Research Report

Approaching The Travelling Salesman with Machine Learning Algorithm

Luther College, Summer 2020

Loi Mai

Research Advisor: Dr. Roman Yasinovskyy

I. Abstract

The Travelling Salesman Problem (TSP) is one of the most studied computational problems in the past and current century. The essence of the TSP is to find the shortest round trip path to all the points in a given set of points.Currently, the state-of-the-art approach to a given TSP problem is by running William J. Cook’s Concorde Solver which employs Lin-Kernighan heuristics and branch-and-bound method, all leading algorithms in this field. The Concorde Solver guarantees that the solution to a given TSP is optimal should it finishes the processing. However, solving instances of thousands of points takes a considerable amount of time.

My initial plan with this research opportunity is to find an original Machine Learning oriented approach to this problem. However, after learning more about the foundational work in the field, the goal of this research becomes finding a heuristics method to approximate the optimal route through a given set of nodes. Ultimately, despite not having been able to craft the learning algorithms for this heuristics, I would like to make an argument that such Machine Learning algorithms exist for finding better heuristics. Thus, my heuristic approach aims to use this Concorde Solver in Divide and Conquer algorithms to minimize both the processing time and the error that arises from my method.

II. Research Outcomes

1. Previous Work On Divide and Conquer (D&C) Approach

During the course of this research, I have come across numerous other papers published in the intention of applying D&C to the TSP problem. For example, this approach [1] both consisted of solving for sub tours and then merging them together to provide a solution to the problem. However, upon closer examination, I hypothesize that merging sub tours would be suboptimal. If one breaks any tour apart, the individual components are segments with two ends instead of a full tour. Thus, to research a D&C method that combines the right sub-elements of a tour becomes the goal of this project.

2. Neighborhood Groupings Approach

This section devotes to describing the specific processes of the heuristic method. We first assume how close the points together to be considered neighbors, *d*, are determined. This can be represented by a circle of radius $\frac{d}{2}$ around every point in the TSP instance. In the visualization below, if the circle of a point comes into contact with a circle of another point, they will be considered neighbors and put into the same group. The question of how to best determine a radius for any cases will be addressed at the end of this section. For the demonstration of the algorithm, we consider a simple TSP instance of 15 points generated in a box of 100×100 pixels with information from Figures 1 and 2. Let the radius be 11.

Step 1: Group all points that are neighbors. As seen in Figure 2, there are 7 distinct neighborhoods that are present when points within the distance of 22 pixels or less are considered neighbors. The information on which point is inside which neighborhood can be stored in a list of lists.

Fig. 1: Graph of a random TSP instance

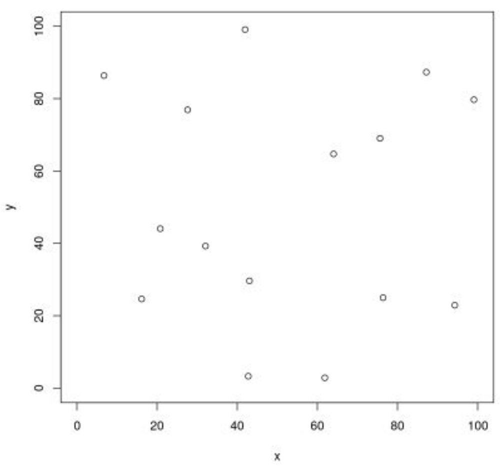
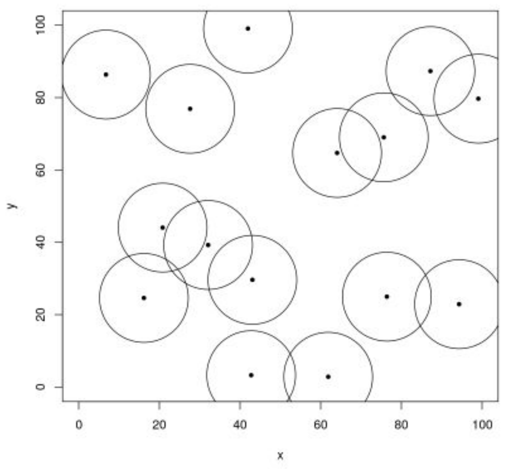
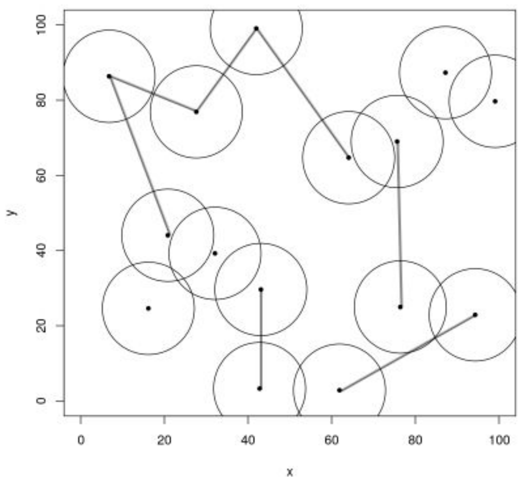


Fig. 2: Circles of Radius 11 px added



Step 2: Use the Concorde Solver to solve for the optimal path to traverse all neighborhoods that are determined by Step 1. It is possible to achieve due to the fact that the solver uses the information of a distance matrix. For example, if the Concorde Solver had to solve this instance alone, it would input the 15×15 distance matrix. In this case, we wish to input the 7×7 distance matrix that reflects the distances between neighborhoods. Note that the distance between any two neighborhoods is the shortest distance between a pair of points, each shared by 1 neighborhood. In this specific case, the Concorde Solver gives the optimal solution to traverse the neighborhoods as follows.

Fig. 3: Visualization of the neighborhood's tour



There is a case where the neighborhood tour leads to conflicts. If one looks closely at the bottom right corner of Figure 4, we can see that simply letting the endpoints be the pair of points that produce the shortest distance does not always work. Consider the neighborhood A containing point A1 and A2. The neighborhood tour instructs that neighborhood A has to be connected to neighborhood B and C. A1 has the shortest distance to the point B1 and C1. However, it cannot be the case that neighborhood A has only 1 endpoint since if A1 connects to both B1 and C1, then A1 cannot connect to A2, which is a necessary condition of a legal tour. The same scenario also happens at the top right neighborhood in Figure 4. Thus, there is a mechanism applied that checks for and accommodates this type of conflict.

Fig. 4: Determination of endpoints

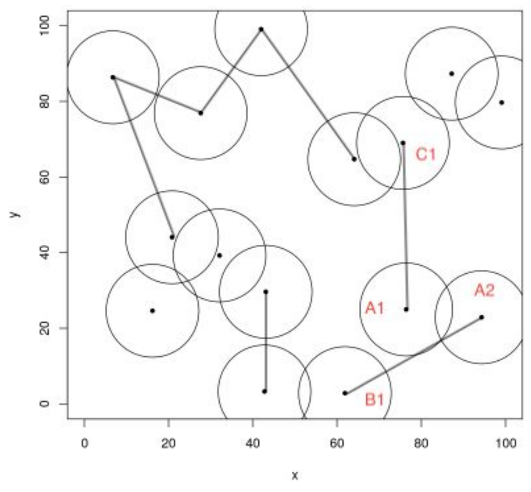
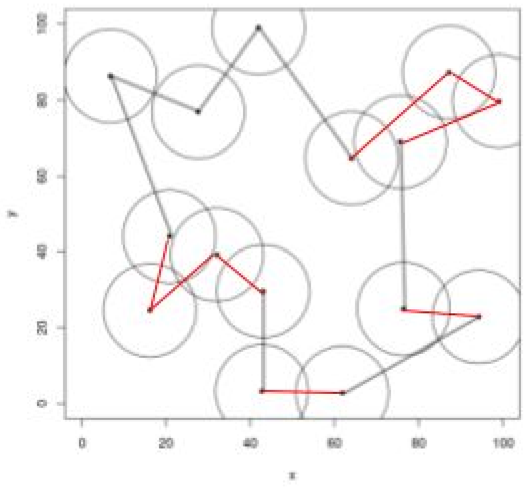


Fig. 5: Two-end Tour Within Neighborhood (red)



Step 3: Solving subcases. Notice for each set of points corresponding to the neighborhood division, there are 2 endpoints (with the exception of neighborhoods containing only 1 point each). In this case, we are interested in an optimal tour that traverses between two endpoints without looping back. Luckily, this problem can be tackled by adding a dummy city with special values to force the optimal tour to behave like we described. The dummy city needs to be infinitely far away from all the points that are not endpoints (represented by value Inf in R)

and has a distance of 0 to the 2 endpoints. Regarding the methodology, recall that the Concorde Solver inputs a distance matrix, with each entry of row *i* and column *j* reflecting the distance between the point *i* and point *j.* One then can add another row and column to the distance matrix reflecting the described dummy city using the built-in function *insert_dummy* function of R's "TSP" package. The tours within their respective neighborhoods are depicted in Figure 5 (red-marked sections).

Step 4: Merge 2-end tours of subcases within the neighborhood according to the neighborhood tour solved in step 2. The resulting tour is presented below alongside with the optimal tour produced by Concorde Solver running alone.

Fig. 6: Tour produced by D&C algorithm

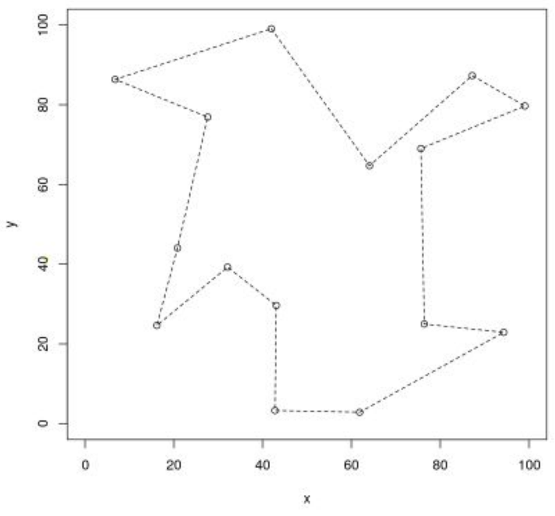
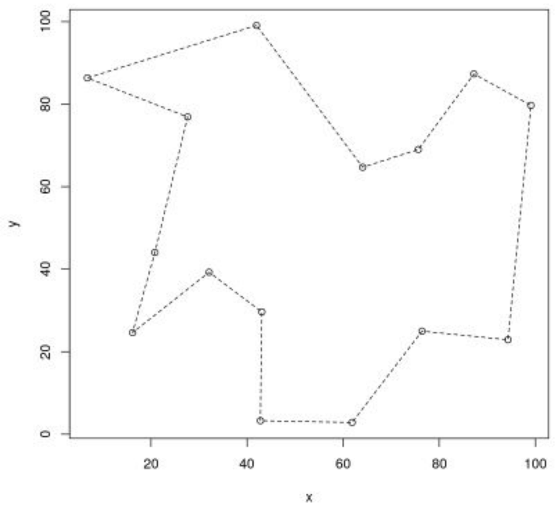


Fig. 7: Tour produced by Concorde Solver



In this example, the tour produced by this neighborhood D&C algorithm is 5.9% longer than the optimal tour. Regarding processing time, since this is a small set example, the method is slower than Concorde Solver running from the start. However, due to the method's ability to reduce the number of points in the TSP which Concorde Solver has to solve, we expect that this method will perform significantly faster than Concorde Solver in larger samples. In the next section, we will be benchmarking both D&C and Concorde Solver larger samples. Since a method of determining the radius is absent, the tests will be performed with a range of radii.

3. Performance results

The tests of the algorithm were performed on Luther College’s server, which notably consists of 24 CPUs and 100 Gb RAM. However, the multicore setting does not affect the running time as our code in the R notebook only utilizes an individual processor. One unfortunate fact is that the memory performance is not recorded, and thus cannot be assessed in this paper. The processors’ model in use is Intel(R) Xeon(R) CPU E5-2440 operating at 2.40GHz. During the course of the research, we have tested out the performance of this algorithm on many cases with the number of points ranging from 734 to 8246, yielding data on processing time and accuracy. The accuracy is determined in this case by the formula

$$\left(\frac{heuristic\ tour\ length}{optimal\ tour\ length} - 1\right) \times 100\ (\%)\,,$$

where lower value is better and the value 0% means having the optimal length. The test cases are taken from the TSPLIB library “National TSP Collection” at TSP Test Data. [insert citation]

Test case 1 is Uruguay 734-city instance (uy734.tsp) with Figure 8.1 and 8.2. The Concorde Solver took 151 seconds to solve this instance.

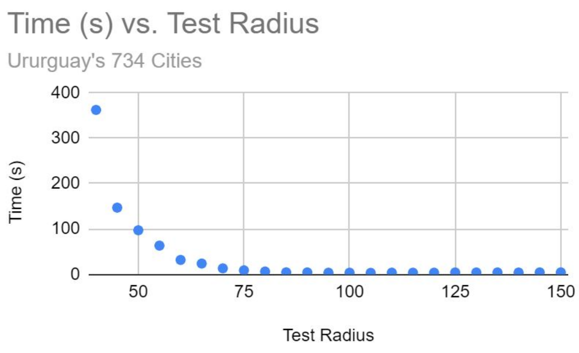


Fig. 8.1: Processing Time vs. Radius

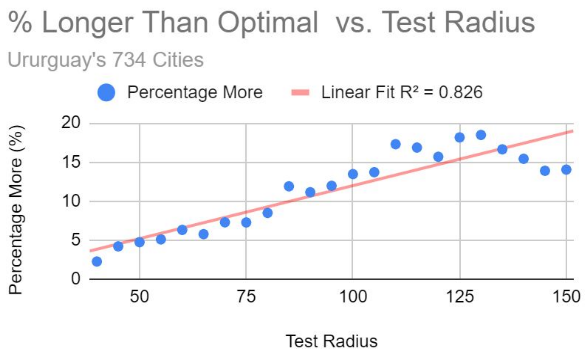
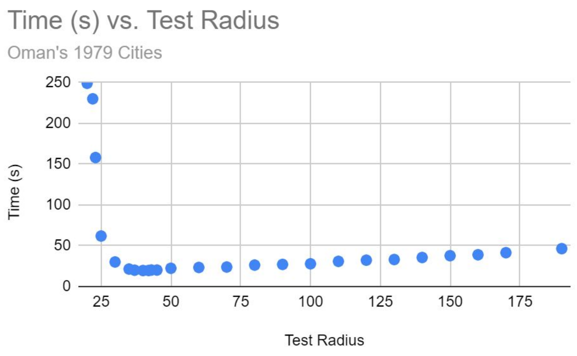


Fig. 8.2: Percentage Over vs. Radius

Note: As for Figure 8.1, even though the trajectory seems flat as the test radius gets bigger, there is still a slight uptick trend towards the end that is hard to discern. As we progress with other larger examples, the slight uptick will become more visible. Also, notice that once the radius is small enough, this method's processing exceeded Concorde's processing time.

Test case 2 is Oman's 1979-city instance (mu1979.tsp) with Figure 9.1 and 9.2. The Concorde Solver took 2493 seconds to solve this instance.



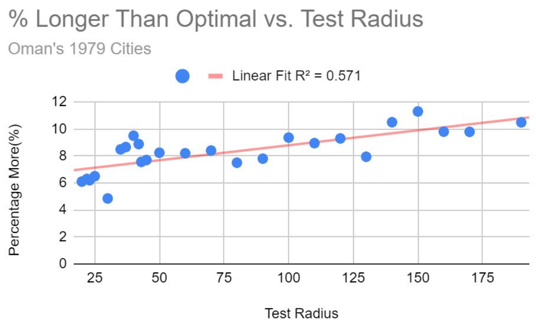
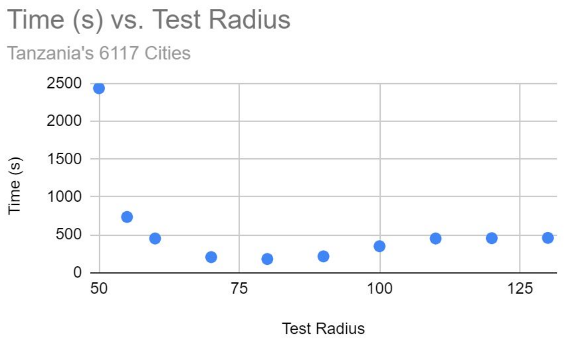


Fig. 9.1: Processing Time vs. Radius Fig. 9.2: Percentage Over vs. Radius

As for test case number 3 and 4, due to Concorde Solver's large computation time, that data is not available. Previous Concorde runtime on this test from other researchers cannot be compared with due to the test machine's different configurations. Test case 3 is Tanzania's 6117-city instance (tz6117.tsp) with Figure 10.1 and 10.2.



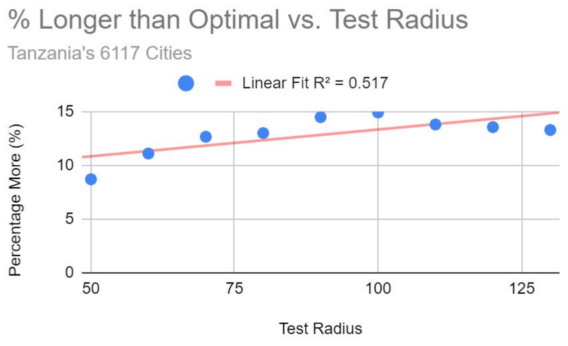
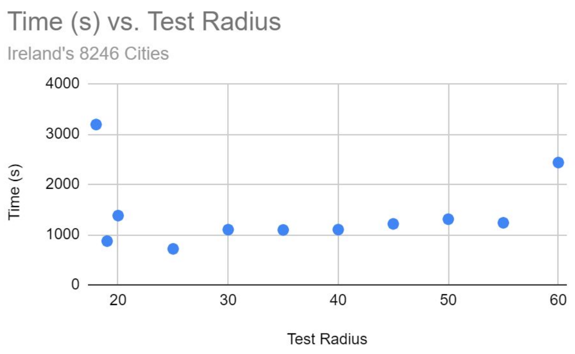


Fig. 10.1: Processing Time vs. Radius Fig. 10.2: Percentage Over vs. Radius

Observation: There might be a trend that is not linear, but rather logarithmic as the data points form a curve rather than a line. However, we do not have a detailed explanation as to why this curve-like trend occurs.

Test case 4 is Ireland's 8246-city instance (ei8246.tsp) with figure 11.1 and 11.2. It is noted that while the processing time chart forms a valley, the percentage of heuristic tour length more than optimal is trending downward with a linear fit. This might be caused by the overall structure of the TSP that we have not been able to identify.



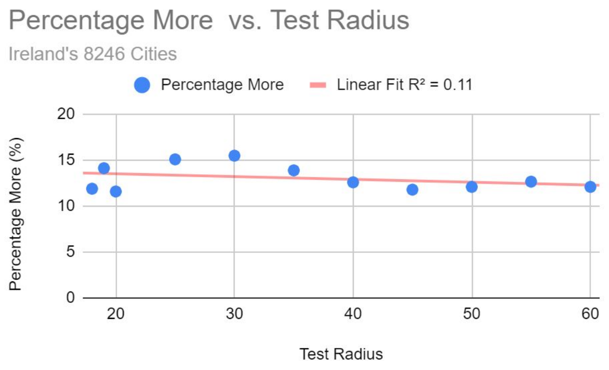


Fig. 11.1: Processing Time vs. Radius Fig. 11.2: Percentage Over vs. Radius

Before analysis, it is important to note that the test radius cannot be compared between instances since the spread of points are not normalized in a controlled environment, such as a square box of a fixed side length. As a result, the radius has no official unit aside from representing pixels on the graph.

From these 4 examples, there are two observable trends reflected by the graphs of test radius versus processing time and versus accuracy (percentage longer than optimal solution). First of all, the processing time correlation forms a valley when radius changes. This suggests that choosing a radius that minimizes the processing time is also an optimization problem. However, the factors that affect a given radius' effectiveness in this method are still unknown. Regardless, we would like to hypothesize that this valley form exists because of the dynamic factor of this Divide and Conquer Neighborhood method. If the radius is small, then the number of neighborhoods will be roughly the same as the number of points, albeit of smaller size. This load will take a long time for the Concorde Solver to solve since not a lot of points are taken out of consideration. This will lead to less error in the final heuristic tour, but takes considerably more time. In contrast, if the radius is too large, then the neighborhood algorithm will take time to sort points into their neighborhoods. This increase is mild as the graphs demonstrated compare to the increase in time when the Concorde Solver has to handle much bigger cases. Unfortunately, the downside is the lack of accuracy where too many points were generalized to be in the same neighborhood. Ultimately, a right mix of these aforementioned factors will lead to the shortest processing time, minimizing both the Concorde Solver's load and the neighborhood determination steps. However, it will take more data on the individual processes in order to verify this hypothesis.

The second trend can be seen in test radius versus accuracy figures. The data suggest that the percentage over the optimal length increases linearly with test radius. Thus, this method has to be used with care as over grouping will create a big margin in error.

4. Assessment

In order to assess the Divide and Conquer Neighborhood method proposed, there is an assumption that needs to be made. Since a reliable determination of radii such that the processing time is optimal is absent, the radii to be considered is taken within the "valley" of the Processing Time graph. This range is reasonable because the main goal of the Divide and Conquer method is to save time. However, because this range of radii in the "valley" are chosen by the author's approximation (Figure 13) instead of being clearly defined, there might be slight error in the evaluation process.

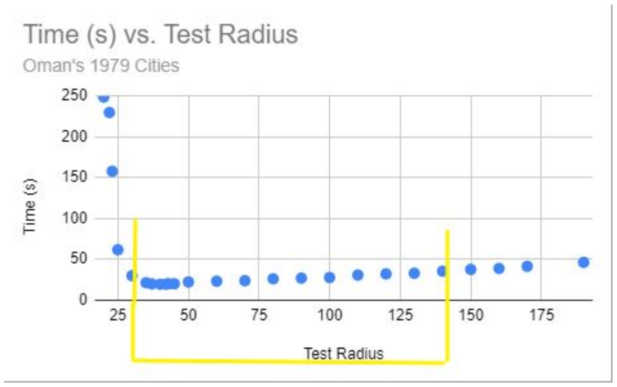
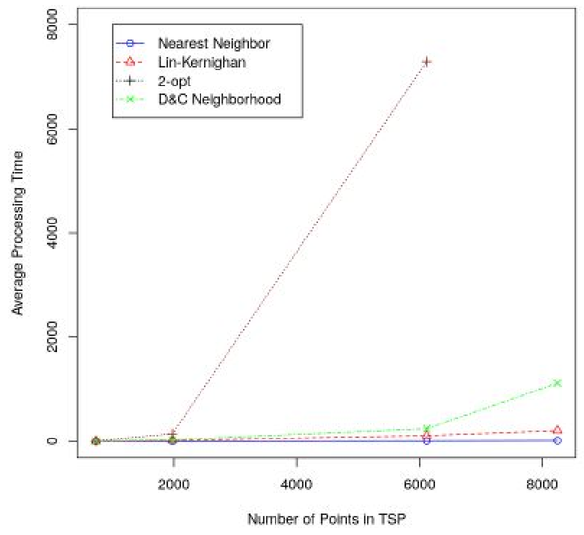


Fig. 13: An Approximation of A “Valley” in Oman’s 1979-city TSP Processing Time

The most effective method to assess this heuristic algorithm performance is to compare with other known heuristics methods. The three other heuristics will be Nearest-Neighbor, 2-opt edge exchange, and Lin-Kernighan which uses dynamic k-opt edge exchange. Each method was tested approximately 10 times in the process. The average accuracy and average processing time from all methods are depicted in the following Figure 12.1 and 12.2 (error bars removed due to small value).



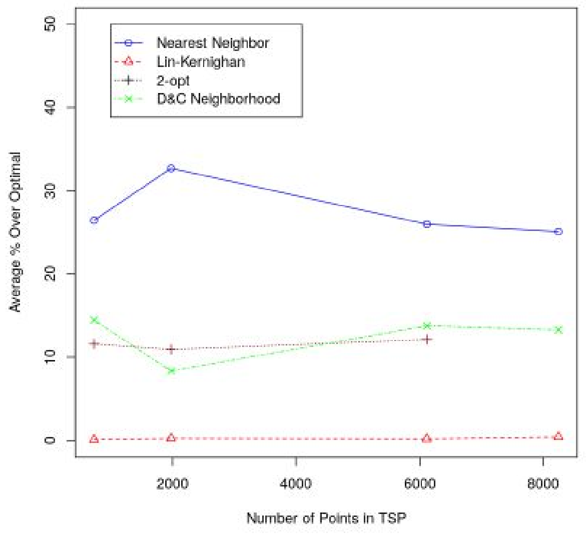


Fig. 12.1: Average Processing Time Fig. 12.2: Average Percentage Over Optimal

From these two performance graphs, there are three major conclusions that can be made. First of all, the Lin-Kernighan (LKH) heuristic performs better than the proposed method in both processing time and accuracy. This result is unsurprising because LKH was ground-breaking in the field of TSP research. Secondly, there cannot be any meaningful comparisons between our proposed method and Nearest Neighbor (NN). The NN method takes minimal time to complete the task with rough ~33% over optimal path length, while our algorithm takes longer to achieve the same task but with higher accuracy of around 13%. However, our method is comparable to the 2-opt edge exchange method in terms of accuracy, but has a much lower processing time. Furthermore, while conducting the 2-opt test on Ireland's 8246-city instance, the 2-opt test takes 5 hours 42 minutes (20844 seconds) to produce a tour that is around 12% longer than optimal. However, due to the long processing time, only one trial was conducted. Thus, this data point is inherently prone to error and was omitted from the graph for better examination of other trendlines. It is worth noting that the 2-opt method does have a good consistency of around 12% over the optimal length. Nevertheless, under the assumption laid out, our algorithm is likely to provide more utility than 2-opt since the differences between accuracies are small while the amount of processing time saved is significant.

5. Future Work

The first imperfection of this algorithm is that a specific way of determining a good radius of any TSP is unknown. Thus, solving for an instance requires a person to first look at the neighborhoods formed by multiple radii to estimate which radius would be likely to give the fastest processing time without sacrificing too much accuracy. This lack of automation means that, currently, this method cannot be automated efficiently.

An additional problem that comes with human estimation is that it is significantly harder to examine the larger instances because there are too many other points to look at. This issue has prevented us from testing larger scale problems since picking the inappropriate radii leads to huge processing time.

However, this problem can be tackled in another research session to look into how Machine Learning (ML) models can substitute human estimation. For example, the ML model can mimic the human process of inputting multiple graphs of neighborhoods to guess the radius. Another approach would be to feed the number of points in each neighborhood for every neighborhood, and run the ML model to see if there is any arising pattern. This ML model would be good enough if it can consistently predict a radius that is on the left of the described "valley" just before the big increase in performance time.

In terms of accuracy, there is a possibility that the approach of setting a fixed radius as a way of determining the neighborhood is an ill-suited strategy. The concept of neighborhood works if the neighborhood is isolated enough from the other points so that the global optimal tour would still pass through the neighborhood the shortest way possible. However, since the surrounding of each neighborhood might be different, one fixed way to determine one neighborhood will highly likely be over grouping and under grouping in many other neighborhoods. Thus, further research can be done to examine the feasibility of classifying non-uniform neighborhoods.

In terms of processing time, more work could be done to improve the individual processes. One notable area that looks for improvement is the generation of the neighborhood's distance matrix. The reason is that usually the steps to identify the new distance matrix for the neighborhood tour takes as much as 90-95% of the total processing time. This issue is caused when attempting to reduce an m-by-m matrix to a n-by-n matrix where m is the number of points and n is the number of neighborhoods. The distance entries in the neighborhood matrix must look for the shortest edge connecting two neighborhoods which cost significant time for the algorithm to check for a lot of pairs of points' distances. Thus, overall examinations of each method used in the process are necessary for this method's competitiveness.

III. Learning Process

1. History of the Travelling Salesman Problem

The most important takeaway is the history of TSP. When I made the first demo of the problem and the proposed Machine Learning method, I wasn't aware of the deep history TSP had with the previous century mathematicians and computer scientists. This research opportunity provides me with not only an opportunity to tackle this complex problem, but also to close the gap by reading books, papers, and the latest work in this field. The book *In Pursuit of The Travelling Salesman* by William J. Cook is fantastic in conveying the essence of this problem as well as a comprehensive list of efficient methods tested before. This research turns out to be a monumental step as the new knowledge convinced me to change the focus of the project since Week 1. The combination of branch-and-bound and Lin-Kernighan heuristics methods yields the Concorde Solver which is by far the most efficient approach to solving the exact optimal solution to a large TSP problem. My goal changed from trying to solve the problem from scratches to attempting to understand the Concorde Solver from activities such as setting up, benchmarking, reading the mathematical model, and learning about the procedures of the solving process.

2. Technical difficulties

During the course of this research experience, there were a lot of factors that I didn't anticipate. Aside from having to change the focus of research, troubles arise when I have to tackle the 2-decade-old-software's compatibility problems. The main programming language I used for this research is R. However, when the specific package in R is used to run the Concorde Solver, it produced an obviously non-optimal tour. In meetings with Dr. Yasinovskyy, I also reproduced the errors to discuss what could be the reasons for its malfunction. In the end, the debugging process took almost 2 weeks to find a makeshift alternative approach that includes both Bash scripts and R scripts. It was only after another 2 weeks did I manage to find a way to code all of the processing algorithms in R-based notebooks on Luther's server. My most important advice I received from Dr. Yasinovskyy during this phase was to be patient and work on other aspects of the project while keeping an eye on the original problem. In retrospect, this is a good reminder not to let myself be bogged down in some steps without achieving the end goal of the plan. Our meetings would always be divided into two phases. The first phase is dedicated to discussing the idea of the heuristics, while the second phase is to look together at the errors that we have been trying to test and debug. In the end, I am glad that we were able to overcome these technical challenges when there are not many documents and forums that discuss these specific issues that we were encountering.

3. R - A Brand New Programming Language

During the course of my study at Luther College, there are a lot of times I have heard of the R programming language that is dedicated to the statistical analysis world. While preparing for this summer research, I also discovered that R happens to have a dedicated package that implements a lot of past approaches, including the Concorde Solver. The two factors were fantastic since my plan is to implement the Concorde Solver and analyze a lot of the data coming from my heuristic methods. During the two months of using this language, I am really amazed by its dedication by a lot of the open-source developers who put out versatile packages that suit my needs. Thus, I am happy to incorporate this language into my future projects that require sophisticated analysis.

Reference

[1] Hoda A. Darwish, Ihab Talkhan, *Reduced Complexity Divide and Conquer Algorithm for Large Scale TSPs.* International Journal of Advanced Computer Science and Applications, Vol. 5, No. 1, 2014. [Online] Available at: https://thesai.org/Downloads/Volume5No1/Paper_10-Reduced_Complexity_Divide_and_Conquer_Algorithm_for_Large_Scale_TSPs.pdf