Discrete Tree-seed Algorithm for Solving Symmetric Traveling Salesman Problem

Paper Draft

Felix Beese, Jens Lohmann

26.01.2021

Abstract

This paper analyzes the discrete tree-seed algorithm (DTSA) which was proposed to solve the symmetric traveling salesman problem (TSP). The algorithm is implemented and also a similar genetic algorithm (GA) is designed, to analyze the relationship and similarities of the DTSA to the GA. For evaluation the results of both algorithms are compared by using statistical metrics like mean, standard deviation etc. The algorithms give similar results, in both cases the running time increases exponentially on the problem size. The DTSA is like a genetic algorithm, because it also uses the concept of evolution, specifically mutation and selection. It converges a little bit faster to a final solution, but gives on average a little worse results than the GA.

1 Introduction

The traveling salesman problem (TSP) is a well researched NP-hard problem in the field of computer science[1]. 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. Applications of the TSP can be found for example in planning and logistics[2]. Instead of finding the optimal solution, it often is sufficient to find a tour, that is close to optimal. Many heuristic and approximate algorithms have been proposed to solve this problem in a reasonable amount of time, including the nearest neighbor algorithm[3], genetic algorithms[4], and the discrete tree-seed algorithm[5].

The asymmetric traveling salesman problem is a special case of the TSP[6]. Here for two cities the distance between them can be of different length in both directions. For example one-way streets or traffic jams would lead to an asymmetric TSP. In this work only symmetric TSPs are considered. Reasons for this are discussed in chapter 4.

The tree-seed algorithm (TSA) is an iterative search algorithm inspired by nature[7]. To iteratively solve traveling salesman problems, it leverages the concept of the life cycle of trees. In the initial set of potential solutions, each one stands for a tree and produces seeds which differ slightly from the initial tree. Then in each iteration of the algorithm for each group of seeds the best ones are selected and they produce new seeds in the next generation.

To solve discrete problems, a slightly modified algorithm called discrete tree-seed algorithm (DTSA) was proposed[5]. In contrast to the original algorithm one of the initial trees is not initialized random but as a nearest neighbor tour[3]. A nearest neighbor tour starts at a random city. The path then gets extended by the city closest to the recent city until all cities are included in the path. The discrete tree-seed algorithm also uses a different method to create new seeds by choosing randomly between the best tour and a pre-selected tour. After convergence it optimizes the final solution with a 2-opt algorithm[8]. This reorders a self-crossing tour in a way that it does not cross itself anymore, which usually results in a shorter path.

This optimization behavior is closely related to genetic algorithms, which are using the biological concept of evolution. They use heuristic methods to estimate solutions for optimization and searching problems[4]. Applications of genetic algorithms can be found in image processing, job shop scheduling and others[9]. A genetic algorithm is initialized with a set of possible solutions, which take a similar role as genes in biology. These are copied and modified by mutation and crossover operations. A mutation makes a small change in one gene. In contrast, a crossover makes a major change. It uses two genes and mixes parts of their sequence to create a new gene, similar to sexual reproduction in biology. However crossover operations are optional for a GA, since asexual reproduction via only cloning and mutating an existing gene is also possible[10, 11]. Out of the full set of possible genes the best genes are selected by a fitness function, which describes the quality of a gene. The selected genes are then copied and modified again until a specific termination condition, which is often a number of repetitions or a threshold for the fitness level, is reached.

As described, the DTSA's approach to solve the TSP is comparable to a genetic algorithm. This raises the question if similar results can be achieved by a GA and if the DTSA might be a GA in disguise. In this paper we will design a GA that is similar to the DTSA and do a comparison of both algorithms. Section 2 explains how both algorithms work and how there are implemented. Also the experiment setup and evaluation methods are presented. In order to analyze the performance of the algorithms and their results, different statistical metrics are calculated and compared in Section 3. Based on this comparison we discuss our findings in Section 4 where we also talk about the limitation on symmetric TSPs. Finally, we present our conclusion with Section 5.

2 Methods

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 [5]. In the next step a genetic algorithm (GA) was designed and implemented to compare them with each other in the last step.

2.1 Implementation of the DTSA

The implementation of the discrete tree-seed algorithm was done strictly according to the paper. The algorithm starts by creating $number\_of\_cities$ initial solutions. These are initialized randomly except for one, which was created by the nearest neighbor approach[3]. This gives a starting solution with a shorter path length on average, than with a random approach. For every tree six seeds are created by the three operations swap, shift, and symmetry. The tree gets replaced by the seed with the shortest path length. This is continued until max_fes iterations are made. As a final optimization step the overall best solution is optimized by a 2-opt algorithm.

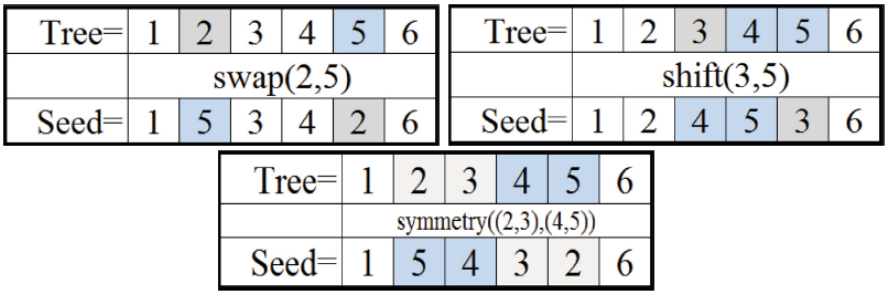


Figure 1: Examples for the three operations swap, shift, and symmetry. Picture taken from Cinar et al.[5].

An example for each operation is given in Figure 1. 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 left except the most left node, which is then set to the right end of the interval. With *symmetry* the order of nodes in a given interval is reversed to create a mirror image of the interval, effectively changing a subpath from $A - B$ to $B - A$.

The same parameters were used as advised by the paper's authors[5] to make the results comparable. $max\_fes = 800000$ indicates the number of seeds that are produced overall. $search\_tendency = 0.5$ indicates the ratio between using the recent tree or the overall best tree for producing the next seed. $population\_size = number\_of\_cities$ indicates how many seeds are selected for the next iteration. Details can be found in the original paper by Cinar et al.[5] and pseudocode is provided in the appendix 6.1.

2.2 Implementation of the GA

For investigating the strengths and weaknesses of the DTSA, a similar genetic algorithm to solve TSPs was implemented and its performance was compared to the DTSA. At the beginning of the algorithm *number_of_cities* initial solutions are created of which one is created by the nearest neighbor algorithm. Every solution gets mutated into three new possible solutions by using swap, shift, and symmetry as described. Of all solutions the best *number_of_cities* solutions are picked for the next iteration of mutations. At the end the overall best solution was optimized by the same 2-opt algorithm.

Three different approaches were tested, which differ by the order of the mutation and selection step. By this it could be investigated if the order of mutation and selection changes the results tremendously. 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 starts with a selection step, has then a mutation step and finally a selection step.

The proposed evolutionary process is comparable to asexual reproduction because it misses crossover mutations. These were not implemented, because it is not possible to simply recombine two parts of different *parent trees* to form a new seed, without interfering with the specification of the TSP to visit each city exactly once.

For the number of genes and the maximum function evaluations the same parameters as in the DTSA were used, to ensure a comparability of the algorithms. Pseudocode for the algorithm, which we called GTSPA (short for *Genetic Traveling Salesman Problem Algorithm*), is provided in the appendix 6.2.

2.3 Experiment setup and statistical evaluation

The problem instances for the evaluation were taken from tsplib95[12]. Six different sets were used ('berlin52', 'st70', 'kroA100', 'eil101', 'ch150' and 'tsp225'). The number at the end of the name gives the number of cities. Every algorithm was run $n = 90$ times for each problem instance.

To compare the different algorithms, different statistical parameters were calculated. These are the mean μ, standard deviation σ, and the return error RE which describes the deviation from the optimal solution in percent. In addition boxplots[13] are used for result visualization for each problem. The complete formulas of the calculated parameters are given here:

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

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

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

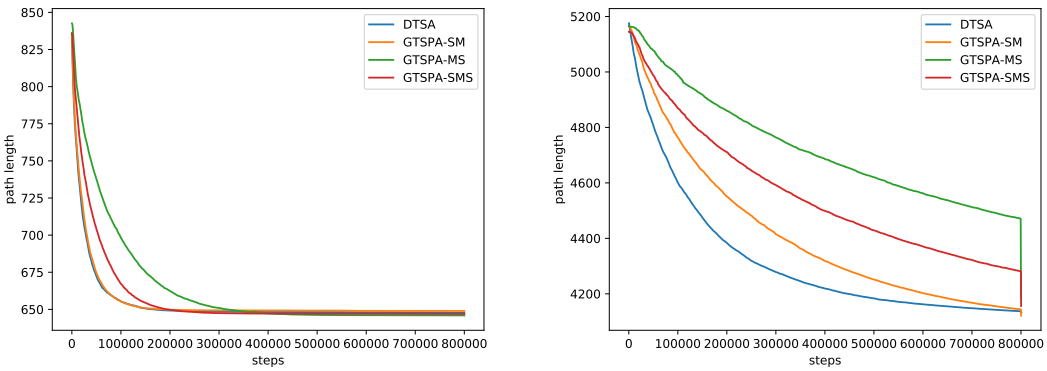
To calculate the return error, the optimal solutions of the tsplib95 documentation [12] were used. The return error was calculated for each sample and then the mean over all samples was calculated. Also the best and the worst minimum path length over all samples was noted.

3 Algorithm comparison

To compare the two algorithms, the main focus was the quality of the results. Because both algorithms are non-deterministic, 90 independent runs were used for evaluation. 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 data sets of different size 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 iterations

The current minimum path length during the algorithms was plotted over the number of iterations. It shows the improvement in each iteration, which indicates the convergence rate. The goal of this investigation was to see which algorithm gets the fastest to a good solution, not to see which one gives the best solution. For 90 runs of the algorithms the mean of the minimum path length for each iteration was calculated.



(a) eil101 (b) tsp225

Figure 2: Convergence of minimum path length over iterations.

Figure 2 shows, that all algorithms require more iterations with an increasing number of data points. Individual graphs for four more data sets are provided in the appendix 6.3, which underline this observation. In all analyzed cases, the DTSA is improving the fastest at the beginning, while the GTSPA-MS converges the slowest. In Figure 2(b), which shows the 'tsp225' problem, not all algorithms

are fully converged before the stopping criterion is met. The sharp cut at the end results from the 2-opt algorithm, which decreases the path length rigorously.

3.2 Final Path Lengths

The results of the different algorithms for the different problems are analyzed. Therefore, the mean, standard deviation, and return error (RE) of the final path lengths are calculated. The results for the six problem instances are shown in Table 1. Detailed boxplots are are illustrated in appendix 6.4.

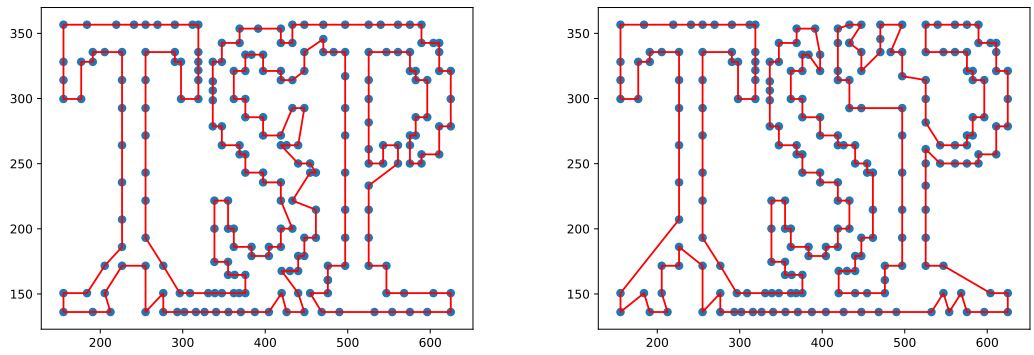
dataset	algorithm	mean	std. dev.	best	worst	RE(%)
berlin52	DTSA	7868.18	185.5	**7542**	**8405**	4.32
	GTSPA-SM	7867.68	193.33	**7542**	8301	4.32
	GTSPA-MS	**7790.32**	175.78	**7542**	8263	3.29
	GTSPA-SMS	7798.98	192.88	**7542**	8281	3.41
st70	DTSA	700.57	16.09	683	738	3.79
	GTSPA-SM	698.26	15.3	**680**	**747**	3.45
	GTSPA-MS	**695.41**	13.09	682	737	3.02
	GTSPA-SMS	695.47	12.2	682	741	3.03
kroA100	DTSA	21831.69	382.83	**21282**	23009	2.58
	GTSPA-SM	21754.38	355.07	**21282**	23136	2.22
	GTSPA-MS	**21588.57**	348.82	**21282**	23096	1.44
	GTSPA-SMS	21719.31	351.1	**21282**	**23224**	2.05
eil101	DTSA	647.76	8.45	**629**	**671**	2.98
	GTSPA-SM	648.99	8.66	630	**671**	3.18
	GTSPA-MS	**645.92**	7.22	631	665	2.69
	GTSPA-SMS	646.9	8.26	630	669	2.85
ch150	DTSA	6688.03	85.32	6549	**6922**	2.45
	GTSPA-SM	6683.87	88.31	6549	6898	2.39
	GTSPA-MS	6665.89	86.8	6552	6913	2.11
	GTSPA-SMS	**6661.59**	77.5	**6544**	6839	2.05
tsp225	DTSA	4132.73	64.11	3995	**4365**	5.45
	GTSPA-SM	**4121.48**	57.02	**3987**	4307	5.17
	GTSPA-MS	4188.68	55.87	4038	4350	6.88
	GTSPA-SMS	4156.21	54.31	4039	4330	6.05

Table 1: Statistical evaluation of the different algorithms for the six problems. The best results for the columns *mean* and *best* are bold. Also the worst results in the column *worst* are bold.

The DTSA produces on average the worst results on every problem instance except for 'eil101' and 'tsp225'. This also applies for the error rate of the DTSA. The best results of all algorithms are for 'berlin52' and 'kroA100' the optimal solution. For the other problems the best results are near to each other. The error rate is the highest for the analyzed problem instance with the largest number of cities.

3.3 Final Tours

To evaluate the final solutions in more detail, the best path given by the algorithm was plotted and compared to the optimal path. The results are included in appendix 6.5 in Figure 7-12. For the 'berlin52' and 'kroA100' problems the results of the four algorithms are identical. They also correspond to the optimal path as it is shown in Table 1. For 'st70', 'eil101' and 'ch150' the results of the algorithms have minor differences. None of the resulting paths matches exactly with the optimal path.



(a) GTSPA-SMS (b) GTSPA-MS

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 and the GTSPA-MS approach (Figure 3). Interestingly, although for the human mind the best results of these two algorithms look quite similar to the optimal solution, they produced the longer best paths (GTSPA-SMS = 4039, GTSPA-MS = 4038) in comparison to the other two algorithms (DTSA = 3995, GTSPA-SM = 3987) as it can be noticed in Table 1.

4 Discussion

As it was described in Section 3.1, the DTSA converges to local minima faster compared to the other algorithms. But this usually only has an impact on the run

time and not on 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 being faster, it gives a little bit worse final solutions on average. It has the highest mean for all analyzed problems except for 'eil101' where it is third best and 'tsp225' where it has the second best mean. On considerably smaller problems, all algorithms give the same best solution as it was the case for 'berlin52' and 'kroA100'.

The results of the algorithms only differ slightly from each other. Therefore, similarities and differences of the algorithms were also examined to analyze the relationship of the DTSA to a genetic algorithm. In the implementation all algorithms use the same operations (swap, shift and symmetry). Also the concept of selection is evident in the DTSA. This could be the reason for the similar results, that the algorithms have these structural similarities. The only difference is that the DTSA is missing is a crossover operation. However, the usage of *best* and *random* instead of only the *current* tree to generate new seeds, could be considered as a kind of crossover. It does not mix parts of each solutions to make a new one, but replace one completely by the other, like a crossover which starts at the first base of a chromosome.

Three different variants of the GTSPA were tested, which only differ by the order of the mutation and selection step. This was done to test if the different approaches would differ from each other significantly. It can be seen in Figure 2 that the GTSPA-SM is most of the time the fastest, while the GTSPA-MS approach takes the most steps to reach the final best path. If given enough time, the GTSPA-MS returns the best average result. A reason for this could be the fact that the different order of the mutation and selection approach only result in different population sizes and therefore lead to different exploration spaces. This leads to the GTSPA-SM having the smallest population during the process of mutation while GTSPA-MS has the largest. The same variants of the algorithms could be achieved by changing only the starting population sizes. A greater population size means a more explored solution space (better results) but also less iterations overall (slower convergence, more time needed).

The implemented algorithms returned overall quite good results, but they also have some limitations. In this case only six rather small problem instances were used. As described in Section 3.2 the algorithms perform better on smaller instances. Real world applications include often more than 225 cities, which would lead to worse results. Also in the real world asymmetric TSPs are more common. A possible way to solve asymmetric TSPs could be to transform them into symmetric TSPs. Jonker and Volgenant (1983) presented a way to perform this transformation[6]. Their idea was to create a $2n \times 2n$ distance matrix, with so called ghost nodes. These ghost nodes need to be included 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.

A possible solution would be $A \to A' \to C \to C' \to B \to B' \to A$. As it can be seen the path goes first to the corresponding ghost node before it continues to the next city. This is a necessity to be able to derive a path from the solution. With this approach the implemented DTSA and GAs can be used to calculate

	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.

asymmetric TSPs, but it needs more time. A problem might be, that the ghost nodes need to be connected to their respective real node. This can not be ensured with the mutation operations and the 2-opt algorithm. A lot of testing with the weight between the nodes and their ghost nodes would be required. Other special issues of this approach are discussed in the corresponding paper[14].

Although theoretically the implemented algorithms could be used to solve asymmetric TSPs as well, they only perform well on symmetric ones. Both algorithms rely on small changes of the best trees, performed by the operations swap, shift and symmetry. While these operations (especially symmetry) can already have a major impact on the solutions, most of the time only small changes are made between the iterations. In an asymmetric TSP already a small operation like swap can change the solution a lot, for better or for worse. This problem might lead to not applying said changes for the next generation, although it could bring even better results in later generations.

5 Conclusion

We have implemented the DTSA and a similar GA that uses the same mutation operations to solve TSPs. Both algorithms were tested with different problem sizes. We have shown that on average our GTSPA got better results in form of slightly shorter paths. However the DTSA converged faster to a solution, therefore it is useful if the task is more time-sensitive. Overall it is recommended to run all algorithms several times and pick the best solution of all runs. This is a better approach than running the algorithms for more iterations. At some point the path does not get any shorter although it is not the optimal path, as it can be seen for the smaller problems. This could be because it is stuck in a local minimum.

The GTSPA was tested with three different orders of selection and mutation to test its influence on the algorithms performance. This lead to different population sizes during the mutation step and the same variants of the algorithm could be achieved by only using different starting population size. We found that the GTSPA-SM converges faster on all problems while the GTSPA-MS needs more time, but also leads to better results on average when enough time is provided.

The DTSA is a genetic algorithm in disguise, because it follows a similar approach and uses the same operations. On average it can provide solutions in fewer

generations than the GTSPA, but provides worse solutions that result in longer paths.

The algorithms could be applied to solve real world problems, but will probably face some limitations. A problem with 225 cities could be solved, but with more cites the run time increases exponentially. Also asymmetric TSPs are more often seen in real world application. A solution to solve these was presented, but will again increase the computational costs and running time.

References

[1] K. Menger. Ergebnisse eines mathematischen kolloquiums. *Springer*, 1932.

[2] Teguh Narwadi and Subiyanto. An application of traveling salesman problem using the improved genetic algorithm on android google maps. *AIP Conference Proceedings*, 1818, 2017.

[3] Wayne Iba Pat Langley. Average-case analysis of a nearest neighbor algorithm. *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, 1993.

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

[5] 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.

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

[7] Mustafa Servet Kiran. Tsa: Tree-seed algorithm for continuous optimization. *Expert Systems with Applications*, 42(19):6686 – 6698, 2015.

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

[9] Mohammad Husian Manoj Kumar and Deepti Gupta Naveen Upreti. Genetic algortihm: review and application. *International Journal of Information Technology and Knowledge Management*, 2(2):451–454, 2010.

[10] Kevin J. Dawson. The advantage of asexual reproduction: When is it twofold? *Journal of Theoretical Biology*, 176(3):341 – 347, 1995.

[11] Jan Engelstädter. Constraints on the evolution of asexual reproduction. *BioEssays*, 30(11-12):1138–1150, 2008.

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

[13] Mary Eleanor Spear. Charting statistics. *McGraw Hill*, page 166, 1952.

[14] Roy Jonker and Ton Volgenant. Transforming asymmetric into symmetric traveling salesman problems: erratum. *Operations Research Letters*, 5(4):215 – 216, 1986.

6 Appendix

6.1 Pseudocode DTSA

The following lines describe the discrete tree-seed algorithm.

Algorithm 1: Discrete tree-seed algorithm

```
Data: maxfes, numcities, numtrees, searchtendency,
Result: Hamiltonian circle
begin
    Set first tree as nearest neighbor tour
    Create numtrees − 1 trees as random permutation between 1 and
     numcities
    Set fes to numtrees
    Set best to tree with shortest tour
    while fes < maxfes do
        for current in all trees do
            Get random tree (except current)
            Set rand to random number between 0 and 1
            if rand < searchtendency then
                Create seed with swap(best)
                Create seed with shift(best)
                Create seed with symmetry(best)
            else
                Create seed with swap(current)
                Create seed with shift(current)
                Create seed with symmetry(current)
            end
            Create seed with swap(random)
            Create seed with shift(random)
            Create seed with symmetry(random)
            fes + = 6
            Set bestseed to seed with shortest tour
            if bestseed < current then
                Set current to bestseed
            end
            Set best to tree with shortest tour
        end
    end
    Set best to 2-opt(best)
    return best
end
```

6.2 Pseudocode GTPSA

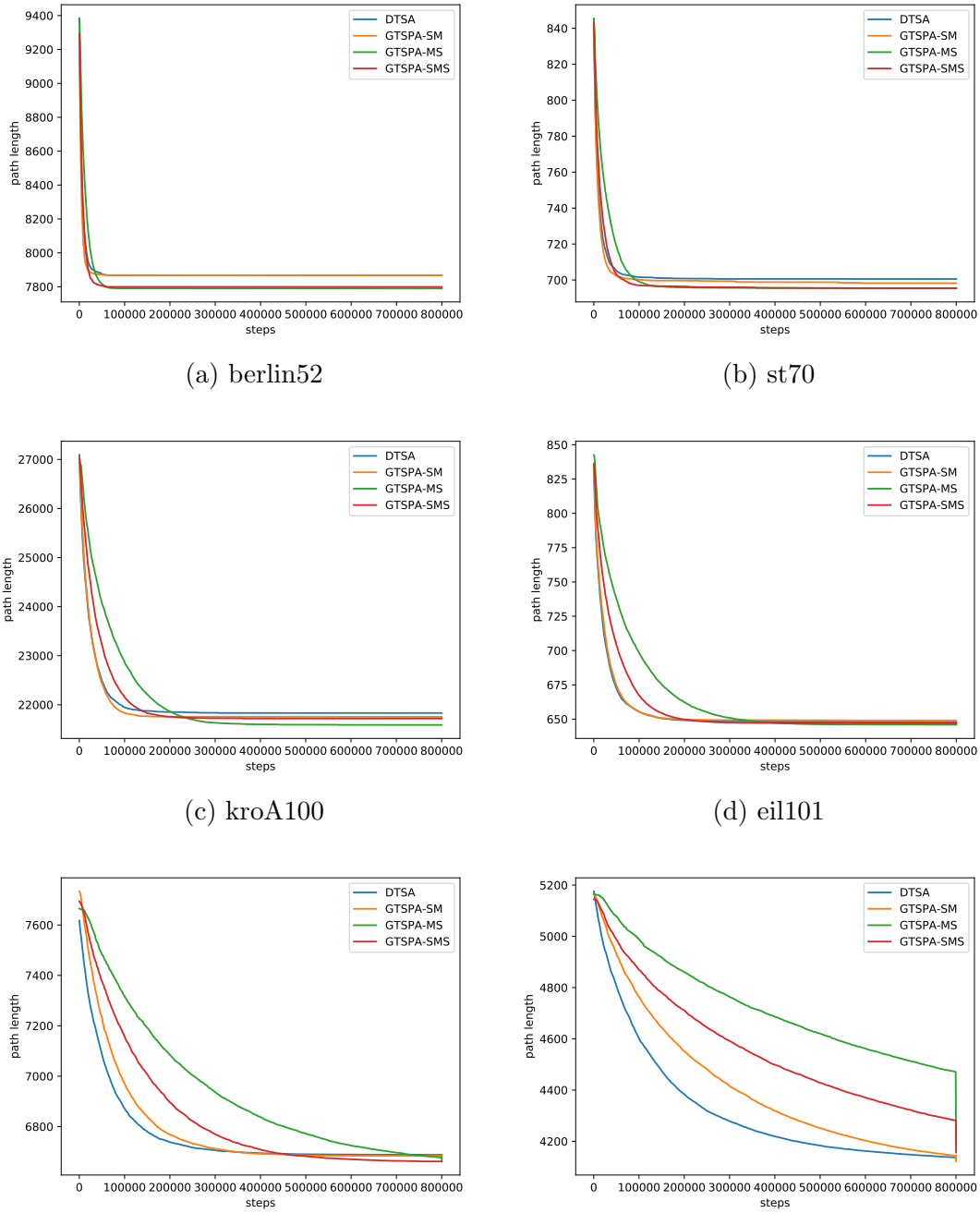
The following lines describe the genetic algorithm.

Algorithm 2: genetic algorithm

```
Data: maxfes, numcities, numtrees, preselection,
Result: Hamiltonian circle
begin
    Set k to numtrees · preselection
    Set first tree as nearest neighbor tour
    Create numtrees − 1 trees as random permutation between 1 and
     numcities
    Set fes to numtrees
    Set best to tree with shortest tour
    while fes < maxfes do
        Get best k trees with shortest tour
        Initialize next generation of trees as an empty list possiblenext
        for current in k best trees do
            Create seed with swap(current)
            Create seed with shift(current)
            Create seed with symmetry(current)
            fes + = 3
            Get best numtrees trees of possiblenext with shortest tour
        end
        Set best to tree with shortest tour
    end
    Set best to 2-opt(best)
    return best
end
```

6.3 Convergence of the paths

Convergence of the algorithms for different data sets by increasing size.



(a) berlin52

(b) st70

(c) kroA100

(d) eil101

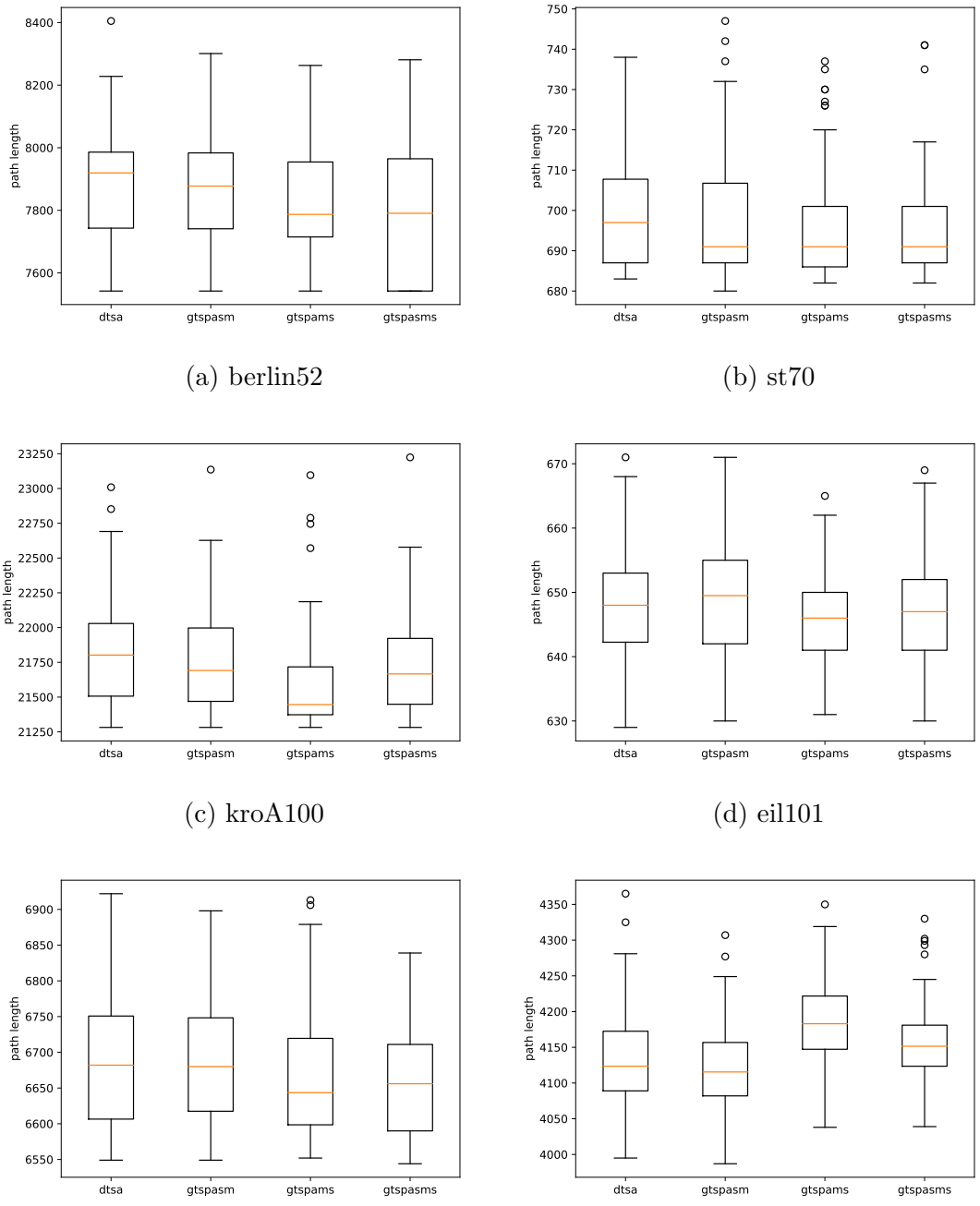
(e) ch150

(f) tsp225

Figure 5: Convergence of minimum path length over time.

6.4 Boxplots for different problems

Boxplots different problem instances by increasing size.



(a) berlin52 (b) st70

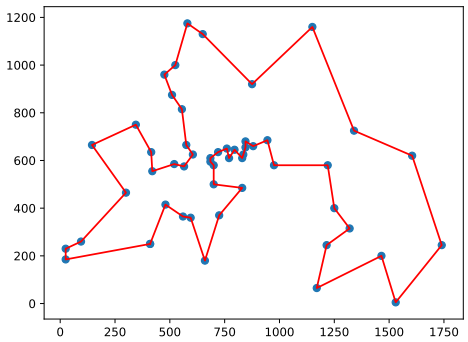
(c) kroA100 (d) eil101

(e) ch150 (f) tsp225

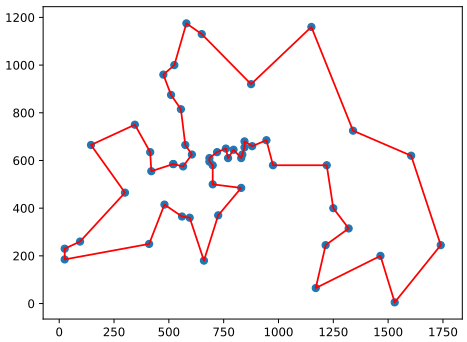
Figure 6: Boxplots with four different algorithms for each problem instance.

6.5 Final paths

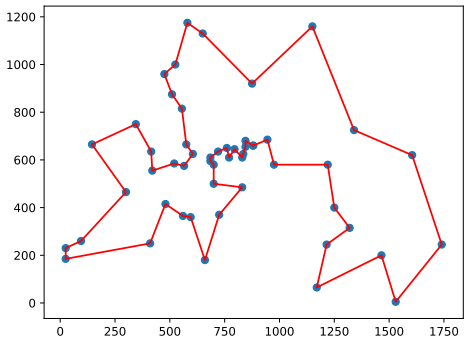
The final paths for all problems and algorithms are given.



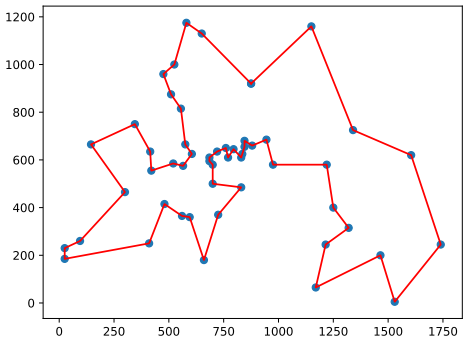
(a) DTSA



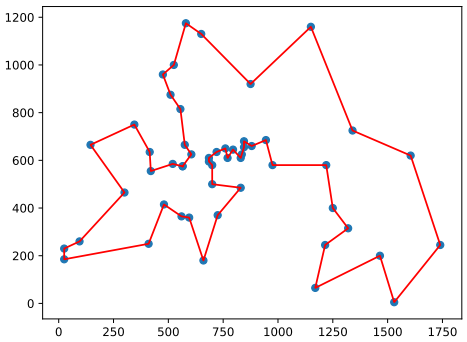
(b) GTSPA-MS



(c) GTSPA-SM

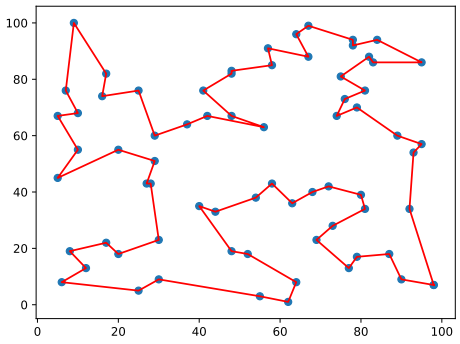


(d) GTSPA-SMS

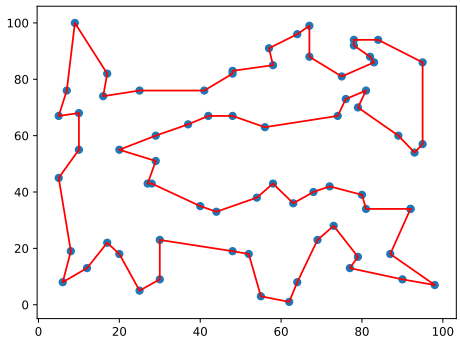


(e) optimal path

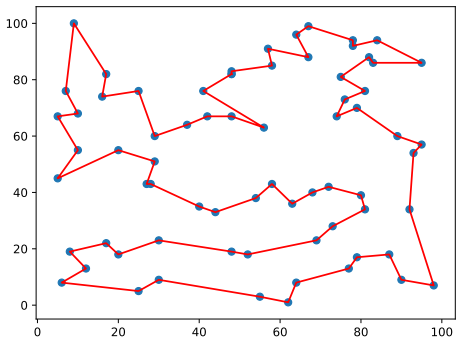
Figure 7: Final best paths for the different algorithms and optimal path for 'berlin52' problem.



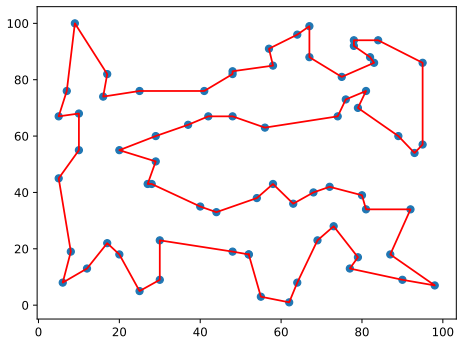
(a) DTSA



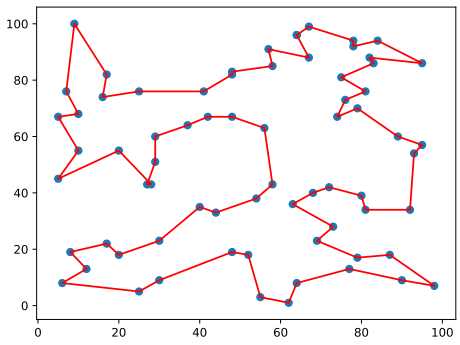
(b) GTSPA-MS



(c) GTSPA-SM

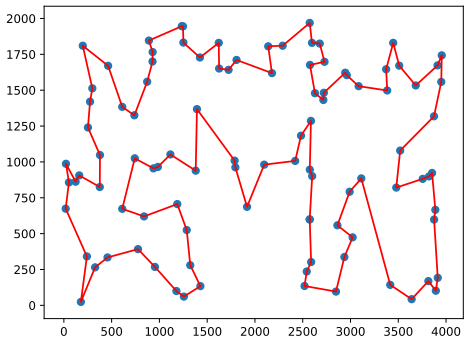


(d) GTSPA-SMS

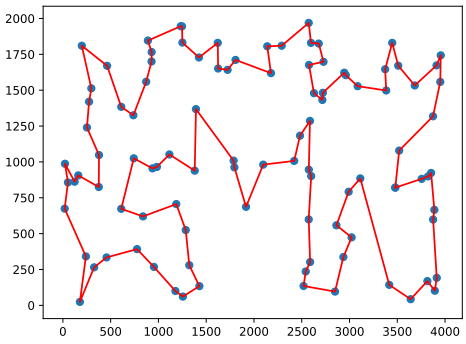


(e) optimal path

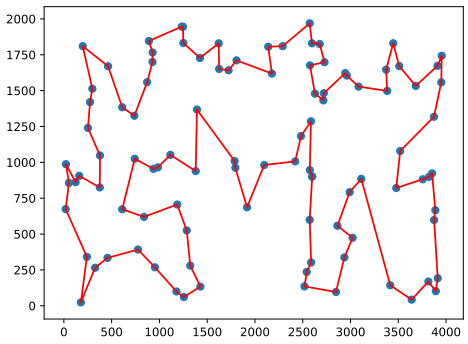
Figure 8: Final best paths for the different algorithms and optimal path for 'st70' problem.



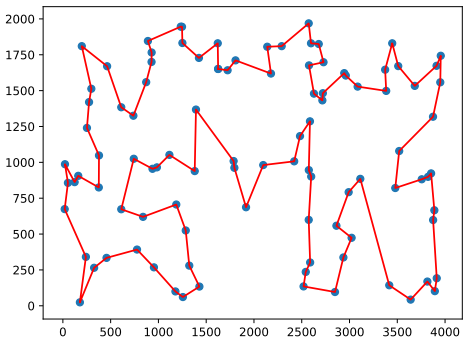
(a) DTSA



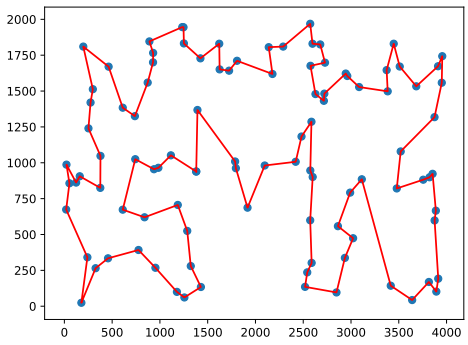
(b) GTSPA-MS



(c) GTSPA-SM

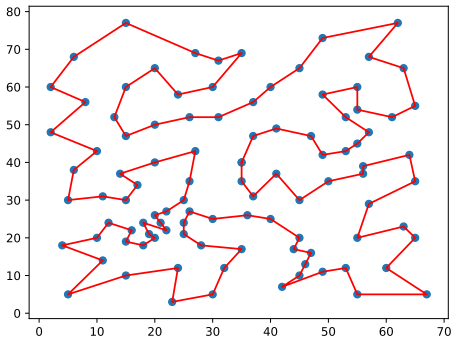


(d) GTSPA-SMS

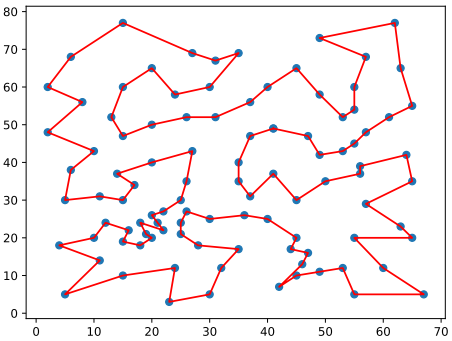


(e) optimal path

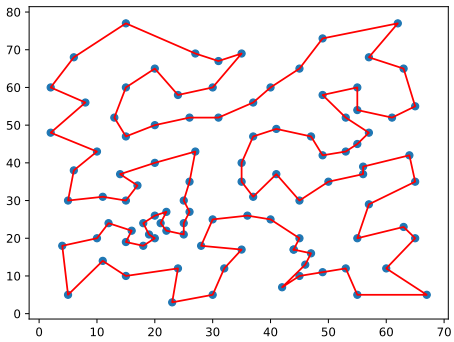
Figure 9: Final best paths for the different algorithms and optimal path for 'kroA100' problem.



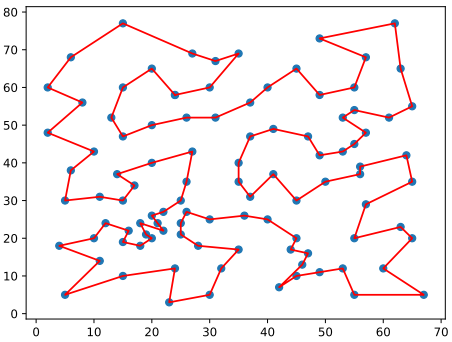
(a) DTSA



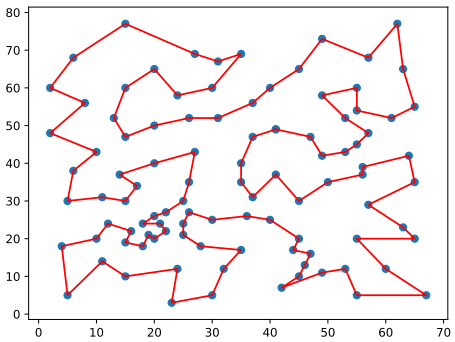
(b) GTSPA-MS



(c) GTSPA-SM

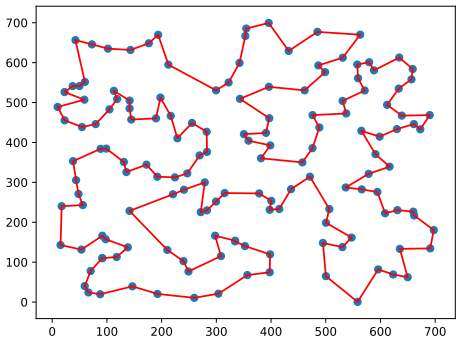


(d) GTSPA-SMS

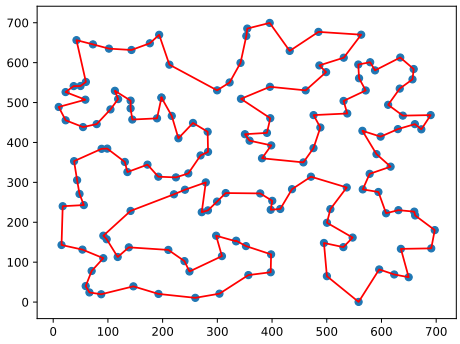


(e) optimal path

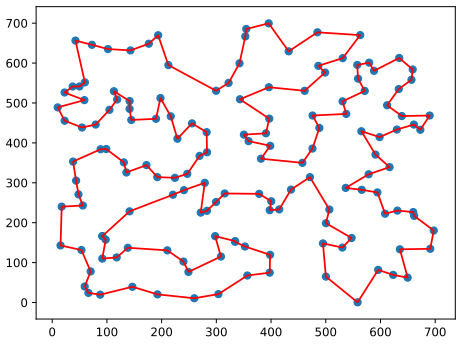
Figure 10: Final best paths for the different algorithms and optimal path for 'eil101' problem.



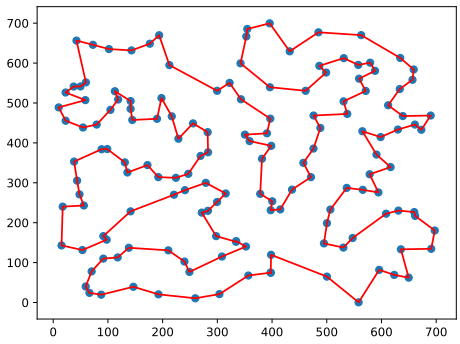
(a) DTSA



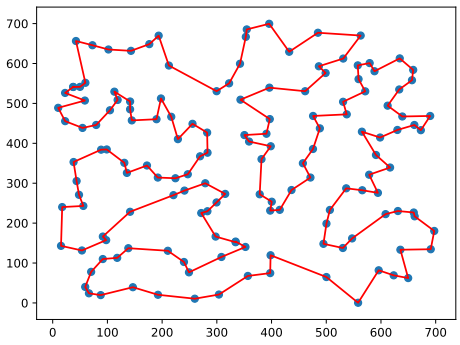
(b) GTSPA-MS



(c) GTSPA-SM

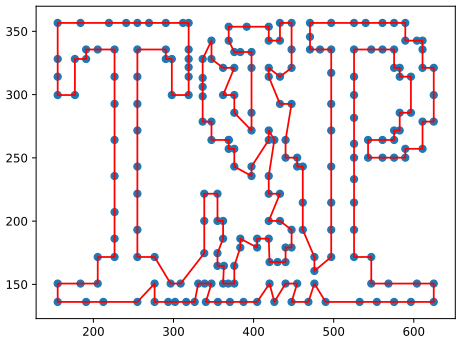


(d) GTSPA-SMS

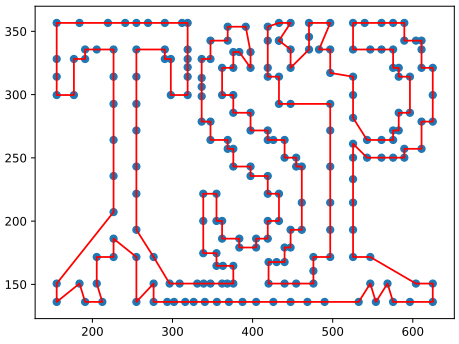


(e) optimal path

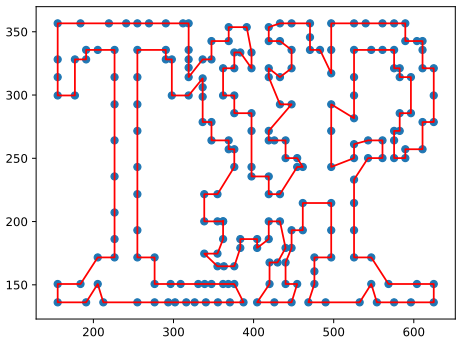
Figure 11: Final best paths for the different algorithms and optimal path for 'ch150' problem.



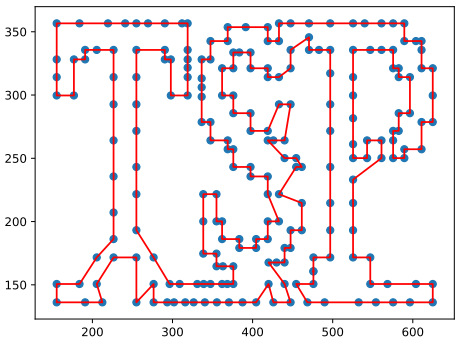
(a) DTSA



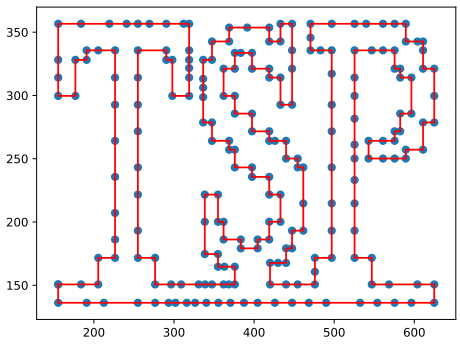
(b) GTSPA-MS



(c) GTSPA-SM



(d) GTSPA-SMS



(e) optimal path

Figure 12: Final best paths for the different algorithms and optimal path for 'tsp225' problem.

6.6 Source code

Link to a git repository with full code: `https://git.mafiasi.de/16beese/BAI-Seminar_Beese_Lohmann`