,

Lab report:
Biologically-inspired algorithms and models

Part I: Local Search, problem: symmetric TSP

April 19, 2023

Lecturer: dr hab. inż. Maciej Komosiński, prof. PP

Authors:	**Radosław Bodus**	inf145457	AI	radoslaw.bodus@student.put.poznan.pl
	Hubert Radom	inf145445	AI	hubert.radom@student.put.poznan.pl

Thursday classes, 16:50.

We declare that this report and the accompanying source code has been prepared solely by the above authors, and all contributions from other sources have been properly indicated and are cited in the bibliography.

Authors' contribution

- RB - algorithms implementation
- HR - experiments, report
- All authors have read and approved the complete, final report.

1 Problem description

The Symmetric Traveling Salesman Problem (TSP) is a well-known combinatorial optimization problem in which a salesman must visit a set of cities and return to the starting city, while minimizing the total distance traveled. The problem is symmetric because the distance between any two cities is the same regardless of the direction of travel.

The TSP has many real-world applications, including logistics, transportation, and scheduling. For example, a delivery company may need to find the shortest route between a set of destinations to minimize travel time and fuel costs. The TSP can also be interpreted as a graph problem, where cities are represented as nodes and the distance between cities is represented as edges.

The TSP is known to be NP-hard, meaning that there is no known polynomial-time algorithm that can solve all instances of the problem. As a result, many approximation and heuristic algorithms have been developed to find near-optimal solutions in a reasonable amount of time. Random search, random walk, nearest neighbor, greedy local search, and steepest local search are examples of such algorithms that can be used to find solutions for the TSP.

Random search and random walk are simple algorithms that explore the search space by randomly selecting cities to visit. Nearest neighbor is a heuristic that chooses closest node when creating a cycle. Greedy local search and steepest local search are more sophisticated algorithms that use local search techniques to gradually improve the quality of the solution. In practice, these algorithms can often find good solutions for real-world instances of the problem in a reasonable amount of time.

2 Implementation

C is the language we chose for our implementation as it is a good choice for implementing algorithms that require high performance and low-level control over program flow. C's syntax allow for efficient implementation of algorithms that require complex computations, and the language's lack of runtime overhead and garbage collector means that C programs can often be more efficient than programs written in higher-level languages like in Python.

Two neighborhood operators were used: two-nodes exchange and two-edge exchange. Two-nodes exchange involves swapping the position of two selected nodes in the tour that gives the best result, while two-edge exchange involves swapping the positions of two edges in the tour to create a new tour. These operators can generate a significant number of potential solutions. In our implementation, we use both at once.

Our implementation has been prepared for the *EUC_2D* data format, because it occurs most often among the instances available to us. When choosing the instances, we tried to have variety in their size.

2.1 Simulated Annealing

Simulated Annealing works by firstly setting high enough temperature so that the majority of the encountered solutions are accepted, then the temperature is lowered iteratively by every k iterations by a value of *alpha*. Higher k as well as *alpha* favors exploration of the solution space i.e. the algorithm is more willing to pick the deteriorating solution. Reversing the k and *alpha* to be lower we would favor the exploitation of the solution space.

We performed tests on ch150 instance of all combinations of alphas set to {0.8, 0.9, 0.95}, lengths of markov chains set to {0.02, 0.05, 0.1, 0.2} * size and acceptance rates set to {0.5, 0.75, 0.85, 0.95}. We obtained best average quality equals to 1.012 for alpha = 0.9, markov = 0.1, acceptance rate = 0.95. We futher explored similar parameters values. We performed tests with alphas set to {0.88, 0.92}, lengths of markov chains set to {0.08, 0.12} and acceptance rates set to {0.93, 0.97}. Average quality improved

to 1.009 with parameters alpha = 0.92, markov = 0.08, acceptance rate = 0.97. This values we used for futher computations.

2.2 Tabu Search

For Tabu Search we the algorithm which uses the "elite candidate moves" as a parameter denoted as k. The second parameter is tenure t which tells how many iterations a given move is tabu. In Tabu search in every iteration we pick the best move that is not forbidden by the tabu list, unless it meets the aspiration criteria which in our case means that the tabu move can be chosen if it creates solution that is better than the best found so far. Also at every iteration the solution that was picked becomes tabu and the tabu values are decreased so eventually they go out of tabu.

For Tabu Search we performed tests on a280 instance of all combinations of size of candidates list set to {0.1, 0.2, 0.5} * size and tabu tenure set to {0.1, 0.2, 0.5} * size. We obtained best average quality candidates = 0.5 and tenure = 0.2.

3 Algorithms comparison

3.1 Quality

Quality is the distance from the optimum defined as normalized measure that represents the performance of the algorithm relative to the best known value, as a percentage of the best known value:

$$\text{Quality} = \frac{\text{Score}}{\text{Best}}$$

This means that values closer to 1.0 are better. We think that such measure is intuitive and easy to understand, as it represents the ratio of the algorithm score to the best known value in a straightforward way. This measure can be used to rank algorithms in terms of their relative performance on a given optimization problem, because the common reference point is the best score itself, not other algorithms.

Average quality

The results of average quality measures are present in Table 1 and in Figure 1.

Random Search: This algorithm consistently performs the worst across all instances, with quality values ranging from 3.279 to 37.083. The standard deviations are relatively small, indicating that the algorithm's performance is consistently poor. As a result, the Random Search algorithm is not recommended for solving the TSP.

Random Walk: Similar to the Random Search algorithm, the Random Walk algorithm exhibits poor performance across all instances, with quality values ranging from 3.202 to 36.936. Although it performs slightly better than Random Search, its overall performance is still far from satisfactory. Thus, the Random Walk algorithm is also not recommended for solving the TSP.

Nearest Neighbor: The Nearest Neighbor algorithm shows a significant improvement in performance compared to the Random Search and Random Walk algorithms. The quality values range from 1.16 to 1.33, indicating that the algorithm can produce solutions reasonably close to the optimal solution. However, its performance is not as good as the Local Search Greedy and Local Search Steepest algorithms. The standard deviations are also larger than those of the latter two algorithms, suggesting less consistent performance.

Local Search Greedy: The Local Search Greedy algorithm demonstrates good performance across all instances, with quality values ranging from 1.057 to 1.081. This algorithm consistently produces solutions very close to the optimal solution. Additionally, the standard deviations are small, indicating stable and reliable performance. In comparison to the Nearest Neighbor and Local Search Steepest algorithms, the Local Search Greedy algorithm performs better in most instances.

Local Search Steepest: The performance of the Local Search Steepest algorithm is also reasonable, with quality values ranging from 1.055 to 1.116. While it performs slightly worse than the Local Search Greedy algorithm in some instances, it outperforms the Nearest Neighbor algorithm in all cases. The standard deviations are generally low, reflecting consistent performance.

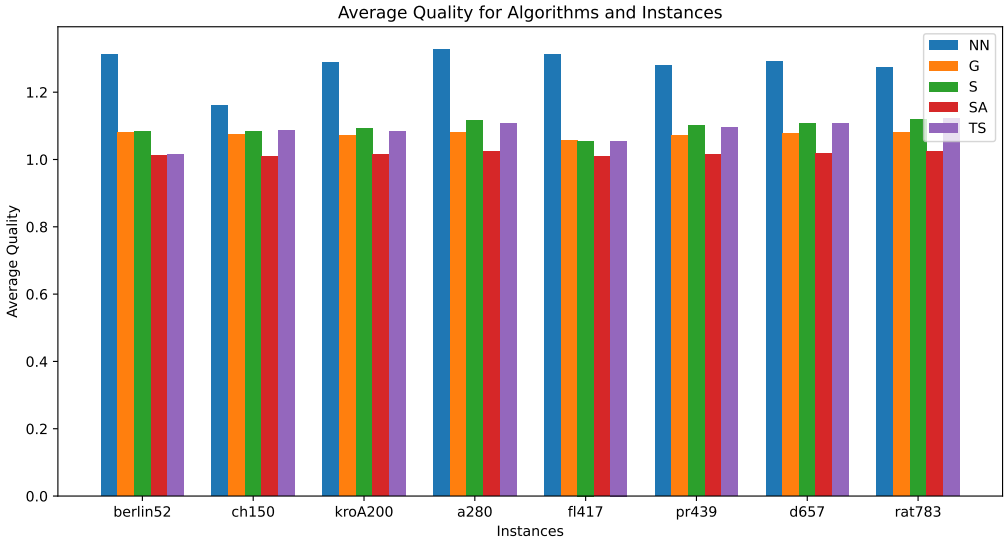

Figure 1: Plot showing average quality for given algorithms and instances. NN - Nearest Neighbor, G - Local Search Greedy, S - Local Search Steepest, SA - Simulated Annealing, TS - Tabu Search

Simulated Annealing: SA consistently outperforms all other algorithms in terms of average values ranging from 1.009 to 1.025 and their stability, exhibiting the best results across all TSP instances.

Tabu Search: TS provides competitive results to Local Search with quality ranges from 1.015 to 1.122. In most cases, it falls between Simulated Annealing and Steepest Local Search in the results.

Table 1: Table with quality for given algorithms and instances in format: average (standard deviation). RS - Random Search, RW - Random Walk, NN - Nearest Neighbor, G - Local Search Greedy, S - Local Search Steepest, SA - Simulated Annealing, TS - Tabu Search

Instance	Algorithms						
	RS	RW	NN	G	S	SA	TS
berlin52.tsp	3.279 (0.119)	3.202 (0.084)	1.312 (0.042)	1.080 (0.028)	1.083 (0.041)	1.011 (0.015)	1.015 (0.012)
ch150.tsp	7.216 (0.123)	7.123 (0.126)	1.160 (0.033)	1.073 (0.010)	1.083 (0.023)	1.009 (0.006)	1.087 (0.022)
kroA200.tsp	10.060 (0.086)	9.888 (0.096)	1.287 (0.046)	1.071 (0.026)	1.092 (0.021)	1.016 (0.008)	1.082 (0.023)
a280.tsp	11.714 (0.151)	11.663 (0.134)	1.328 (0.042)	1.081 (0.026)	1.116 (0.023)	1.024 (0.011)	1.108 (0.023)
fl417.tsp	37.083 (0.263)	36.936 (0.257)	1.311 (0.021)	1.057 (0.024)	1.055 (0.017)	1.009 (0.003)	1.055 (0.02)
pr439.tsp	16.024 (0.124)	16.087 (0.078)	1.28 (0.034)	1.070 (0.019)	1.101 (0.012)	1.015 (0.01)	1.095 (0.009)
d657.tsp	16.202 (0.073)	16.202 (0.068)	1.291 (0.022)	1.077 (0.011)	1.108 (0.014)	1.018 (0.009)	1.107 (0.009)
rat783.tsp	18.818 (0.074)	18.81 (0.116)	1.274 (0.018)	1.080 (1.120)	1.12 (0.008)	1.025 (0.007)	1.122 (0.004)

Best quality

The results of best qualities obtained for given instances are present in Table 2.

Random Search: The best quality values for the Random Search algorithm are consistently poor across all instances, ranging from 3.064 to 36.444. This further confirms that the Random Search algorithm is not suitable for solving the TSP due to its inferior performance.

Random Walk: Similar to the Random Search algorithm, the best quality values for the Random Walk algorithm are also poor, ranging from 3.035 to 36.422. While it outperforms Random Search in some instances, its overall performance is still far from satisfactory. Consequently, the Random Walk algorithm is also not recommended for solving the TSP.

Table 2: Table with best quality for given algorithms and instances.
RS - Random Search, RW - Random Walk, NN - Nearest Neighbor, G - Local Search Greedy, S - Local Search Steepest, SA - Simulated Annealing, TS - Tabu Search

Instance	Algorithms						
	RS	RW	NN	G	S	SA	TS
berlin52.tsp	3.064	3.035	1.227	1.039	1.000	1.000	1.000
ch150.tsp	6.939	6.928	1.087	1.058	1.053	1.000	1.048
kroA200.tsp	9.857	9.730	1.219	1.033	1.055	1.007	1.037
a280.tsp	11.421	11.363	1.244	1.038	1.082	1.009	1.060
fl417.tsp	36.444	36.422	1.288	1.019	1.037	1.006	1.024
pr439.tsp	15.801	15.885	1.205	1.053	1.083	1.004	1.049
d657.tsp	16.029	16.045	1.260	1.059	1.086	1.008	1.092
rat783.tsp	18.701	18.538	1.241	1.069	1.108	1.019	1.116

Nearest Neighbor: The Nearest Neighbor algorithm shows a substantial improvement in performance compared to the Random Search and Random Walk algorithms. The best quality values range from 1.087 to 1.288, indicating that the algorithm can produce solutions reasonably close to the optimal solution. However, its performance is not as good as the Local Search Greedy and Local Search Steepest algorithms.

Local Search Greedy: The Local Search Greedy algorithm demonstrates good performance across all instances, with the best quality values ranging from 1.019 to 1.069. This algorithm consistently produces solutions very close to the optimal solution.

Local Search Steepest: The performance of the Local Search Steepest algorithm is also good, with the best quality values ranging from 1.0 to 1.108. While it performs slightly worse than the Local Search Greedy algorithm in most instances, it still outperforms the Nearest Neighbor algorithm across all cases and in fact, it even achieves the optimal solution for the berlin52.tsp instance.

Simulated Annealing: The performance of the Simulated Annealing algorithm as was with the averages beats every other algorithm, with the best quality values ranging from 1.0 to 1.019. It easily outperforms every other algorithm.

Tabu Search: As was the case with average results, depending on the instance, Tabu Search achieves results similar to Local Search.

3.2 Efficiency, Time and Steps

Table 3: Table with average number of solutions evaluations
RS - Random Search, RW - Random Walk, NN - Nearest Neighbor, G - Local Search Greedy, S - Local Search Steepest, SA - Simulated Annealing, TS - Tabu Search

Instance Algorithms	RS	RW	G	S	SA	TS
berlin52.tsp	411	7951	107093	117572	745507	200020
ch150.tsp	4526	237625	4603948	3397200	15602783	600060
kroA200.tsp	9660	609583	12686728	8419911	41533511	800080
a280.tsp	16320	1502887	33242054	23308541	95532080	914377
fl417.tsp	43209	5300731	130640673	86138485	483003872	1660166
pr439.tsp	47571	6317586	180802177	99046595	427101051	1740174
d657.tsp	117850	19508847	574485871	336604752	1338900592	2620262
rat783.tsp	169046	29654975	1017117272	572287429	2349418122	3120312

The average numbers of solutions evaluations are present in Table 3. The average number of steps in local search algorithms are present in Table 4. The average running times of local search algorithms

Table 4: Table with average number of steps in local search

Instance	Algorithms	
	Local Search Greedy	Local Search Steepest
berlin52.tsp	176	44
ch150.tsp	677	152
kroA200.tsp	1016	212
a280.tsp	1450	298
fl417.tsp	2522	497
pr439.tsp	2773	515
d657.tsp	4446	781
rat783.tsp	5556	935

Table 5: Table with average running time in microseconds

Instance	Algorithms				
	Nearest Neighbor	LS Greedy	LS Steepest	Simulated Annealing	Tabu Search
berlin52.tsp	5	369	462	15855	12837
ch150.tsp	28	13405	10404	315101	50020
kroA200.tsp	51	49162	33870	868366	77588
a280.tsp	98	100655	70965	1952433	1396872
fl417.tsp	221	409604	274535	10481999	245068
pr439.tsp	245	564666	317962	9302031	266376
d657.tsp	614	1989497	1225700	33885664	592867
rat783.tsp	884	3705421	2216171	56258142	857427

Table 6: Table with efficiency for given algorithms and instances
RS - Random Search, RW - Random Walk, NN - Nearest Neighbor, G - Local Search Greedy, S - Local Search Steepest, SA - Simulated Annealing, TS - Tabu Search

Instance	Algorithms						
	RS	RW	NN	G	S	SA	TS
berlin52.tsp	1505	1481	6	397	499	16041	13025
ch150.tsp	75120	74067	32	14367	11260	317804	54328
kroA200.tsp	340808	334639	66	52506	37039	882494	83922
a280.tsp	831185	827628	129	108788	79186	1998837	1531176
fl417.tsp	10179484	10139970	290	431742	289653	10575772	258624
pr439.tsp	5095176	5114954	313	604532	350029	9442370	291786
d657.tsp	19859609	19859005	793	2142665	1357586	34476074	656282
rat783.tsp	41680906	41660672	1127	3999160	2483293	57664817	962136

are present in Table 5. Note that the Random Search and Random Walk algorithms are not included in the time analysis, as their running times are assumed to be equal to the average time of the Local Search Steepest algorithm. Random Search evaluates the smallest number of solutions due to the need to generate a new one in each step. Random Walk scans more solutions but less than Local Search, which only counts deltas, while Random Walk also changes the solution each time.

It is visible that considering only the quality of a given solution can be detrimental to the Nearest Neighbor algorithm. To measure efficiency of algorithm we propose such formula:

$$\text{Efficiency} = \text{Quality} * \text{Time}$$

This means we multiply quality times time. Both components mean the less the better. We want to punish Local Search and reward Nearest Neighbor for giving reasonably good results in a very short time.

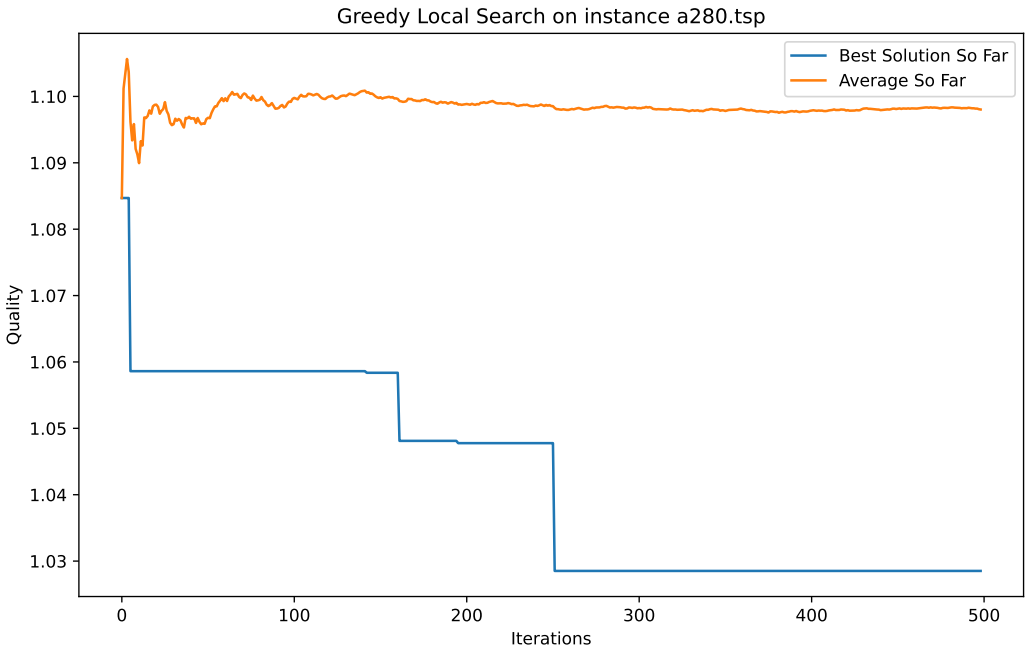

Figure 2: Plot showing the number of restarts vs. average and the best of solution in Greedy Local Search algorithm on a280.tsp instance.

Comparing results in Table 6 we clearly see the dominance of the Nearest Neighbor algorithm. Random algorithms remain the worst, but there is also a preference for Steepest Local Search over Greedy, which has longer average execution times. Simulated Annealing scores even worse than random because of the long execution time. Taking into account the execution time, Tabu Search turns out to be a good choice with similar results to Local Search, but shorter execution time.

3.3 Restarts

After analyzing the plots comparing the number of restarts versus the average and best solutions found so far for the Local Search Greedy (see Figure 2 and Figure 4) and Local Search Steepest (see Figure 3 and Figure 5) algorithms, we can draw several conclusions based on the selected TSP instances.

For a larger TSP instance of size 280, we observe that there is no significant improvement in the solution quality for the Local Search Greedy algorithm after 250 repetitions. This suggests that running the algorithm more than 250 times may not yield better results. However, for the Local Search Steepest algorithm, we still see an improvement in the solution quality even after 450 repetitions, where 500 is our maximum limit. This indicates that it might be worth continuing to run the Steepest algorithm for a larger number of restarts to potentially achieve better solutions.

For a smaller TSP instance of size 150, we find that there is no significant difference in the solution quality for both the Local Search Greedy and Local Search Steepest algorithms after 100 repetitions. This suggests that running either algorithm more than 100 times may not lead to substantial improvements in the solution quality for this particular instance.

3.4 Initial solution impact

Based on the analysis of the plots shown in Figure 6, in Figure 7 and in Figure 8 depicting the relationship between the quality of the initial solution and the quality of the final solution for the Local Search Greedy and Local Search Steepest algorithms, we can conclude that there is no significant correlation

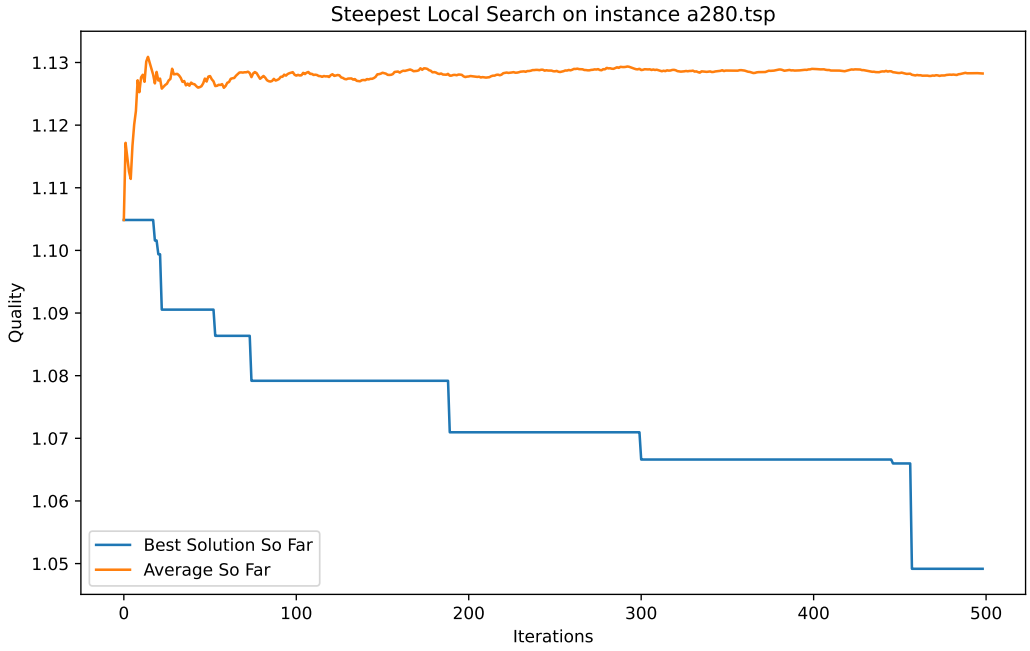

Figure 3: Plot showing the number of restarts vs. average and the best of solution in Steepest Local Search algorithm on a280.tsp instance.

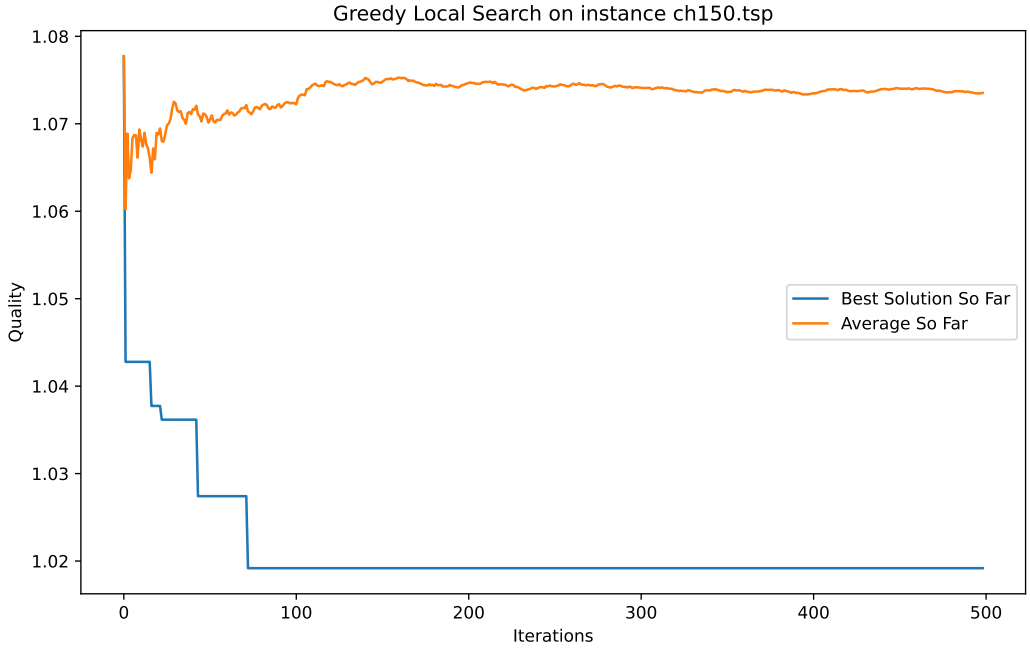

Figure 4: Plot showing the number of restarts vs. average and the best of solution in Greedy Local Search algorithm on ch150.tsp instance.

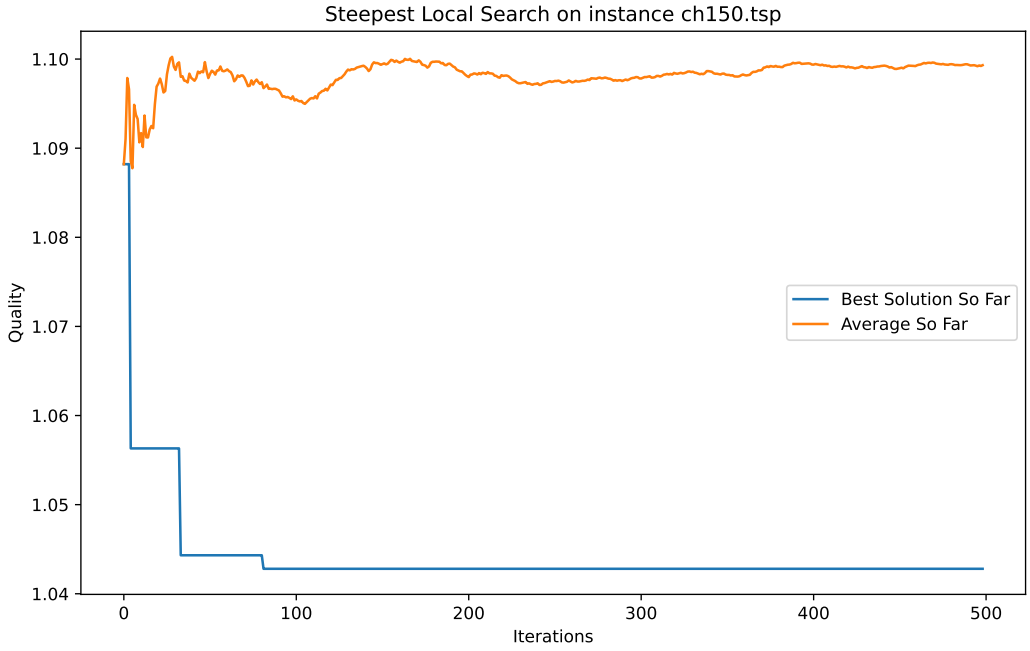

Figure 5: Plot showing the number of restarts vs. average and the best of solution in Steepest Local Search algorithm on ch150.tsp instance.

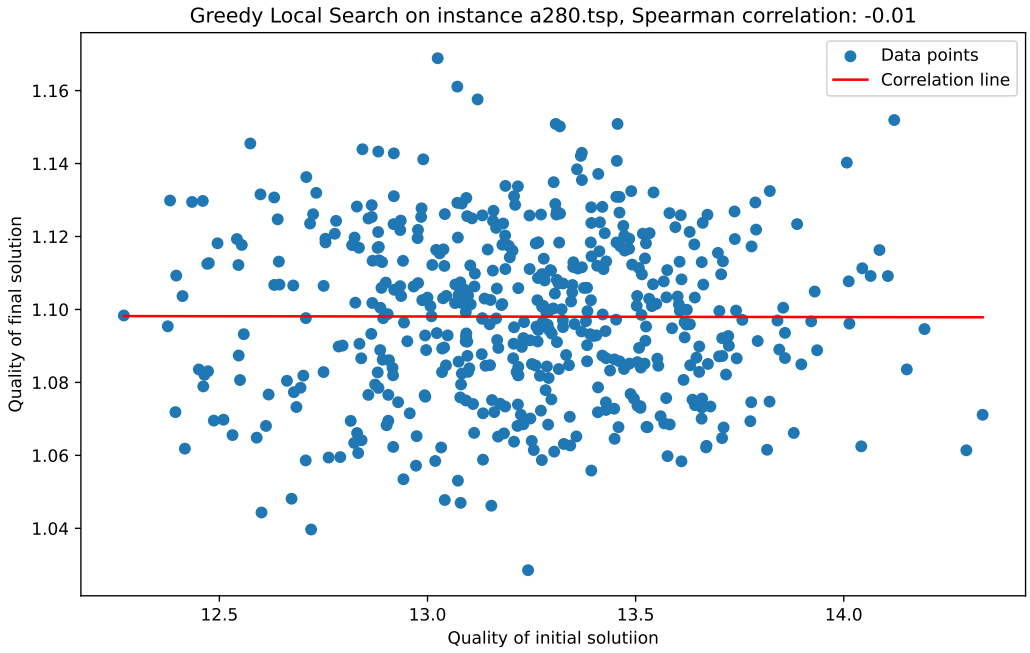

Figure 6: Plot showing the quality of initial solution vs quality of the final solution in Greedy Local Search algorithm on a280.tsp instance.

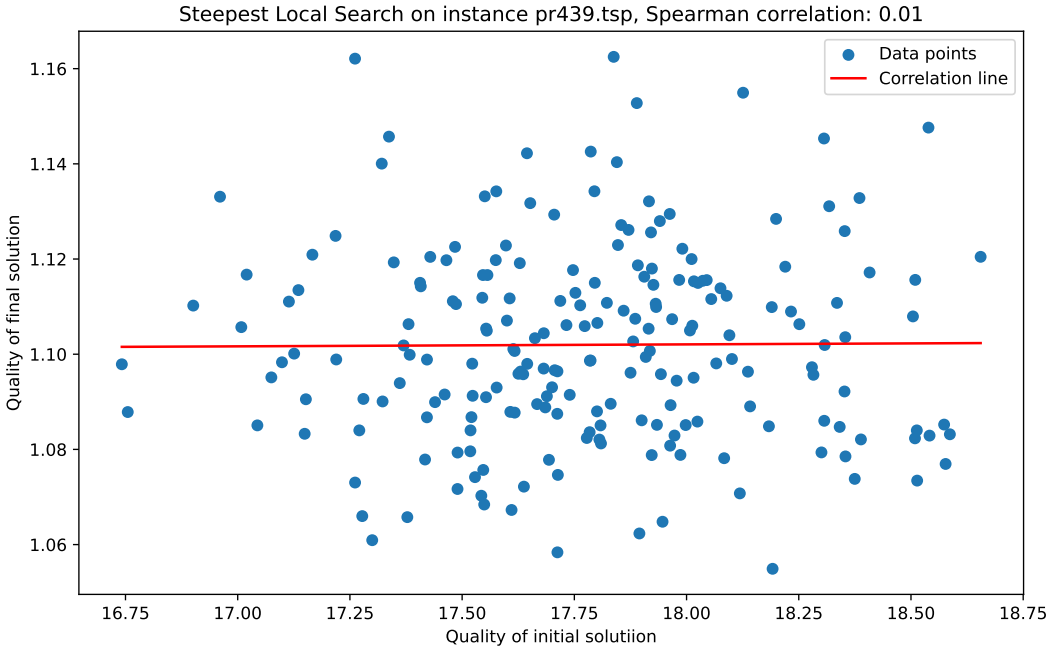

Figure 7: Plot showing the quality of initial solution vs quality of the final solution in Steepest Local Search algorithm on pr439.tsp instance.

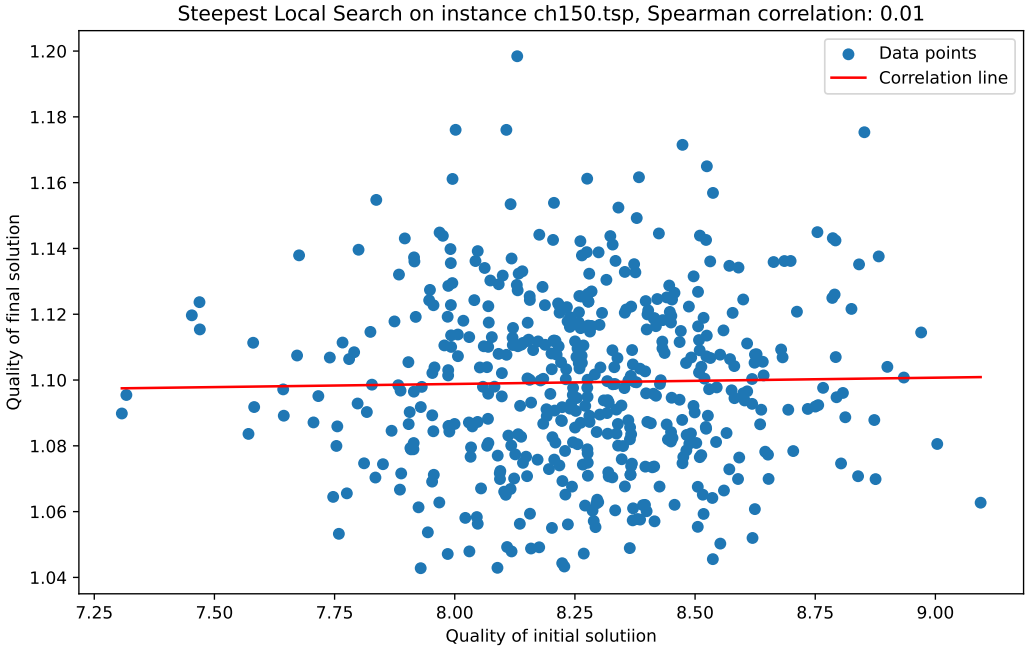

Figure 8: Plot showing the quality of initial solution vs quality of the final solution in Steepest Local Search algorithm on ch150.tsp instance.

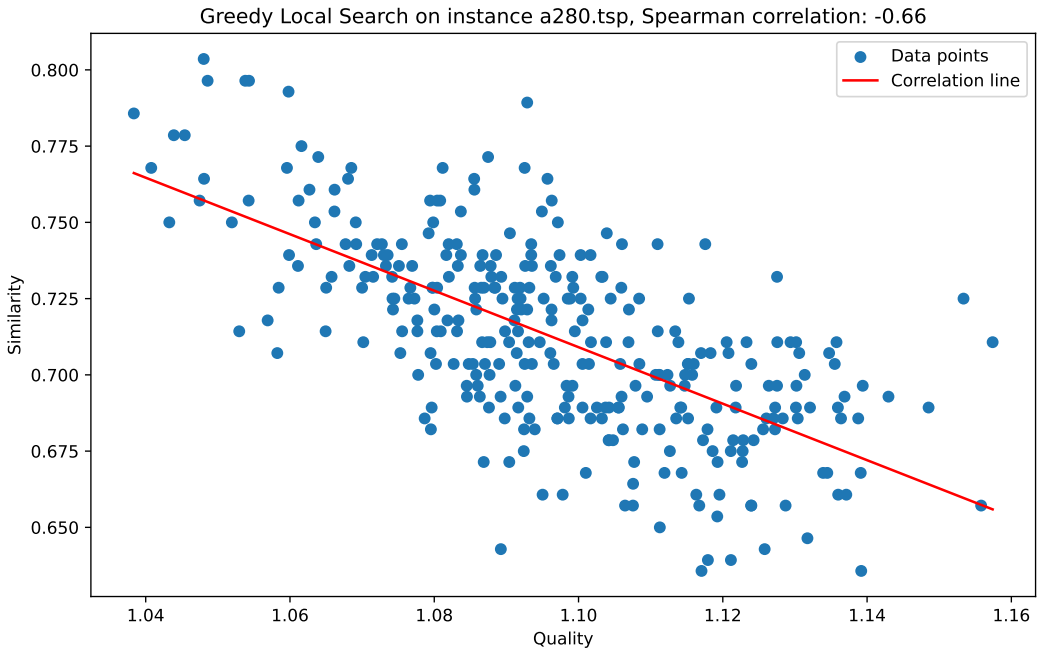

Figure 9: Plot showing the quality of solution vs similarity to optimum solution in Greedy Local Search algorithm on a280.tsp instance.

between these two factors. The lack of correlation between the initial and final solution quality implies that the performance of the Local Search Greedy and Local Search Steepest algorithms is not primarily influenced by the quality of the starting solution. Instead, it suggests that these algorithms are capable of exploring the solution space effectively, escaping local optima, and converging towards high-quality solutions regardless of the initial solution provided.

3.5 Similarity

We chose two instances to investigate correlation between quality of solution and similarity to the optimum solution. The similarity measure we used in this analysis is the number of common edges between the compared solutions divided by number of all edges in solution. Our analysis indicates that better-quality solutions tend to be more similar to the optimal one. This suggests that as the algorithms progress towards higher-quality solutions, they also converge to solutions that share more similarities with the optimal solution in terms of their structure. There is no difference between algorithms as for a280.tsp instance Greedy Local Search algorithm (see Figure 9) shows a greater correlations than Steepest Local Search algorithm (see Figure 10), but for ch150.tsp instance Steepest (see Figure 12) shows greater correlation than Greedy (see Figure 11).

4 Summary

The Local Search Greedy, Local Search Steepest, Simulated Annealing and Tabu Search algorithms deliver superior and more consistent results than the Nearest Neighbor, Random Search, and Random Walk algorithms for the instances tested. Simulated Annealing algorithm, in particular, demonstrates the best overall performance in terms of solution quality.

The Local Search Greedy and Steepest algorithms also demonstrates respectable results, making them a viable alternative in certain scenarios. Both the Nearest Neighbor, Random Search and Random Walk

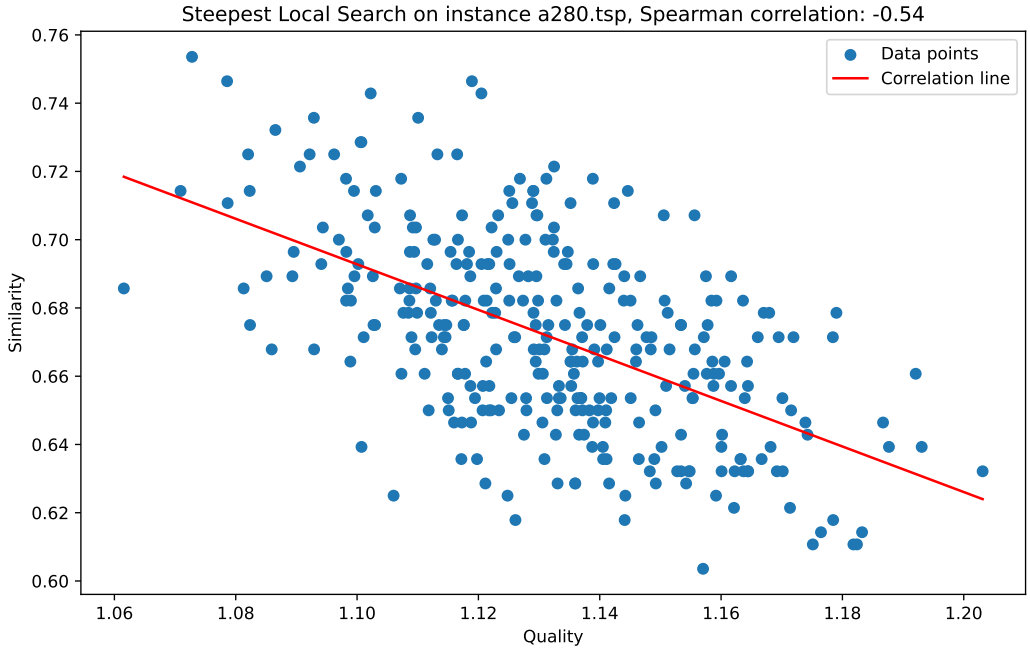

Figure 10: Plot showing the quality of solution vs similarity to optimum solution in Steepest Local Search algorithm on a280.tsp instance.

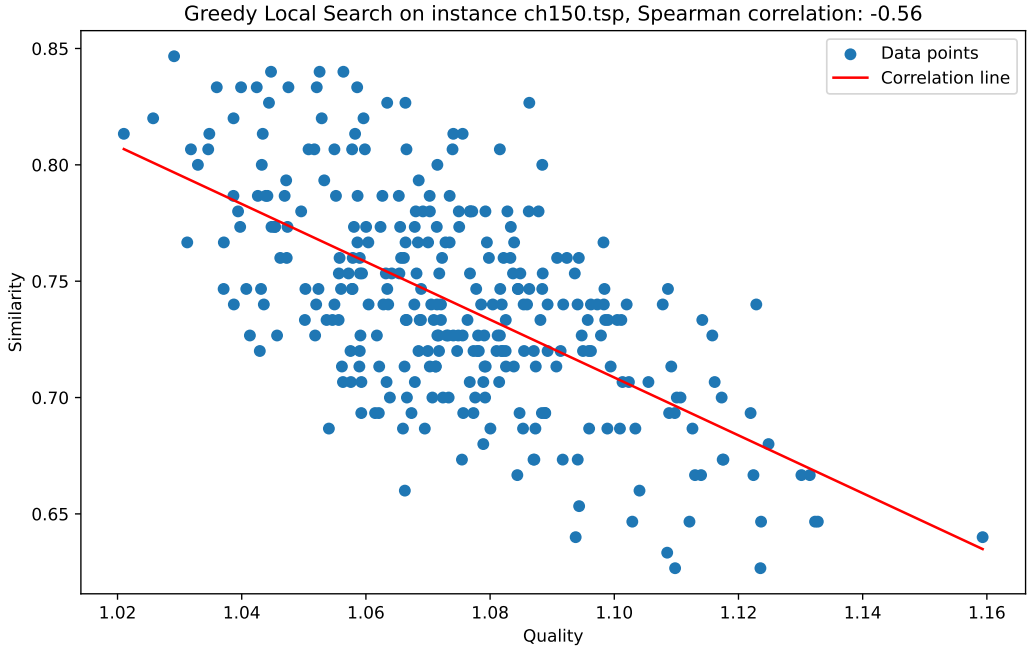

Figure 11: Plot showing the quality of solution vs similarity to optimum solution in Greedy Local Search algorithm on ch150.tsp instance.

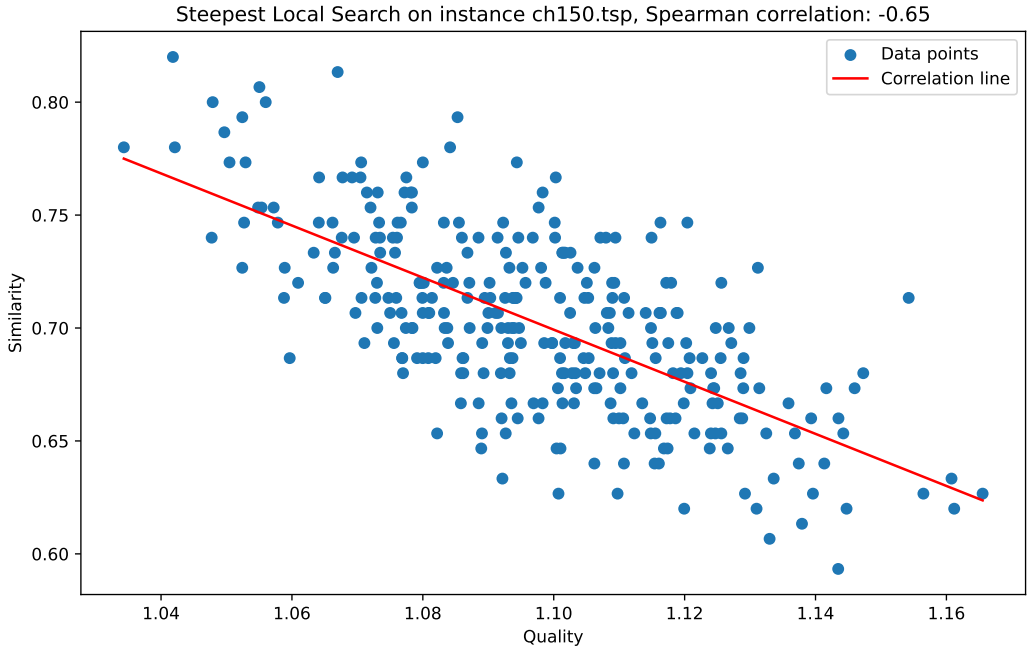

Figure 12: Plot showing the quality of solution vs similarity to optimum solution in Steepest Local Search algorithm on ch150.tsp instance.

are not recommended for solving the TSP due to their inferior performance. However, Nearest Neighbor is a good choice if you want to find a relatively good result (much better than random) in a very short time due to its high efficiency. Simulated Annealing and Tabu Search algorithms perform well in comparison to other algorithms, demonstrating their effectiveness in solving TSP instances. Both algorithms consistently achieve solution qualities near the optimal solution, with Simulated Annealing outperforming Tabu Search and all the other algorithms. These algorithms offer reliable alternatives to Local Search Greedy and Local Search Steepest algorithms. All algorithms offer a tradeoff between time and score for example Simulated Annealing as the best algorithm comparing quality, loses comparing execution times.

The optimal number of repetitions varies depending on the algorithm and the problem instance size. For the larger TSP instance (size 280), it may be beneficial to run the Local Search Steepest algorithm with more restarts (beyond 500) to potentially achieve better results, while the Local Search Greedy algorithm does not show significant improvements beyond 250 repetitions. For the smaller TSP instance (size 150), both algorithms reach a plateau in solution quality after 100 repetitions, suggesting that further restarts may not be necessary.

Observation of lack of correlation between initial and final solutions is valuable, as it indicates that the choice of the initial solution is not a critical factor in determining the success of the Local Search Greedy and Local Search Steepest algorithms in solving TSP instances. Instead, the focus should be placed on other aspects of the algorithms, such as their search strategies, and parameters, to further improve their efficiency in solving the TSP.

This finding about similarity of optimum and good solutions demonstrates that the similarity between locally optimal solutions and the global optimum can provide insights into the performance of the algorithms and can help guide the search towards creating better algorithms. Additionally, it emphasizes the role of the edge structure in determining the quality of solutions in the TSP and can inform the design of heuristics or search strategies that exploit this relationship.

One of the main problems encountered during the implementation of the algorithm was how to debug badly allocated memory. Memory allocation was supposed to occur in the program only once during the execution of the algorithm, but when running the experiments many times, we automated the program

so that we would not run it multiple times, but that the program would load subsequent instances many times during one run and run the algorithms on them. This part required memory allocation. As for the TS the most difficult part was to get the best parameters which we haven't probably manage to get the optimal set.