**PROJ 201 Project Final Report**

**Project Title:** Route My Electric Van

**Name, Surname & ID of group members:** Sadiq Qara (30410), Mustafa Harun Şendur (31209), Umut Yunusoğlu (32158), Arif Kemal Sarı (28999)

**Supervised by:** Bülent Çatay

**Month, day, year:** 01.06.2023

**Abstract**

This paper describes various implementations of algorithms along with using local search techniques applied to the Travelling Salesman Problem also known as TSP. Throughout this project, each project member aimed to use different algorithms in their code and present their solution to the problem using numerous computer languages (C++, JavaScript and Python). The algorithms that were used to find near-optimal solutions include GRASP, Ant Colony Optimisation, Sweep and Tabu Search. In addition to this; in this paper, the individual result of the implemented codes of the project members is compared with one another. These comparisons are based on benchmarking to understand which implementations produce the best results.

1. **Introduction**

The Travelling Salesman Problem also known as TSP is a well-known problem in theoretical computer science as well as operations research (Traveling Salesman Problem – Dynamic Programming Approach, not dated). TSP was first expressed mathematically by Merrill M. Flood in 1930 when he seeked to solve the problem of school bus routing (Lawler et al., 1985). In the Travelling Salesman Problem, a salesman has to visit each city in the set of cities and return back to the starting point (Lawler, 1985, p. 1). This particular problem highlights the challenge of finding the shortest and most efficient route for travelling the given route.

The Travelling Salesman Problem is classified as a combinatorial optimisation problem that is identified as an NP-hard problem in the academic literature (Ilhan, 2016). The reason for this particular classification is that the complexity of the problem to produce the optimal solution increases correspondingly as you add more cities to the Travelling Salesman Problem (Nuriyeva & Kizilates, 2017). Due to the city's increase in the problem, complexity of TSP increases exponentially as well. In order to overcome this problem many researchers have put forward heuristic and metaheuristic solution techniques. Many algorithms have been implemented to tackle the problem as a brute-force approach would not be efficient (Hoffman et al., 2013). With brute force, for a 16-city TSP 653,837,184,000 distinct routes would be required to be examined. Hence instead of investigating all the possible routes that could be produced, successful algorithms are used to eliminate unuseful routes without ever considering some of them (Hoffman et al., 2013).

Many algorithms have been discussed and tested in academic circles. However, the topic remains to promise many unfound methods for researchers.

In this context, this project aims to review the literature about the Travelling Salesman Problem and examine heuristic and metaheuristic algorithms, implement them to find near-optimal values, compare different implementations of algorithms and illustrate an analysis of the results of the selected algorithms.

1. **Methods & Materials**

**2.1. Introduction to Methods**

In this project we use four different algorithms, Ant Colony Optimization (ACO), Greedy Randomized Adaptive Search Procedure (GRASP), Tabu Search and Sweep. These methods are implemented in 3 programming languages, those being JavaScript, C++, and Python, and are supported with local search methods.

The final versions of the algorithms are tested across eight different TSP datasets which are derived from “TSP Test Data”(*TSP Test Data*, not dated) and “VLSI Data Sets” (*VLSI Data Sets*, not dated) . The datasets used differ in both size and complexity. These tests are done in a restricted time criteria in order to be able to compare the results. The runtime for each dataset is set proportionally to the size of the dataset.

The datasets and the times given for the algorithms are listed below:

* wi29, 15 seconds
* dj38, 15 seconds
* xqf131, 1 minute
* qa194, 1 minute
* uy734, 10 minutes
* zi929, 15 minutes
* lim963, 15 minutes
* xit1083, 15 minutes

The results of these tests are evaluated with different metrics such as their distance from optimum, improvement rate from initial solution etc. using benchmarking techniques. Additionally, for each implementation different visualization methods are used to gain an understanding of the results.

**2.2. Description of the Algorithms**

**2.2.1. Ant Colony Optimisation**

Ant colony optimization is a metaheuristic algorithm that is inspired by real ants, used to solve computational problems which require finding above average paths through a given data.

Artificial ants create a route and increase pheromone values between arcs inversely proportional to the length of the total route(Dikmen et al. , 2014).

After a finite number of iterations, pheromones accumulate around optimal routes (Şahin, 2019). Also after every iteration, some percentage of pheromones evaporate in order to avoid local minimum traps.

In this study ACO used an elitist ant system where the pheromones were only updated by a limited number of high performing ants. Two-opt local search method is also used to enhance the routes found by elite ants. Due to this approach an increase in performance is aimed. Another method that is used to increase performance and escape from local minimums is the min-max pheromone system. This system sets the minimum and maximum pheromone levels every arc can ever have and limits the system to exceed these boundaries.

And in order to tune the parameters, there 198 controlled tests were made. In these tests initial values for parameters are set from the paper “Ant Colony Optimization Algorithm Parameter Tuning for T-way IOR Testing” (Ramli et al., 2018). After these tests optimal values of alpha, beta, evaporation rate and number of elites for different sizes of datasets are determined.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Best Alpha For Dj38** | **Best Alpha For Qa194** | **Best Alpha For Zi929** |
| **Average Distance**  **From Optimum (%)** | 0.816367617 | 6.98264328 | 23.39615061 |
| **Alpha Value** | Alpha=2 | Alpha=2 | Alpha=1 |

Figure 2- Alpha Test Results

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Best Beta For Dj38** | **Best Beta For Qa194** | **Best Beta For Zi929** |
| **Average Distance**  **From Optimum (%)** | 0.125342099 | 6.311009982 | 24.07711 |
| **Beta Value** | Beta=5 | Beta=1.8 | Beta=0.5 |

Figure 3- Beta Test Results

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Best Evaporation Rate For Dj38** | **Best Evaporation Rate For Qa194** | **Best Evaporation Rate For Zi929** |
| **Average Distance**  **From Optimum (%)** | 6.324668917 | 7.737834834 | 24.26467612 |
| **Evaporation Rate**  **Value** | Evap=0.5 | Evap=0.05 | Evap=0.1 |

Figure 4- Evaporation Rate Test Results

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Best Number of Elites For Dj38** | **Best Number of Elites For Qa194** | **Best Number of Elites For Zi929** |
| **Average Distance**  **From Optimum (%)** | 4.24900507 | 6.904690781 | 25.13041814 |
| **Number of Elites** | Elites=5 | Elites=1 | Elites=2 |

Figure 5- Number of Elites Test Results

**2.2.2. Greedy Randomized Adaptive Search Procedure**

GRASP (Greedy Randomized Adaptive Search Procedure) is a metaheuristic search algorithm that constructs a route by adding the best available options at every step through greedy algorithms (Feo & Resende, 1995). In addition to this, randomness is used by allowing a certain number of candidates to randomly be selected which assists to create variety in the solution without being only dependent on the best available option. Along with this approach, the Swap algorithm is used as a local search technique which can be applied to advance the route quality for the Travelling Salesman Problem through swapping pairs of cities and assess the swapping’s outcome. (Gutin & Punnen, 2002). Through the swapping, if the route shortens, swapping is accepted and the process is continued.  
  
 In the implementation of GRASP and the swap algorithm, firstly the datasets is selected and the starting point is chosen by the user of the program. The greedy approach was then used to choose the nearest available locations to create a candidate list, and along with that, the route was continued to go to the next city through random selection from the candidate elements of the particular city’s candidate list. The candidate list size used in the implementation varies from 2 to 20 considering the dataset that is selected to examine. During the randomized greedy approach, in each iteration, multiple times of swapping was used to have a shorter route to have a better outcome. Lastly, multiple times of swapping is once more done after the route is constructed to improve the solution quality by looking for route shortening that may be available. Furthermore, like the multiple iterations of swapping, GRASP is also being done multiple times to have a nearer optimal value for the TSP. Through this technique, the algorithm is examined to present a more accurate solution compared to the optimum value in the time limit that is designated for each dataset.

See Appendix D for the pseudo-code of the algorithm.

**2.2.3. Tabu Search**

Tabu search is a meta-heuristic algorithm for solving TSP. Tabu search heuristic is based on making certain moves of local search as tabu in order to escape from local optima (Pedro et al., 2013). Local search (also known as hill climbing) is a heuristic that makes small changes on initial solution thus moves from one solution to another to search for better solutions in its neighborhood (Misevičius, 2004).

Local search makes improvements until there are not any moves. In other words there is no neighbor that has a better solution therefore local search methods cannot find a better solution and this solution is the optimum solution for ls. However these optimum states might not be the optimal solution. Tabu search forbids some number of moves and it allows to move neighbors has no improvement therefore it escapes from local optimum and searches for global optimum state.

Tabu search is meta-heuristic because it can be used for different optimization problems and it is not a simple search procedure ,it uses short term memory and hierarchical search thus it is an efficient way to find shortest path (Fiechter, 1994). Tabu search uses tabu list to escape local optima , tabu list has some aging process, it removes old moves that are included in the tabu list. Tabu search avoids cycling of solutions and it searches for different optima with the help of tabu list (Pedro et al., 2013). In this implementation , a vector is used for tabu list and it is allowed to grow eighth moves only ,if more than eight moves are forbidden, the oldest moves removed from the list.

In this implementation of tabu search for the TSP, 2- opt is used for local search . The 2-opt algorithm works by iteratively improving a solution to the TSP by swapping the position of two non-adjacent cities in the route (Croes,1958).Distance matrix is used to avoid unnecessary calculations. Two opt first iterates through all pairs of non-adjacent edges in the current solution and calculates the improvement in total distance if the order of those two edges is reversed. If reversing the order of the edges results in a shortening of the total distance or if there is no result, improve the solution, the existing solution is used. The function reverses the order of the edges and updates the distance matrix with the new order of the edges.

The Nearest neighbor algorithm used to construct the initial solution .The nearest neighbor algorithm is a simple heuristic for TSP . The nearest neighbor algorithm is based on the idea of constructing a solution to the TSP by starting at a given city and visiting the nearest unvisited city, adding the distance between the two cities to the total distance traveled. This process is repeated until all cities have been visited, at which point the algorithm returns to the starting city.

**2.2.4. Sweep**

Sweep is a heuristic search algorithm that moves a line through a set of nodes and connects the path to whichever node the line contacts. In this implementation of the Sweep algorithm, two types of sweep are used: Angular Sweep, which moves the line clockwise or counterclockwise starting from a point and Linear Sweep, which moves a line horizontally or vertically starting from a point.(Line Sweep Algorithms, 2018)

The linear sweep works in the following way, it checks what the angle of the angular sweep is and depending on the angle either moves horizontally or vertically. It also alternates between moving away from the starting point and moving into it. The path is finished after it goes through a local search algorithm to see where it could be improved and then improving it accordingly. The result is an algorithm that finds a route quickly at the cost of giving a longer route on average.

1. **Results**

To categorise and compare the results of the project, we have examined each project member’s chosen algorithm in designated categorisations under a table titled “Comparison Table”. These categorisations to compare every implementation include “Average Distance from Optimum”, “Average Best Cost”, and “Best of All”.

“Average Distance from Optimum” assists to compare each selected algorithm by their average distance calculated by numerous runs and how close they are to the optimal value by percentage. The optimal value of each dataset is given under the name of every data set below the title “Dataset Name”. Furthermore, “Average Best Cost” highlights the calculated average total distance from numerous runs of project members’ program runs. Lastly, “Best of All” categorisation signifies the best outcome that was produced from the project member’s algorithm. The best outcome underlines the shortest route that could be reached.

In addition to this, during the numerous runs of the GRASP algorithm, “Average Distance from Optimum” and “Average Best Cost” values were examined to be identical as in each run of the program GRASP algorithm was repeated multiple times depending on the dataset size and the best outcome was presented to the user, the same was also observed in the Sweep algorithm as well.

**3.1. Comparison Table**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | ***Dataset Name*** | ACO | Sweep | Tabu Search | Grasp |
| ***WI29***  ***Optimal Value: 27603*** | Average Distance From Optimum | 2.20 | 1.50 | 1.62 | 0.06 |
| Average Best Cost | 28225.17 | 28019 | 28056.2 | 27620.8 |
| Best of All | 28145.16 | 28019 | 27750 | 27620.8 |
| ***DJ38***  ***Optimal Value: 6656*** | Average Distance From Optimum | 1.09 | 0.05 | 2.19 | 0.05 |
| Average Best Cost | 6729.62 | 6659 | 6804 | 6659.43 |
| Best of All | 6664 | 6659 | 6662 | 6659.43 |
| ***XQF131 Optimal***  ***Value:564*** | Average Distance From Optimum | 12.98 | 9.5 | 9.73 | 8.64 |
| Average Best Cost | 648.89 | 618 | 624.8 | 612.18 |
| Best of All | 620.32 | 618 | 598 | 612.18 |
| ***QA194 Optimal***  ***Value: 9352*** | Average Distance From Optimum | 16.13 | 8.40 | 3.70 | 9.87 |
| Average Best Cost | 11162.36 | 10145 | 9711.67 | 10275.5 |
| Best of All | 10727.04 | 10145 | 9513 | 10275.5 |
| ***UY734***  ***Optimal Value:***  ***9352*** | Average Distance From Optimum | 13.33 | 11.30 | 9.13 | 17.07 |
| Average Best Cost | 91302.55 | 88084 | 87062.7 | 92622.6 |
| Best of All | 90245.2 | 88084 | 85505 | 92622.6 |
| ***ZI929***  ***Optimal Value:***  ***95345*** | Average Distance From Optimum | 11.91 | 8.60 | 5.81 | 17.25 |
| Average Best Cost | 108249.27 | 103570 | 101234.25 | 111793 |
| Best of All | 107512.72 | 103570 | 99344 | 111793 |
| ***LIM965***  ***Optimal Value:***  ***2789*** | Average Distance From Optimum | 18.70 | 9.10 | 11.62 | 19.76 |
| Average Best Cost | 3313 | 3043 | 3155.95 | 3340.59 |
| Best of All | 3313 | 3043 | 2843 | 3340.59 |
| ***XIT1083***  ***Optimal Value:***  ***3558*** | Average Distance From Optimum | 16.40 | 14.20 | 18.07 | 19.80 |
| Average Best Cost | 4257.89 | 4082 | 4342.95 | 4266.96 |
| Best of All | 4135.2 | 4082 | 3910 | 4266.96 |

Through the implementations of different algorithms to find a near-optimal value for the Travelling Salesman Problem, we can observe that the ACO algorithm tend to find the a result 1-3% close to the maximum, and Sweep algorithm behaves in a similar way, while sometimes finding the optimum when the dataset contains 25 - 150 locations. Meanwhile, the GRASP algorithm examined through the same datasets that have lesser locations produces better results compared to other examined algorithms with a nearer value to the optimal solution. Tabu search is successful in big data search however in small data sets the average case is not as successful as Sweep or GRASP. One possible reason for that is stucking too many local optima search processes terminate before finding global optimum because of reaching maximum iterations. The same problem occurred in DJ38, too many local optima decreased the performance of Tabu Search and performed poorly compared to other algorithms such as Sweep and GRASP. Although QA194 has more nodes thus its complexity is higher, Tabu Search did find the nearest to the optimum solution.

Moreover, it is clearly evident that the ACO and Sweep algorithms produce results that are more distinct from one another when the dataset contains more than 150 locations. This can be seen from the dataset QA194 where the difference between ACO and Sweep is more than 7%. On the other hand, GRASP on the same data set, QA194, presents a similar solution compared to the Sweep algorithm as its average distance from the optimum by percentage is 9.87. Tabu search, however, produces the closest value to the optimum.

Furthermore, when the algorithms are tested using the medium-sized dataset UY734, ACO and Sweep algorithms are again observed to have a similar result in comparison to the average distance from optimum by percentage, with Sweep having better accuracy. In the medium-sized dataset, the GRASP algorithm is examined to produce a result that has more of a poor quality in the accuracy comparison. In mid-size datasets, Tabu Search achieved acceptable average results compared to the other algorithms. It is noticeable that, although Tabu search found solutions near to the optimum, it also found solutions far from the optimum in some searches.

Lastly; as the bigger datasets which have 900-1000+ locations are examined, it is noted that GRASP continues to have an average distance from the optimum by percentage values between 17-20%. In comparison, it is noticed that the Sweep algorithm produces better results, especially in the dataset ZI929 where it has a result that is nearly 8% better than the GRASP algorithm. On the other hand, ACO is observed to have a closer value to the Sweep but not with a more accurate solution. Tabu Search could find solutions nearest to the optimum given time and find lots of local minimums; however, it did not find a good neighbour and stuck with local optimums in some executions, thus average performance worse than other algorithms. Additionally, when LIM963 and XIT1083 are examined, it is remarked that all of the algorithms, except Sweep, are producing a result with more than 15% accurate values compared to the optimum. In the biggest dataset in our examination, XIT1083, the Sweep algorithm produces the best result with the nearest value to the optimal solution. While on the other hand, the Tabu search could find solutions which are near to the optimum solution, its average case was high at 18.07.

Overall, GRASP seems to be the perfect pick for datasets up to about 190 cities, after which point Tabu search becomes the ideal solution, and at about 950 cities Sweep becomes preferable, while the ACO algorithm remains as an all around good solution for all database sizes.

Through these observations and results of the algorithms, it is proven that different algorithms that are implemented to find a near-optimal value produce distinct results in comparison to their values by percentage in “Average Distance from Optimum”. It has been observed that the accuracy of each algorithm’s solution can differ from one another by more than 10% if the dataset is big enough. However, the solutions are close to each other as the percentage value is less than 5 when the dataset contains lesser locations.

1. **Discussion and Conclusion**

This study is mainly a guideline to four algorithms for solving the Travelling Salesman Problem and also puts forward a frame for further research about how ACO, Sweep, GRASP and Tabu Search perform in different sizes of datasets at the same time criteria. Throughout the project, different implementations of algorithms have been used along with different local search techniques.

Our study can be improved in different datasets which vary in both complexity and size. This kind of research would be crucial for TSP researchers to understand the behaviour of algorithms in a deeper perspective. Furthermore, the study can be advanced by analyzing the algorithms using the same computer language instead of using numerous computer languages like in this project. Moreover, the study can be improved by examining different algorithms and local search techniques that can be applied to the Travelling Salesman Problem to further investigate the accuracies of distinct implementations and find more near-optimal solutions.

1. **References**

*Croes, G. A. (1958). A Method for Solving Traveling-Salesman Problems. Operations Research, 6(6), 791–812.* [*http://www.jstor.org/stable/167074*](http://www.jstor.org/stable/167074)

*Dikmen, H., Dikmen, H., Elbir, A., Ekşi, Z., & Çelik, F. (2014). Gezgin Satıcı Probleminin Karınca Kolonisi ve Genetik Algoritmalarla Eniyilemesi ve Karşılaştırılması. Süleyman Demirel Üniversitesi Fen Bilimleri Enstitüsü Dergisi, 18(1), 8–13.* [*https://doi.org/10.19113/sdufbed.74660*](https://doi.org/10.19113/sdufbed.74660)

*Feo, T. A., & Resende, M. G. C. (1995). Greedy randomized adaptive search procedures. Journal of Global Optimization, 6(2), 109-133.*

*Fiechter, C. N. (1994). A parallel tabu search algorithm for large traveling salesman problems. Discrete Applied Mathematics, 51(3), 243–267.* [*https://doi.org/10.1016/0166-218x(92)00033-i*](https://doi.org/10.1016/0166-218x(92)00033-i)

*Gutin, G., & Punnen, A. (2002). The traveling salesman problem and its variations. Springer Science & Business Media.*

*Hoffman, K. L., Padberg, M., & Rinaldi, G. (2013). Traveling Salesman Problem. Encyclopedia of Operations Research and Management Science, 1573–1578.* [*https://doi.org/10.1007/978-1-4419-1153-7\_1068*](https://doi.org/10.1007/978-1-4419-1153-7_1068)

*Ilhan, İ. (2016). Solution for the Travelling Salesman Problem with a Microcontroller-based Instantaneous System. International Journal of Intelligent Systems and Applications in Engineering, 4 (4), 122-127. Retrieved from https://dergipark.org.tr/tr/pub/ijisae/issue/27840/294649*

*Lawler, E. L. (1985). The traveling salesman problem: A guided tour of combinatorial optimization. New York, NY: John Wiley & Sons.*

*LineSweepAlgorithms.(2018).* [*https://www.topcoder.com/thrive/articles/Line+Sweep+Algorithms*](https://www.topcoder.com/thrive/articles/Line+Sweep+Algorithms)

*Mazidi, A., & Damghanijazi, E. (2017). Meta-Heuristic Approaches for Solving Travelling Salesman Problem. International Journal of Advanced Research in Computer Science, 8(5), 18–23.* [*https://doi.org/10.26483/ijarcs.v8i5.3560*](https://doi.org/10.26483/ijarcs.v8i5.3560)

*Misevičius, A. (2004, October 5). Using Iterated Tabu Search for the Travelling Salesmen Problem | Information Technology and Control.* [*https://www.itc.ktu.lt/index.php/ITC/article/view/11858*](https://www.itc.ktu.lt/index.php/ITC/article/view/11858)

*Nuriyeva, F., & Kizilates, G. (2017). A NEW HEURISTIC ALGORITHM FOR MULTIPLE TRAVELING SALESMAN PROBLEM. TWMS Journal of Applied and Engineering Mathematics, 7(1).*

*Pedro, O., Saldanha, R., & Camargo, R. (2013). A Tabu Search Approach for the Prize Collecting Traveling Salesman Problem. Electronic Notes in Discrete Mathematics, 41, 261–268.* [*https://doi.org/10.1016/j.endm.2013.05.101*](https://doi.org/10.1016/j.endm.2013.05.101)

*Ramli, N., Othman, R. R., & Fauzi, S. S. M. (2018). Ant Colony Optimization Algorithm Parameter Tuning for T-way IOR Testing. Journal of Physics: Conference Series, 1019, 012086.* [*https://doi.org/10.1088/1742-6596/1019/1/012086*](https://doi.org/10.1088/1742-6596/1019/1/012086)

*ŞAHİN, Y. (2019). SEZGİSEL VE METASEZGİSEL YÖNTEMLERİN GEZGİN SATICI PROBLEMİ ÇÖZÜM PERFORMANSLARININ KIYASLANMASI. Bolu Abant İzzet Baysal Üniversitesi Sosyal Bilimler Enstitüsü Dergisi, 19(4).* [*https://doi.org/10.11616/basbed.v19i51339.558208*](https://doi.org/10.11616/basbed.v19i51339.558208)

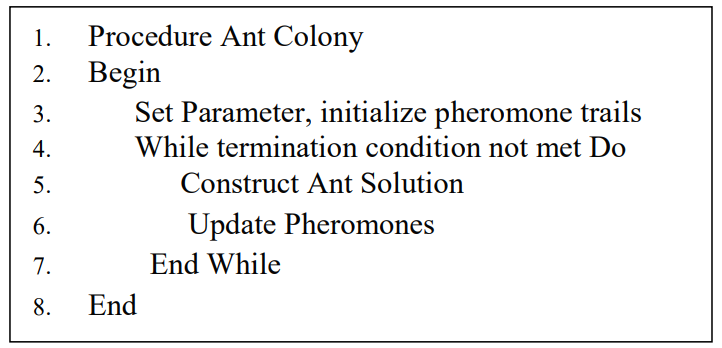
*Traveling Salesman Problem – Dynamic Programming Approach. (n.d.). Baeldung. Retrieved January 5, 2023, from* [*https://www.baeldung.com/cs/tsp-dynamic-programming*](https://www.baeldung.com/cs/tsp-dynamic-programming)

*TSP Test Data. (n.d.).* [*https://www.math.uwaterloo.ca/tsp/data/index.html*](https://www.math.uwaterloo.ca/tsp/data/index.html)

*Yousefikhoshbakht, Majid & Sedighpour, Mohammad. (2012). A combination of sweep algorithm and elite ant colony optimization for solving the multiple traveling salesman problem. Proceedings of the Romanian Academy - Series A: Mathematics, Physics, Technical Sciences, Information Science. 13. 295-301*

**Appendices:**

**Appendix A:** The pseudocode for the Ant Colony Optimization algorithm:



**Appendix B:** The pseudocode for the Sweep algorithm:

BEGIN

path = angular\_clockwise\_sweep(sorted\_points)

// use angular clockwise sweep to find the initial path

start = path[0] // initialize a variable to store the starting point

direction = "out" // initialize a variable to store the current direction of the linear sweep

for angular\_clockwise\_sweep(points): // for every iteration in the angular sweep

if direction==”out”: // sweep according to the direction and toggle direction afterwards

linear\_outward\_sweep(points)

direction = “in”

else:

linear\_inward\_sweep(points)

direction = “out”

while path\_improved: // loop until the path is improved

improved\_path = local\_search(new\_path) // use local search to find a better path

if path\_improved(path, improved\_path): // update the path if it has been improved

path = improved\_path current\_angle = get\_angle(start, path[1])

direction = toggle\_direction(direction)

return path

END

**Appendix C:** The pseudocode for the Tabu Search algorithm:

BEGIN

TABU\_SEARCH(size, distance\_matrix)

tabu\_list <- empty list

choose random index

current\_solution <- CONSTRUCT\_INITIAL\_ROUTE(,index, size, distance\_matrix)

current\_distance <- CALCULATE\_TOTAL\_DISTANCE(size, distance\_matrix, current\_solution)

best\_solution <- current\_solution

best\_distance <- current\_distance

for i = 1 to MAX\_ITERATIONS

candidate\_solution <- LOCAL\_SEARCH(size, distance\_matrix, current\_solution)

candidate\_distance <- CALCULATE\_TOTAL\_DISTANCE(size, distance\_matrix, candidate\_solution)

if candidate\_distance < current\_distance

current\_solution <- candidate\_solution

current\_distance <- candidate\_distance

if candidate\_distance < best\_distance

best\_solution <- candidate\_solution

best\_distance <- candidate\_distance

tabu\_list <- ADD\_MOVE\_TO\_TABU\_LIST(tabu\_list, candidate\_solution, size)

else

if MOVE\_NOT\_TABU(tabu\_list, candidate\_solution, size)

current\_solution <- candidate\_solution

current\_distance <- candidate\_distance

tabu\_list <- ADD\_MOVE\_TO\_TABU\_LIST(tabu\_list, candidate\_solution, size)

tabu\_list <- REDUCE\_TABU\_LIST(tabu\_list)

return best\_solution

END

**Appendix D:** The pseudocode for the GRASP algorithm:

1. Program Starts - dataset & the starting point is entered

* function GRASP

1. The candidate list size for the dataset selected is defined.

* WHILE cities are remaining

1. The nearest candidate elements are pushed back to each city’s candidate list.
2. Route is continued by randomly selecting the next city from the current city’s candidate list.

* FOR the candidate list size

1. Swapping multiple times is done for each iteration after selecting the next city for the route.

- function SWAP (called in function GRASP)

* END FOR the candidate list size
* END WHILE cities are remaining

1. After the route is constructed multiple swapping is again done to improve the solution quality. (In a for loop that is dependent on the dataset size)

* GRASP function called multiple times (dependent on the dataset size)

1. This process of GRASP along with swaps is done multiple times to find a nearer optimal solution in the time limit designated.
2. Program Ends - outputs the route with coordinations & total distance travelled.