A survey of the Traveling Salesman Problem using parallel simulated annealing and greedy heuristics

Trevor Degruccio, Aaron Fox
Google Drive Link:
https://drive.google.com/drive/folders/1XExeW2LJhJDNMZ5vOyqFBrzbR0a3InNX?usp=sharing
Final Project for the Fall 2020 CSE 625 Combinatorial Optimization Course
Computer Science and Engineering Department, University of Louisville
Louisville KY 40292 USA
trevor.degruccio@louisville.edu
aaron.fox@louisville.edu

***Abstract*— This experiment sets out to survey the benefits of using parallel simulated annealing and compare the strengths and weaknesses of this metaheuristic to a more classical method that has long been in the literature, the local greedy heuristic. We are investigating the simulated annealing algorithm as it has become an increasingly popular optimization algorithm that has many real-world applications for solving optimization problems today. We would therefore like to compare the impacts of parallelizing simulated annealing and evaluate the impact on solving an NP-hard problem. In order to accomplish this, we use both parallel simulated annealing and the local greedy heuristic as a means of solving the famous Traveling Salesman Problem to evaluate the time complexity, optimality of results, and requirements of tweaking the input parameters and setup of both algorithms. The Traveling Salesman Problem was chosen as it is a well-known problem in the literature and will therefore serve as a useful metric for comparison of the two techniques evaluated in this experiment.**

***Keywords*— Simulated Annealing, Traveling Salesman Problem, Greedy**

1. Introduction

The Traveling Salesman Problem (TSP) is a well-known problem in computer science and is one of the most studied combinatorial optimization problems known in the literature [1]. The gist of the traveling salesman problem concerns the following: given a set of cities that are connected via roads, what is the shortest path that goes to every city exactly once and then returns to the starting city? The TSP has been the face for NP-hard problems in the field of combinatorial optimization since its original formulation in 1930. Despite the inherent computation complexity of TSP, with brute-force exhaustive searches heading toward worst cases of O(N!), there have been numerous metaheuristics and exact algorithms that serve to solve the TSP efficiently, with the algorithms trying to balance computational complexity, optimality of the final solution, and ease of use in adjusting the parameters for finding each solution. The applications of the generalization of the TSP stretch far and wide: some examples are: finding the shortest path of balancing a network of virtual machine server farms to most efficiently place them for communication, finding an optimal route for a GPS system, modelling which holes to most effectively drill in a circuit board, and how to efficiently organize book stockers in a library, to name a few real-life applications.

Our main goal of investigating how to solve the TSP is through the use of simulated annealing. Simulated annealing is a metaheuristic function that can be used to approximate the global optimum of a function. In our approach we will use the simulated annealing technique and parallelize it to take an asynchronous approach to further reduce execution time and make better use of the hardware we have available.

The following table includes a brief survey of relevant literature on previous efforts of solving the Traveling Salesman Problem:

Reference	**Name of Algorithm**	**Category (as in question 8 below)**	**Known complexity**
S. Zhan, J. Lin, Z. Zhang, Y. Zhong, "List-Based Simulated Annealing Algorithm for Traveling Salesman Problem", *Computational Intelligence and Neuroscience, vol.* 2016.	List-Based Simulated Annealing	CompleteEvaluation, Approximate Solution: Simulated Annealing	O(m*n*k), where m represents the size of the rows of the cities distance matrix, n represents the size of the columns of the cities distances matrix, and k represents the number of cities to include.
J. D. C. Little, K. G. Murty, D. W. Sweeney, and C. Karel, "An Algorithm for the Traveling Salesman Problem," Operations Research, vol. 11, no. 6, pp. 972–989, 1963.	Branch and Bound	Total and exact evaluation: Branch and Bound	Worst case of O(N!), where N represents the number of cities in the TSP.
J. Grefenstett, R. Gopal, B. Rosmaita, D. Van Gucht. "Genetic algorithms for the traveling salesman problem." Proceedings of the first International Conference on Genetic Algorithms and their Applications. Vol. 160. No. 168. Lawrence Erlbaum, 1985.	Genetic Algorithm for TSP:	Total and Approximate evaluation: genetic algorithm.	O(G *(N*M)), where G represents the number of generations, N the population size, and M the number of cities in the TSP.
D. J. Rosenkrantz, R. E. Stearns, and P.M. Lewis, II. "An Analysis of Several Heuristics for the Traveling Salesman Problem." SIAM J. Comput., vol. 6, no. 3, pp. 563–581, 1976.	Nearest Neighbor	Total and Approximate Evaluation : local greedy heuristic.	O(N^2*(log(N))

Table 1. References to TSP optimization problem

2. Optimization Problem

In the TSP problem the optimization goal is trivial. The salesman wants the shortest tour given a set of cities because he wants to make all of his sales in the shortest amount of time. The shortest tour is defined by the distance it takes to get around the tour as defined by the cost function below.

Cost Function:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

For Simulated Annealing there are specific variables required that define the algorithm and the way it works as defined below

Variables:

- T = Temperature
- Snew = Initial State
- Sfinal = Final State
- E = Energy
- N() = Candidate Generator Function
- P() = Acceptance Probability Function

Parameters:

- Tmax = Max Temperature
- Tmin = Minimum Temperature
- Egoal = Energy Goal
- N = Number of cities

Constraints:

- No Constraints

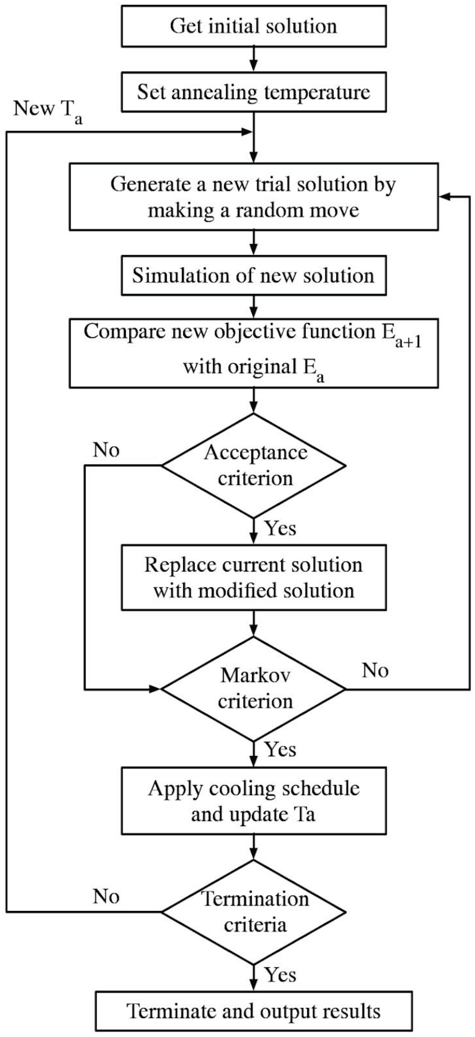

Get initial solution
Set annealing temperature
New T_a
Generate a new trial solution by making a random move
Simulation of new solution
Compare new objective function E_{a+1} with original E_a
Acceptance criterion
No
Yes
Replace current solution with modified solution
Markov criterion
No
Yes
Apply cooling schedule and update Ta
Termination criteria
No
Yes
Terminate and output results

Figure 1. Simulated Annealing Flowchart [2]

3. Search Space

The search space of a TSP problem since there are no hard constraints has a feasible search space of size N! as noted in Figure 2 in the Appendix, where N corresponds to the number of cities to place in the path. One constraint of the search space is that all potential paths must include every city exactly once in the route, with the exception of the starting city having to be returned to at the end of the route to connect the cyclical path.

4. Difficulties

A central difficulty encountered while executing this experiment concerned choosing the optimal parameters to use for both the parallel simulated annealing metaheuristic and the local greedy heuristic. Because we judged that it is only fair to use the best possible formulation of both of these algorithms when comparing them to allow both algorithms to be on level ground, we had to spend much time tweaking the multivariate equations that arose in adjusting the parameters of the functions to determine which inputs and variable adjustments could lead to the optimal results. For a small example of this difficulty, consider how changing the Acceptance Probability function of the simulated annealing algorithm and how its results perform much better with a higher maximum temperature variable. These two variables and parameters therefore can greatly impact the overall performance of the metaheuristic, and, if we did not find the optimal inputs and variables to use, our results would be skewed and misinformed. After determining that we had adjusted and tweaked the respective algorithm's functions, we took great care in ensuring they were impacting each algorithm's performance in the best way.

Another issue we ran into is consulting the true optimality of each search space. As there are N! possible combinations of routes to consider in the search space, even using specialized exact algorithms made finding the optimal solution's cost difficult given the nature of the NP-hard problem. The analysis of each method therefore took a fair amount of time in tabulating our data and comparing it with the best solutions. Additionally, the computational complexity of each solution being O((N^2+N)*log(N))) and O(N^2*(log(N)) for the simulated annealing and local greedy heuristics, respectively, increasingly highlighted the factor of time in this experiment as one of the main difficulties.

5. Related Problems

Given that we are indeed solving the classical Traveling Salesman Problem in this experiment, we are working on a problem that has been shown many times over in the literature to be NP-hard. The fact that the TSP is NP-hard can be proven by reducing the Hamiltonian Cycle problem down to the Traveling Salesman Problem [5]. Similar problems, such as the generalization of the TSP with the travelling purchaser problem and the vehicle routing problem, are also NP-hard in nature. Other NP-hard problems, such as finding a Hamiltonian cycle with the least weight of a weighted graph and the Bottleneck TSP also prove to be NP-hard due to their relationship with the original TSP proposed in 1930 that is used in this experiment.

6. Libraries Used

In order to carry out these simulations, the open source simanneal module from PyPi, the official third-party software repository for Python. The module was adjusted to fit the needs for parallelization required in this function as it was only originally designed for single threads use. Additionally, the matplotlib Python module was used for visualizing the results graphically. The data used in finding the top 30 most populous US cities and their latitude and longitude come from Wikipedia.

7. Optimization Methods

The TSP problem has a variety of optimization methods that can be utilized to attempt the minimum tour. In this experiment we compared the results between a traditional Greedy approach to the TSP and our parallel simulated annealing approach. A traditional greedy approach is rather trivial in the approach to the TSP problem where given a starting city the greedy algorithm will use the minimal distance to choose the nearest neighbor. This approach is good to an extent but it can cause problems for the general Symmetric and Asymmetric TSP [4]. The Simulated Annealing metaheuristic algorithm is a probabilistic technique for solving the global optimum of a function; in the case of the TSP, this is the minimum possible distance of a tour.

When comparing the main difference between Simulated Annealing and a greedy approach in the case of a TSP, it is that simulated annealing has a much lower chance of getting stuck in a local optima and continue to explore a larger search space over a large number of iterations and find the best solution [6]. Greedy approaches will only solve the TSP problem one initial way however the Simulated Annealing approach iterates over and over until it can converge on a solution and that's why it ultimately performs and gives a better solution.

Additionally, the simulated annealing approach was parallelized. We chose to parallelize the simulated annealing algorithm because of the nature of its algorithm. Due to the random nature of simul;ated annealing, sometimes the resulting energies and therefore routes of the the TSP are excellent and fine; other times, the results are lackluster. Therefore, by parallelizing the code, we are able to run the code in parallel and use the most optimal route from all the threads that were running the same function. This leads to faster and better performing results. In order to do this, we used the built-in threading module of Python.

8. Algorithm's Pseudocode

The pseudocode of the simulated annealing algorithm is as follows:

```
1: For each thread:
2: Let state = initial_state
3: For iteration = 0 to maximum_iteration_num:
4:          state_new = neighbor(state)
5:          temp = calculate_temp(iteration, temp)
6:          if probability of energy of the state and temp
7:            > random variable from 0 to 1:
8:                    state = state_new
9: Return state
```

Pseudocode for neighbor function:

```
1: Let neighbor be a copy of the current city array
2: Simply swap two cities in neighbor to make it
3: different from original city array
4: Return neighbor
```

Pseudocode for energy of state:

```
1: Energy = 0
2: For i = 0 to length of current city array - 1:
3:          energy += distance(city[i], city[i+1])
4: Return Energy
```

Pseudocode for distance used in energy:

```
1: distance = square root of input ((city_2.x - city1.x)
2: ^2 + (city_2.y - city1.y )^2)
```

Pseudocode for temperature calculation:

```
1: If T = 0:
2:          Attempt Guess for T based on progression
3:          with steps
3:          Solve for Tmax that give 98% acceptance
4:          Solve for Tmin that gives 0% improvement
```

Pseudocode for candidate generation function:

```
1: If e^(ΔE/T) < random value from (0,1):
2:          Select neighbor candidate determined by
3:          neighbor function
4: Else:
5:          Keep current state
```

The pseudocode is fairly straightforward and illustrates the process of how the annealing metaheuristic functions. Starting at an initial state supplied as an input parameter, continue iterating until the required number of maximum_iteration_num have been met in the for loop. Every iteration sees a call to the neighbors of the current state and then, depending on random probabilities and the current temperature (with a hotter temperature increasing the likelihood of the probability being true), set the value of the current state to the new state found in the neighbors solution. The schedule of annealing is therefore determined by the calculate_temp method which determines the rate of temperature to use and is helped to determine by the input parameters to the function.

The pseudocode for the candidate generation function with the following equation of

$$e^{\frac{\Delta E}{T}} > \mathrm{rand}(0,1)$$

shows how, at higher temperatures, a bad choice is more likely to be selected even if it has a worse change of energy (meaning little improvement was made with the current solution), allowing for the simulated annealing approach to escape local optima, an important factor in expanding the search space of the algorithm. At lower temperatures, however, a bad solution is less likely to be selected, helping to ensure good solutions are still kept in the algorithm as well.

The pseudocode for the greedy algorithm consists of the following:

```
1: Do for every starting city and take best route:
2: Initialize array holding indices of cities = indices
3: Initialize array for results including first city = route
4: Mark visited cities array =  visited
5: min = infinity
6: Initialize city indices = city_i, city_j
7: while results array does not have every city and there are
still cities left to add:
8:      if path has not be visited and current city < min:
9:          min = dist(city_i, city_j)
10:     increment city_j
11:     Add all paths from city_i to visited
12:     Update final city with best cost visited to route
```

And the pseudocode for the neighbor function of the greedy code is as follows:

```
Neighbor Function:
1: min_distance = INF
2: min_city = NULL
3: For every unvisited city:
4:      if distance from city to city < min_distance:
5:          min_distance = this distance
6:          min_city = this city
7: return min_city, min_distance
```

Effectively, the greedy heuristic works by holding two arrays of data: one for the indices of the cities based on the input distance matrix of all the cities, and one array that holds the results of all the cities which have already been visited. Using these arrays, the algorithm traverses over the city distance matrix for every city and, if the given distance to go to any city from the current city of the algorithm is the lowest, update the minimum city. Then, based on the minimum city to reach from each current city, greedily generate the final route path based on the minimum distance from each city.

9. Algorithms Relation To Existing Method

With respect to algorithms studied in class lectures and projects, the simulated annealing algorithm was one of the metaheuristics we studied. The simulated annealing metaheuristic is a complete evaluation and is also an approximate algorithm and doesn't always return the (exact) optimal solution. In comparison the greedy algorithm is also a total evaluation and approximate solution as often the greedy algorithm will produce a poor result.

10. Complexity

The time complexity of both the greedy and simulated algorithms are the same. The input size of the problem depends on the total n cities that are needed to traverse in the TSP problem. In simulated annealing to solve for the TSP it goes through O(log(n)) temperature steps and for each temperature the search examines O(n) attempts and accepts any potential changes. Rejected changes are done in O(1) time while accepted changes require O(n) reversal in city exchanges [7].

Algorithm	**Time Complexity**
Greedy	$O(N^2 \log(N))$
Simulated Annealing	$O((N^2 + N)\log(N))$

Table 2. Complexities

11. Experimental Methodology

To run the experiments, both the parallel simulated annealing and greedy algorithms were designed and implemented in Python and ran on a Windows 10 computer. Each experiment was run 10 times and the best path results were determined to see how, on average, each algorithm did in determining the optimality of a set of cities. Each city set was derived from the top 30 most populous cities in the United States and their respective latitudinal and longitudinal coordinates. The distance between each latitudinal and longitudinal coordinate of the cities was then calculated based on the radius of the earth.

The four different sizes of city sets compared include the five most populous cities, the ten most populous cities, and the twenty and thirty most populated cities. These varied city sizes were useful in determining how both algorithms compared in this study did with each size of cities.

12. Results

The results of the experiment can be seen in the Appendix.

13. Results Discussion

After running the experiments ten times for each city set and each method, the results were tabulated, analyzed, and graphed. It can be seen that in the results, while differences on smaller data sets aren't as prominent between the optimality of the greedy and simulated annealing heuristics, it can be seen that, once we reach at least 10 cities in size and larger, the results begin to bifurcate much more starkly, as seen in Table 3. The table illustrates how, as the dataset gets larger, the difference in finding a more optimal answer increases heavily toward the parallel simulated annealing metaheuristic.

Number of Cities	Simulated Annealing Miles	Greedy Miles	Difference in Miles
5	5248	5248	0
10	5898	5924	26
20	6801	7030	229
30	8423	8882	459

Table 3. Results Analysis

When visualizing the results figures in the appendix, it becomes apparent that, as more steps occur, the rate of acceptance dwindles as the tour becomes increasingly optimized, such as in Figure 11. Figure 11 serves to illustrate how simulated annealing works: by fluctuating the acceptance rate of which candidate solutions to accept while the temperature is higher (the correspondence between Figure 13 and Figure 11 is apparent), less optimal solutions are able to be accepted earlier on as the temperature is high, allowing for the escaping of local optima. This is the central reason why simulated annealing consistently outperforms the greedy heuristic: the accepting of poor solutions also allows it to accept better solutions in the long run. In Figure 12, the rate of change in the distance covered by each step also shows how the rate of change dwindles as simulated annealing mimics heating up and cooling down. Rather than shooting immediately for the best distance as the greedy heuristic does, the simulated annealing algorithm will counterintuitively go toward worse distances and find less optimal routes. The relationship between the increased temperature and higher variance of change in the distance of the routes corresponds: as the temperature is higher, more rapid changes in slopes from negative to positive are apparent. In contrast, as the temperature drops, the changes in slopes are less dramatic as only more optimal candidates are accepted. Lastly, Figure 13 the temperature vs steps shows how the simulated annealing algorithm drastically heats up and then cools down quickly, simulating the metallurgy metaphor.

14. Conclusion

In conclusion, we were able to successfully compare and contrast a well known metaheuristic and greedy approach against each other as a means of finding useful routes for the classical TSP problem. Simulated annealing proved to be a better solution to the TSP problem vs the greedy approach which didn't explore outside of the local optima. Through the Heating and cooling approach the simulated annealing heuristic was able to explore more possible solutions and ultimately return better results than the greedy approach. When comparing the time analysis of the two optimization methods the complexity was about that same however, the simulated annealing produced the better result and is thus the better choice. By solving the TSP with simulated annealing we were able to prove the capability of simulated annealing in other real world problems that can be derived from the classical TSP. By making the simulated annealing parallel we were able to further reduce execution time and make more use of the CPUs we have available to us. The simulated annealing approach can be utilized on a variety of problems and there is a lot more future work that can be done for problems that need a global optima solved!

15. References

[1] M. Bellmore and G. L. Nemhauser, "The Traveling Salesman Problem: A Survey," *Operations Research*, vol. 16, no. 3, pp. 538–558, 1968.

[2] S. Zhan, J. Lin, Z. Zhang, Y. Zhong, "List-Based Simulated Annealing Algorithm for Traveling Salesman Problem",

Computational Intelligence and Neuroscience, vol. 2016.

[3] OptaPlanner. “Is the search space of an optimization problem really that big?,” *OptaPlanner*. 2014.

[4] G. Gutin, A. Yeo, and A. Zverovich, “Traveling salesman should not be greedy: domination analysis of greedy-type heuristics for the TSP,” Mar. 2002.

[5] OpenDSA Project, “Reduction of Hamiltonian Cycle to Traveling Salesman¶,” in *OpenDSA Data Structures and Algorithms Modules Collection*, OpenDSA Project, Ed. .

[6] P. Akella, “*Comprehensive Simulated Annealing*.”

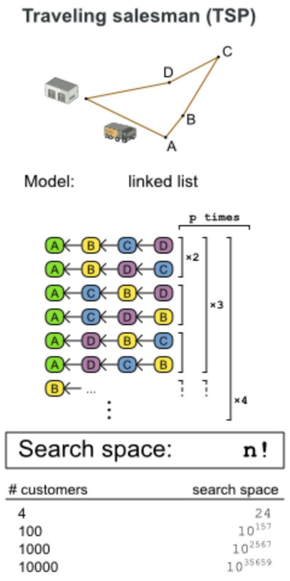
[7] P. B. Hansen, “*Simulated Annealing*,” 1992.

16. Time Plan and Team Plan

Task	Who	When	Completed?
Research for Final Presentation Topic	Aaron/Trevor	11/15/2020	11/18/2020
Research of Simulated Annealing & TSP Research papers	Aaron/Trevor	11/18/2020	12/09/2020
Find Simulated Annealing Code Resources	Aaron/Trevor	11/30/2020	12/4/2020
Create Github	Aaron	12/5/2020	12/5/2020
Modify Code and Plot Graphs using Matplotlib	Aaron/Trevor	12/5/2020	12/7/2020
Create Initial Powerpoint Intro	Aaron/Trevor	12/5/2020	12/5/2020
Create Final Report and Powerpoint	Aaron/Trevor	12/5/2020	12/09/2020

Table 3. Task Completion

17. Appendix



# customers	search space
4	24
100	10^{157}
1000	10^{2567}
10000	10^{35659}

Figure 2. TSP Search Space Example [3]

The resulting figures from each city set variant can be seen below:

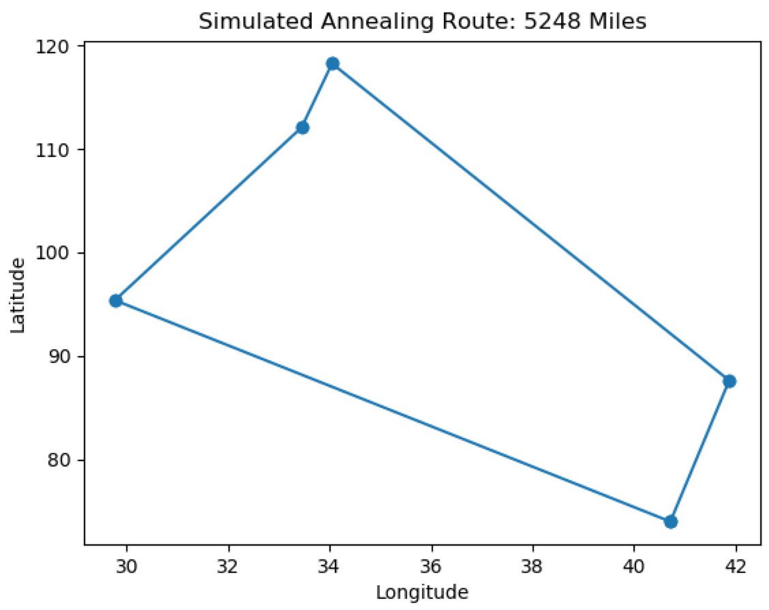


Figure 3. Route determined by the parallel simulated annealing metaheuristic on a set of 5 cities

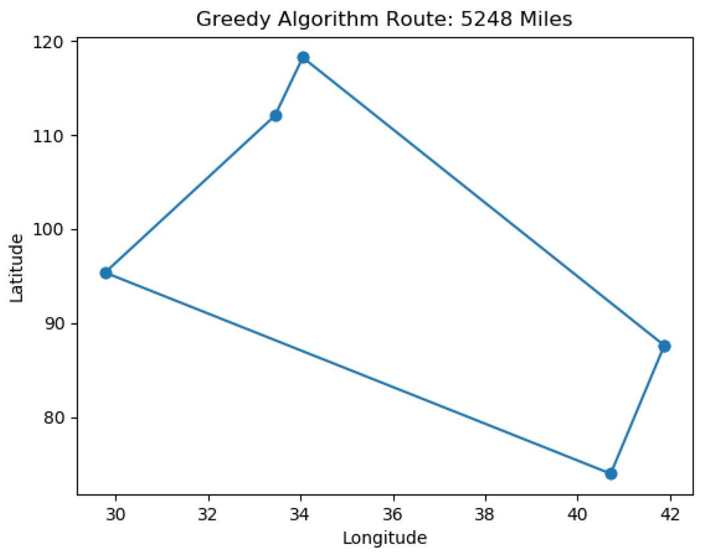


Figure 4. Route determined by the local greedy heuristic on a set of 5 cities

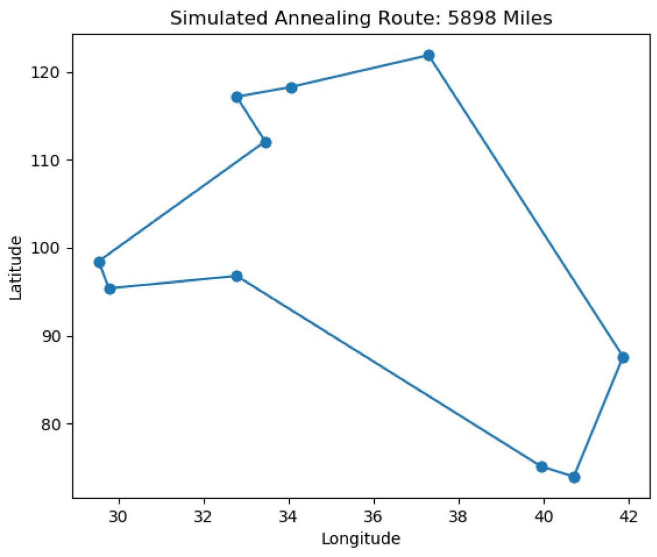


Figure 5. Route determined by the parallel simulated annealing metaheuristic on a set of 10 cities

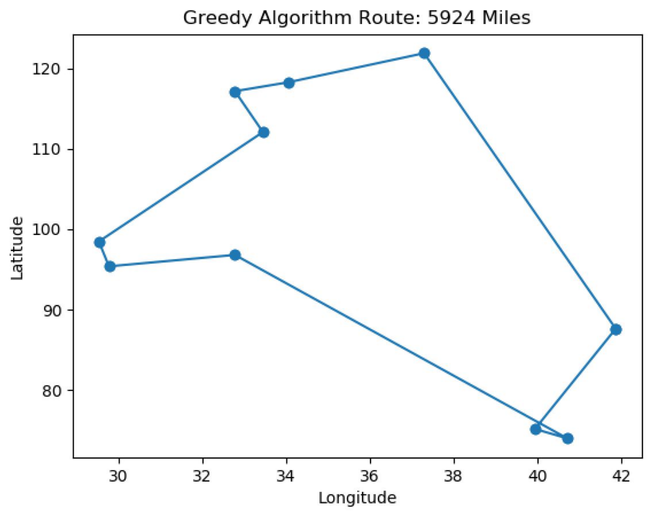


Figure 6. Route determined by the local greedy heuristic on a set of 10 cities

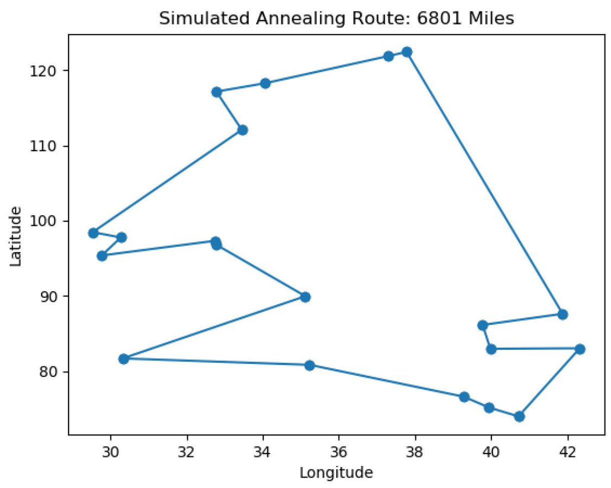


Figure 5. Route determined by the parallel simulated annealing metaheuristic on a set of 20 cities

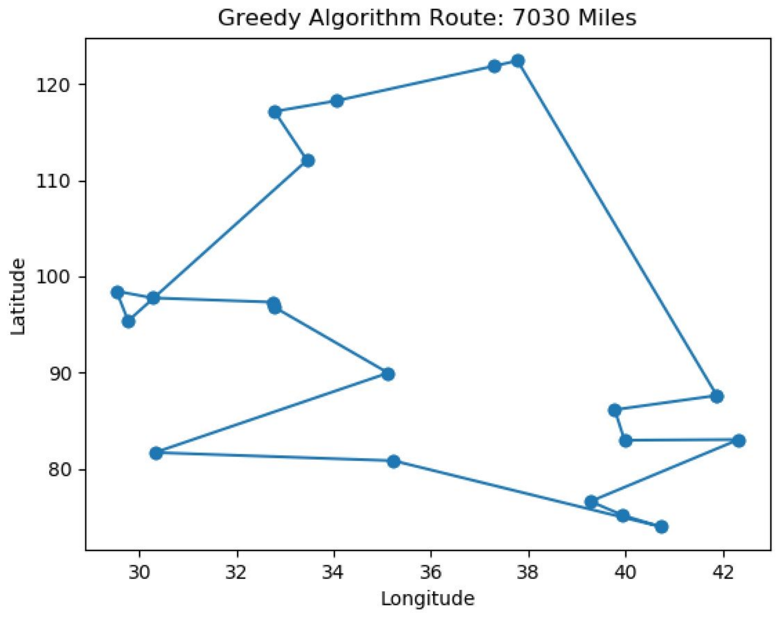


Figure 8. Route determined by the local greedy heuristic on a set of 20 cities

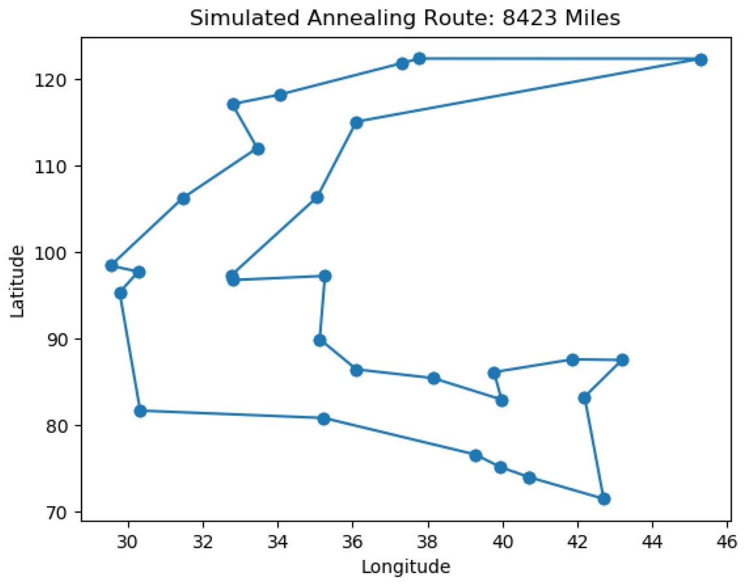


Figure 9. Route determined by the parallel simulated annealing metaheuristic on a set of 30 cities

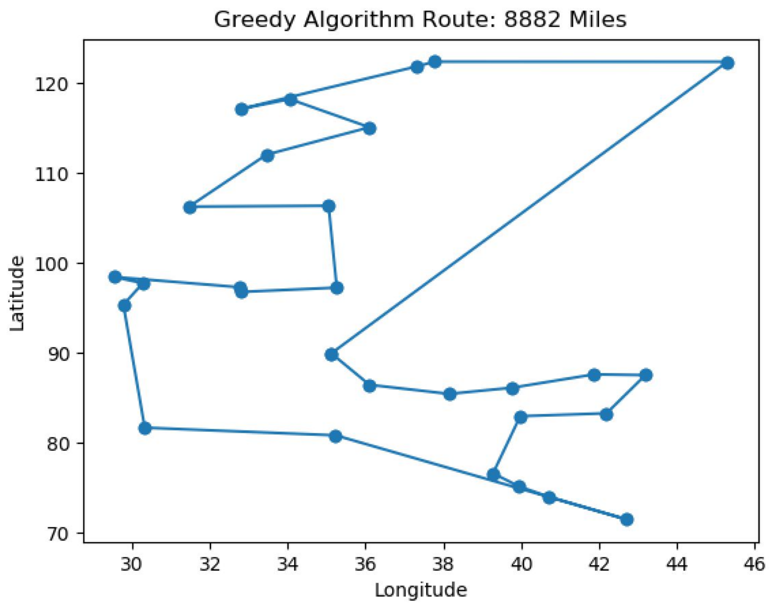


Figure 10. Route determined by the local greedy heuristic on a set of 30 cities

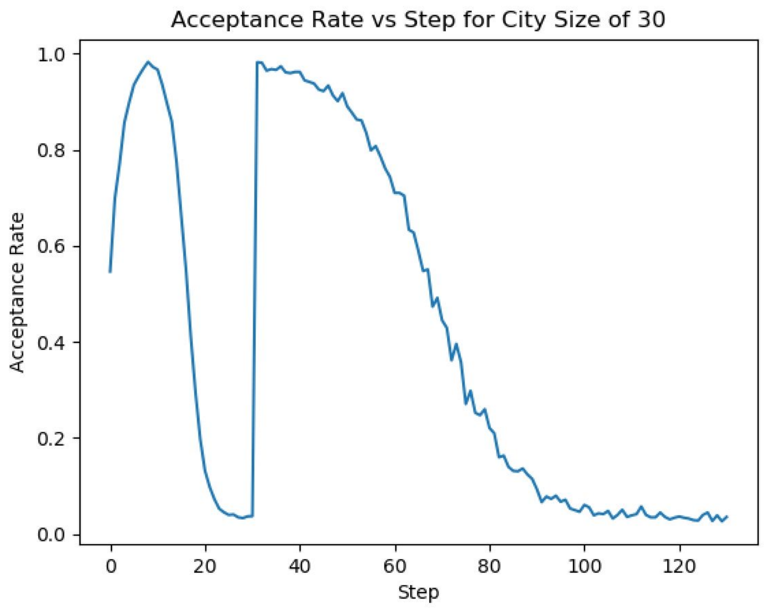


Figure 11. Acceptance rate of simulated annealing algorithm for each step

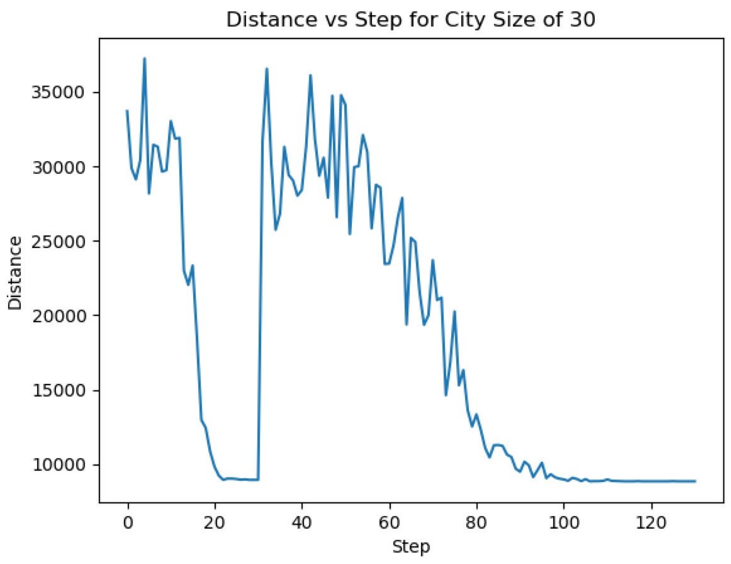


Figure 12. Distance of route found for each step size of simulated annealing algorithm

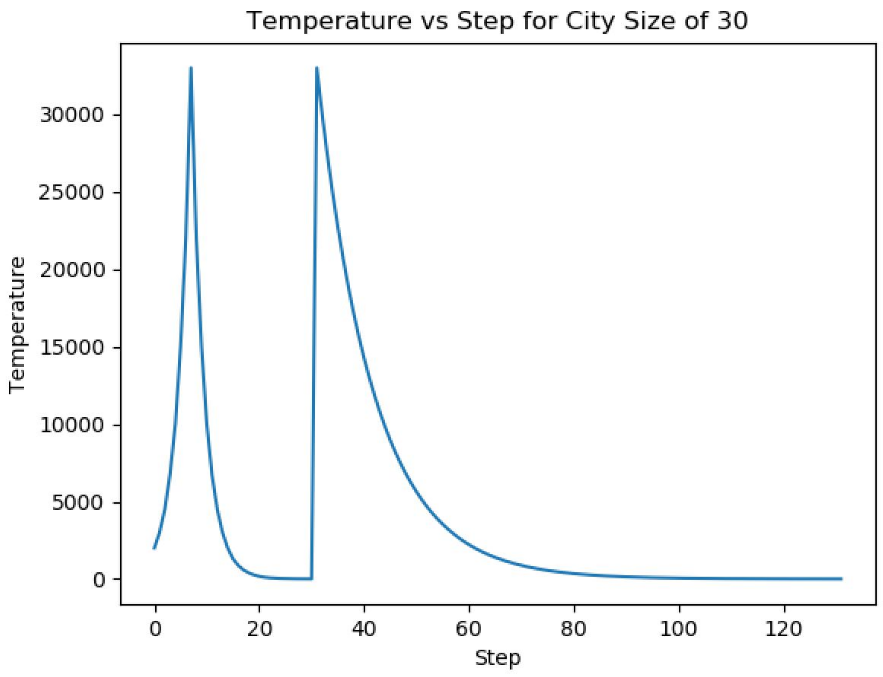


Figure 13. Temperature of simulated annealing algorithm over each step