Speed vs. Accuracy: Heuristic Traveling Salesman Problem Approaches

Moro Bamber, Youji Seto, Hiromi Kageyama

Department of Computer Science and Engineering, University of Alaska Anchorage
{mwbamber, yjseto, hpkageyama}@alaska.edu

***Abstract*— The Traveling Salesman Problem (TSP) is about speed, but when does speed outweigh the importance of accuracy? This paper explores this balance by presenting an evaluation of three heuristic algorithms – the Christofides Algorithm (CA), Ant Colony Optimization (ACO), and Genetic Algorithm (GA) – in terms of their performance on both speed and shortest path metrics. Through a series of experiments conducted on TSP instances from Reinelt Gerhards's TSPLIB dataset, encompassing varying problem sizes, we collected data on runtime and path length. Our findings reveal that while further optimization of heuristic TSP solutions incurs overhead, it also leads to improved path lengths.**

***Index Terms* - TSP, Speed, Accuracy, Heuristic, Christofides, Ant Colony Optimization, Genetic Algorithm, Performance, Optimal Solution**

I. INTRODUCTION

What are the computational limits of a computer? At the core of computing is the P NP problem, where NP problems are beyond the limits of computational power. The Travelling Salesman problem is one of these NP problems [1].

The TSP is when given a graph and distances between vertices, the goal is to visit every vertex in the graph and return to the start on the shortest path possible. For a graph with n vertices, there are (n-1)!/2 possible solutions, and only 1 is the optimal solution. Running a computation with factorial time complexity to find the length of every path possible is futile, computing resources quickly diminish rapidly [2]. Heuristic - approximating - approaches were developed to solve this issue and give a sub-optimal solution but within reasonable time complexity [3]. Solving TSP problems has applications outside of computer science, as it is a problem in everyday life as well [4]. Take, for example, a bus route in a city, using a TSP algorithm to create the shortest path to hit all the stops would be of importance to the bus driver, or problems that arise in DNA sequencing [5].

In this study we used three algorithms, the Christofides, Ant Colony Optimization, and Genetic Algorithm, to study runtime and optimal path length. All three algorithms were implemented in C++. We chose C++ due to its speed at runtime [6] and our desire to test larger problem sets within the limits of computing power available. To start, we studied the basics of these algorithms from Christiran Nilsson's paper [3]. The algorithms were chosen due to their allowance of varying levels of optimization. To streamline the process of problem selection for TSP algorithms, a TSP problem library created by Reinelt Gerhard was used as our dataset [11]. It allowed us to run the same problem across multiple algorithms in the same input format.

II .CHRISTOFIDES AND 2 OPT SWAP

The Christofides Algorithm (or heuristic) is based on the idea that finding a minimum spanning tree (MST) in a graph is not computationally expensive. A MST is simply the smallest weighted tree for a given set of points. It can be found using Prim's algorithm. Prim's algorithm starts at an arbitrary vertex and adds the smallest distance edge - meaning it's greedy; it does this for every vertex in the graph until all vertices have been reached [14]. For the CA we used, both used Prim's algorithm to initially find a MST. From the MST, a Hamiltonian path must be created to solve the problem. A Hamiltonian path is a one where every node is visited once and returns to the starting node. To do this, one must find the perfect matches for vertices of an odd degree, this is to close loops in the graph. Then one must create a Euler Circuit (a path that visits every vertex and returns to the start), which is then changed to a Hamiltonian Path by removing edges that allow a visitation of a node twice. And thus the MST has been transformed to a path that visits each vertex of the tour once, creating a path for the TSP.

The 2-opt algorithm examines each edge in the path and looks for optimizations. For all non adjacent edges it determines if swapping its edge with an edge between two other vertices would shorten the path. If so, the path is altered. 2-opt does this until no more improvements can be made. The code used to run the algorithm was modified for our purposes, the base Algorithm was written by Dilson Lucas Pereira [15] and the 2-opt version was written by Rebecca Sag [16].

III. ANT COLONY OPTIMIZATION

The Ant Colony Optimization (ACO) algorithm takes inspiration from the behavior of real ants to a MST using combinations of heuristic information and pheromone scent.

In terms of the TSP, an ant initialized to a starting city will traverse to the next city by first evaluating the heuristic information, such as the shortest edge to the next city. The first group of ants will continue to find the shortest path given the heuristic knowledge until all cities have been visited and returned to the home city. Once an ant completes a tour, a pheromone scent will be placed on the edges the ant traversed. The subsequent group of ants in the next iteration will use the heuristic information and the pheromone scent to try to find the shorter path. Increasing the number of iterations will optimize the output with the cost of increased runtime. It is also important to know that the optimization from each subsequent has diminishing returns [12].

IV. GENETIC ALGORITHM

The Genetic Algorithm (GA) is a metaheuristic algorithm inspired by the process of natural selection and has 5 phases. The first phase is the initialization phase which generates a population of potential solutions. These solutions are typically represented as a string of numbers or symbols and are often referred to as genotypes [18], [17].

The second phase is the evaluation phase. In this phase, each solution, or individual, is run through a fitness function which quantifies how good or bad the solution is with respect to the problem being solved. This function acts like the environment to help guide the evolutionary process, determining individuals who are more likely to survive and reproduce [18], [17].

The third phase is the selection phase where after the fitness of each individual is evaluated, a selection process picks the individuals with higher scores for reproduction. This ensures the fitness score will increase over each generation, mimicking the process of natural selection [18], [17].

The fourth phase is the reproduction phase. The individuals selected will create offspring for the next generation through genetic operators such as crossover and mutation. Crossover involves combining genetic information from two parent individuals to produce new offspring, while mutation introduces random changes to the offspring's genetic material [18], [17].

The fifth and final phase is the replacement phase. In this phase, the offspring replace the least fit individuals in the population, forming the next generation. Other than the initialization phase, the other phases repeat until a termination condition is met. This condition could be the max number of generations reached, or a satisfactory fitness level of the population [18], [17].

V. RELATED WORK

Prior research in the field of TSP problems have discussed different approaches to address the challenges of finding solutions heuristically [7]. Some studies have looked at alternative ways to solve TSP instances [8][9]. We recognized the need to evaluate tradeoffs between speed and solution quality. Our study was inspired by the multithreaded approach described by Wei et al. [10] which described the need for more optimization for modern computing systems leveraging multi core usage. And unlike other studies comparing different algorithms, we focused on the impact of optimization levels for a single heuristic algorithm . In sum, existing research has contributed valuable insights into the speed and methodology of other heuristic algorithms. We intended to expand this knowledge further by trying to gain insight into configuration of different heuristic TSP approaches.

VI. METHODS

A. Objective

The goal of this experiment is to determine which heuristic approach is most applicable to a given need. We are studying the tradeoffs between speed and optimization.

B. Testing Environment

We have configured an environment on a Virtual Machine to run the code. Running on Oracle VM Virtual Box was an Arch Linux OS with 10Gb. of memory and using 8 processors of an AMD Ryzen 7.

C. Programming Language

All algorithms were compiled and executed using C++ version 11.

D. TSP Testing Data

Our testing data consists of TSP files with varying numbers of vertices that we obtained from TSPLIB [4]. Four ‘tours’ were chosen with 51, 150, 1002, and 4461 vertices respectively. The names of these files were eil51, kroB150, pr1002, and fnl4461. Since the programs were running in C++, we ran make files to compile the code with adjustments to fit our input and output files.

E. Christofides Implementation

Once the code to run was established, we used a base CA and a CA with 2-opt optimization. 10 tests per problem were run and we used the average runtime as a final runtime result for time and the path length (which did not change on different runs).

F. ACO Implementation

We used the ACO algorithm published on a public repository on Github for our experiment [13]. The program comes with both sequential and parallel models which we tested separately. Across iterations ranging from one to twenty, we measured the average runtime and output tour length produced by the algorithm using specific TSP data. Our goal was to observe how the ACO algorithm optimizes tour length with increasing iterations and understand the trade-off between runtime efficiency and tour length reduction. This analysis allowed us to identify the iteration point where efficiency is optimized and record how multithreading affects performance.

G. Genetic Algorithm Implementation

To implement the GA, we found a repository on Github and modified the code to suit our experiment [19]. After setting up the GA, 5 tests were run for each TSP. We took the average of the first and last generation's best path along with the runtime and put the results in a table; runtime includes the time it took to initialize the population. Comparing the first and last generation allows us to observe the algorithm's efficiency in finding the most optimal solution after running. The termination condition of this algorithm was set to if the average tour length subtracted by the best tour length in the population equaled less than 0.001 in value [19].

VII. RESULTS

A. Christofides Test Results

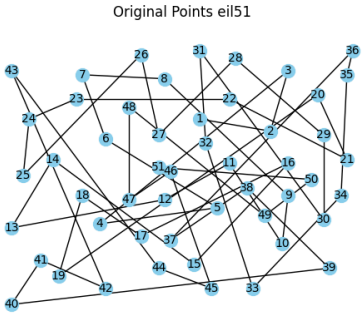

Figure-1: eil51 base graph

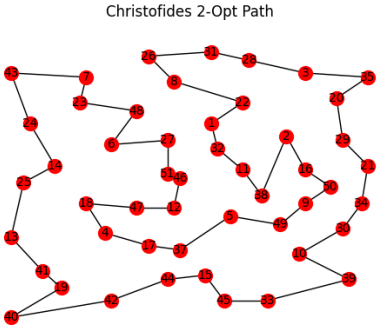

Figure-2: Result Path of Christofides 2-opt
Path Length: 432

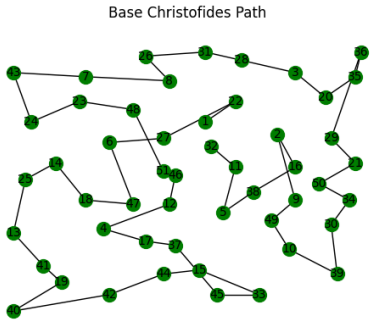

Figure-3: Result Path of Christofides no 2-opt
Path Length: 484

Comparing Figure-1 to both Figures 2 & 3 gives us insight into what a TSP heuristic approach is doing. It sees a mess of points and edges, and determines a better way to visit all the points. We can also discern some differences between the path generated from the Christofides 2-opt path (Figure-2) and the graph generated from the base Christofides algorithm (Figure-3). These differences amount to a Christofides base algorithm path length difference of 52 compared to the 2-opt graph, an extra 12% longer than the 2-opt graph. However, if we compare the runtime, the base Christofides algorithm was 0.02 seconds quicker.

problem	**eil51**		**kroB150**	
algo	time(s)	path	time(s)	path
ca_base	0.0007	484	0.0082	30168
ca_2opt	0.0299	432	0.0934	27291
% diff	190.72	11.35	167.78	10.01

Table-1: Results of CA Tests Part 1

problem	**pr1002**		**fnl4461**	
algo	time(s)	path	time(s)	path
ca_base	1.155	288129	92.2858	207400
ca_2opt	17.263	275440	112.841	195625
% diff	174.91	4.50	20.04	5.84

Table-2: Results of CA Tests Part 2

We found with the CA that the base algorithm was consistently faster than the 2-opt. With the largest percent difference in time being for the small problem with 51 cities at a percent difference of 190%. The smallest percent difference in runtime was the largest problem with 4461 points. The percent difference was only 20%. We also found that 2-opt outperformed the base algo over every test in terms of path

length. The largest difference was for the 150 point problem at 10%, and the smallest was for 1002 points at 4.5%, the average path length difference was 8%.

B. ACO Test Results

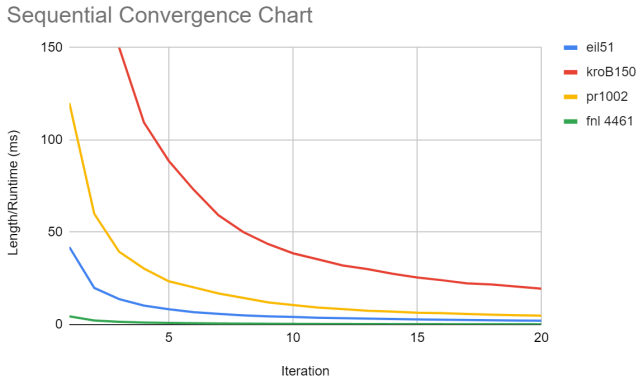

Figure-4: Convergence Chart of ACO Sequential Test

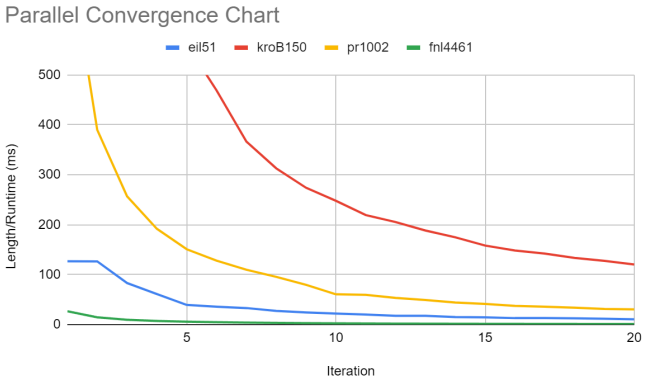

Figure-5: Convergence chart of ACO Parallel Test

The convergence chart shows how the greatest change in efficiency occurred during the first 5 iterations but begins to converge afterwards especially for the TSP with smaller datasets.

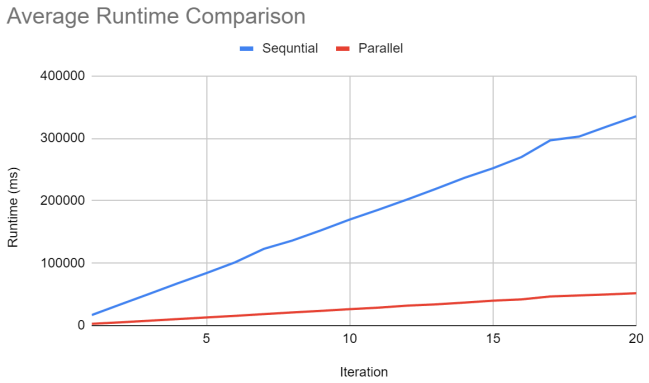

Figure-6: Average Runtime Comparison Between ACO sequential and parallel

The average runtime as the number of iterations increases follows a positive linear trend for both sequential and parallel. This data gives a glimpse of how the runtime increases as the number of iterations increases. This chart also shows the average runtime difference between running the algorithm in parallel using 10 threads versus sequentially. The exact percent difference between the runtime ranged between 137 and 149 percent.

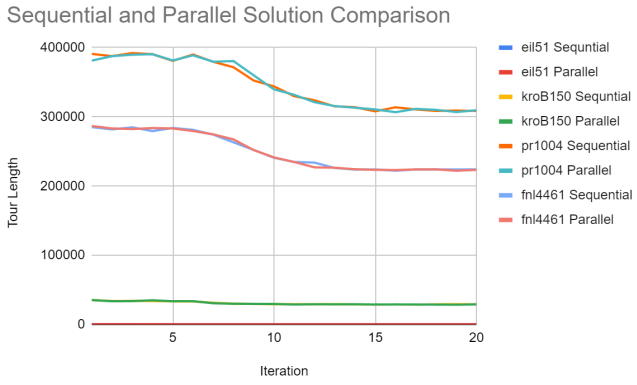

Figure-7: Tour Comparison Between ACO Sequential and Parallel

Running the ACO algorithm sequentially vs. multithreaded does not impact the solution. This chart shows that both had very similar tour lengths despite the difference in runtime..

problem	eil51		kroB150	
algo	time(s)	path	time(s)	path
Sequential	0.21	459.469	1.286	28732.1
Parallel	0.031	451.147	0.225	28681.8
% diff	148.55	1.83	140.44	0.18

Table-3: Shortest Tour Length Results of ACO Tests Part 1

problem	pr1002		fnl4461	
algo	time(s)	path	time(s)	path
Sequential	48.242	308016	1027.84	222283
Parallel	8.219	306849	189.499	222205
% diff	141.77	0.38	137.73	0.04

Table-4: Shortest Tour Length Results of ACO Tests Part 2

C. Genetic Algorithm Test Results

Tables 5 and 6 reveal the performance of the GA. We took the average of 5 trials and separated the data into 2 tables. Iteration represents the generation of the population and always initializes at 0. The generation continues to increment until the termination condition is met. The average last generation is listed below the initial generation. Path represents the most optimal solution in the population at the current generation.

Problem	Iteration (Gen)	Path	Run Time (s)
eli51	0	428.2	0.3238
	104	426	
kroB150	0	26597	0.7806
	112.6	26130	

Table-5: Genetic Algorithm Results Part 1

Running TSPs with 150 cities doubles in run time and has a difference of about 8 iterations when compared to a TSP with 51 cities.

Problem	Iteration (Gen)	Path	Run Time (s)
pr1002	0	273317.4	13.5736
	231	259045	
fnl4461	0	192321.6	181.0544
	794	182570.2	

Table-6: Genetic Algorithm Results Part 2

Increasing the city count to 1002 doubles the iteration count and increases the run time to more than 17 times compared to a TSP with 150 cities. When the city count further increases to 4461, the iteration count almost bumps up 3.5 times and the run time increases more than 13 times.

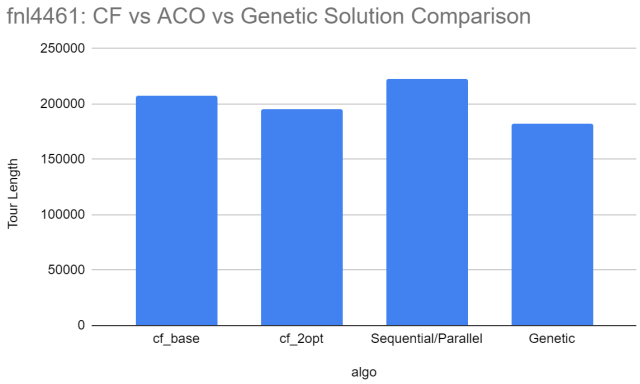

Figure-8: fnl4461: CA vs ACO vs Genetic Solution Comparison

This chart compares the solution tour length for each of the three algorithms for the fnl4461 TSP dataset. The lower the number or smaller the bar graph, the more optimal the solution is. Sequential and Parallel have the output as running the ACO algorithm with multiple threads only affects runtime.

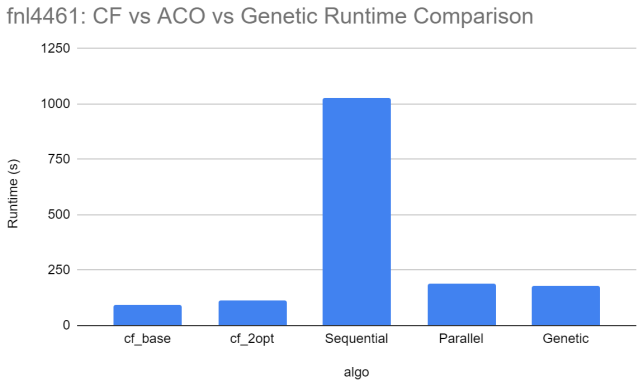

Figure-9: fnl4461: CA vs ACO vs Genetic Runtime Comparison

The CA algorithm had the most efficient runtime whereas the sequential ACO algorithm was by far the slowest, almost ten times slower than the CA algorithm. It is also important to note that the ACO running in parallel with 10 threads performed at the same, if not slower, than the CA and GA which are not multi-threaded.

VIII. DISCUSSION

The main goal in our experiments was to identify the speed and path length for a given TSP for different algorithms of varying optimization. We have achieved this goal. While some results may not be surprising, we are content with the data we gathered and hope it gives insight into heuristic approaches to the TSP.

For the tests involving the CA, we saw that for both metrics - speed and path length - that one implementation was better than the other. The base algorithm was always faster than the 2-opt, and vice-versa for runtime. The speed difference was extremely noticeable at lower point counts, but not so for the largest problem we tested. It is implied that the 2-opt swap will add time to the computational process at an order of O(m*n^2) where m is the number of times 2-opt is run, and n is the number of points in the graph. However, it outperformed the base algorithm in terms of path length. It was able to consistently optimize the base Christofides path length. The goal of a heuristic algorithm is to find a suboptimal solution in reasonable time. We believe that both CA's do this well. However, if one is in need of pure speed, the base CA will do fine, if more accuracy is needed, then the 2-opt swap will need to be performed, adding extra runtime. We feel that the use of either depends on the context of the situation, especially on a larger problem set. In the future we would like to build on the CA to use simulated annealing in C++. Simulated annealing allows for suboptimal 2-opt swaps to occur on the chance it shortens the path length eventually. We believe threading the Simulated Annealing optimization would be an excellent addition to the data presented in this paper.

The ACO algorithm test results when comparing how the number of iteration affects solution and runtime shows us that the algorithm efficiency decreases overall as the number of iterations increase and that the greatest change of convergence occurred between iterations 1 through 5 for all of the TSPs. By iteration 20, our results show that the graph has reached a stage of diminishing returns where after a certain number of iterations the solution quality plateaus despite increasing runtime (Figure -1,2).

Comparing the Tour Length and runtime between Sequential and Parallel versions of the ACO algorithm, shows us that although running the ACO algorithm in parallel using 10 threads did not make noticeable change in tour length, it made drastic improvements in runtime (figure 6,7). This difference was calculated to be between 137.73% to 148.55% difference (Table-3,4).

Looking at the results for the GA, eli51 and kroB150 have similar iteration counts (difference of about 8 on average) despite having a difference in city count of about 3 times. The initial best path of the population for each trial had a nice spread of randomness which allowed the algorithm to show its efficiency in finding the most optimal solution. The last generation of each trial outputs the same path for eli51, kroB150, and pr1002. The only exception for this trend is fnl4461 since the path of the last generation changes for each trial. This can be due to reaching the termination condition before finding the most optimal solution. Running TSPs with more cities will help us understand why the last generation for each trial outputs different results for each trial's path.

If we take a look at the path efficiency, GA provides the best results when compared to CA and ACO. Despite being the best path finding algorithm, the runtime is not as efficient. By far, CA is the most efficient algorithm in runtime with ACO in second place when running in parallel. However, it seems that ACO might run slower with more cities compared to the GA, as we can see in problem fnl4461.

Figure 8 compares the path efficiency of all 3 algorithms and figure 9 compares the runtime efficiency. We can visualize the differences of each algorithm with the help of these two figures and one thing to note is that the tour length of each algorithm is fairly similar. As stated above, the best path finding algorithm was the Genetic Algorithm, but in terms of runtime the Christofides Algorithm was the most efficient. The Sequential version of ACO was extremely slow needing about 1,000 seconds while the other algorithms took less than 190 seconds. If we were to choose an algorithm based on these results, the criteria of selection would be the most optimal tour length as a higher priority than runtime.

IX. CONCLUSION

Our study focused on evaluating three heuristic algorithms and how they balance runtime efficiency and solution quality in solving the Traveling Salesman Problem. Our findings revealed distinct characteristics of the Christofides (CA), Ant Colony Optimization (ACO), and Genetic Algorithm (GA) that demonstrate the differences in solving and optimizing the TSP. While the ACO algorithm provides optimization efforts, its computational overhead introduces diminishing solution quality and increased runtime. The CA algorithm with 2-opt added computational overhead, but with the guarantee of a shorter path length. With larger problems, the difference in runtime with added optimization is decreased, so the tradeoff between performance and path length is desirable. Both the Christofides and Genetic Algorithm provided a shorter solution with much more efficient runtime leading us to believe that not all optimization efforts enhance solution quality, although ACO was more efficient in executing when running in parallel compared to the Genetic Algorithm.

The results we gained from evaluating the three algorithms underscores the importance of considering multiple factors, including resources, problem complexity, and desired solution accuracy, when selecting an appropriate heuristic algorithm for the TSP. By recognizing the different trade-offs between runtime and solution efficiency, practitioners can make informed decisions tailored to their specific requirements and constraints.

Moving forwards, our study sets the stage for future research with the goal to refine heuristic algorithms and develop hybrid approaches that leverage the strength of different optimization strategies. By utilizing the data obtained from our experiment,

future researchers can achieve more optimal solutions for the TSP while reducing computational overhead and runtime complexity.

REFERENCES

[1] Kochkarov, R. (2021, October 31). Research of NP-Complete Problems in the Class of Pre-Fractal Graphs. *Mathematics*, *9*(21). https://www.mdpi.com/2227-7390/9/21/2764

[2] Borwein, P. B. (1985, September). On the Complexity of Calculating Factorials. *Journal of Algorithms*, *6*(3), 376-380. https://www.sciencedirect.com/science/article/abs/pii/0196677485900069

[3] Nilsson, Christian. (2003). Heuristics for the Traveling Salesman Problem.

[4] Lenstra, J. K., & Rinnooy Khan, A. H.G. (1975, November). Some Simple Applications of the Travelling Salesman Problem. *Operational Research Quarterly*, *26*(4), 717-733. https://www.jstor.org/stable/3008306?seq=4

[5] Lenstra, J. K., & Rinnooy Khan, A. H.G. (1975, November). Some Simple Applications of the Travelling Salesman Problem. *Operational Research Quarterly*, *26*(4), 717-733. https://www.jstor.org/stable/3008306?seq=4

[6] Performances of Sorting Algorithms in Popular Programming Languages - Durrani et al. 2022.

[7] C. Zhang and P. Sun, "Heuristic Methods for Solving the Traveling Salesman Problem (TSP): A Comparative Study," *2023 IEEE 34th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, Toronto, ON, Canada, 2023, pp. 1-6, doi: 10.1109/PIMRC56721.2023.10293957

[8] Abdulkarim, Haider & Alshammari, Ibrahim Fadhil. (2015). Comparison of Algorithms for Solving Traveling Salesman Problem. International Journal of Engineering and Advanced Technology. ISSN. 2249 – 8958.

[9] Halim, A.H., Ismail, I. Combinatorial Optimization: Comparison of Heuristic Algorithms in Traveling Salesman Problem. *Arch Computational Methods Eng* 26, 367–380(2019).https://doi.org/10.1007/s11831-017-9247-y

[10] Wei, X., Ma, L., Zhang, H., & Liu, Y. (2021). Multi-core-, multi-thread-based optimization algorithm for large-scale traveling salesman problem. *Alexandria Engineering Journal*, *60*(1), 189-197. https://www.sciencedirect.com/science/article/pii/S1110016820303227

[11] Wei, X., Ma, L., Zhang, H., & Liu, Y. (2021). Multi-core-, multi-thread-based optimization algorithm for large-scale traveling salesman problem. *Alexandria Engineering Journal*, *60*(1), 189-197. https://www.sciencedirect.com/science/article/pii/S1110016820303227

[12]Ja-k."GitHubJa-k/Ant-Colony-Optimization-for-TSP-Problem: Ant Colony Optimization (ACO) for Solving Traveling Salesman Problem (TSP) Using Multi-Threading as Well as Sequential Programming in C++11." *GitHub*, https://github.com/Ja-k/Ant-Colony-Optimization-for-TSP-Problem. Accessed 13 Apr. 2024.

[13] Stutzle, Thomas, and Marco DORIGO. ACO Algorithms for the Traveling Salesman Problem†.

[14] BSTJ 36: 6. November 1957: Shortest Connection Networks And Some Generalizations. (Prim, R.C.)

[15] Pereira, Dilson L. "christofides-algorithm." *GitHub*, https://github.com/dilsonpereira/christofides-algorithm. Accessed 21 March 2024.

[16] Sag, Rebecca. "traveling-salesman." *GitHub*, https://github.com/beckysag/traveling-salesman. Accessed 22 March 2024.

[17] G. Ye and X. Rui, "An improved simulated annealing andgenetic algorithm for TSP," 2013 5th IEEE International Conference on Broadband Network & Multimedia Technology, Guilin, China, 2013, pp. 6-9, doi: 10.1109/ICBNMT.2013.6823904.

[18] Zhu, Yu, and Lin Wu. "Structure Study of Multiple Traveling Salesman Problem Using Genetic Algorithm | IEEE Conference Publication | IEEE Xplore." *IEEE Xplore*, IEEE, June 2019, ieeexplore.ieee.org/document/8787633. Accessed 14 Apr. 2024.

[19] Sugia. "Genetic Algorithm for Traveling Salesman Problem." *GitHub*, github.com/sugia/GA-for-TSP. Accessed 13 Apr. 2024.

APPENDIX

Member Contributions

Moro Bamber

- Abstract
- I. INTRODUCTION
- II. CHRISTOFIDES ALGORITHM AND 2-OPT SWAP
- V. RELATED WORKS
- VI METHODS (A-E)
- VII. RESULTS
    - A. CHRISTOFIDES ALGORITHM

- VII DISCUSSION
- CONCLUSION

Youji Seto

- III. ANT COLONY OPTIMIZATION
- VI. METHODS
    - F. ACO IMPLEMENTATION
- VII RESULTS
    - B. ACO Test Results
- VIII. DISCUSSION
- CONCLUSION

Hiromi Kageyama

- IV. GENETIC ALGORITHM
- VI. METHODS
    - G. Genetic Algorithm Implementation
- VII. RESULTS
    - C. Genetic Algorithm Test Results
- VII. DISCUSSION
    - Last 3 paragraphs
- CONCLUSION
    - Last sentence of 1st paragraph

We feel like work was distributed equally.