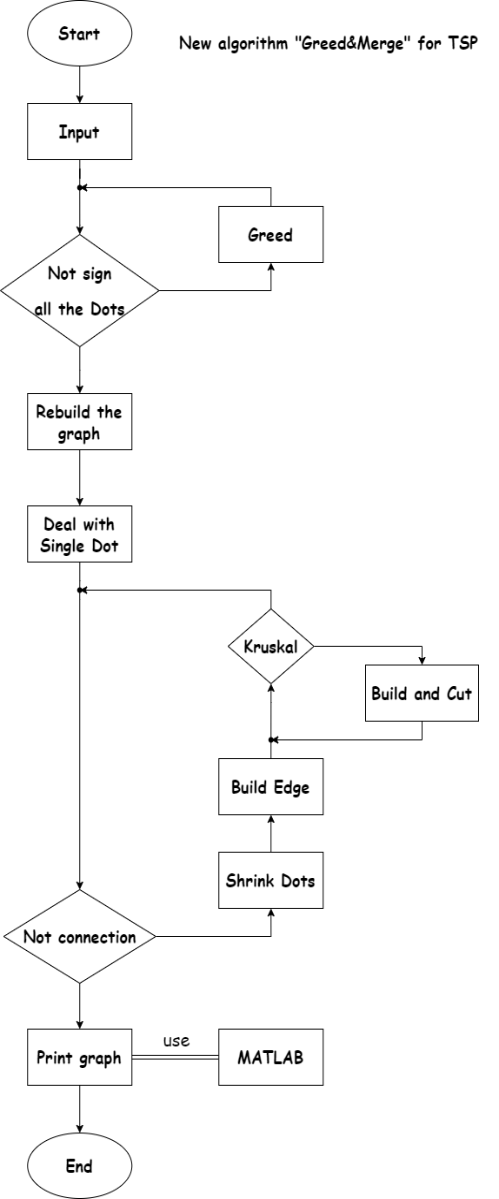


Figure 8

The block diagram of the G&M algorithm