Discrete Tree-seed Algorithm for Solving Symmetric Travelling Salesman Problem

Paper Draft

Felix Beese, Jens Lohmann

16.12.2020

Abstract

This paper takes a look at the discrete tree-seed algorithm (DTSA) which was proposed to solve the symmetric travelling salesman problem (TSP). The algorithm was implemented and also a similar genetic algorithm was designed. The results of both algorithms were compared by using statistical parameters like standard deviation and error rate. Also the progress over time was looked at. This paper also tries to answer the questions whether the DTSA could be a genetic algorithm in disguise.

Contents

1 Introduction **2**
1.1 Travelling Salesman Problem . . . 2
1.2 Discrete Tree-seed algorithm . . . 2
1.3 Genetic Algorithms . . . 4

2 Implementation **5**
2.1 Implementation of the DTSA . . . 5
2.2 Implementation of the GA . . . 5

3 Comparison of the Two Algorithms **7**
3.1 Change of the Path Length over Time . . . 7
3.2 Final Path Lengths . . . 8
3.3 Final Tours . . . 9

4 Discussion **10**
4.1 Performance of the algorithms . . . 10
4.2 Difference of the algorithms . . . 10
4.3 Limitation on symmetric TSP's . . . 10

5 Appendix **12**

1 Introduction

As a start a short introduction of the travelling salesman problem and genetic algorithms in general are given.

1.1 Travelling Salesman Problem

The travelling salesman problem (TSP) is a famous NP-hard problem in the field of computer science[4]. Given a set of cities along with distances between them, the task is to find the shortest path between all cities, while each city is visited exactly once. Instead of finding the optimal solution, it often is sufficient to find a tour, that is close to the optimal one. So naturally, many heuristic and approximate algorithms have been proposed to solve this problem in a reasonable amount time, including the nearest neighbour algorithm, ant colony optimization, and the discrete tree seed algorithm.

The symmetric travelling salesman problem is a special case of the TSP. Here for all cities the distance between two cities has to be the same in both directions. For example one-way streets or traffic jams would lead to an asymmetric TSP.

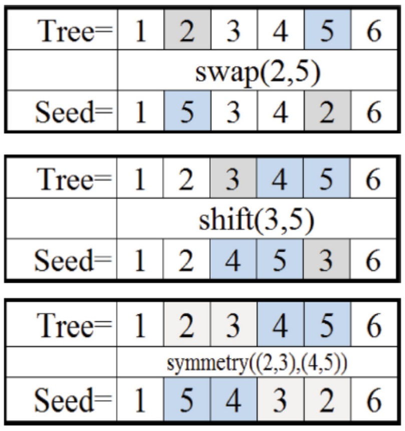
In this work only symmetric TSP's are considered. Reasons for this are discussed in chapter 4.

1.2 Discrete Tree-seed algorithm

The tree-seed algorithm (TSA) is an iterative search algorithm inspired by nature. It uses the concept of the life cycle of a tree. In the initial set of potential solutions, each one stands for a tree and produces seeds which differ slightly from the tree. Then for each group of seeds the best ones are selected and they produce new seeds in the next generation.

The discrete tree-seed algorithm (DTSA) is slightly modified. At first one of the initial trees is not initialized random, but as a nearest neighbour tour. It also uses a different method to create new seeds. Three operations called swap, swift and symmetry are used to explore the solution space. With *swap* the positions of two given nodes in the path get swapped. *Shift* is a kind of rotation, where every node in a given interval moves one place to the right except the most right node, which is then set to left end of the interval. With *symmetry* the order of nodes in a given interval is turned around to be a mirror image of the interval before, effectively changing a subpath from $A - B$ to $B - A$. For each operation an example is given in figure 1.

As a last difference the discrete tree-seed algorithm optimizes the final solution with a 2-opt algorithm[2].



Tree=	1	2	3	4	5	6
	swap(2,5)					
Seed=	1	5	3	4	2	6

Tree=	1	2	3	4	5	6
	shift(3,5)					
Seed=	1	2	4	5	3	6

Tree=	1	2	3	4	5	6
	symmetry((2,3),(4,5))					
Seed=	1	5	4	3	2	6

Figure 1: Examples for the three operations swap, shift and symmetry. Picture taken from Cinar et al.[1].

The following lines describe the discrete tree-seed algorithm.

- Determine the maximum function evaluation number ($maxfes$)
- D is the total number of cities in the problem
- Determine the number of trees (N)
- Determine the search tendency between 0 and 1 (ST)
- Determine the first tree as nearest neighbour tour
- Create all trees (except the first one) with random permutations between 1 and D
- Calculate the objective function of all trees
- Set function evaluation number $fes = N$
- Determine the best tree via using the objective function values ($best$)
- While fes is smaller than $maxfes$:
  - For all trees ($Tree(1)$ to $Tree(N)$):
    - Determine a random tree (except the current tree) ($Tree(k)$)
    - Create a number between 0 and 1 ($rand$)
    - If $rand$ is smaller than ST:
      - Create a seed with the *swap* operator using the *best* tree
      - Create a seed with the *shift* operator using the *best* tree
      - Create a seed with the *symmetry* operator using the *best* tree
    - Else:

            · Create a seed with the *swap* operator using the *current* tree
            · Create a seed with the *shift* operator using the *current* tree
            · Create a seed with the *symmetry* operator using the *current* tree
        ∗ Create a seed with the *swap* operator using the $Tree(k)$
        ∗ Create a seed with the *shift* operator using the $Tree(k)$
        ∗ Create a seed with the *symmetry* operator using the $Tree(k)$
        ∗ Calculate the objective function of the seeds
        ∗ Increase fes by adding 6
        ∗ Determine the best seed (*bestseed*)
        ∗ If the *bestseed* has a shorter path than *current* tree, replace the *current* tree with *bestseed*
    – Determine the best tree via using the objective function values (*best*)

- Apply the 2-opt algorithm using the *best* tree
- Return the optimized *best* tree as solution of the algorithm

1.3 Genetic Algorithms

Genetic algorithms are working based on the biological concept of evolution. They use mutation, crossover and selection methods to estimate solutions for optimization and searching problems[5].

A genetic algorithm is initialized with a set of possible solutions. These are copied and modified by mutation and crossover operations. A mutation makes a small change in a temporal solution. In contrast, a crossover makes a major change. It uses two solutions and mixes parts of their sequence to create new solutions, similar to sexual reproduction in biology. However crossover operations are optional for a GA, since asexual reproduction via only cloning and mutating an existing solution is also possible. Out of the full set of possible solutions the best solutions are selected. The selection is made via a so called fitness function, which describes the quality of a solution. Now the process starts again with the selected solutions which get copied and modified again. This way, the selected solutions slightly improve with every iteration. The process continues until a specific termination condition, which could be a number of repetitions or a threshold for the fitness level, is fulfilled.

2 Implementation

The work splits up into three main steps. First the discrete tree-seed algorithm (DTSA) was implemented as it is explained in the paper by Cinar, Korkmaz and Kiran [1]. In the next step a genetic algorithm (GA) was implemented. The results of both algorithms were optimized by a 2-opt algorithm[2].

The data sets for the evaluation were taken from tsplib95[1], which was also used by the Cinar et al. [1],[6]. Three different datasets were used. ”berlin52” with 52 data points, ”ch150” with 150 data points and ”tsp225” with 225 data points. All code is written in Python and provided in the appendix.

2.1 Implementation of the DTSA

The implementation of the discrete tree seed algorithm was done strictly according to the paper. The following parameters were used:

- $max\_fes = 800000$
- $search\_tendency = 0.5$
- $population\_size = number\_of\_cities$

These are the same as advised by the paper's authors to get as close to the original as possible. All starting solutions were initialized randomly except for one, which was created by the nearest neighbour approach.

2.2 Implementation of the GA

We designed and implemented a similar genetic algorithm to solve TSP's. As mutation operations swap, shift and symmetry were reused to explore the solution space in a similar way compared to the DTSA. However, no crossover operators were used since it is not possible to simply recombine two parts of different *parent trees* to form a new seed. Otherwise in most cases this would interfere with the specification of the TSP to visit each city exactly once. This means, that our proposed evolutionary process compares to asexual reproduction.

The algorithm, which we called GTSPA (short for *Genetic Travelling Salesman Problem Algorithm*), works as followed:

- Determine the maximum function evaluation number ($maxfes$)
- D is the total number of cities in the problem
- Determine the number of trees (N)
- Determine the rate of selection that should happen before mutation between 0.25 and 1 ($pre\_selction$)

[1]`https://pypi.org/project/tsplib95`

- Calculate k as $N \cdot pre\_selection$
- Determine the first tree as nearest neighbour tour
- Create all trees (except the first one) with random permutations between 1 and D ($trees$)
- Calculate the objective function of all trees
- Set function evaluation number $fes = N$
- While fes is smaller than $maxfes$:
  - Determine the k best trees via using the objective function values (k_best)
  - Initialize the next generation of trees as an empty list ($possible\_next$)
  - For all trees in k_best ($Tree(1)$ to $Tree(k)$):
    - Create a seed with the *swap* operator using the *current* tree
    - Create a seed with the *shift* operator using the *current* tree
    - Create a seed with the *symmetry* operator using the *current* tree
    - Add the three seeds and the current tree to the list of $possible\_next$ trees
    - Calculate the objective function of the seeds
    - Increase fes by adding 3
  - Determine the next generation of trees as the N best of the $possible\_next$ trees via using the objective function values
- Determine the best tree via using the objective function values (*best*)
- Apply the 2-opt algorithm using the *best* tree
- Return the optimized *best* tree as solution of the algorithm

Three different approaches were tested. They differ by the order of the mutation and selection step. In GTSPA-SM the solutions are first selected and then mutated. GTSPA-MS has those operations switched, first mutation then selection. GTSPA-SMS combines these approaches and has at first a selection step, then a mutation step and then again a selection step.

For the number of trees and the maximum function evaluation number the same parameters as in the DTSA were used, so the algorithms can be compared better in the next step. Again all starting solutions were initialized randomly except for one, which was created by the nearest neighbour approach. The algorithms' results were also optimized by the same 2-opt algorithm.

3 Comparison of the Two Algorithms

To compare the two algorithms the main focus was the quality of the results. Because both algorithms are heuristic, each of them was run 60 times. The change of the minimum path length was plotted for the different algorithms. Also the mean and the standard deviation of the minimum path lengths was calculated. At last the best paths were plotted and compared to the paths of the optimal solutions. This was done for the three datasets "berlin52", "ch150" and "tsp225" to see if the performance of the algorithms are dependent on the size of the data set.

3.1 Change of the Path Length over Time

The current minimum path length during the algorithms was plotted over the number of iterations to see how fast they get to the final path. We used data from 60 runs of the algorithms and calculated the mean of the minimum path length for each iteration.

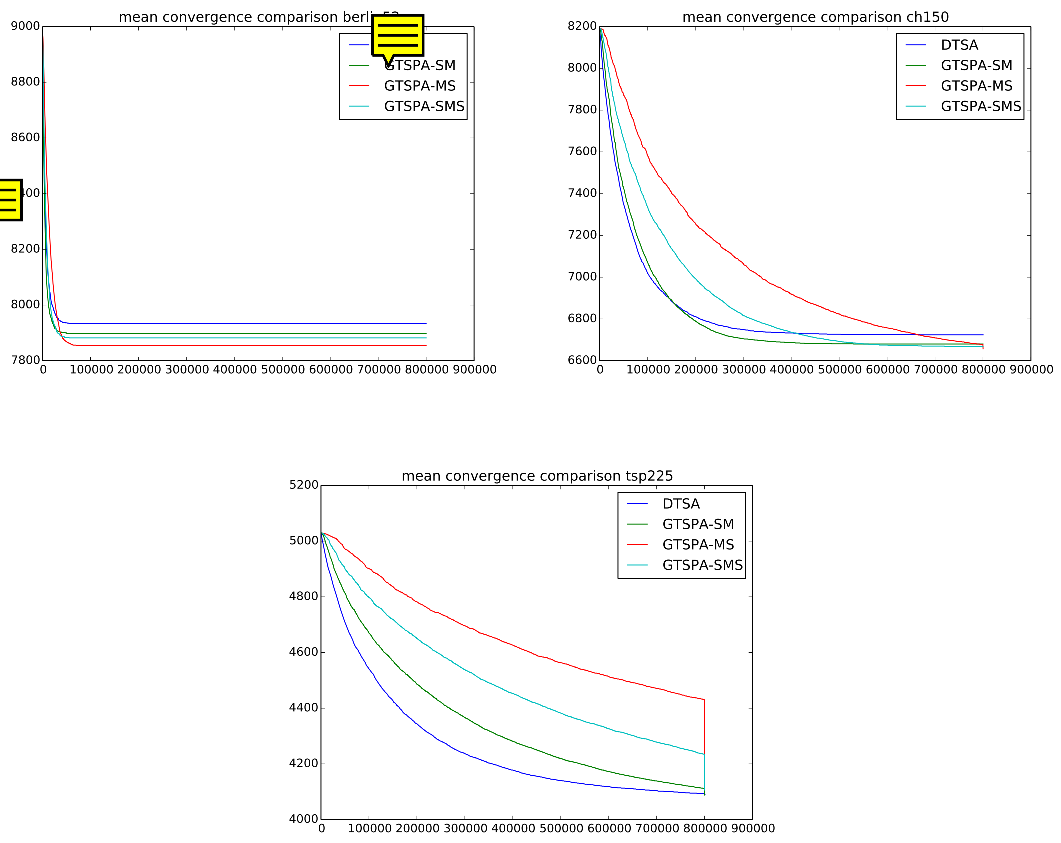


Figure 2: Convergence of minimum path length over time for berlin52, ch150 and tsp225

In figure 2 it can be seen, that all algorithms need more time with increasing number of data points. In all cases the DTSA is converging the fastest at the beginning. In the third picture, which shows the 'tsp225' data set, not all paths are fully converged before the stopping criterion is met. The sharp cut at the end comes from the 2-opt algorithm, which decreases the path length rigorous.

3.2 Final Path Lengths

The results of the different algorithms working on the different problems were analysed. Therefore the mean, standard deviation and return error (RE) of the final path lengths were calculated. For all calculations, except for the return error, pre-implemented functions in numpy were used. The sample size for each of the problems and algorithms was $n = 60$. The complete formulas are given here:

$$\mu = \frac{1}{n} \sum_{n}^{i=1} x_i \tag{1}$$

$$\sigma = \sqrt{\frac{\sum (x_i - \mu)^2}{n}} \tag{2}$$

$$RE = \frac{Result - Optimum}{Optimum} * 100 \tag{3}$$

To calculate the return error, the optimum solutions of the tsplib95 documentation [6] were used. The return error was calculated for each sample and then the mean over all samples was calculated. Also the best and the worst minimum path length over all samples was noted.

The results for the three data sets can be seen in table 1. The DTSA gives in average the worst results on data set 'berlin52' and 'ch150'. For these two data sets all of the algorithms give the same best tour.

Table 1: Statistical evaluation of the differrent algorithms for the three problems

dataset	algorithm	mean	std. dev.	best	worst	RE(%)
berlin52	dtsa	7933.17	90.02	7715	8134	5.19
	gtspasm	7897.37	97.67	7715	8055	4.71
	gtspams	7854.95	104.4	7715	8122	4.15
	gtspasms	7882.45	99.96	7715	8055	4.51
ch150	dtsa	6724.23	81.78	6563	6900	3.01
	gtspasm	6680.03	77.66	6563	6870	2.33
	gtspams	6657.53	57.65	6565	6793	1.98
	gtspasms	6667.5	75.75	6563	6876	2.14
tsp225	dtsa	4090.62	34.42	4023	4192	4.38
	gtspasm	4088.78	33.01	4023	4154	4.33
	gtspams	4150.43	43.51	4078	4252	5.91
	gtspasms	4106.45	35.38	4009	4209	4.78

3.3 Final Tours

To better evaluate the final solutions, the best path given by the algorithm was plotted and compared to the optimal path. The results are included in the appendix in figure 5, 6 and 7. The results of the algorithms look the same for the 'berlin52' data set. They differ only in small regions from the optimal path. This is the same, for the ch150 problem. The results only differ slightly from the optimal path.

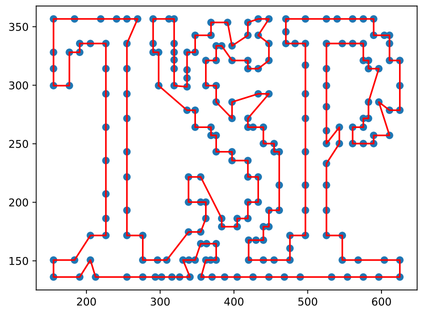


Figure 3: Best path for the tsp225 problem made with GTSPA-SMS.

For the tsp225 problem the results are different. The "T" and the "P" can be recognized quite good in all figures. The "S", which is in the middle of the figures is only noticeable in the GTSPA-SMS approach (figure 3).

4 Discussion

The results the experiments are discussed to see if the DTSA produces better results than a similar genetic algorithm. The similarities and differences of the algorithms are also examined. At last a small outlook on the asymmetric TSP and its solution is given.

4.1 Performance of the algorithms

As it was described in 3.1 the DTSA findes faster local minima compared to the other algorithms. But this only has an impact on the run time and not the result, because of the 2-opt algorithm, which is executed at the end. This can be seen in figure 2 in the 'tsp225' plot. Though the DTSA gives worse final solutions on average (table 1). If the data set is small enough all algorithms give the same best solution.

Therefore the DTSA is useful if the task is more time-sensitive. It is also recommended to run all algorithms several times, to get a little better solution.

4.2 Difference of the algorithms

The results of the algorithms only differ little from each other. Therefore the question arises if the DTSA could be a genetic algorithm in disguise. In the implementation all algorithms use the same operations (swap, shift and symmetry). Also the concept of selection can be seen in the DTSA. This could be a reason for the similar results.

The only thing that the DTSA is missing is a crossover operation. However, the usage of *best* and random $Tree(k)$ to generate new seeds, could be seen as a kind of crossover.

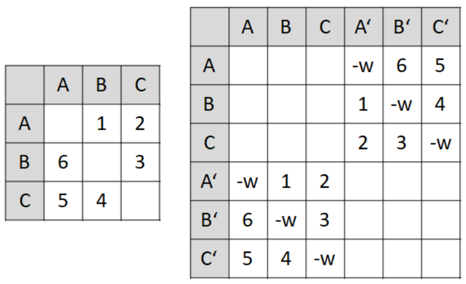
Could DTSA be a genetic algorithm in disguise? Yes. Swap, shift, symmetry are mutations, selections via selection of best seed and even some kind of crossover by using *best* and random $Tree(k)$ to generate new seeds.

4.3 Limitation on symmetric TSP's

The implemented algorithms only work on the symmetric TSP. A possible way to solve asymmetric TSP's could be to transform them into symmetric TSP's. Jonker and Volgenant presented a way to perform this transformation[3]. Their idea was to create a $2n \times 2n$ distance matrix, with so called ghost nodes. These ghost nodes need to be connected in the optimal solution. With this approach both distances between two cities can be denoted. A 3×3 example of this approach is given in figure 4.

What are the reasons for that?

With this approach the implemented DTSA and GA's can be used to calculate asymmetric TSP's, but it needs more time.



	A	B	C
A		1	2
B	6		3
C	5	4	

	A	B	C	A‘	B‘	C‘
A				-w	6	5
B				1	-w	4
C				2	3	-w
A‘	-w	1	2			
B‘	6	-w	3			
C‘	5	4	-w			

Figure 4: Example of the approach to convert an ATSP into a TSP.

References

[1] Ahmet Cevahir Cinar, Sedat Korkmaz, and Mustafa Servet Kiran. A discrete tree-seed algorithm for solving symmetric traveling salesman problem. *Engineering Science and Technology, an International Journal*, 23(4):879–890, 2020.

[2] G. A. Croes. A method for solving traveling-salesman problems. *Operations Research*, 6(6):791–812, 1958.

[3] Roy Jonker and Ton Volgenant. Transforming asymmetric into symmetric traveling salesman problems. *Operations Research Letters*, 2(4):161 – 163, 1983.

[4] K. Menger. ergenisse eines mathematischen kolloquiums. *Springer*, 1932.

[5] M. Mitchell. An introduction to genetic algorithms. *MIT Press*, 1996.

[6] Gerhard Reinelt. Tsplib95. *Interdisziplinäres Zentrum für Wissenschaftliches Rechnen (IWR), Heidelberg*, 338:1–16, 1995.

5 Appendix

The final paths for all problems and algorithms are given.

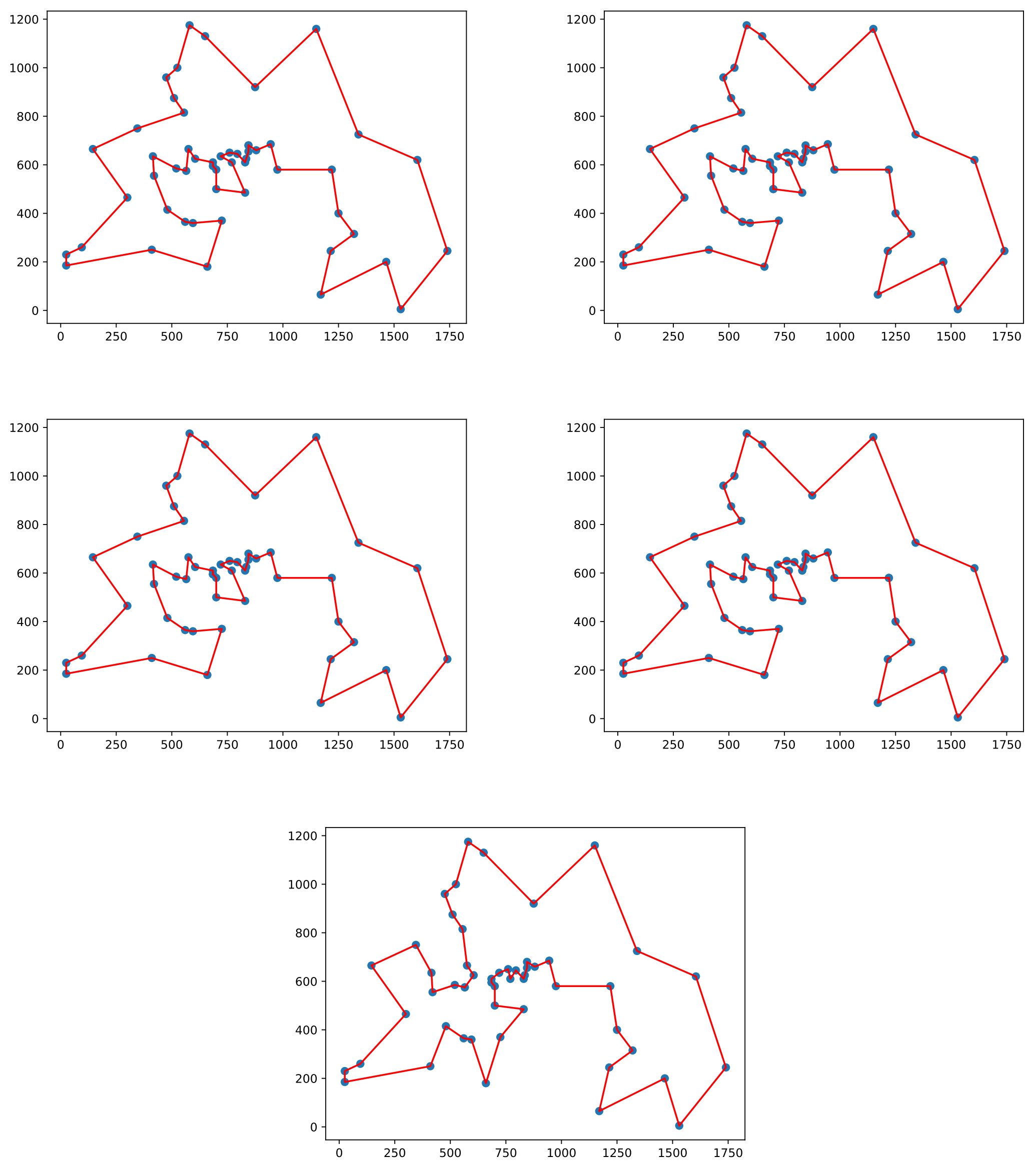


Figure 5: Final best paths for the different algorithms (from top left to bottom right: DTSA, GTSPA-MS, GTSPA-SM, GTSPA-SMS, Optimal path) on 'berlin52' problem.

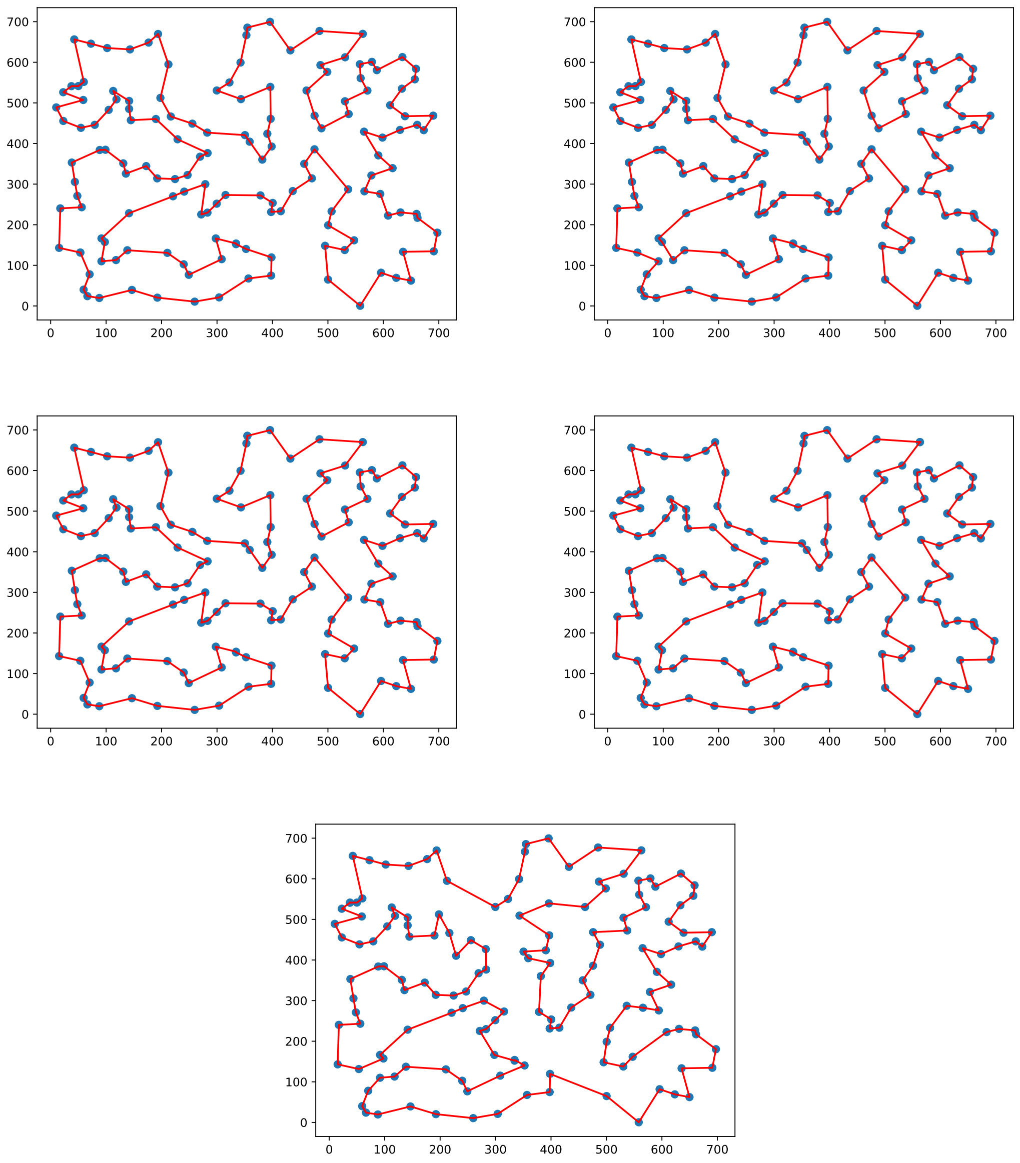


Figure 6: Final best paths for the different algorithms (from top left to bottom right: DTSA, GTSPA-MS, GTSPA-SM, GTSPA-SMS, Optimal path) on 'ch150' problem.

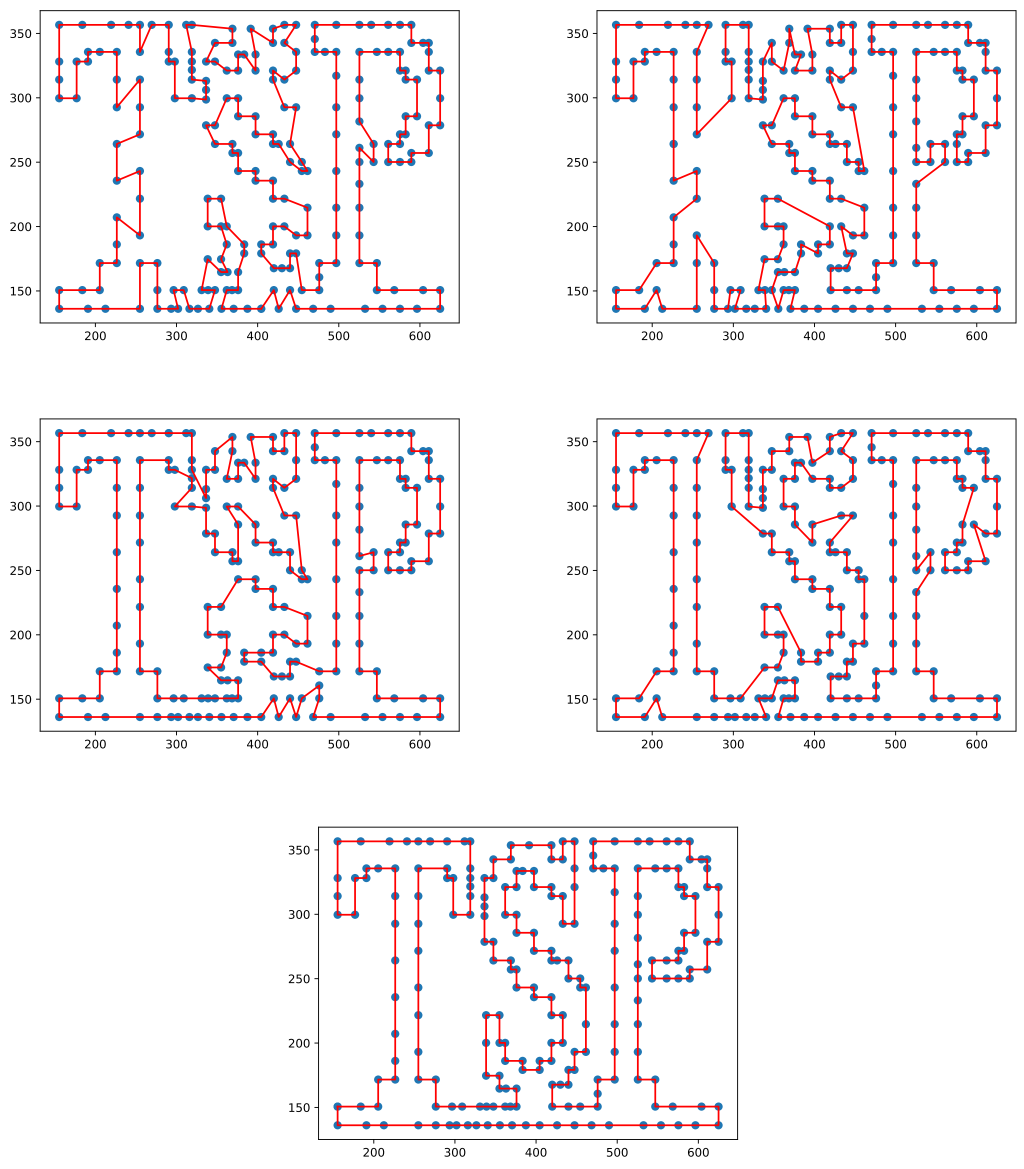


Figure 7: Final best paths for the different algorithms (from top left to bottom right: DTSA, GTSPA-MS, GTSPA-SM, GTSPA-SMS, Optimal path) on 'tsp225' problem.

include Code

This section is going to contain the code we wrote.