Density-Based Clustering Heuristics for the Traveling Salesman Problem

Matthew Agostinelli
University of Rhode Island, magostinelli@my.uri.edu

DENSITY-BASED CLUSTERING HEURISTICS FOR THE TRAVELING SALESMAN PROBLEM

BY

MATTHEW AGOSTINELLI

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

IN

SYSTEMS ENGINEERING

UNIVERSITY OF RHODE ISLAND

2017

MASTER OF SCIENCE THESIS

OF

MATTHEW AGOSTINELLI

APPROVED:

Thesis Committee:

Major Professor	Manbir Sodhi
	David Chelidze
	Frederick Vetter
	Nasser H. Zawia
	DEAN OF THE GRADUATE SCHOOL

UNIVERSITY OF RHODE ISLAND

2017

ABSTRACT

The Traveling Salesman Problem (TSP) is one of the most ubiquitous combinatorial optimization problems. Given a set of cities, the objective of the TSP is to generate a solution that ultimately minimizes the total distance traveled and ensures that each city on the tour is visited exactly once. The TSP is classified as $\mathcal{NP}$-hard, which implies that there is no polynomial time algorithm to solve the problem to optimality. Consequently, exact algorithms cannot be utilized to generate solutions in reasonable computing time. Metaheuristics have drawn much attention in recent years and many advancements have been facilitated by hybrid approaches wherein inspiration is drawn from other fields of study. Less research has focused on the utilization of hybrid strategies for the advancement of classic heuristic approaches.

This thesis presents a novel design conjoining two classic construction heuristics with density-based clustering. The density-based spatial clustering of applications with noise (DBSCAN) is used in conjunction with both the nearest neighbor and greedy heuristics. The efficacy of this method is evaluated by comparing non-aided greedy and nearest neighbor heuristics with those utilized in combination with DBSCAN. The results show that heuristic methods utilizing DBSCAN can facilitate a significant reduction in computation time while improving the quality of solutions obtained when compared with classic construction heuristics.

ACKNOWLEDGMENTS

I would like to offer my most sincere gratitude to Dr. Manbir Sodhi. He has been a mentor and friend throughout my time as a student at the University of Rhode Island. Dr. Sodhi's counsel, encouragement, and guidance has motivated me to explore new concepts and challenge myself throughout this research process.

I would also like to thank my two additional committee members, Dr. David Chelidze and Dr. Frederick Vetter for their participation, support, and commitment during this process.

Lastly, I would like to thank my entire family for their unwaivering encouragement.

TABLE OF CONTENTS

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Optimization

The past decade has witnessed a significant expansion in both applied mathematics and theoretical computer science. One field that has drawn considerable attention is numerical optimization. Optimization is at the heart of the rapid advancement of artificial intelligence, transportation and manufacturing systems, as well as of software development. The core objective of optimization is to either maximize or minimize some quantitative measure that can ultimately be represented by a single number. In order to do so, a model that is representative of the system characteristics must be formulated by identifying constraints and variables that directly impact the parameter to be optimized. Optimization algorithms are then used to compute the value of the system variables that facilitate optimality.

An optimization problem can take on one of two compositions; discrete or continuous form. Continuous optimization refers to problems in which model variables can take on any value within a particular range, while discrete optimization involves cases where variables can only equate to a value belonging to a distinct set. Generally, continuous optimization problems are easier to solve because the smoothness of such functions allows for the constraints and objective at a particular point to be used for the estimation of behavior at all nearby points. Discrete problems are increasingly difficult to solve, as nearby points do not necessarily have similar function values. There are several subfields of continuous optimization, including problems involving linear and nonlinear programming, quadratic programming, and unconstrained optimization. Discrete optimization problems consist of those involving integer linear programming and combinatorial optimization [32, 43, 71].

1.1.1 Combinatorial Optimization

The primary objective of combinatorial optimization problems is to solve for the optimal "object" (i.e., an integer number, permutation, or graph structure) from a discrete set [14, 74]. Naturally, as the number of objects increases, the ability to concisely represent the set in a graph or matrix becomes increasingly more difficult. Thus, it is often necessary to employ *heuristics*, or algorithms that can be used to generate approximate solutions [84].

One of the most widely used performance evaluations for optimization algorithms is the time complexity, or total computating time. In combinatorial optimization problems it is often infeasible to construct an algorithm that allows for all problem instances to be solved for optimality within a time bound that is polynomial [69]. Intuitively, problems that are solvable in polynomial time are referred to as $\mathcal{P}$, while $\mathcal{NP}$ indicates that the problem cannot be solved in polynomial time. A further classification is that of $\mathcal{NP}$-Hard problems, which are considered to be the most difficult class of $\mathcal{NP}$, as the computational time grows exponentially with the problem size [60, 69]. Thus, heuristics techniques are imperative for solving such problems, more specifically by allowing for approximate solutions to be obtained in reasonable computing times.

The fundamental motivation for expanded research within the field of combinatorial optimization is the array of applications such an optimization problem can be employed. Some of the most common problems involving combinatorial optimization include assignment problems, shortest path problems, and the traveling salesman problem [57, 69]. The traveling salesman problem (TSP) is a quintessential combinatorial optimization problem. Given a set of cities, and the corresponding distance between each pair, the problem objective is to determine the shortest possible route that visits each city exactly once and finishes at the initial starting point. As an $\mathcal{NP}$-Hard problem, optimal solutions are not guaranteed in polynomial time and hence the difficulty of the problem scales exponentially with its size. Consequently, exact approaches for solving the TSP are only feasible for very small-scale problems. The importance of the TSP is not only reinforced by the numerous applications of the problem principles, which are prevalent in vehicle routing, workshop scheduling, and computer wiring. However, the TSP itself is also highly representative of a larger class of problems, often serving as a benchmark for optimization algorithms. Moreover, if an algorithm capable of solving the TSP in polynomial time were discovered, this entire class of problems would also become solvable.

1.1.2 Clustering

Clustering has become an essential component to advancements in several fields including that of data mining, machine learning, as well as statistical data and pattern analysis [49, 16]. As a useful paradigm, clustering involves the aggregation of data into groups or clusters based on patterns or similarities. In practice, clustering can be considered as an assignment problem, as data points are designated to a particular group based on some sort of similar characteristic. Thus, clustering can be highly effective for the identification of relationships among several data points.

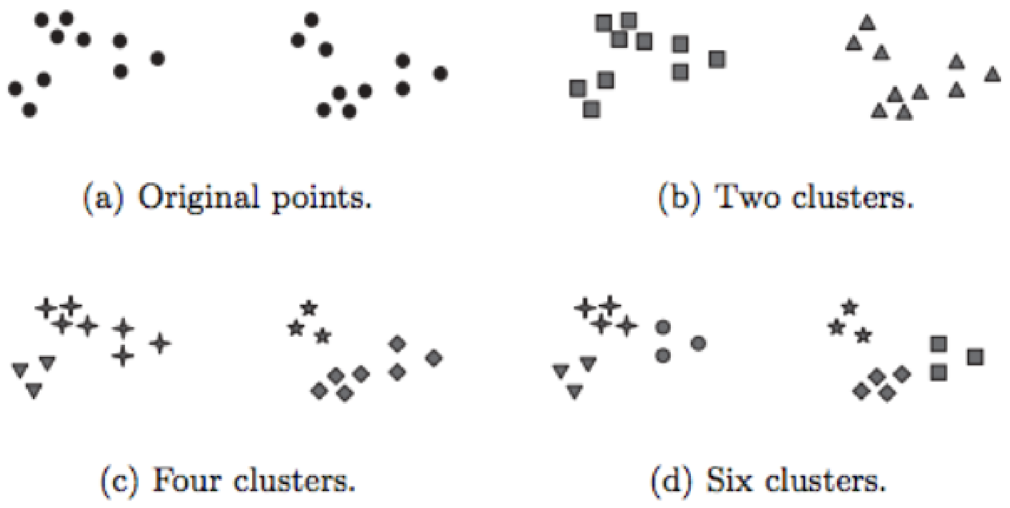


Figure 1: Varying methods of data clustering with the same set of points [89]

There are several different methods of data clustering which can be chosen directly based on the data and its application. As illustrated in Figure 1, the same data set can be clustered in several different forms based on the criterion for a pattern and the ultimate objective of the data analysis. Generally the entire collection of clusters is constructed by one of two methods: hierarchical or partitional clustering. In partitional clustering, each data point is in exactly one subset, whereas in hierarchical clustering each cluster could also include a subcluster [89]. There are also several approaches for the overall formation of a cluster, including that of well-separated, prototype-based, graph-based, and density-based clustering.

One of the most widely used forms of density-based clustering is DBSCAN (Density-Based Spatial Clustering of Applications with Noise) [15, 75]. DBSCAN was first proposed by Ester, Kriegel, Sander, and Xu in 1996 as a tool for detecting arbitrarily shaped clusters, as well as noisy outliers in data points [30]. With DBSCAN the density is determined by computing the number of points within a region of a specified radius around a particular point. Despite its proven usefulness in large databases [29, 92], DBSCAN has limited applications within combinatorial optimization problems. It is conjectured to be advantageous because it is uniquely suited to detecting arbitrarily shaped clusters and is also designed to incorporate noise and thus is not constrained to incorporate all data points.

1.2 Overview

This thesis presents the application of DBSCAN as a clustering assignment for existing TSP heuristics. Moreover, two novel heuristics are proposed, namely the clustered greedy heuristic and the clustered nearest neighbor heuristic. Although these two heuristics are of primary examination in this thesis, these clustering algorithms can also be employed with alternative heuristic methods. The efficiency of DBSCAN utilized in combination with the greedy and nearest neighbor heuristics is evaluated based on performance improvements when compared with non-clustered results. The methods demonstrate that DBSCAN facilitates an improvement in solution quality for both heuristics and a significant reduction in computing time for the greedy heuristic.

The remainder of this thesis is structured as follows. Chapter 2 provides a detailed background on the TSP and its applications. In Chapter 3 clustering methods with application to heuristics for solving the TSP are reviewed. Chapter 4 presents an examination of parameter derivation methods for DBSCAN, as well as an evaluation of the overall performance of both the clustered greedy and nearest neighbor heuristics. Conclusions and recommendations for future work are given in Chapter 5.

Chapter 2

The Traveling Salesman Problem

2.1 Origins

The traveling salesman problem (TSP) requires finding an optimal path for a salesman to travel through a predetermined set of cities. Optimality is defined as the shortest possible path in which the salesman visits each city only once and returns to the same city in which he started. The TSP has a rich history in applied mathematics and holds particular importance in theoretical mathematics as well. Naturally, the TSP can be found in numerous transportation, logistic, and vehicle routing applications. However, the application of the TSP is becoming more relevant in a growing number of fields including genetic engineering, satellite maneuvering, semi-conductor manufacturing, communication networking, and numerous others [81, 2, 7, 8, 86].

One of the first documented appearances of a routing problem in mathematics was that of the *knight's tour* where a solution was presented by Leonhard Euler in 1757 [31]. The knight's tour problem consists of finding a sequence in which every square of a chess board is visited exactly once and the knight ends it's tour on the same square in which it started. As evident from the problem's name, only valid knight moves are permitted. The knight's tour problem is recognized as a precursor to more generalized routing problems as well as to the TSP.

TSP related mathematical formulations and problems were later studied by Sir William Rowan Hamilton and Thomas Penyngton Kirkman in the 1800's. Hamilton presented a game in which a player needed to complete a tour through twenty points using only the connections specified as part of the game. The early work of Hamilton and Kirkman is presented in great detail in *Graph Theory 1736-1936* [13]. The general form of the TSP, which is the focus of this thesis, was first studied in the 1930's by Karl Menger in Vienna and Harvard [85]. The TSP gained more prominence after the problem was presented by Hassler Whitney and Merrill Flood at Princeton in 1934 [33]. A robust accounting of the TSP and its origins can be found in *In Pursuit of the Traveling Salesman*, authored by William Cook [18]. In this work, Cook makes note of original approaches to the TSP masked behind real world applications prior to the problem being commonly referred to as the TSP, such as in farmland equipment routing, school bus routing, and police watch tours.

Evolving research methods and solution strategies naturally spawned variants and abstractions of the TSP. Generally, there are three major variants of the TSP in which most routing problems are derived. The majority of abstractions simply introduce an additional constraint on the solution space, or seek to generalize a previously presented problem instance. The first form of the TSP is the symmetric traveling salesman problem (sTSP); this is the case of the problem that is most synonymous with the TSP. In the symmetric case, the costs of traveling from one city to another is the same regardless of the direction of travel. The asymmetric traveling salesman problem (aTSP) is that in which the cost of traveling from one city to another is dependent on the direction traveled. Lastly, the multiple traveling salesman problem (mTSP) is that in which multiple salesman start at a single city and each city must be visited by only one salesman with each traveler returning to the starting city. The most ubiquitous mTSP's include the vehicle routing problem (VRP) and the capacitated vehicle routing problem (CVRP), which were initially introduced by Dantzig as the Truck Dispatching Problem [21]. In this variant of the mTSP each truck is restricted by its capacity and must either deliver or pickup some quantity of goods in each city it visits, always returning to the starting city. Evidently, a tremendous number of objective criterion and constraints can be introduced into these problems. Some of the most frequently studied include multi depot routing, time window constraints, precedence constraints, and variable salesman or vehicles [83].

While the concepts and applications of this research may also be employable within other TSP variants, the primary focus will be on the sTSP, which will hereby be referred to simply as the TSP.

2.2 Problem Formulation

As previously stated, this research focuses on the TSP and more specifically the symmetric variant of the problem where costs between cities are symmetric. The TSP can thus be formally constructed using the following mathematical representation. Let $G = (V, A)$ be an undirected, complete graph where $V = \{v_1, \ldots, v_n\}$ is a set of nodes representing the cities and $A = \{\{i, j\} : i, j \in V\}$ is a set of arcs representing the edges between cities. C defines a cost matrix in which $c_{ij} = c_{ji}$ is the cost associated with edge $\{i, j\} \in A$. Additionally, $c_{ij} = 0 \, \forall \, i, j$ where $i = j$. Typically the distance between nodes is the Euclidean distance, or some similar metric as defined later in this section. The cost matrix satisfies the triangle inequality when $c_{ij} \leq c_{ik} + c_{kj} \; \forall \; i, j, k$. The objective of the TSP is to find a tour that visits each node once and returns to the starting node while minimizing the total distance traveled.

2.2.1 Mathematical Programming Formulations

The TSP can be mathematically represented in numerous forms, as illustrated by Orman and Williams' survey of Integer Programming Formulations where eight unique models are presented [72]. Although the research focuses exclusively on the aTSP, the survey highlights the robust research and literature dedicated to finding suitable mathematical representations of the problem. The most cited mathematical formulation for TSP was formulated in 1954 by Dantzig, Fulkerson, and Johnson [20]. The formulations presented in [72] rely on the following conventions.

The set of cities:

$$N = \{1, 2, 3, \ldots n\}$$

The path variables:

$$x_{ij} = \begin{cases} 1 & \text{if } arc(i,j) \text{ is in the tour} \\ 0 & \text{otherwise, } i \neq j \end{cases}$$

The costs (distances):

$$c_{ij} = \text{distance of } arc(i,j)$$

The primary difference between the aTSP and the sTSP in the following formulations is the binary assignment variables and the rules restricting the iterators for i and j. In the symmetric case, only the upper triangle (or lower) of the cost matrix, C, needs to be considered because the corresponding distance value is identical where $i, j = j, i$. Although this distinction is important, for the following formulations the generalized aTSP case is presented for robustness and clarity.

2.2.2 Conventional Formulation [20]

$$\min \sum_{ij\forall i\neq j}^{n} c_{ij}x_{ij} \tag{1}$$

Subject to:

$$\sum_{j\forall j\neq i} x_{ij} \qquad \forall i \in N \tag{2}$$

$$\sum_{i\forall i\neq j} x_{ij} \qquad \forall j \in N \tag{3}$$

$$\sum_{ij\in M\forall i\neq j} x_{ij} \leq |M| - 1 \qquad \forall M \subset N \mid \{1\} \notin M \quad |M| \geq 2 \tag{4}$$

This formulation has $2^n + 2n - 2$ constraints and requires $n(n-1)$ binary variables. The exponential number of constraints make the formulation impractical for most applications. To overcome this, (4) is appended as a subtour elimination constraint only when it is violated by an otherwise optimal solution when only considering (2) and (3). Subtours occur when a path is produced that does not originate at the first city and cycles between a subset of cities in N. Constraint (4) prevents this by prohibiting the total number of binary assignments to exceed the total number of variables that can be reached from the first city. An illustrative example of subtours and how they can influence the behavior of TSP solutions is presented in Figure 2. The subtour paths are indicated by a red color to delineate them from the primary path in black. The illustrated solution would satisfy both 2 and 3 because each city is entered and exited exactly once. However, 4 would be violated as the subtour cities would not exist in M. Numerical methods for implementing the subtour elimination and further problem relaxations that are based on this formulation can be found in [53].

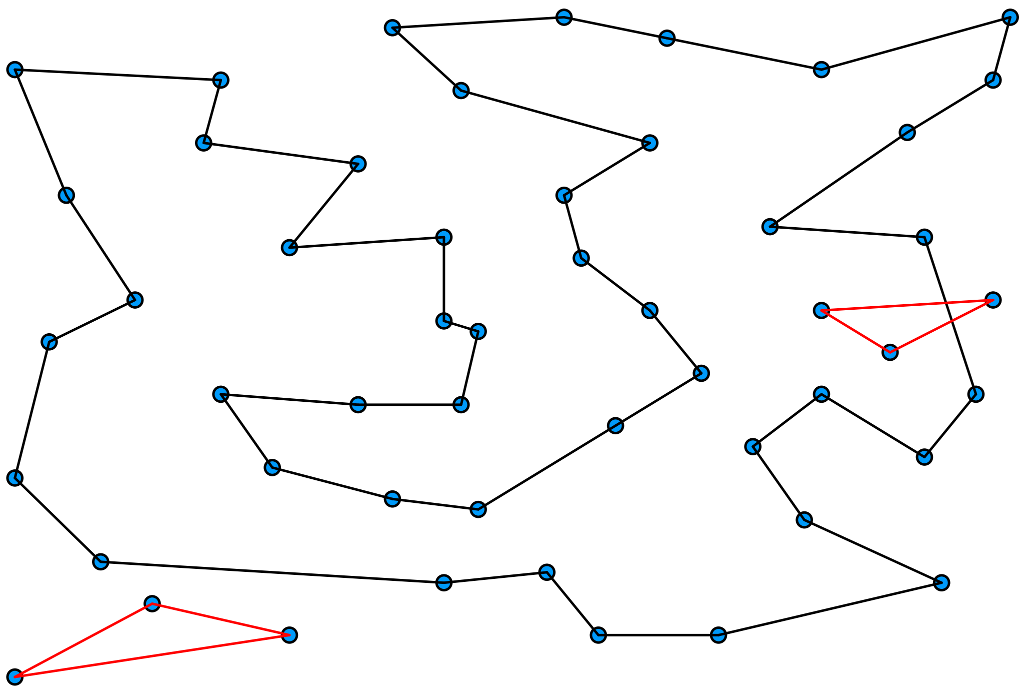


Figure 2: Presence of subtours in a TSP solution

The basis for the conventional formulation and for every other mathematical programming formulation for the TSP are 1,2, and 3. Together they construct a classic problem known as the Assignment Problem [68]. The alternative mathematical formulations presented here differ from the Conventional Formulation only by the way in which they seek to constrain the assignment of binary variables to produce full tours and to prohibit subtours or incomplete tours from being valid.

2.2.3 Sequential Formulation [67]

The sequential formulation of the TSP includes continuous variables to measure the position at which a city is visited within a tour, defined as follows.

$$u_i = \text{sequence in which city } i \text{ is visited}, i \neq 1$$

$$\min \sum_{ij \forall i \neq j}^{n} c_{ij} x_{ij} \tag{5}$$

Subject to:

$$\sum_{j \forall j \neq i} x_{ij} \qquad \forall i \in N \tag{6}$$

$$\sum_{i \forall i \neq j} x_{ij} \qquad \forall j \in N \tag{7}$$

$$u_i - u_j + n x_{ij} \leq n - 1 \qquad \forall i, j \in N - \{1\}, i \neq j \tag{8}$$

The sequential formulation contains $n^2 - n + 2$ constraints and requires $n(n-1)$ binary variables along with $(n-1)$ continuous variables to represent the sequence. The objective function and the first two constraints are the same as in the conventional formulation. Constraints (8) ensure that a sequence must contain every city and that for any two cities they exist in the same sequence.

2.2.4 Commodity Flow Formulation [38]

The commodity flow formulation again introduces a continuous variable to constrain the problem and eliminate the possibility of subtours. The new continuous variable is defined by the following:

$$y_{ij} = \text{Flow in arc}(i, j) \ , i \neq j$$

$$\min \sum_{ij\forall i \neq j}^{n} c_{ij}x_{ij} \tag{9}$$

Subject to:

$$\sum_{j\forall j\neq i} x_{ij} \qquad \forall i \in N \tag{10}$$

$$\sum_{i\forall i\neq j} x_{ij} \qquad \forall j \in N \tag{11}$$

$$y_{ij} \leq (n-1)x_{ij} \qquad \forall i,j \in N, i \neq j \tag{12}$$

$$\sum_{j} y_{1j} = n-1 \tag{13}$$

$$\sum_{j\forall i\neq j} y_{ij} - \sum_{j\forall i\neq k} y_{jk} = 1 \qquad \forall j \in N - \{1\} \tag{14}$$

Equations 13 and equations 14 serve to restrict $n-1$ units of a single commodity to flow into the first city and only one unit to flow out of each other city. The flow of a commodity is restricted by equation 12 as a unit is only available to flow through an arc if it exists at the starting node. The commodity flow formulation requires $n(n+2)$ constraints, $n(n-1)$ binary variables, and $n(n-1)$ continuous variables.

2.3 Exact Algorithms

As an $\mathcal{NP}$-hard problem, the TSP is difficult to solve to optimality for non-trivial problem sizes. As is the case with many other difficult optimization problems, numerous strategies and algorithms for finding quality solutions have emerged. Generally, these approaches can be classified as either exact approaches or heuristic approaches.

Exact algorithms are those that guarantee that the optimization problem will be solved to optimality. In regard to the TSP specifically, exact algorithms are primarily derived from the Integer Linear Programming formulations presented in Section 2.2.1. The primary issue with exact algorithms is the computational time that is required to reach a solution. The largest verified solved instance of a TSP to date contains 85,900 cities, which required a total 136 CPU years for solving [6, 5]. In practice, for a problem that may have dynamic distances over a daily, weekly, or monthly time horizon, this length of time nullifies the usefulness of this approach.

When Dantzig et al. [20] first introduced their formulation, there were no algorithms that could solve integer linear formulations. Instead of an algorithmic approach, the relaxation solutions were examined for subtours and then constraints were added until a valid solution was presented. Since that time the preferred exact method for solving the TSP has remained that of branch and bound algorithms. A diverse and thorough survey of these approaches and lower bound calculations for linear programing methods are detailed in [59, 6, 81].

2.4 Heuristic Methods

Heuristic approaches are concerned with generating reasonable solutions that are not necessarily optimal, however that can be attained in reasonable computing time. Heuristics to the TSP have garnered significant attention by researchers because of the time complexity of the problem and the lack of a polynomial bound method for providing optimal solutions. Many heuristic approaches seek to ensure a guaranteed worst case performance while others are more concerned with empirical performance [59]. Heuristic approaches are typically evaluated by two fundamental elements. The first is the computational complexity of the approach and the computational resources required to implement the algorithm. The second is the quality of the solution that is ultimately provided. Quite obviously, the most popular heuristic methods often excel in one area and suffer in the other following the *No Free Lunch Theorems for Optimization* developed by Wolpert and Macready [91]. An extreme example of this is comparing a random permutation to an exact approach. The random permutation generates a solution as fast as possible, however, the probability of optimality is near zero and quality cannot be guaranteed. Alternatively, the exact approach will guarantee optimality, however, could take years to find a solution.

TSP heuristics are generally classified in two distinct categories, tour construction methods and tour improvement methods. Tour construction algorithms are described as methods that find a solution by immediately terminating upon generating a feasible tour. Tour improvement algorithms on the other hand implement mechanisms that often seek to explore more of the solution space, however do not perform robust searches. In practice, tour construction algorithms are often predecessors to tour improvement methods that use the tour constructor as a means of initializing a solution. Extensive reviews of tour construction and tour improvement heuristics are detailed by Kurz [58], Laporte [59], and Reinelt [81] among many others.

2.5 Tour Construction Methods

As with many heuristic methods for optimization, numerous variants exist for any given algorithm. Efforts to improve empirical and computational performance may exist only for a specific class of problem. Presented here are fundamental descriptions of core algorithmic techniques for constructing TSP tours.

2.5.1 Nearest Neighbors

The nearest neighbor algorithm is a simple and straightforward approach to the TSP originally presented by Bellmore and Nemhauser in 1968 [10]. The algorithm has also been referred to as the “next-best method” by Gavett in 1965 where the algorithm was used for production scheduling [37]. The fundamental rule that defines the nearest neighbor heuristic is to always move to the nearest city from the current city. Moreover, the algorithm initializes with a random city and indexes that city to 1, $i = 1$. The next city visited is the minimum of c_{ij} for all j not in the current tour. This process repeats until all cities are visited, at which time the edge from the last city to the first city is selected. The nearest neighbor approach has been empirically shown to typically generate solutions within 25% of optimal and has a computational complexity of $\mathcal{O}(n^2)$ [50, 70].

If the tour is viewed based on the iteration of the algorithm it can be quickly realized that the algorithm initially performs well by adding the shortest edge available from the current point. However, as the tour grows in length and the number of cities in the set of unvisited cities approaches zero, large edges are forced into the tour and the length of the tour begins to suffer tremendously. This is due to the concept of forgotten cities; a problem illustrated in Figure 3, where the final edge additions are highlighted in red. The forgotten cities are bypassed because a path exists around or near them in which they are not the nearest neighbor to any of the points selected. The nearest neighbor algorithm and supporting proofs are further detailed in [82] by Rosenkrantz, Stearns, and Lewis.

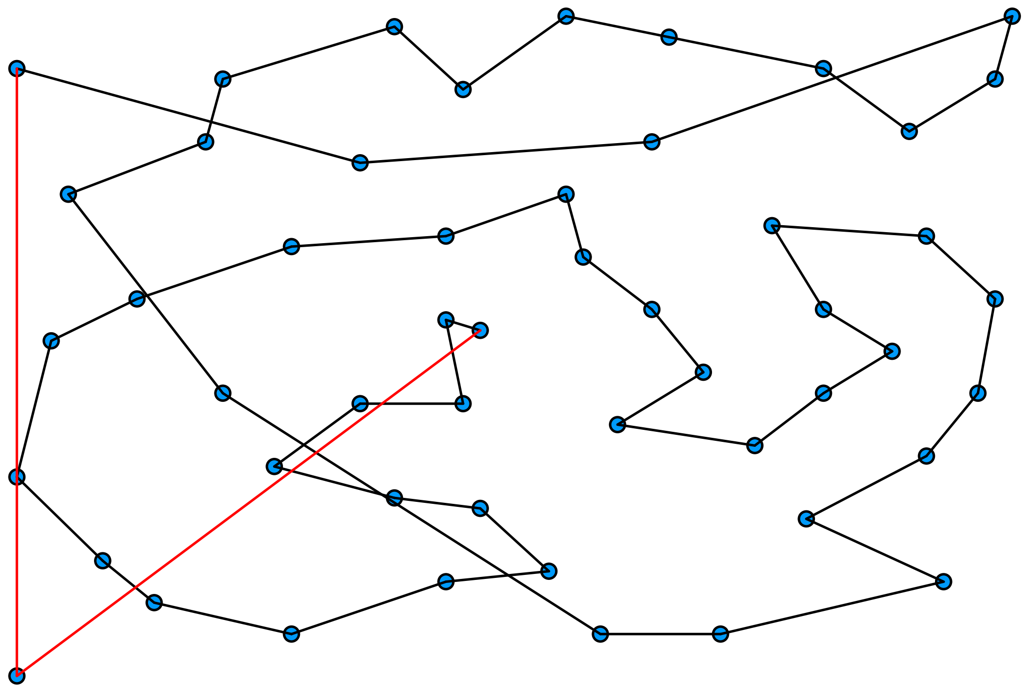


Figure 3: Final two edge additions utilizing the nearest neighbor algorithm

2.5.2 Greedy Edge Addition

The greedy edge addition or multi-fragment heuristic is more commonly referred to simply as the greedy heuristic. Since the nearest neighbor algorithm is a greedy technique, it is often mistakingly referred to as the greedy heuristic. Nonetheless, the true greedy heuristic is based on seriation methods initially presented by Gelfand in 1971 with applications in archaeological chronology [39]. The ordering of a distance matrix for clustering was later improved upon by Gruvaeus and Wainer by cascading search trees such that the smallest distance of one tree was immediately followed by a subtree beginning with the remaining smallest distance [44]. The adaption and analysis of the greedy heuristic specific to the TSP was presented by Bentley in 1992 [11].

The greedy heuristic starts with a set of all edges defined by their distance, c_{ij}. A tour is constructed by selecting the shortest available edge that does not violate either the cycle constraint or the degree constraint. The cycle constraint ensures that subtours are not constructed and restricts any cycle from having less than N edges. The degree constraint ensures that each node is restricted to two, such that each city is arrived at and departed from only once. The algorithm terminates when a cycle is produced with N edges.

The greedy heuristic typically provides a tour within 15-20% of the Held-Karp lower bound which makes it a higher performing constructor than the nearest neighbor approach, however, it is subject to providing far worse solutions in particular instances [50]. This does, however, come at a computational cost greater than nearest neighbors, with a complexity of $\mathcal{O}(n^2 \log_2 n)$ [50, 70]. While not subject to forgotten nodes in the same way that nearest neighbors is, the greedy heuristic does incur an increasing cost per edge as it iterates. Finding edges that satisfy the cycle and degree constraint becomes increasingly difficult as the tour grows. This is illustrated in Figure 4, where the final edge additions are highlighted in red.

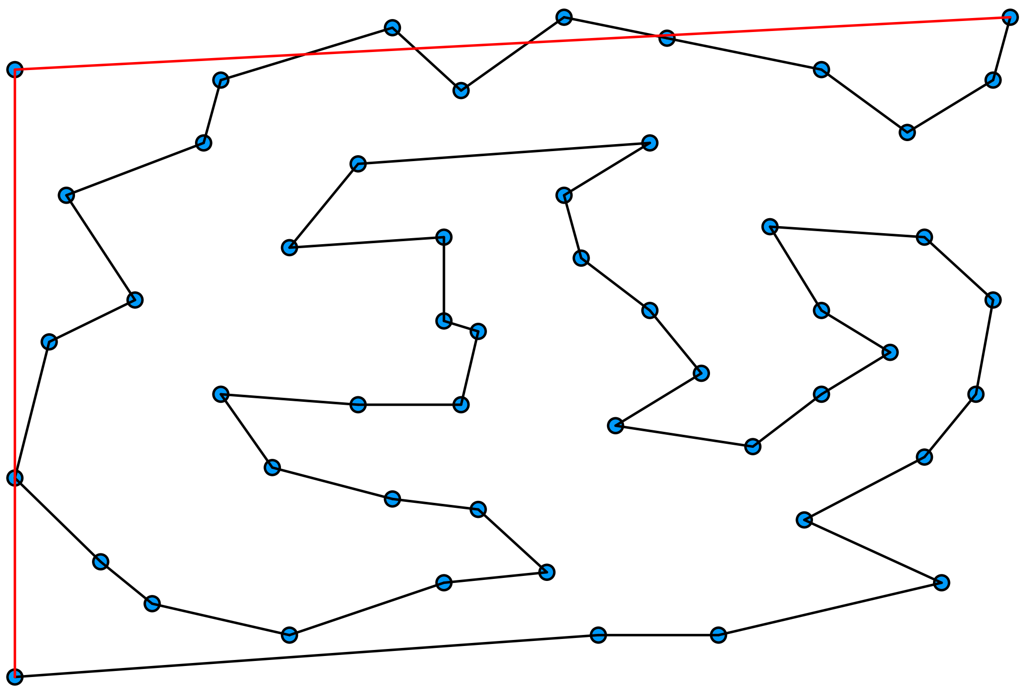


Figure 4: Final two edge additions utilizing the greedy heuristic

2.5.3 Insertion

Insertion heuristics have been used as an intuitive approach to build a tour by starting with a subtour and then extending the tour by inserting one new city k at a time. The objective is to minimize the insertion cost, namely the tour's length, by inserting the city in between two consecutive cities that are already on the tour. Varying schemes for insertion can be used to alter the order in which nodes are inserted into the tour. The most common variants include the following:

Nearest insertion: The nearest city k is added to the tour.

Farthest insertion: The farthest city k is added to the tour.

Cheapest insertion: The city k with the minimal cost of insertion is added to the tour.

Arbitrary insertion: The city k is chosen randomly from the set of cities not yet added to the tour.

The cheapest insertion technique, which was first proposed by Karg and Thompson in 1964, allows for the node that will result in the smallest increase in the tour length to be added successively [54]. Both the cheapest and nearest insertion techniques are rooted to the minimum spanning algorithm first proposed by Prim in 1957 [78]. The spanning algorithm allows for all vertices to be connected together with the minimum possible total edge weight. Insertion heuristics correspond with adding an edge to a partial spanning tree, whereupon the insertion of each city is equivalent to a new edge.

Perhaps nonintuitively, the arbitrary insertion technique is also efficient in deriving tours mainly due to the fact that choosing arbitrary or random points requires little to no computation time for selection [82]. Farthest insertion heuristics begin by selecting each successive point farther from the preceding, so that an outline of the entire city is established. The construction of a general outline can often facilitate improved performance when there are a limited number of nodes, with solutions that are within 2/3 of the optimal length [52]. Interestingly, the farthest insertion heuristic has been shown to produce improved tour solutions when compared with nearest and cheapest insertion techniques, as well as with nearest neighbor heuristics [82].

An alternative approach that builds upon the cheapest insertion heuristic was proposed by Bass and Schubert in 1967. The approach is referred to as the Convex Hull heuristic, as it finds the smallest convex set containing all points, namely the Convex Hull, and correspondingly finds its cheapest insertion [9]. The Christofides algorithm, which has a guaranteed worst case ratio of 3/2, is an alternative technique that was first proposed in 1976 [17]. The algorithm allows spanning trees to become Eulerian, by first computing a minimum spanning tree and then adding a minimum weight for odd degree nodes so that a Eularian graph can be obtained. To this day the Christofides heuristic provides the best "worst-case" performance guarantee for the TSP [4].

2.6 Tour Improvement Methods

Tour improvement methods generally include any algorithmic methodology that takes a tour and generates an improved tour. However, tour improvement heuristics typically are limited to simple and local tour modifications and exclude more complicated approaches. More complex methods that employ multiple layers of logic and explore globally are commonly referred to as metaheuristics and are detailed in Section 2.7. Here, tour improvements are restricted to local search processes where each improved tour is adjacent to the previous tour, which requires that an improved tour be reachable from the previous tour by a single move.

2.6.1 2-opt

The k-opt algorithm and all subsequent variants are based on the 2-opt algorithm originally formulated by Croes in 1958 [19]. Generally, a 2-opt move consists of removing two edges from an already constructed TSP tour and reconnecting the nodes for an improved result. Upon the two edge removal, four nodes remain in the tour that have a degree of 1 and thus two separate paths exist that must be connected. It can be seen that based on the disconnected nodes, there exist only two ways of connecting the nodes that would result in a valid tour, one of which was just disconnected. Thus, the 2-opt has only one alternative option to connect the nodes. An illustration of a 2-opt operation is presented in Figure 5, wherein the edges chosen to be removed and the new edges after the 2-opt move has been made are highlighted. If the tour is viewed as a permutation vector, the 2-opt operation is essentially the reversal of a segment in the permutation between two nodes. If $\{1, 2, 6, 5, 4, 3, 7, 8\}$ were the path in Figure 5 before 2-opt, the reversal would occur such that the corresponding path segments would be $\{1, 2\} \ldots \{6, 5, 4, 3\} \ldots \{7, 8\}$ resulting in $\{1, 2, 3, 4, 5, 6, 7, 8\}$.

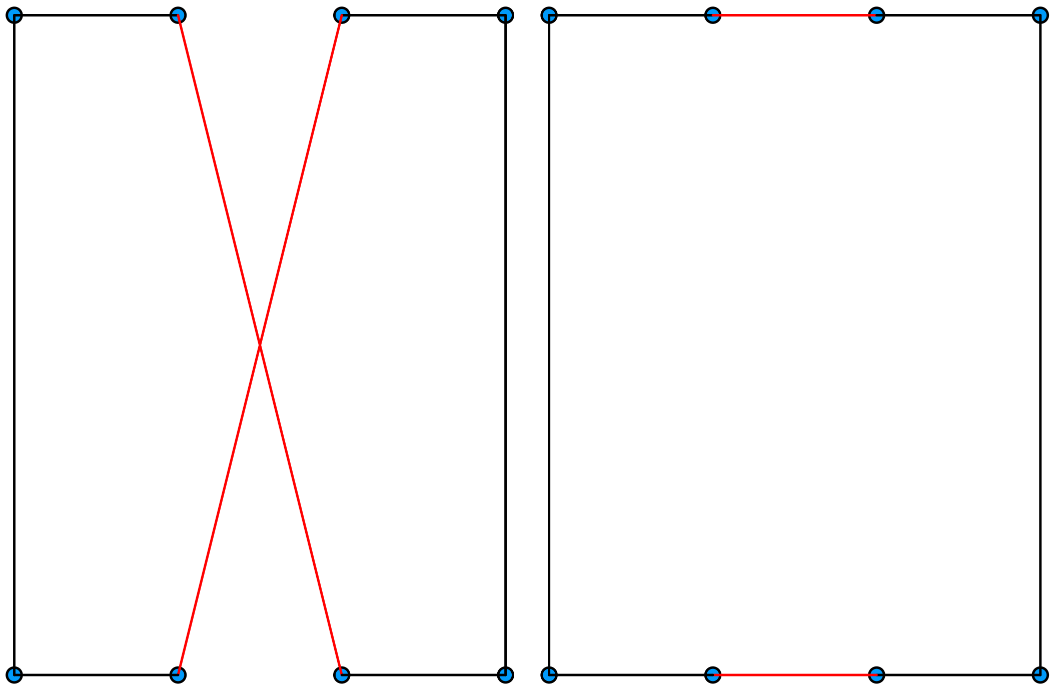


Figure 5: 2-opt edge removal (left) and the subsequent edge additions (right) highlighted

Evidently the edge removal and subsequent edge addition is not always an improvement. Qualifying the move to be made is the basis for the 2-opt improvement algorithm. The algorithm iterates through all pairs of edges and executes 2-opt if the operation is profitable. If the nodes indexed by $\{i, j, k, m\}$ and the cost matrix C are considered, then the search is based on selecting C_{ij} and searching all C_{km}. A move is only executed if the following relationship is true: $C_{ij} + C_{km} > C_{jk} + C_{im}$ This continues until no 2-opt move exists that satisfies the relationship at which time the tour is considered to be 2-opt optimal.

The performance of 2-opt is very efficient and often results in tours within 5% of the Held-Karp lower bound [50]. However, one potential drawback to 2-opt is the fact that a single move can take up to $\mathcal{O}(n)$ to complete in a worst case scenario [82]. Thus, considerable work has been done to ensure the best data structures are employed and search reduction techniques are used. One of the most meaningful is the observation by Steiglitz and Weiner that $C_{ij} > C_{jk}$ must hold for a 2-opt to be profitable [88]. This can trim the search space significantly but does come at a cost in terms of memory. The search space can be even further constrained by choosing only the m nearest neighbors for 2-opt consideration but reduces the quality of the solution [50].

2.6.2 k-opt and Lin-Kernighan

The success of 2-opt naturally inspired similar mechanisms for higher dimensions. The complete generalization of this technique, known as k-opt or r-opt, was forumalated by Lin in 1965 [61]. Similarly to 2-opt, a 3-opt procedure would remove 3 edges from the tour and then reconnect the degree 1 nodes to complete the tour. This procedure continues until no 3-opt operation is available that improves the tour. Any k-opt scales the same way. Naturally a higher dimension k-opt routine would seem to provide a better tour but the computational penalty can be extremely high and the relative improvement to 2-opt and 3-opt is small. An analysis of k-opt routines by Christofides in 1972 and experimental results from Johnson in 1997 indicate that any check of optimality requires $\mathcal{O}(n^k)$ operations [62, 50]. Interestingly, the anlysis by Christofides found a measurable improvement exists between 2-opt and 3-opt, fails to exist between 3-opt and 4-opt, and exists again between 4-opt and 3-opt [62]. These results prompt the question of how to determine what is a worthwhile dimension for k-opt. If speed is the primary concern then 2-opt will easily outperform other k-opt routines. However, if solution quality supersedes the speed of a solution then higher dimension k-opt routines would be justified. As with 2-opt procedures, considerable research has focused on speeding up k-opt procedures and making them as efficient as possible [51].

The challenge of choosing which k-opt routine to perform was considered by Lin in 1973, whereupon the Lin-Kernighan heuristic also known as the LK heuristic was first introduced [62]. The Lin-Kernighan heuristic addresses the exact problem of determining which k-opt routine will provide the greatest improvement and dynamically determines which dimension to proceed with. In practice k is restricted to 2, 3, and 4. At each iteration the algorithm makes the best dimension determination, completes the improvement at that dimension, and then repeats until no improvement is available. Lin-Kernighan has a computational complexity of $\mathcal{O}(n^{2.2})$ and is able to achieve this by determining that any k-opt move can be completed as a series of 2-opt moves [62]. The Lin-Kernighan heuristic has proven to be one of the most effective heuristics for the TSP and has spawned a tremendous number of variants seeking to improve its performance [47]. A thorough review of current best practices and Lin-Kernighan variants can be found in [79].

2.7 Metaheuristics

The final category of approaches to solving the TSP are categorized as metaheuristics. Unlike tour construction heuristics and tour improvement heuristics, metaheuristics employ a deeper search of the solution space [73]. Fred Glover introduced the term "metaheuristic" in 1986 to characterize a heuristic approach that can be described generally but employs heuristic methods specific to the problem being solved [40]. Glover predicted that metaheuristics would become very effective for solving combinatorial optimization problems. Since that time a multitude of metaheuristics have been developed for several variations of combinatorial optimization problems.

Most metaheuristics are defined by a strategy that controls a search procedure. Underlying the strategy are typically the same operations that define heuristic methods for the given problem. The power of metaheuristics is that they can move out of local optimum and explore new solution spaces, make the best use of heuristic procedures, and benefit from the cooperative use of multiple heuristics methods [64]. Numerous metaheuristics have been shown to find near optimal solutions in multiple combinatorial optimization problems [14].

Within metaheuristics for the TSP there exist three primary categories of approaches: local search, population search, and learning mechanisms. For the sake of brevity, only the most successful and widely used approaches will be briefly described with some indication of their strengths and weaknesses.

2.7.1 Simulated Annealing

Simulated annealing was first introduced by Kirpkpatrick in 1983 as an optimization technique based on annealing in metallurgy [56]. The TSP was used as the first problem to test the approach for combinatorial optimization by Kirkpatrick. The general structure for simulated annealing as proposed by Kirkpatrick found in [55] is as follows. At every iteration of the algorithm, a solution x is randomly generated from the neighborhood of the current solution x_t. If the cost of x is less than the current solution x_t, $f(x) \leq f(x_t)$, then the current solution is updated: $x_{t+1} = x$. When the solution is not an improvement, the current solution is updated with a probability p_t. The probability of updating with a worse solution is typically a decreasing function of both time (iteration) and relative fitness, t and $f(x) - f(x_t)$. Normally the probability function takes the following form [1]:

$$p_t = \exp \frac{-[f(x) - f(x_t)]}{\theta_t} \tag{15}$$

In Equation 15, θ_t is the temperature at iteration t and is updated by the cooling schedule or cooling function. The cooling schedule normally decreases as the iterations increase. The result is that the probability of moving to a worse solution decreases over time. The stopping criteria typically depends on a fixed number of iterations or the number of iterations without a certain amount of improvement.

The cooling schedule and probability function used in simulated annealing guide the strategy of the algorithm and have a significant impact on its performance. However, the mechanisms that perform the neighborhood search can be just as critical. In most cases for the TSP, a swap of two permutation members or a 2-opt move are used to advance from the current solution to a neighbor [81]. The starting solution is typically generated by a construction heuristic. Simulated annealing performs similarly to 2-opt in terms of solution quality and is capable of generating solutions comparable to Lin-Kernighan if the runtime is extended. However, simulated annealing on large instances requires speedups when 2-opt is used.

2.7.2 Tabu search

Another local search metaheuristic for the TSP is tabu search. Tabu search was originally introduced by Glover in 1986 [40]. Tabu search works in much the same way as simulated annealing does as it seeks to move from one solution to a neighborhood solution in each iteration. The major difference is that tabu search forces a move at each iteration even if the neighbooring solution is worse. Intuitively, constantly moving in a neighborhood would eventually create a cycle between solutions. To avoid this, recently visited solutions are excluded from the search and become “tabu”. What makes specific implementations differ is the way in which tabu solutions are stored in memory, how long they remain tabu, and the mechanisms that explore the neighborhood [42].

In most tabu implementations for the TSP a 2-opt move is the preferred method for exploring a neighborhood. The use of the restricted solutions increases the efficiency of 2-opt because many of the moves would be restricted before all 2-opt moves could be explored. As such, some of the most important aspects of tabu search are the efficiency of the tabu constraint and the parameters controlling the tabu solutions. Parallel implementations and the use of more robust search functions have proven tabu search to be very effective for generating near optimal tours for the TSP [41].

2.7.3 Ant Colony Optimization

Ant Colony Optimization is a learning metaheuristic for optimization that is based on the behavior of ants in colonies searching for food. The introduction of the concept came in 1991 by Colorni, Dorigo, and Maniezo in 1991 [66]. Ant colonies find food by having ants initially wander away from the colony. As the ants move away from the colony, they release a pheromone trail that marks the path in which they have traveled. The pheromone acts as a communication agent that signals to other ants how long the path is and the quality of food that can be found along that path. However, in time the pheromone dissipates and only the most frequently traveled paths are able to maintain a high level of pheromone. This process results in a greater number of ants releasing a greater amount of pheromone on the most rewarding paths.

The original formulations of ant colony algorithms were focused on application to the TSP. Subsequent improvements and benchmarks additionally focused on the TSP and the general form can be taken as follows as found in [66, 23, 24]. Each edge has two associative values, where the edge is defined by $(v_i, v_j$. The first value is the visibility of the edge defined as $v_{ij} = \frac{1}{d_{ij}}$ where d_{ij} is the distance between the vertices. The second value is the pheromone level of the edge, Γ_{ij}, which is updated after each ant iteration. At each iteration, n artificial ants are generated and set out on a tour based on nearest neighbors. The distance however is based both on v and the strength of the pheromone, Γ. At the end of an iteration, the pheromone level of each edge decays by 1-p and the edges traversed in the resulting tours are increased. The pheromone between two vertices is updated by the following function:

$$\Gamma_{ij} = p\Gamma_{ij} + \sum_{k=1}^{n} \delta_{ij}^{k} \tag{16}$$

In Equation 16, δ is the inverse of the length of the total tour by the k ant traversing that path. The algorithm typically iterates for a fixed number of iterations or until a path becomes dominant based on the pheromone level. Ant colony optimization has proven to perform well on the TSP but is often difficult to scale to large problems because of its high usage of memory. The approach has inspired other swarm and colony based variants for a larger set of problems. The most recent advances and techniques are reviewed in [25].

2.8 Benchmark Problem Sets

A significant number of problem sets and benchmark problems have been used to analyze the performance of TSP algorithms. The most well known and comprehensive collection of benchmark problems is referred to as TSPLIB [80]. The collection includes symmetric and asymmetric TSP instances as well as capacitated vehicle routing problems, sequential ordering problems, and Hamiltonian cycle problems. The value of such a well known repository is that it allows for an equal comparison of algorithms and provides a set of problems that are feature rich enough to represent a variety of benchmarks. The reason feature richness is important is because randomly generated TSP instances often are categorized as uniform or clustered. Some algorithms perform well for uniformly generated cases and others perform well for clustered instances, however not for both. TSPLIB features a broad class of instances, many of which are derived from real world geographic schemes. Among symmetric TSP instances, which are the focus of this thesis, problem sizes range from 14 to 85,900 cities in the repository.

Chapter 3

Applications of Clustering for the TSP

3.1 Previous Approaches

The TSP has seen a myriad of approaches taken to produce high quality solutions using heuristic methods. Many of the predominant heuristics used can be found in Section 2.4. However, the relative success of many of these heuristics has spawned more robust methods such as meta-heuristics as detailed in Section 2.7 and hybrid approaches inspired by numerical methods in alternative fields and applications. One such approach is the incorporation of clustering algorithms into previously defined methods for solving the TSP in order to improve the result obtained by the standalone TSP solver.

3.1.1 K-Means

K-means clustering is one of the most classic and widely used clustering methods in multivariate classification, data mining, and unsupervised machine learning. Originally introduced in signal processing, k-means aims to partition n observation vectors into k clusters where each observation is a member of the cluster with the nearest mean [46, 65, 34]. The problem posed in k-means clustering can be formulated by the following as found in [46]: Given a set of observations $(x_1, x_2, \dots, x_n)$ where each observation is a d-dimensional real vector, partition the n observations into k sets, S, where $k \leq n$, minimizing the following objective function:

$$\min \sum_{i=1}^{k} \sum_{x \in S_i} ||x - \mu_i||^2 \tag{17}$$

In the objective function 17, μ_i is the representative mean of the set S_i. Finding the optimal solution to this partitioning problem is proven to be $\mathcal{NP}$-hard for all d-dimension metric instances [36]. If k is fixed then the problem can be solved to optimality with a time complexity of $\mathcal{O}(n^{dk+1})$ [48].

Due to the computational complexity of the k-means clustering problem heuristic algorithms are generally used. By far the most common implementation and what has come to be known as the standard approach is Lloyd's algorithm [63]. Lloyd's algorithm initializes by placing k points in the input plane of dimension d. The algorithm then iteratively constructs a Voronoi diagram across the k points. From the resulting Voronoi cells in the Voronoi diagram, the centroid of each cell is computed and the kth site is moved to the centroid of the corresponding cell. Lloyd's algorithm terminates after convergence occurs, which is typically an approximate measure of the relative convergence due to the limitations of preciseness and increasingly slower convergence rate. Most common implementations therefore terminate after the maximum distance update for all centroids is less than a predefined lower bound, which for most applications provides a solution that is practical and as useful as optimal. A sample k-means result is illustrated in Figure 6 which includes the data points, Voronoi cells, and centroids. The generalized k-means algorithm requires only one argument to be passed for solving: a cost matrix which represents the distance between the vectors being clustered for all n. However, for the purposes of speed and to incorporate domain specific knowledge, k is also passed in as an argument to determine the number of clusters that are to be used for assignment.

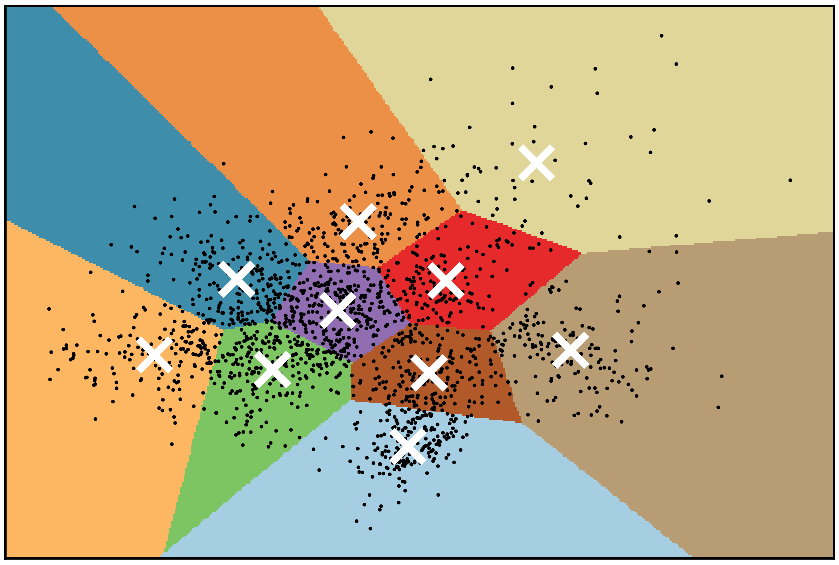


Figure 6: Sample k-means result [76]. Centroids (white), Voronoi Cells (colored)

The incorporation of clusters identified by k-means into an algorithm for solving the TSP is intuitive and the concept of marrying the two seems simple. The k-means clustering algorithm seeks to cluster the nodes that are closest to each other relative to the distance in which they lie from the center of another cluster. Abstractly, a good TSP solution would primarily contain edges between nodes that are close together and closely connected nodes could be referred to as a cluster of nodes given a good TSP solution. Therefore, generating a good solution to a TSP with k-means would contain two solution steps, generating solutions within identified clusters and then finding ways to connect the clusters to produce a full TSP tour. Given that k-means also provides an imaginary centroid node that is representative of the cluster, solutions can be abstracted from a TSP tour that passes only through the centroids and then used as a guide for incorporating the actual nodes into the TSP tour.

In current literature, direct implementation of k-means clustering as a tour construction method for the TSP was presented by Deng, Liu, and Zhou in [22] and by Phienthrakul in [77]. Both presented their k-means strategy as a method for initializing a population for an evolutionary algorithm. The results taken from their approaches are difficult to assess because the solutions are only presented after evolving the population for a large number of iterations through the genetic algorithm. However, compared to a random initialization the results were significantly improved. This lends credence to the concept that a clustering mechanism can aide in constructing quality TSP solutions. The strategy employed in [22] is illustrated in the following steps:

1 Cluster n cities into k groups using $k = \text{Int}(\sqrt{n} + 0.5)$

2 Generate locally optimal paths within each cluster using a genetic algorithm

3 Generate a global optimal path that traverses the k centroids representing each cluster

4 Moving through the global path, remove one edge from each local path and connect the local path to the global path

In the research of Deng, Liu, and Zhou's, the process is repeated until an initial population is produced. This results in differing solutions because the edge removal at each local path is done randomly and no calculations are made to remove the edge that would result in the shortest global path from the local paths.

The strategy in [22] alludes to an interesting methodology for solving the TSP. The first technique to note is the clustering of the cities in order to divide the problem into smaller sets of cities. What this allows for is locally optimal solutions to be generated that can then be used in a greater context of a potentially globally optimal ordering of the sets. While the genetic algorithm was used by Deng, Liu, and Zhou on three levels; locally, global sets, and then global paths, any method for generating a tour could be used in its place at all or some of the levels of the problem. Phienthrakul in [77] used the nearest neighbors heuristic to connect the clusters but used evolutionary computation to arrange paths within clusters and eventually the final tours.

3.1.2 Affinity Propogation

Affinity propagation is a far more recent clustering method than k-means and is based on message passing between data points and views the points as members of a network in which information is exchanged. The algorithm was originally presented by Frey and Dueck in 2007 [35]. The goal of the algorithm is to identify the "exemplar" data points that are representative of a larger number of points by finding the points in which the greatest amount of information is passed through. Affinity propogation uses a graph structure to determine clusters and allows individual points to "vote" on the data point they prefer most to be their exemplar. In effect each node has an edge between itself and its most preferred exemplar. The result of these connections are clusters in which each point is reachable from any other point in the cluster through the exemplar but is not reachable from any other cluster.

The formulation for affinity propagation as found in [35] is presented as follows: The observations $(x_1, x_2, \dots, x_n)$ define the data points representing the nodes of the graph. A similarity function, s, measures the similarity between any two data points such that $s(x_i, x_j) > s(x_i, x_k) \iff x_i$ is more similar to x_j than it is to x_k. A responsibility matrix, R, is initialized with zeros and will store values $r(i, k)$ which quantifies how acceptable x_k is in serving as the exemplar for x_i in relation to all other candidate exemplars for x_i. A similar availability matrix, A, is also initialized with all zeros and stores values for the appropriateness for x_i to choose x_k as its exemplar while considering the preference for other points to also choose x_k as an exemplar. The algorithm then iterates over two main update procedures until a termination criteria is reached. The first step is the responsibility update to R:

$$r(i,k) = s(i,k) - \max_{k \neq k'}[a(i,k') + s(i,k')] \tag{18}$$

The second update step is to update the availability. Of key importance is the two calculations that occur in this step, the second of which updates the diagonal of A. This value represents the total availability of that node as an exemplar to all other nodes.

$$a(i,k) = s(i,k) - \min_{i \neq k}[0, \sum_{i' \notin (i,k)} \max(0, r(i',k))] \tag{19}$$

$$a(k,k) = \sum_{i' \neq k} \max(0, r(i',k)) \tag{20}$$

The algorithm iterates through Equations 18, 19, 20 until the cluster members go unchanged for a predetermined number of iterations. The exemplars after termination are determined by the sum of the responsibility and availability matrices for that node to itself:

$$\textit{exemplar} \iff r(i,i) + a(i,i) > 0 \tag{21}$$

Affinity propagation results in clusters in which each member of a cluster can be directly measured against its similarity to the exemplar of that cluster. From a network perspective in which the algorithm is frequently used, the most important nodes for communication throughout the network are identified as exemplars. An illustrative example is presented in Figure 7. In comparison to k-means clustering, affinity propagation presents a number of advantages that were motivating to the original formulation of the algorithm. The biggest advantage is that a predetermination of the number of desired clusters does not need to be made a priori [26]. Instead, the best number of representative clusters are used to describe the data. However, in many cases a more malleable result might be desirable. In this case certain points may be suggested as exemplars or a limit on the maximum number of cluster members that can be introduced into the algorithm [27].

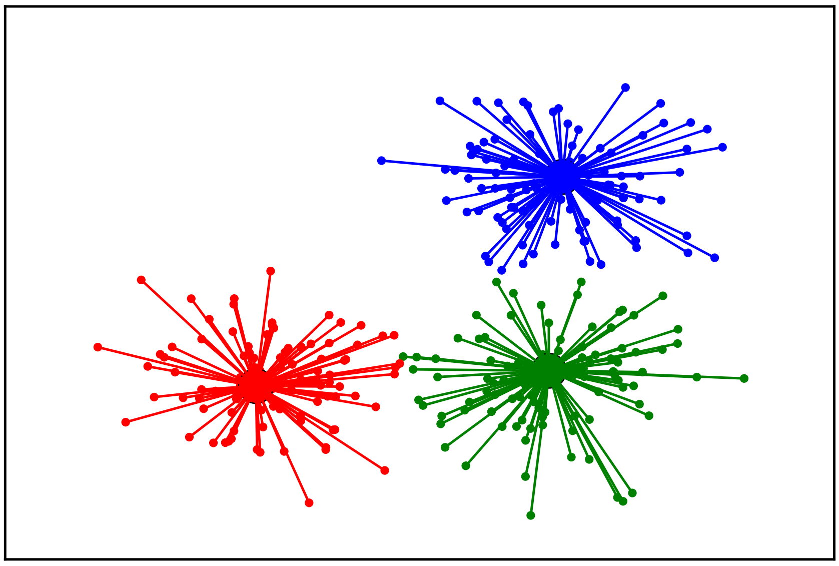


Figure 7: Sample Affinity Propagation result [76]

As a clustering method affinity propagation has proven to be useful in computer vision, computational biology, text mining and networking [45]. With respect to the TSP, [28] proposed the use of affinity propagation as a means of constructing solutions. El-Samak and Ashour detail a two level genetic algorithm similar to alternative approaches that use k-means as the clustering mechanism but instead utilize affinity propagation as the initial clustering mechanism [28]. The methods employed in [28] are detailed as follows:

1 Cluster n cities into k groups using affinity propagation

2 Generate locally optimal paths within each cluster using a genetic algorithm

3 Generate a global path by moving iteratively through each k cluster until all clusters are connected

The steps outlined by El-Samak and Ashour are not completely clear as to how exactly clusters connect into a global path. According to their description, they produce the same path over and over until a population is produced. However, for a genetic algorithm this does not make sense as every chromosome in the population would be identical. It can be inferred that low level cluster paths are connected randomly to initialize a population. Similarly to previous clustering approaches with the TSP, the results are promising compared to a standard genetic algorithm but the quality of initial solutions from the clustering are difficult to deduce.

3.2 DBSCAN

Current research has shown that the incorporation of clustering into algorithms for solving the TSP is feasible and potentially promising. However, a limited number of clustering methods have been employed and the incorporation of clustering has not influenced a large number of heuristic or meta-heuristic applications. As one of the most frequently used clustering algorithms, it is no surprise that k-means clustering has been used with the TSP. Applications of k-means in [77] and [22] both use it as an initialization strategy for evolutionary algorithms. Affinity propagation has been used similarly as an initialization strategy in [35]. However, other clustering algorithms have been used in different capacities related to the TSP. Most notably perhaps is analyzing the performance of particular algorithms on the TSP and using clustering as a method of feature recognition for a TSP instance. In [87] the authors use GDBSCAN, an extension of DBSCAN, to extract features from TSP instances in an effort to produce clustering related meta-data that is then learned from to choose the most appropriate TSP algorithm. The authors of [90] use GDBSCAN to measure the structural properties of TSP instances. The use of GDBSCAN in this capacity is partially responsible for the exploration of DBSCAN as an aide to heuristic algorithms in this research. While GDBSCAN is a useful extension of DBSCAN, Euclidian metrics are used for the TSP and thus an extension beyond DBSCAN is not required.

DBSCAN is an acronym for density-based spatial clustering of applications with noise and was originally formulated in 1996 [30]. As the name alludes, DBSCAN was originally formulated for use in spatial databases where objects are often irregularly clustered, arbitrary shapes exist, and large numbers of objects are present. The authors make note of two key differences between DBSCAN and more popular clustering algorithms. The first is the reliance on users to have domain specific knowledge enough that they can appropriately conclude the number of clusters a priori. As mentioned in Section 3.1.1 with k-means, choosing the appropriate number of clusters can be challenging and unintuitive. Secondly, partition and hierarchal clustering assumes all objects in a database to be part of some cluster and thus the entire space must be considered as is the case with affinity propagation. Therefore, partitioning of outliers and low density areas in the space can significantly affect the final assignments of high density regions resulting in poor overall classification. In effort to overcome the shortcomings of other approaches, DBSCAN was formulated with three requirements in mind: Minimal requirement of domain knowledge to derive input parameters, ability to discover arbitrarily shaped clusters, and good runtime efficiency for large databases.

3.2.1 Density-Based Clustering

DBSCAN's core functionality is to detect clusters based on the density of points in a region relative to the overall density of the database. The authors of DBSCAN generalize their approach by describing the algorithm with concepts not typically found in other clustering methods. For clarity and rigor the core definitions that embody DBSCAN are detailed along with the parameters and methods in which they are applied and outlined in [30].

Of utmost importance to DBSCAN are the input parameters that drive the behavior of the algorithm. The first parameter is *Eps*. *Eps* is the maximum allowable distance between two points for them to be considered part of the same neighborhood. More formally, the eps-neighborhood of a point p, can be defined by $\{q \in D | \text{dist}(p,q) \leq Eps\}$. The second parameter of DBSCAN is *MinPts*. *MinPts* is the minimum number of points that are required to constitute a cluster. Incorporating *MinPts* with the definition of an eps-neighborhood establishes the concept of density reachability. The authors of DBSCAN make a distinction between direct density reachability and density reachability. A point p is direct density reachable from a point q if p is in the Eps-neighborhood of q and the Eps-neighborhood contains at least *MinPts*. For core points, direct density reachability is symmetric, but this does not extend to the relationship between a core point and a border point. Core points are points that exist inside a cluster and border points exist on the edge of a cluster. The eps-neighborhood of a core point contains at least *MinPts* and is therefore in what would be considered a higher density region. However, a border point exists in the Eps-neighborhood of a core point but does not have *MinPts* in its own neighborhood and cannot be considered core to the cluster. A point p is considered density reachable from a point q if there is a chain of points that are directly density reachable that connect p and q. A more stringent density reachability is density connectedness in which p and q are density reachable from the same point. These definitions and relationships between points are what determine the identification of clusters in DBSCAN and are illustrated in Figure 8.

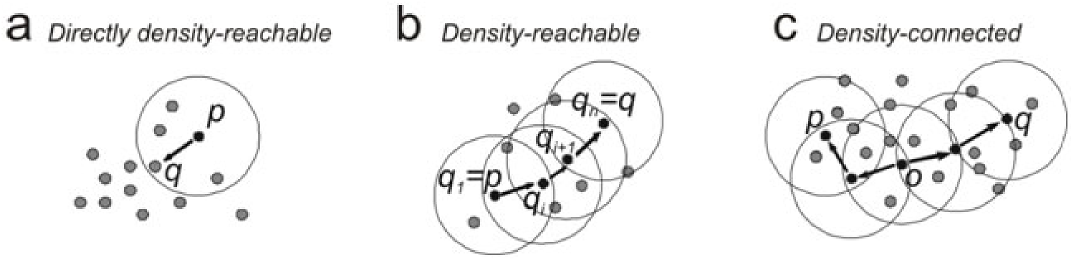


Figure 8: DBSCAN connectedness and reachability [3]

The DBSCAN algorithm as formulated in [30] is presented in Figure 9. It is important to note that the iteration order does not affect the final assignments or the discovery of clusters. This is because once a point is determined to be part of a dense enough region to constitute a cluster, the *explandClust* function finds all subsequent eps-neighborhoods of those points and the process continues until all density reachable points are included in the cluster. Each point does undergo the *regionQuery*, which can run in $\mathcal{O}(log(n))$ given an indexing structure and thus the average runtime complexity is $\mathcal{O}(n\, log(n))$ [30].

Figure 9 Algorithm: DBSCAN

```
procedure DBSCAN(D, eps, MinPts)
    C = 0
    for p in D do
        if p is unvisited then
            p.visited = true
            Neighborhood = regionQuery(D, p, eps)
            if size(Neighborhood) < MinPts then
                p.id = noise
            else
                C += 1
                expandClust(p, Neighborhood, C, eps, MinPts)

procedure regionQuery(D, p, eps)
    return points in D within eps of p

procedure expandClust(p, Neighborhood, C, eps, MinPts)
    assign p to C
    for q in Neighborhood do
        if q is unvisited then
            q.visited = true
            Neighborhood' = regionQuery(D, q, eps)
            if size(Neighborhood') ≥ MinPts then
                Neighborhood = Neighborhood + Neighborhood'
        if q is unassigned then
            assign q to C
```

The connectedness of points and reachability definitions are what constitute the definitions of clusters using DBSCAN. As previosuly mentioned, DBSCAN does not require all points to be considered part of a cluster before termination.

As such, after the algorithm is complete, points are considered as one of three possible classifications: core points, border points, and noise points. Core points are directly density reachable within a cluster, border points are density reachable from the core points of a cluster, and noise points are points that do not fall within a density region as defined by *Eps* and *MinPts.* A cluster with respect to *Eps* and *MinPts* is a non-empty subset of the database, D, such that the following conditions are satisfied:

- $\forall p, q$: if $p \in C$ and q is density reachable from p given *Eps MinPts*, then $q \in C$
- $\forall p, q \in C$: if p is density connected to q given *Eps MinPts*

This establishes a cluster as any set of points that meet the reachability and connectedness criteria given *Eps* and *MinPts.* This simply establishes clusters as any set of points where there are enough points in a small enough area to constitute a dense region.

The connectedness of points in extensive dense regions is what gives DBSCAN the flexibility to identify arbitrarily shaped clusters without having to look beyond locally connected points from already dense areas. This flexibility is illustrated in a toy dataset taken from [76] that compares the results of k-means and affinity propagation with that of DBSCAN found in Figure 10. The results give a good illustration of the positive and negative aspects to using DBSCAN. The positive is that DBSCAN can identify clusters within clusters as illustrated by the two ring dataset. Both affinity propagation and k-means generate cluster assignments that exist in both rings. DBSCAN also demonstrates tremendous flexibility in identifying the two separate crescent shapes where as k-means and affinity propagation divide the crescent regions by assigning them to different clusters. For application to the TSP, DBSCAN is capable of following a "path" that naturally aligns with the objective of the problem. As illustrated, this path need not have its shape predetermined. However, as a density-based clustering algorithm, DBSCAN assigns all elements to a single cluster in the case of a uniformly distributed dataset. Whether or not this would be useful or correct would require domain specific interpretation. It should also be noted that each algorithm's performance is subject to choosing of input parameters as previously mentioned. The adjustment of the number of clusters for k-means and eps for DBSCAN could alter the results tremendously. However, for purposes of comparing the behavior of the algorithms the parameters chosen are well suited.

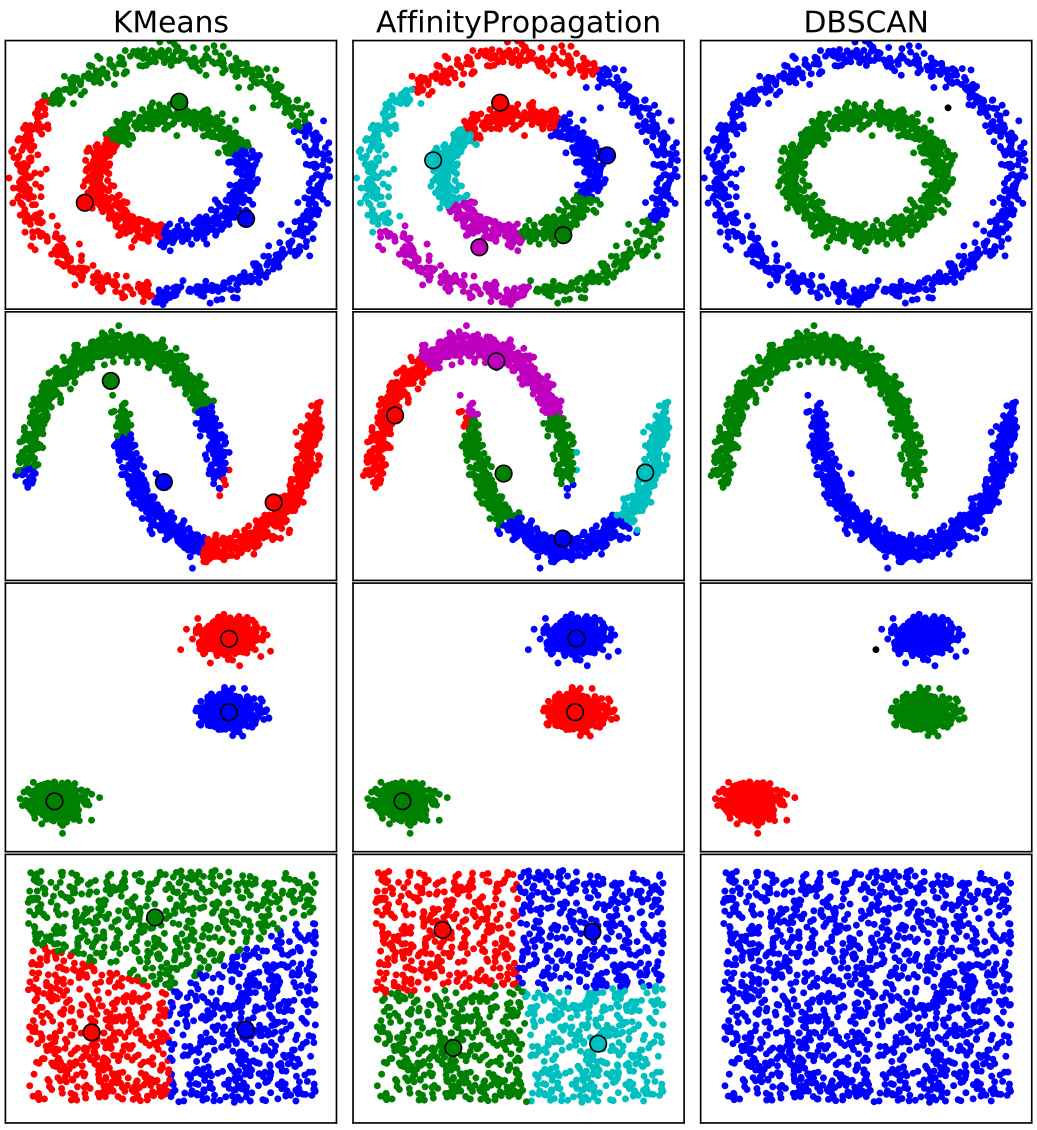


Figure 10: Comparison of clustering methods on toy dataset [76]

3.2.2 Parameter Analysis

Parameter selection for DBSCAN can significantly impact the discovery of clusters within a dataset and can influence the final clustering assignments of points. To date, there is no single automated method for extracting the "best" DBSCAN parameters and as such human reasoning and domain expert knowledge is often the primary method for determining suitable parameters. Typically the selection of *MinPts* and *Eps* are very influential on the selection of the other and most methodologies rely on first establishing one of the parameters in order to derive the other. This section serves to identify existing methodologies that are primarily automated and could be used in conjunction with heuristics for the TSP. The methods will be further examined in Chapter 4, however, visual inspection of sample TSP instances can be effective for understanding the behavior of the derivation methods versus alternatives and the easiest methods for determining parameters.

The original authors of DBSCAN [30] suggest an interactive approach when establishing suitable parameters. For *MinPts*, the authors note that no discernible difference exists between $MinPts = 4$ and $MinPts \geq 4$ for all two-dimensional data with which they experimented. Increasing *MinPts* would require additional computational time and thus working with values less than or equal to four significantly reduces the search space. The proposed values for *MinPts* in this research are restricted to two, three, and four. The reason for considering $MinPts \leq 4$ is because for applications to the TSP the relationship between two nodes can be significant enough to constitute the basis for a cluster. While a cluster arrangement with $MinPts = 4$ might be more meaningful, there is no reason to eliminate the possibility that $MinPts \leq 4$ could produce favorable results. $MinPts = 1$ is not logical because every point would become a cluster in and of itself regardless of *Eps*.

In [30] the authors derive *Eps* having already chosen the *MinPts* parameter.

This is common amongst most parameter derivation methodologies and helps guide the choice of *Eps* significantly. *MinPts* can be referred to as k in order to generate a k^{th} nearest neighbor set of distances. Sorting and plotting a list of k^{th} nearest neighbors allows for the creation of a k-distance plot that plots data points against the distance to its k^{th} nearest neighbor. Formulating the k^{th} nearest neighbor list and subsequent plot is the most common basis for determining *Eps*. A sample k-distance plot for the berlin52.tsp instance is featured in Figure 11.

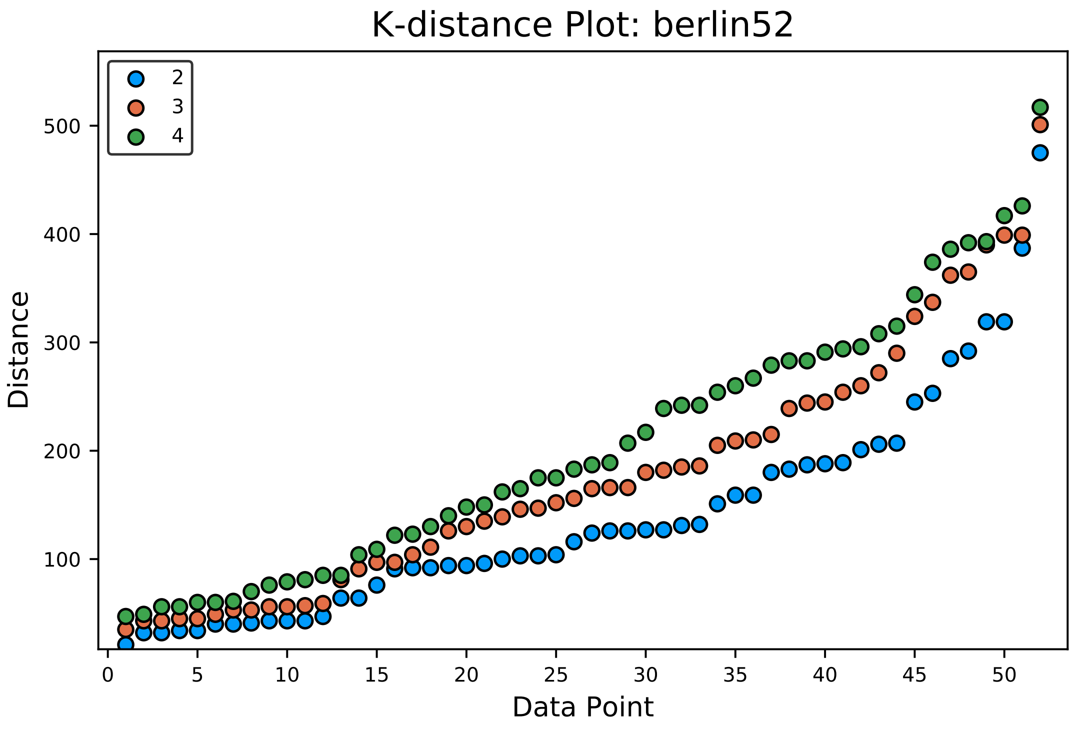


Figure 11: Sample multi k-distance plot for berlin52.tsp

In [30], the authors suggest visually inspecting the k-distance plot for the first valley starting with the greatest k-distances. The algorithm would then be run with the estimated parameter and tuned based on inspection of the results by the user. This approach has two flaws when considering it for the TSP. The first is that it operates under the assumption that the majority of points will be core points and sets *Eps* based on a valley where the k-distance would consider the minority of points to be noise. Secondly, the determination of *Eps* require user interpretation and could be difficult to justify or measure the robustness of the choice. This difficulty is illustrated in Figure 12 where two potential valleys are identified. Not only are they difficult to identify, but the exact point that can be considered a valley is not clear. The shaded regions highlight potential areas to choose as the valley and extract corresponding *Eps* values from. The arrows approximate the sampling range that would be used under this method.

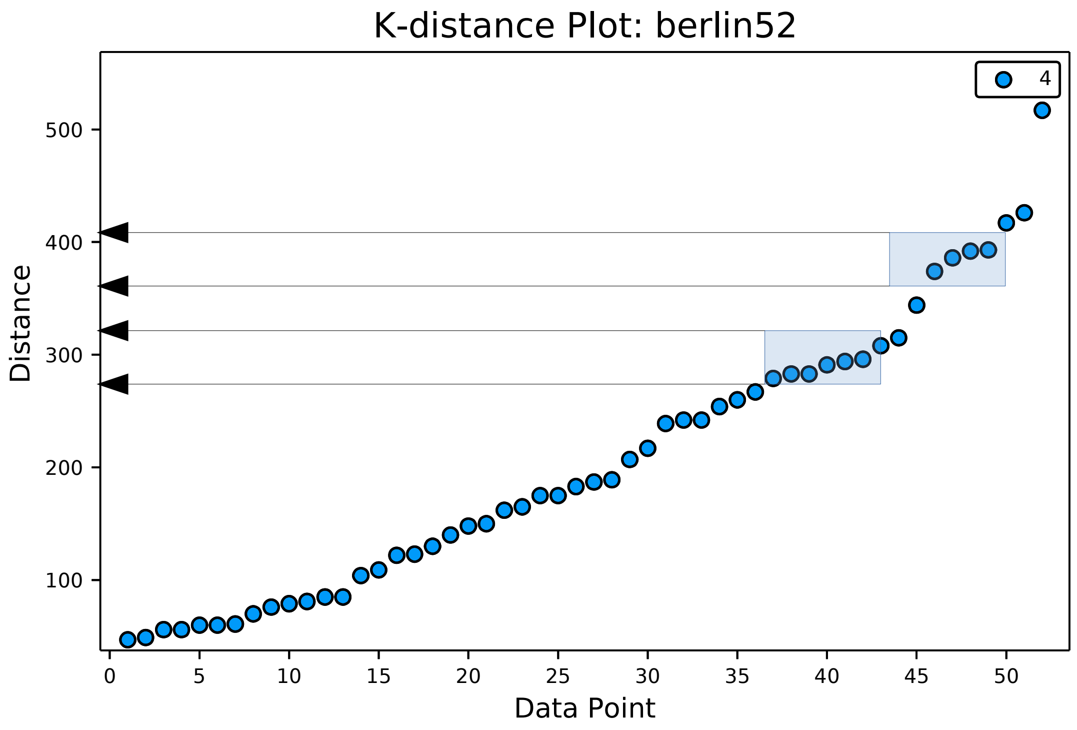


Figure 12: Valley identification k-distance plot for berlin52.tsp

Another approach to choosing *Eps* is based on the expected k-distance under an assumption of evenly spaced points in a grid. A process based on this idea was used by the authors in [87]. The problem environment which they worked in was uniform across all problems generated and had the following features: Each TSP instance consisted of 100 cities generated within a [0,400] by [0,400] grid. Using the same grid and the same number of cities allowed the authors to establish an $Eps = 40$. This value roughly corresponds to the expected distance of the 4th nearest neighbor assuming all points are evenly distributed throughout the region. A generalization of this concept can be used given a

$b \times b$ grid containing n points:

$$d_4 = Eps = \frac{b}{\sqrt{n} - 1} \tag{22}$$

Using this formulation the authors would have chosen $Eps = 44.44$ but there approximation validates this technique was used. To employ this strategy with other TSP instances the problem would either need to be scaled to a square grid prior to establishing *Eps* or an approximation that follows the same methodology could be employed. In this research an approximation method is used based on the maximum distance between the furthest most points within a dataset to establish a "grid" that can then be used to estimate the distance of the 4th nearest neighbor. The procedure for this estimation is based on the maximum distance between two nodes in C. Notating the maximum distance between two nodes as G, this value can be assumed as the hypotenuse of a triangle covering a square grid of the region with dimensions $b \times b$. Using the pythagorean theorem the grid estimate is as follows:

$$G^2 = b^2 + b^2 \tag{23}$$

Rearranging Equation 23 results in the following grid approximation:

$$b = \frac{G}{\sqrt{2}} \tag{24}$$

The approximation of b then allows for calculating *Eps* using Equation 22 as if the grid had previously been known.

The two methods presented thus far for identifying an appropriate *Eps* are not necessarily automated or flexible enough for use in conjunction with the TSP. Determining *Eps* from a k-distance plot might be useful in a data mining application where a subject matter expert could help guide the search but the method is certainly not reliable enough to be used in an automated way. As for a grid approximation with a fixed calculation for *Eps*, the value might work well as a starting point by a user but if it does not yield a positive result there is no obvious path forward for tuning the value. As these two methods have obvious shortcomings and the goal of this research is to develop useful clustering results in conjunction with TSP heuristics, more flexible and automated methodologies for determining *Eps* are explored.

One fully automated strategy for both *Eps* and *MinPts* was presented by Zhou, Wang, and Li in [93]. The authors note that a sorted k-distance value for a particular k is approximately Poisson distributed and thus the maximum likelihood value for the distribution is the geometric mean. Following this strategy, *Eps* is calculated as follows:

$$Eps = \bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i \tag{25}$$

This determination of *Eps* requires that a k level be determined in advance. However, this value is not automatically used for *MinPts*. Instead, each Eps-neighborhood in the dataset is queried for the number of points that lie within it and then the average is taken to calculate a suitable value:

$$MinPts = \frac{1}{n}\sum_{i=1}^{n} p_i \tag{26}$$

The authors compared results on benchmark clustering datasets to the methods proposed by the original DBSCAN authors and were able to achieve greater accuracy. The use of the mean does pose an issue in a dataset that has extreme outliers as this could artificially inflate the value for *Eps* above what is suitable for the dataset. An extension of this research would be the use of the median in place of the mean and then performing the same procedure. This extension will be explored in this research. The most significant element of the research conducted was that the most desirable clustering results on two-dimensional data came when *MinPts* was not equal to 4 as prescribed by the original DBSCAN authors but was smaller. The results of the approach prove that an automated strategy for determining *Eps* and *MinPts* can be effective.

Research conducted by van Hemert in [90] also used the geometric mean as a starting point to determine *Eps*. Initially, the mean and standard deviation of k-distance values were taken and the working range for *Eps* then became $[\bar{X}, \bar{X} + 2\sigma]$. Instead of choosing a singular value for *Eps*, s values were sampled at equal distances throughout the range. In [90], s was initially set to 10. Subsequently, DBSCAN was run given the s values for *Eps* and the predetermined value for *MinPts*. After the runs complete, the number of clusters for each *Eps* are saved and the values corresponding to the most frequent number of clusters are saved. From this list, the median is taken and that value is used as *Eps*. In the event that the most frequent number of clusters is not unique, the process is repeated but instead with $s = 2s$ and thus doubles the number of samples taken in the range. This process repeats until the greatest number of clusters in the range is unique. The method proposed works well but suffers similarly to other methods explored in that the mean and standard deviation are susceptible to outliers that skew the range.

Having research affirming the geometric mean as a method for calculating *Eps* naturally leads to the use of other centrality measures. The most obvious of which would be the median of a k-distance set. Using such an approach in conjunction with a set of *Eps* values would eliminate the effect that outliers could have on alternative calculation methods that involve the mean. Additionally, most proposed methods have a tendency to focus on values for *Eps* that are greater than the mean or median values obtained in a k-distance plot. As previously mentioned, this approach is rooted in data mining applications where the assumption is that points exist in clusters and the primary goal is the discovery of those clusters. For application to the TSP however, it cannot be assumed that useful clusters exist a priori, thus more flexibility must be given to *Eps* values such that consideration is given for situations where a minority of points might constitute a cluster.

With the idea of incorporating flexibility and robustness into a parameter determination method, this research explores a larger set of possible *Eps* values than previous research. The primary motivation for exploring a larger set of *Eps* values is that clustering for solving the TSP may not require the same set of rules that are typical for most clustering applications. In addition, a single method for determining *Eps* may not be suitable for all TSP instances and thus a larger set should be sampled. This research presents an additional methodology for determining both *Eps* and *MinPts* that is based on the best practices of other research yet incorporates a larger set of possible values by drawing from grid search techniques. The grid requires a discrete candidate set for *MinPts* to establish one dimension of the grid and is initialized, $X_{MinPts} = [2, 3, 4]$. The second dimension of the grid is also initialized as a discrete set of values from the k-distance set. The granularity of the *Eps* set can be controlled by the number of samples taken within the range bounded by the 0.1 quantile and the 0.9 quantile. The set is initialized with 10 evenly spaced samples taken within the range, $X_{Eps} = [0.1q, \ldots, 0.9q]$. The grid values are then evaluated based on the effectiveness of the parameters in conjunction with the chosen heuristic. If particular regions of the grid perform more favorably the grid can be reduced to a smaller region with finer granularity centered around that region. This method does incur a greater computational cost than a singular value determination but if a singular value did not perform well alternative values would still need be explored.

The parameter determination methods presented in this research are evaluated in conjunction with TSP heuristics that are adapted to work in conjunction with cluster assignments. Therefore, reviewing the clustering assignments generated alone without a TSP tour is not useful in this research. The methods used to generate suitable *Eps* and *MinPts* parameters throughout the rest of this research are summarized in the following:

1. Estimate the size of the grid encompassing all points by setting the diagonal, G , to the maximum distance between two nodes. Having the diagonal, calculate the size of the grid assuming it to be square, $b = \frac{G}{\sqrt{2}}$. Use $MinPts = 4$ along with $Eps = \frac{b}{\sqrt{n-1}}$.

2. Determine a k-distance set, X, for determining *Eps* by taking the mean of the k-distance set, $Eps = \frac{1}{n}\sum_{i=1}^{n} X_i$. Use *Eps* to query the Eps-neighborhood of every point and set *MinPts* to the mean of the query values, $MinPts = \frac{1}{n}\sum_{i=1}^{n} p_i$.

3. Determine a k-distance set, X, for determining *Eps* by taking the median of the k-distance set, $Eps = \text{median}(X)$. Use *Eps* to query the Eps-neighborhood of every point and set *MinPts* to the mean of the query values, $MinPts = \frac{1}{n}\sum_{i=1}^{n} p_i$.

4. Develop a grid of points by establishing a possible set of *MinPts*, X_{MinPts}, and a candidate set of *Eps* values, X_{Eps}, that is constructed from evenly spaced quantiles in the range $[10^{th}\% \ldots 90^{th}\%]$. The product of these sets creates a grid of parameters to be evaluated.

3.3 Heuristic Methods with Clustering

The novelty of this research is the incorporation of DBSCAN clustering assignments into existing heuristics for the TSP. The goal is that clustering assignments will be useful in improving the performance of well established heuristic methods through this incorporation. Related research has been partly conducted by incorporating k-means clustering and affinity propagation with a genetic algorithm for solving the TSP [22, 77, 28]. In each of these previous approaches a similar strategy was used. First, a path was produced within each cluster with the goal of creating a locally optimal solution within each cluster. Common amongst all previous research was the use of a genetic algorithm in this capacity. Secondly, the locally optimal solutions are connected into a global path that traverses all nodes. In [22] the clusters were connected globally using a nearest neighbors approach. In [77, 28], a genetic algorithm was again used to complete the global tour. The low level first high level second approach taken by previous researchers is important in utilizing the clustering assignments. However, it is not always necessary but can be exploited to take advantage of parallel computing. Instead the heuristic method being adapted can be constrained to incorporate the clustering assignments. The exact implementation can be adapted to utilize best computing practices. This research presents two novel methods based on well known heuristics. The first is the *Clustered Greedy Heuristic*. The second is the *Clustered Nearest Neighbor*. Both of these approaches are incorporated with DBSCAN as the clustering algorithm but any clustering algorithm that maintains independent clusters could be employed with these methods.

3.3.1 Clustered Greedy Heuristic

The Clustered Greedy Heuristic extends the greedy heuristic by incorporating clustering assignments of nodes into the process of building a tour. As detailed in Section 2.5.2, the greedy heuristic is a tour construction heuristic that has been proven as one of the most reliable and best performing tour construction heuristics. The greedy heuristic initializes by sorting the distance matrix of TSP nodes into a queue such that the shortest edges are prioritized ahead of longer edges. The algorithm then attempts to add edges into an undirected graph based on this priority while it does not violate the construction of a Hamiltonian cycle.

To simply describe the modified procedure, edge additions are made in the same way they would be without clustering assignments but the additions are constrained such that nodes within a cluster can only connect to other nodes in the same cluster. The result of this is disjoint paths that traverse all of the nodes within a cluster but do not complete a global tour. The remaining nodes, of which any not connected at all would be noise, are connected by adding edges to connect the clusters and noise points to complete the tour. The goal is that a clustering method that performs well can improve the performance of the greedy heuristic when not considering the clustering assignments. The high level procedure for the Greedy Clustering Heuristic is presented in Figure 13. From a computational standpoint, the only additional complexity added to the greedy heuristic is DBSCAN. However, the isolation of nodes in clusters allows the edges to be added independently within the clusters before the global space is considered. This process lends itself to parallelism, which can be used to decrease the computational time of the algorithm. In addition, the sorting of the distance matrix can be reduced to only the distances pertaining to the nodes within the cluster being considered and further reduces the computational complexity.

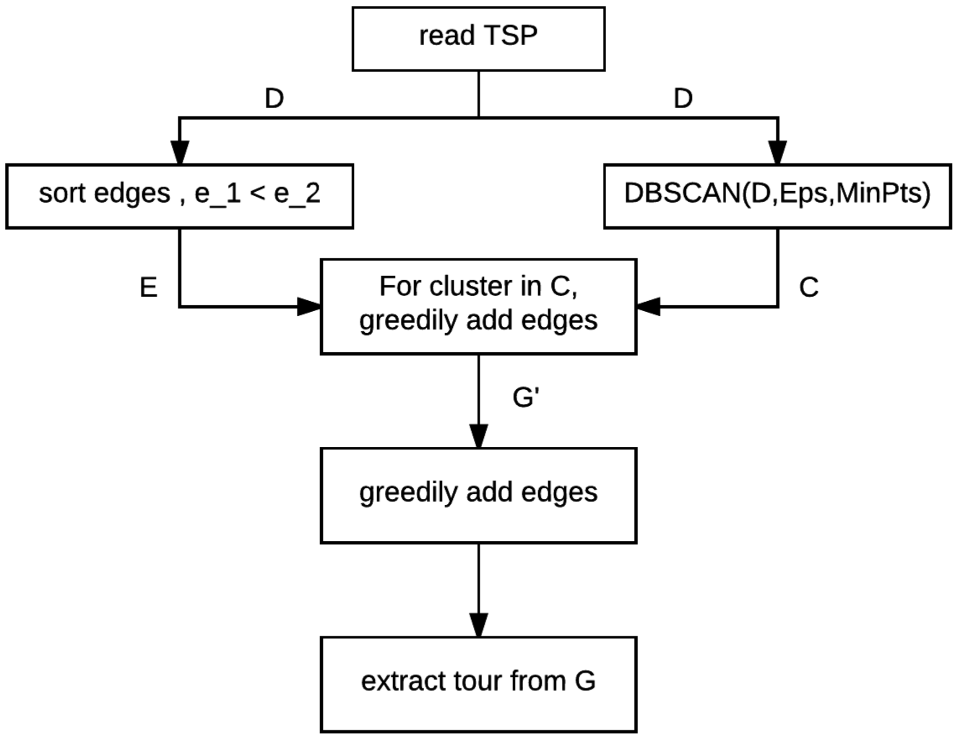


Figure 13: Flow Diagram for Clustered Greedy Heuristic

The Clustered Greedy Heuristic starts by reading a TSP instance and extracting the nodes. The pairwise distance matrix, D, is then constructed based on the given distance metric, which in most cases is Euclidean. The distance matrix is then passed to both the greedy heuristic and to DBSCAN. For the greedy heuristic, the distance matrix must be sorted to produce a vector of edges, denoted by E, such that the shortest edges will be attempted to be added first to the fully disconnected graph, G. For DBSCAN, the distance matrix is paired with the inputted *Eps* and *MinPts* parameters to produce cluster assignments for all nodes, C. As previously noted, the restriction of edges withing a cluster allows for efficiencies to be gained during computation such that the entire distance matrix need not be considered during sorting. For every cluster generated by DBSCAN, edges are greedily added within the cluster to produce a path through the nodes in the cluster. The resulting cluster now is fully connected with the exception of two nodes that have only a single connection, which will serve as the nodes to connect the cluster path to the global path. After the cluster edge additions are made, the entire space is considered, including the noise points and a global tour is greedily constructed. The algorithmic procedure is presented in Figure 14. As can be seen in Figure 14, the main algorithm relies on two supporting functions for determining whether an edge addition would be valid. The first is the *degreeCheck*, which determines whether or not either of the nodes that the edge would connect can be connected validly. The degree of a node is measured by the number of edges connected to that node. A disconnected node has a degree of zero. In the case when either node has a degree of 2, the node is already fully connected and an edge addition involving that node would compromise the construction of a valid tour. Otherwise, the edge can be assessed for the more computationally expensive procedure of determining whether or not it forms an invalid cycle. This is performed by *cycleCheck*.

The *cycleCheck* function determines whether or not the two nodes being connected by the edge are already reachable from one another, in which case

Figure 14 Algorithm: Clustered Greedy

```
procedure ClusteredGreedy(D, eps, MinPts)
    $C$ = DBSCAN(D, eps, MinPts)
    $E$ = sorted edge list s.t. $e_1 < e_2$
    $G$ = Fully disconnected undirected Graph of $D$
    for $cluster$ in C do
        while edgeCount $< m - 1$ do
            $G' = e_i$ from $E$ added to $G_C$
            if degreeCheck(G', $e_i$) then
                if cycleCheck(G') then
                    $G_C = G'$
            next $e_i$
    $G = \sum G_C$
    while tour incomplete do
        $G' = e_i$ from $E$ added to $G$
        if degreeCheck(G', $e_i$) then
            if cycleCheck(G') then
                $G_C = G'$
        next $e_i$

procedure degreeCheck(G',e)
    valid = $true$
    for node in e do
        if degree(node) $> 2$ then
            valid = $false$

procedure cycleCheck(G')
    valid = $true$
    if $G'$ contains a cycle then
        if cycle is non Hamiltonian then
            valid = $false$
```

if they are, the function determines whether the edge would complete a full Hamiltonian tour. In practice, paths are stored as lists and as they are connected the lists are merged. When the two nodes being connected are in the same list, it is known that the edge cannot be added as it would complete a non-Hamiltonian tour. The addition of the final tour does not undergo this same check, however, because the tour would never complete if the number of edges already in the graph was not considered. As such, in the case of m nodes, the *cycleCheck* is bypassed when the number of edges, $edgeCount$, is equal to $m-1$. This hold because the number of edges required to complete the tour is always equal to m. Alternatively, the edge addition loop could be terminated at $m-1$ nodes as it is in the cluster portion of the edge additions and the two nodes with $degree = 1$ connected.

The final step in terminating the algorithm is the extraction of the path from the graph, G. The path is extracted by performing a self avoiding walk for m steps. The self avoiding walk moves from node to node with the restriction that the next node must not be in the already traversed path. The result is a permutation vector containing the constructed path for the TSP of length m. In practice, the path is already in memory because it is required in *cycleCheck* and can easily be extracted from the list of connected components.

3.3.2 Clustered Nearest Neighbor

The Clustered Nearest Neighbor Heuristic extends the Nearest Neighbor Heuristic by incorporating clustering assignments of nodes into the process of building a tour. The Nearest Neighbors Heuristic is detailed in 2.5.1 in and is one of the most widely used heuristic methods for constructing solutions to the TSP. It is often used as a tour constructor as an initialization for singular or population based improvement methods. One reason for this is the simplicity of the algorithm; start at a node and always move to the nearest available node until you are able to return to the starting node. The other reason is that for situations in which multiple candidate solutions are required, such as initialization for a population of solutions in an evolutionary algorithm, Nearest Neighbors is capable of producing numerous unique tours based on the choice of the starting node.

The modification of the Nearest Neighbors is somewhat similar to the modification employed in the Clustered Greedy Heuristic. The movement from the current node to the nearest neighbor proceeds in the same way it would in the standard algorithm with the exception that nodes in a cluster can only move to its nearest neighbor that also exists in the same cluster. Obviously this procedure would get stuck in a cluster and not complete a tour, therefore this constraint is relaxed when there are no unvisited nodes remaining in the cluster. Any nodes that are not part of a cluster, such as the nodes designated as noise from DBSCAN, are free to move to their nearest neighbor without restriction. However, once a node is visited that is within a cluster, the nearest neighbors are restricted to nodes that are within that cluster. The process diagram for Clustered Nearest Neighbors is presented in Figure 15. As can be seen in the diagram, once a node in a cluster is visited all nodes in that cluster must be visited before nodes outside of the cluster are considered.

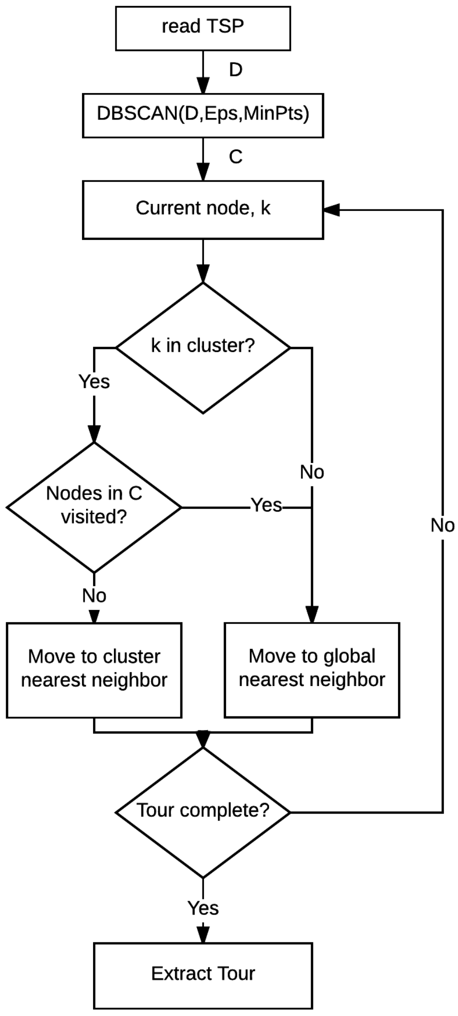


Figure 15: Flow Diagram for Clustered Nearest Neighbor

The Clustered Nearest Neighbor algorithm starts by reading a TSP instance and extracting the nodes. The pairwise distance matrix, D, is then constructed based on the distance metric provided. The distance matrix is paired with *Eps* and *MinPts* and passed to DBSCAN for the generation of cluster assignments, C. A set of candidate nodes, T, is initialized to include all nodes in the TSP instance and a fully disconnected undirected graph, G, is comprised of these nodes. To initialize the process of constructing a tour, a random node, j, from T is chosen as the starting point. The starting point can optionally be supplied to the algorithm as an argument. The current node is determined by finding the nearest neighbor to j. The current node is then removed from the candidate set. If the current node is a member of a cluster, the candidate set is restricted to nodes that are also members of that cluster until the candidate set is empty. Otherwise, the candidate set includes all remaining unvisited nodes. The algorithmic procedure for performing the traversal is detailed in Figure 16.

Figure 16 Algorithm: Clustered Nearest Neighbors

```
 1: procedure ClusteredGreedy(D, eps, MinPts)
 2:     $C$ = DBSCAN(D, eps, MinPts)
 3:     $G$ = Fully disconnected undirected Graph of $D$
 4:     $T$ = set of all nodes in $D \setminus \{m\}$
 5:     $j$ = random node in $T$
 6:     while $T \neq \emptyset$ do
 7:         $T' = \{T_i\} \;\; \forall \;\; i.cluster = j.cluster$
 8:         if j.cluster = noise or $T' = \emptyset$ then
 9:             $T' = T$
10:         $k = min\{c_{jk} | k \in T'\}$
11:         add edge $e_{jk}$ to $G$
12:         $j = k$, remove $k$ from $T$
```

The next node is always the nearest unvisited node to the current node as would be expected from the name nearest neighbor. As a node is visited, the edge between it and the previous node is added to the graph, G, and simultaneously removed from the candidate set, T. In practice, the candidate set of nodes need not be altered directly. Instead, a binary or boolean decision set can be used track the eligibility of nodes based on what has already been visited and whether or not the nodes are part of the same cluster.

The algorithm requires a cluster candidate set to only be constructed once, at which point every node in the cluster is traversed and then the candidate set of the cluster is empty and the global set of points is considered. The result is that a tour enters and leaves a cluster exactly one time. This is in common with the Clustered Greedy Heuristic. As with nearest neighbors, any of the nodes in the TSP can be chosen as a starting node regardless of whether it is a noise point or a member of a cluster. The algorithm terminates when there are no longer any candidate nodes in the set and the path is extracted from G.

Chapter 4

Implementation and Results

In order to examine the efficacy of the clustered greedy and nearest neighbor heuristics, the algorithms performance were numerically evaluated through a series of tests. Moreover, the clustered heuristics were utilized to solve several benchmark TSP instances and thereupon compared to the results generated by the classic, non-clustered approaches. This chapter presents a detailed description of the experimental setup utilized to analyze the clustered heuristics across a wide range of problems. Further, the performance of several parameter determination methods for DBSCAN are examined. The performance improvements facilitated by the clustered greedy and clustered nearest neighbor heuristics are presented in regard to the determination method that results in the best performance improvement.

4.1 Implementation

The development of reusable and flexible utilities is paramount in being able to conduct research efficiently. Having a structure in place that organizes problem information allows for the majority of time to be spent on development and experimentation. In addition, an environment that is well suited to numerical computation, but is also easy to use and adapt as the research process evolves, is necessary. In this research the Julia Programming Language was used for all development, experimentation, and analyses. Julia is a high-level and high performance dynamic programming language that has particular focus on numerical computing [12]. Julia integrates high performance with ease of use and is perfectly suited to the challenges of implementing optimization heuristics involving the TSP. All experimentation and subsequent computational runtime analyses were performed on a machine running Ubuntu 16.04 featuring an Intel Xeon CPU E5-2665 2.40GHz processor.

4.1.1 TSPLIB

For the experimentation of this research, TSPLIB [80] was utilized for benchmark problem instances. TSPLIB hosts 111 different symmetric TSP instances ranging in size from 14 cities to 85,900 cities. At the core of TSPLIB are *.tsp* files that store summary problem information and relevant data pertaining to the particular problem instance. These files are text files that feature keywords followed by the information elaborating that keyword. The authors of TSPLIB break the information contained in a *.tsp* into two separate categories: specification information and data.

The specification information is required to understand the problem and also provides the required logic for correctly interpreting the data. The basic information supplied in the specification is the name of the problem instance, the dimension of the problem, the type of data provided, and the distance metric to be used. The data portion of the file specifies the type of data provided by the keywords in the file and also provides the actual problem data. The problem data takes two general forms. The first option is that node coordinates for cities are given and the distance metric provided by the specification section is to be used to compute the distance matrix. The second option is that an explicit distance matrix is given. Even when an explicit distance matrix is provided, the coordinate nodes or a set of display nodes may be included that is intended only for graphical representation of the problem.

In order to consolidate the data in a *.tsp* file and to generate a uniform interface across all problem instances, a library was created that exports a single type in Julia, namely the TSP. The TSP type is similar to an object that is featured in object oriented programming languages and is a data structure that holds the useful information taken from a TSP instance. The fields of the type and the type of data stored in each field are displayed in Figure 17.

```julia
type TSP
    name::String
    dimension::Integer
    weight_type::String
    weights::Matrix{Float64}
    nodes::Matrix{Float64}
    Dnodes::Bool
    ffx::Function
    pfx::Function
    optimal::Float64
  end
```

Figure 17: TSP type structure

The field information is obtained by parsing the *.tsp* file for keywords and extracting the relevant data. In the case when a distance matrix is not explicitly provided, the distance matrix is calculated based on the provided distance metric. The type is constructed via two functions; *readTSP()* which accepts a file path to a *.tsp*, and *readTSPLIB()* which accepts the name of a TSPLIB instance to load from the library. The aptly named fields contain information that is obvious. Some of the less obvious are described in more detail here to understand how the type is to be utilized. The Dnodes field indicates whether or not the nodes provided are for display only. If the nodes are not provided, the nodes field is initialized as zeros. The ffx and pfx fields are fitness functions to determine the length of a tour. The ffx field calculates the entire cost of a tour, where the final node in a path vector must return to the first node. Alternatively, the pfx function can determine the cost of any given sized path by calculating the path traversal cost between the nodes provided. Lastly, the optimal field stores the optimal tour length if provided, which is the case for all TSPLIB instances.

4.1.2 Experimental Setup

The experimental setup deployed in this research examines the effectiveness of the clustered greedy heuristic and the clustered nearest neighbor against their standard versions. To provide a thorough and effective comparison of both heuristics, 108 TSP problems ranging in size from 14 to 18,512 cities were chosen from TSPLIB. The two largest problems of TSPLIB (i.e. greater than 18,512 cities) were excluded due to limitations in the available memory required to solve such large problems. For both the clustered greedy and the clustered nearest neighbor, multiple parameter determination methods are employed that seek to find useful input parameters to DBSCAN. As detailed in Section 3.2.2, four parameter determination methods are explored in this research. The first method is referred to as the Square Estimate. It is used to estimate the size of the tour space by calculating the diagonal and assuming the space to be square, at which point *MinPts* is fixed to four and *Eps* is the expected distance of the fourth nearest neighbor. The second method is based on taking the mean of a k-distance set and then querying the average of the Eps-neighborhoods for *MinPts*. This method will be referred to as Flexible Mean. The third method is performed exactly as Flexible Mean but instead uses the median and thus will be referred to as Flexible Median. The last method is a grid search across two candidate sets, one for *MinPts* and the other for quantiles of a k-distance set to determine *Eps*. The quantile values chosen for a broad exploration of points are taken starting with the $10^{th}\%$ quantile and proceed for every $10^{th}\%$ quantile up to the $90^{th}\%$. The subsequent grid results in 27 different parameter combinations. This last method will be referred to as Grid Search.

It must also be noted that each parameter determination method produces a different quantity of parameter combinations and thus requires the clustered heuristic to be executed proportionate to this quantity. Square Estimate derives a single parameter and thus requires a single run, whereas Flexible Mean and Median are each run with three candidate values for determining the k-distance level. The grid search is the most exhaustive but obviously the most computationally expensive and examines twenty-seven different combinations of *Eps* and *MinPts*.

With respect to the heuristics and their clustered counterparts, consideration is given to the stochasticity of the nearest neighbor and clustered nearest neighbor. For the nearest neighbor each iteration for a given TSP instance may generate a unique solution from each starting point. To this end, the evaluation of the performance of the parameter determination methods with respect to the nearest neighbor considers only the best tour provided from the candidate starting points. This is justified by the fact that in practice only the shortest tour generated would be utilized as a solution and the effectiveness of the parameter determination methods is most suitably measured by their best performance.

4.2 Parameter Determination Evaluation

The parameter determination methods that require little guidance in determining parameters for DBSCAN are referred to as "automated". In particular the Flexible Mean, Flexible Median, and Square Estimate can all be considered as such. While Square Estimate requires zero input parameters, Flexible Mean and Flexible Median require only, k, which determines the k-distance level from which to start. The best case scenario would be for any of these methods to consistently generate high quality parameters for constructing TSP tours. Each of the parameter determination methods examined are computationally inexpensive and would mark a major step forward in automating the entire process of determining parameters for constructing a quality TSP tour. The Square Estimate is only capable of generating a single parameter combination for *Eps* and *MinPts*. Flexible Mean and Flexible Median are both presented at their respective k-level, signifying the k-distance set that is used for generating parameters.

The analysis of the Grid Search parameter determination method is performed similarly to the analysis of the automated methods. The major difference is that the Grid Search is an exhaustive approach and would be difficult to implement on larger problems without the incorporation of parallel computing or some discerning criteria to significantly reduce the size of the grid. Thus, the results of Grid Search serve as a basis for understanding the effectiveness of deriving *Eps* across quantiles of the k-distance set in conjunction with a predetermined value for *MinPts*. As previously mentioned, the experiments performed in this research evaluate three $MinPts$ values and samples *Eps* from the k-distance set at nine evenly spaced quantiles.

To properly assess the performance, robustness, and consistency across the entire class of benchmark TSP instances, the methods are measured based on their overall performance and in relation to their counterparts. A comparative summary of the performance of these different methods is broken into two separate sections each featuring performance comparisons for the clustered greedy and clustered nearest neighbor. Each table features the mean percent error and the standard deviation of the error with respect to optimality across all benchmark problems. The table also includes the relative percent best value, which is a measure of the relative strength of the parameter determination method against the other methods in the table. The value of the metric indicates the percentage in which that method outperformed all other methods in the table.

4.2.1 Automated Method Results

Table 1 presents a comparison between the automated parameter determination methods for the clustered greedy heuristic. Across all benchmark problems, the standard greedy heuristic obtained a mean error of 17.03% with a standard deviation of 6.21. None of the automated methods were able to establish a lower average error across all benchmark problems. However, the mean error for both Flexible Mean with $k = 2$ and Flexible Median $k = 2$ are comparable to the standard greedy with means of 19.47% and 18.09% respectively. Given the standard deviations of the standard greedy and the automated clustering methods, there is significant variability across in its performance across the benchmark problems. Comparing the Square Estimate to each of the adaptive parameter methods, it obtained the worst average performance but shows promise having obtained the best solution 16.67% of the time relative to the other methods. This again reinforces the parameter variability across the different TSP instances tested. Having the lowest mean errors, Flexible Mean and Median also achieve the relative best solution 26.85% and 27.78% of the time respectively.

Parameter Method		**% Error**		**Relative**
Epsilon	**k-level**	**Mean**	**Std. Dev**	**% Best**
Flexible Mean	2	19.47	7.29	26.85
Flexible Mean	3	21.78	8.18	5.56
Flexible Mean	4	21.74	8.40	10.19
Flexible Median	2	18.09	6.34	27.78
Flexible Median	3	20.23	7.69	10.19
Flexible Median	4	21.80	8.32	2.78
Square Estimate	4	21.81	8.62	16.67

Table 1: Automated Parameter Determination Comparison: Clustered Greedy

One of the key takeaways from Table 1 is that both Flexible Mean and Flexible Median perform significantly better with a k-level of 2. With $k = 2$, the magnitude of both *Eps* and *MinPts* would be smaller than higher values of k. If utilized in combination, this would assist in the identification of small dense areas as core points but would be highly likely to flexibly expand due to the expected smaller value of *MinPts*. This gives some indication that allowing the clustering to be as flexible as possible, thus allowing for small clusters to form, lends itself to a better solution than only qualifying larger clusters.

Table 2 presents a comparison between the automated parameter determination methods for the clustered nearest neighbor heuristic. Across all benchmark problems, the standard nearest neighbor obtained a mean error of 18.45% with a standard deviation of 5.63. None of the automated methods were able to establish a lower average error across all benchmark problems and they performed significantly worse than standard nearest neighbors. In all cases, the automated methods have a higher mean and standard deviation than nearest neighbors. Amongst the automated methods, Flexible Mean with $k = 2$ and Flexible Median $k = 2$ can be clearly identified as the most superior parameter determination methods with means of 21.95% and 20.26% respectively. They each also show the lowest standard deviations amongst the methods but comparatively to the mean the standard deviations indicate highly variable performance. The real dominance of the two superior methods is in the relative strength against the alternative methods. Flexible Median with $k = 2$ obtained the best relative solution on 42.59% of problems and Flexible Mean with $k = 2$ was best on 28.70% of problems.

Parameter Method		**% Error**		**Relative**
Epsilon	**k-level**	**Mean**	**Std. Dev**	**% Best**
Flexible Mean	2	21.95	7.23	28.70
Flexible Mean	3	24.73	8.54	5.56
Flexible Mean	4	25.84	8.66	2.78
Flexible Median	2	20.26	6.59	42.59
Flexible Median	3	23.06	8.21	6.48
Flexible Median	4	24.30	8.34	5.56
Square Estimate	4	24.84	9.22	8.33

Table 2: Automated Parameter Determination Comparison: Clustered Nearest Neighbor

Similar to the results of the methods utilized in conjunction with the clustered greedy, Flexible Mean and Flexible Median at the k-level of 2 exhibit the strongest performance. While the results do not indicate competitiveness with the standard nearest neighbors algorithm, the performance of the clustered greedy heursitic demonstrates that utilizing that particular k-distance set yields the most desirable results. The consistency across the clustered greedy and the clustered nearest neighbor is important in examining the usefulness of the parameter determinations, even when the performance in the clustered greedy was much stronger than the clustered nearest neighbor.

4.2.2 Grid Search

The relative performance comparison for all of the evaluated points used in Grid Search for the clustered greedy heuristic can be found in Table 3. By examining the grid quantiles, it is evident that lower quantiles, and thus smaller values of *Eps* provide more useful clustering assignments resulting in better solutions. The $10^{th}\%$ quantile across all k-levels outperforms all other grid parameters on 47.23% of problems. Averaged across all problems, the $10^{th}\%$ with $k = 2$ outperforms the standard greedy, achieving mean error of 16.97% versus 17.03%, as does the $30^{th}\%$ quantile with $k = 2$. Amongst the rest of the grid, the strongest results are found with $k = 2$ as they were with the adaptive methods. The strongest relative points fall amongst the lower quantiles, namely the $10^{th}\%$, $20^{th}\%$, and $30^{th}\%$. Identifying better performance in lower quantiles is important in reducing the size of Grid Search as well as for increasing the granularity of samples in the promising regions. A further reduction could be made to only include $k = 2$ as it consistently outperforms across most quantiles.

Grid Values		**% Error**		**Relative**
Quantile	**k-level**	**Mean**	**Std. Dev**	**% Best**
0.1	2	16.97	6.16	35.19
0.1	3	17.21	6.03	5.56
0.1	4	18.3	6.39	6.48
0.2	2	17.14	6.03	2.78
0.2	3	17.79	6.18	5.56
0.2	4	19.99	7.26	2.78
0.3	2	16.85	5.85	3.7
0.3	3	18.76	6.64	2.78
0.3	4	21.11	8.27	2.78
0.4	2	17.55	5.94	0.93
0.4	3	20.55	7.82	1.85
0.4	4	22.16	8.24	0
0.5	2	18.18	6.42	0.93
0.5	3	20.9	7.95	0.93
0.5	4	22.33	8.65	0.93
0.6	2	19.1	7.06	0.93
0.6	3	21.01	7.92	0.93
0.6	4	22.29	8.1	5.56
0.7	2	20.2	7.73	0.93
0.7	3	21.66	8.31	3.7
0.7	4	22.78	8.16	4.63
0.8	2	21.3	8.14	0
0.8	3	22.31	8.67	2.78
0.8	4	22.37	8.4	3.7
0.9	2	21.95	8.39	0
0.9	3	21.42	8.17	0
0.9	4	20.33	7.52	3.7

Table 3: Grid Search Comparison: Clustered Greedy

The results of the parameter Grid Search for the clustered nearest neighbor are presented in Table 4. Relative to the performance of the Grid Search for the clustered greedy, clustered nearest neighbors does not fare as well. The best performing grid points are at the $10^{th}\%$ and $20^{th}\%$ quantile with $k = 2$, achieving a mean performance of 18.77% and 18.96% respectively. None of the grid points obtain a better average error than the standard nearest neighbors mean error of 18.45%. The Grid Search results do further evidence that the best performing regions of the grid are the lower quantiles with $k = 2$, consistent with the results of the clustered greedy.

Grid Values		**% Error**		**Relative**
Quantile	**k-level**	**Mean**	**Std. Dev**	**% Best**
0.1	2	18.77	5.89	40.74
0.1	3	19.54	5.93	3.7
0.1	4	20.74	6.72	5.56
0.2	2	18.96	5.72	7.41
0.2	3	19.83	6.17	3.7
0.2	4	21.8	7.39	0.93
0.3	2	19.09	6.14	4.63
0.3	3	20.95	6.83	3.7
0.3	4	23.89	8.52	1.85
0.4	2	19.81	6.38	1.85
0.4	3	22.21	7.63	1.85
0.4	4	24.68	9.28	2.78
0.5	2	20.23	6.63	2.78
0.5	3	23	8.01	0
0.5	4	25.16	8.87	1.85
0.6	2	21.43	7.44	0.93
0.6	3	23.96	8.63	3.7
0.6	4	26.19	9.12	0
0.7	2	22.55	7.88	0
0.7	3	25.09	8.89	1.85
0.7	4	26.63	9.54	3.7
0.8	2	23.78	8.78	0
0.8	3	25.22	9.24	0.93
0.8	4	25.81	9.33	1.85
0.9	2	24.81	9.1	1.85
0.9	3	23.89	8.61	0.93
0.9	4	22.73	8.18	0.93

Table 4: Grid Search Comparison: Clustered Nearest Neighbor

4.3 Clustered vs. Standard Heuristic Performance

The performance comparison of the parameter determination methods gives insight into the relative strength of each method and assesses the average performance against optimality. Here the performance of the clustered greedy and clustered nearest neighbor are compared to the performance of the standard versions of the perspective algorithms without clustering. The comparison of the clustered versions are made across the different parameter determination methods. The improvement frequency is presented as a measure of the percentage of problem instances in which that particular parameter determination method outperformed the standard application of the particular algorithm. The maximum improvement measures the greatest absolute improvement when comparing the best result obtained by that method with the standard heuristic. The maximum improvement is taken from the entire set of solutions obtained by the method. For example, in a Grid Search for a given problem twenty-seven solutions are obtained from which the best performing solution is then compared with the solution from the standard heuristic.

The improvement comparison between the clustered greedy methods and the standard greedy are presented in Table 5. The worst performing method was the Square Estimate with a 33.53% improvement frequency. This is not surprising but the result is still meaningful considering it is the only method that generates a single comparable solution. Flexible Mean and Flexible Median perform comparably, as expected from their previous comparison against each other, with improvement frequencies of 43.52% and 52.78% respectively. The greatest improvement frequency is from Grid Search, with 83.33% improvement frequency. The Grid Search also obtained the largest maximum improvement at 23.35%.

Method	Improvement Frequency (%)	Maximum Improvement (%)
Flexible Mean	43.52	11.36
Flexible Median	52.78	12.19
Square Estimate	33.33	11.64
Grid Search	83.33	22.35

Table 5: Clustered Greedy Improvement vs Standard Greedy

Improvement comparisons for the clustered nearest neighbor are presented in Table 6. The worst performing method was Square Estimate with 23.15% improvement frequency and 4.09% maximum improvement. Flexible Mean and Median achieved 33.33% and 48.15% improvement frequency and 6.5% and 5.67% maximum improvement respectively. Comparing the clustered nearest neighbor improvement to the results in the clustered greedy, each parameter method performed worse for both metrics with the exception of Grid Search. Grid Search was able to outperform standard nearest neighbor on 94.44% of problems with a maximum improvement of 10.28%. For each of the parameter determination methods, maximum improvements for clustered nearest neighbor were less than the maximum improvement of the clustered greedy.

Method	Improvement Frequency (%)	Maximum Improvement (%)
Flexible Mean	33.33	6.5
Flexible Median	48.15	5.67
Square Estimate	23.15	4.09
Grid Search	94.44	10.28

Table 6: Clustered Nearest Neighbor Improvement vs Standard Nearest Neighbor

For both the clustered greedy and the clustered nearest neighbor, the improvement results indicate that each parameter determination method is capable of achieving better solutions than the standard approaches. However, the improvement frequency and maximum improvement do not take into account on which particular problems the improvements are made or the distribution of improvement. To effectively corroborate the improvements facilitated by the clustered greedy heuristic, the ten most improved problem instances are presented in Table 7. Moreover, the greedy and clustered greedy heuristic's total tour length, percent error from optimal, the method obtaining the result, and the percent improvement are examined. A comparable evaluation is performed for the best improvement results obtained by the clustered nearest neighbor in Table 8.

	Greedy		**Clustered Greedy**			
Problem	**Tour**	**Error (%)**	**Tour**	**Error (%)**	**Method**	**Improvement (%)**
pr76	147496	36.37	123323	14.02	Grid Search	22.35
dantzig42	1003	43.49	898	28.47	Grid Search	15.02
berlin52	9951	31.94	8861	17.49	Grid Search	14.45
pr226	97601	21.44	87940	9.42	Grid Search	12.02
ch150	7809	19.62	7098	8.73	Square Estimate	10.89
eil51	531	24.65	487	14.32	Grid Search	10.33
gr229	163844	21.72	150467	11.79	Grid Search	9.94
lin105	16766	16.60	15382	6.98	Flexible Mean	9.63
bier127	141339	19.49	130757	10.55	Grid Search	8.95
pr299	63333	31.42	59131	22.70	Grid Search	8.72

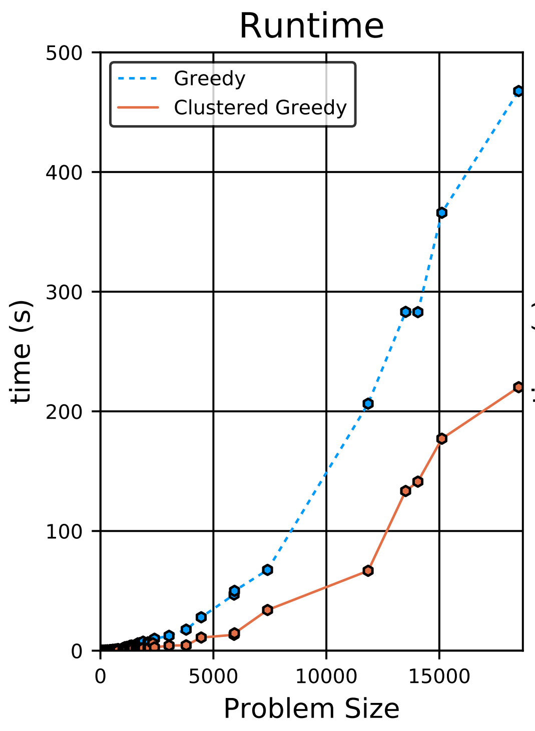
Table 7: Top 10 Improvements for Clustered Greedy vs Greedy

The best improvements obtained by the clustered greedy in Table 7 indicate that the Grid Search method is responsible for the greatest improvements on the best performing problems. The clustered greedy heuristic promoted a rather significant improvement in generating solutions when compared to the standard greedy. The most considerable advancement was seen in a TSP problem with 72 cities, wherein the clustered greedy heuristic showed a 22% improvement over the classic greedy method. It is also clear that the clustered approach allows for the algorithm to find solutions much closer to optimality. This is particularly visible in the "lin105" problem, which includes 105 cities, wherein the generated solution has only 6.98% error versus optimality. The clustered greedy also offers the greatest improvement for problems in which the greedy produces some of its worst solutions across the benchmark problems. More specifically, 90 of the 108 tested problems showed an improvement over the classic greedy, 4 problems produced exactly equivalent solutions, and 14 problems exhibited a 0.25% or less increase in total tour length. Three of the problems that showed a decrease in performance can be classified as the large problems, as they included 13,509, 14,051, and 18,512 cities respectively. However, there is not a distinct relationship between the performance and the problem size, as a problem with 15,112 cities showed an improvement of 0.75%. Thus, there does not appear to be a discernible relationship between the improvements and the problem size, however, it is clear that the largest improvements are seen in problems considered to be of small and medium size.

	Nearest Neighbor		**Clustered Nearest Neighbor**			
Problem	**Tour**	**Error (%)**	**Tour**	**Error (%)**	**Method**	**Improvement (%)**
p654	43253	24.85	39691	14.57	Grid Search	10.28
kroB100	26388	19.18	24675	11.44	Grid Search	7.74
berlin52	8920	18.27	8360	10.84	Grid Search	7.43
pr226	94122	17.11	88461	10.07	Grid Search	7.04
kroB150	32044	22.63	30345	16.13	Flexible Mean	6.5
bier127	135737	14.76	128052	8.26	Grid Search	6.5
lin105	17143	19.22	16232	12.89	Grid Search	6.34
gr96	68308	23.73	64854	17.47	Grid Search	6.26
swiss42	1547	21.52	1468	15.32	Grid Search	6.21
pr439	129060	20.37	122433	14.19	Grid Search	6.18

Table 8: Top 10 Improvements for Clustered Nearest Neighbor vs Nearest Neighbor

The ten best improvements for the clustered nearest neighbor are presented in Table 8. Similarly to the clustered greedy results, Grid Search is responsible for the greatest improvement amongst the best performing problems. The greatest improvement was obtained on a TSP problem with 654 cities, wherein the clustered nearest neighbor showed a 10.28% improvement. The best result obtained is on a fairly large problem given the size of the other most improved problems. The majority of significant improvements came on smaller and medium sized problem instances. This is consistent with the improvement results of the clustered greedy. Overall, the clustered nearest neighbor obtained better solutions than the standard nearest neighbor on 70 of the 108 problems tested, equal solutions on 33 problems, and was worse on 5 of the problems. Similarly to the clustered greedy, the clustered nearest neighbor obtained its worst relative results on the largest problems in the test set.



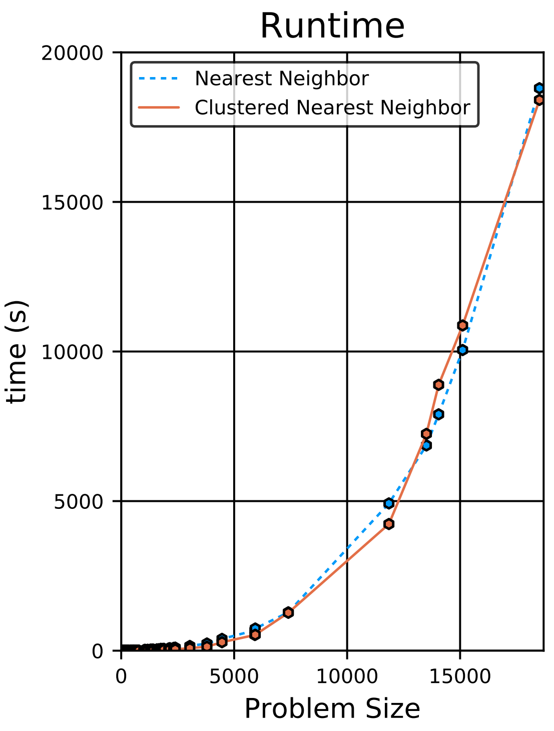


Figure 18: Runtime Performance Comparison (a) Greedy vs. Clustered Greedy (b) Nearest Neighbor vs. Clustered Nearest Neighbor

An additional element to consider when introducing DBSCAN as a clustering method into the greedy and nearest neighbors is the effect that it has on the runtime performance of the two algorithms. The runtime performance comparison between the clustered and standard versions of the algorithms is presented in Figure 18. The total runtime for the clustered greedy is calculated as the mean across all clustered runs on a particular problem dimension and includes the runtime for DBSCAN. In other words, this runtime does not account for the number of runs that may occur given a parameter determination such as Grid Search that would require the algorithm be run 27 times. Instead, the time is taken as if all instances had been run in parallel or a single instance had been run.

For nearest neighbors, the runtime for a given problem dimension is the summation of all run times across all potential starting nodes. The introduction of DBSCAN into the nearest neighbor algorithm does not appear to affect the runtime performance of the algorithm. This is logical because the smaller search tree that is utilized while moving inside of a cluster offers a performance gain that is roughly equivalent to the computational cost of running DBSCAN. The net result is that no significant gain or penalty is incurred from DBSCAN. On the other hand, the introduction of DBSCAN to the greedy heuristic brings about a significant decrease in computation time. The primary reason for the decrease in runtime is that utilizing clustering assignments significantly reduces the cost of some of the most expensive operations that the greedy is required to perform. Inside of each cluster, a far smaller number of edges need to be sorted and a much smaller dimension of constraint checks need to be made. While the total number of edges across the clustered version and the standard version are the same, the sum of the smaller dimension operations is much less expensive than considering the entire dimension of the problem. As the DBSCAN algorithm runs comparatively fast to the other operations and introduces a performance gain through dimension reduction, its runtime results in faster computation times, particularly as the dimension of the problem scales.

Method	**Mean**	**Std. Dev**	**% Best**
Standard	17.03	6.18	36.11
DBSCAN	16.85	5.82	46.30
k-means	25.98	9.96	6.48
Affinity Propagation	24.00	6.94	11.11

Table 9: Clustered Greedy Comparison

Method	**Mean**	**Std. Dev**	**% Best**
Standard	18.45	5.61	66.00
DBSCAN	18.77	5.87	13.00
k-means	27.29	9.59	24.00
Affinity Propagation	22.68	7.79	5.00

Table 10: Clustered Nearest Neighbor Comparison

Choosing the best performing parameters obtained from Grid Search and using them to produce a single tour offers a valid comparison to alternative clustering methods found in other research. The results of running k-means and Affinity Propagation are offered in comparison to DBSCAN and without clustering in Figure 9 and Figure 10 for the greedy and nearest neighbor heuristic, respectively. The results indicate that DBSCAN was not only the strongest performing clustering method but also the best performing method overall when compared to the standard greedy. The performance of these DBSCAN parameters indicates that it provides a more useful clustering assignment than k-means and affinity propagation when incorporated with the greedy heuristic. However, the incorporation of clustering with the nearest neighbor heuristic did not prove stronger for any of the clustering methods as the standard nearest neighbor obtained the best result 66% of the time. DBSCAN performed best only 13% of the time with k-means offering better clustering assignments 24% of the time. The comparison for nearest neighbors validates the results across all parameter determination methods that clustering assignments do not consistently improve the standard nearest neighbor as they do with the greedy heuristic.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

This thesis presents methods for improving classical TSP heuristics through the incorporation of density-based clustering. The algorithm presented for clustering TSP instances is DBSCAN and the two heuristic approaches in which it is integrated are the greedy heuristic and the nearest neighbor heuristic. Four methods for deriving suitable input parameters from TSP instances for DBSCAN are also examined. The effectiveness of these DBSCAN parameters are measured by running all combinations of methods and algorithms on 108 benchmark instances taken from TSPLIB. This includes results from the clustered greedy and clustered nearest neighbor. From these results, the overall performance of the clustered heuristics is evaluated based on absolute performance, performance relative to the standard version of the heuristic, and computational runtime performance.

It is shown that constraining the classic heuristics by the clustering assignments obtained from DBSCAN can introduce a significant improvement in tour lengths for the TSP. The clustered greedy heuristic resulted in a maximum improvement of 22.35% when compared with the standard greedy algorithm. The clustered nearest neighbor also improved results relative to the standard nearest neighbor algorithm with a maximum improvement of 10.28%. The inclusion of clustering into the greedy heuristic also brought about a significant decrease in computation time, particularly on the largest problem instances, indicating the clustered greedy is more scalable than the standard greedy. For the nearest neighbor algorithm, the computational expense of running DBSCAN was offset by small performance gains and netted no performance penalty or gain.

While the results in this thesis are promising, the clustered versions of the heuristics are shown to be highly sensitive to DBSCAN's input parameters. Amongst the three automated parameter determination methods presented, there is not one derivation that consistently outperforms the standard heuristic for either the clustered greedy or the clustered nearest neighbor. Each method does increase performance on particular problems. Consequently, a grid search was employed to explore points across the parameter input space. The grid search proved that given suitable parameters, the clustered greedy and clustered nearest neighbor can outperform standard results and generate near optimal solutions. The increased performance and the computational expense of the grid search highlight the trade-off that exists between qualifying input parameters and obtaining improved solutions.

5.1.1 Future Work

This thesis demonstrates the effectiveness of increasing heuristic performance through the integration of density-based clustering. The heuristics employed in this research are simple and thus their shortcomings are not entirely overcome by incorporating clustering assignments. However, the results prove that DBSCAN is an effective algorithm for partitioning a TSP instance by reducing the dimensions of the problem given appropriate input parameters. This has led to the goal of extending this research in two main areas.

The first extension of this research involves finding a more robust density-based clustering algorithm that either requires no parameters or is less sensitive to its input parameters, as is DBSCAN. Ideally, an algorithm that is able to adaptively generate clusters based on the problem space would be most preferable. In recent years, numerous extensions of DBSCAN have been evaluated and have proven to be effective in other domains of research. One interesting algorithm is the hierarchal DBSCAN (HDBSCAN), which requires only one parameter and could lead to more flexible clustering and thus more desirable results. An additional consideration is developing new automated methodologies for determining suitable DBSCAN parameters specific to the TSP. Moreover, developing a relationship between TSP attributes and the clustering input parameters could also improve the effectiveness of DBSCAN.

A further extension of this research could focus on incorporating density-based clustering with other heuristic or metaheuristic algorithms. Other research has partially integrated clustering concepts, however, no effort to date has been made specifically with DBSCAN. One potential method would be to generate clusters through DBSCAN and then employ a more robust method, such as simulated annealing, to develop shortest paths through the clusters. This would then be followed by joining the shortest paths with the remaining points to construct a tour. Future research will focus on integrating more robust TSP algorithms with a potentially more flexible density-based clustering algorithm.

Bibliography

[1] Emile Aarts, Jan Korst, and Wil Michiels. Simulated annealing. In *Search methodologies*, pages 265–285. Springer, 2014.

[2] Richa Agarwala, David L Applegate, Donna Maglott, Gregory D Schuler, and Alejandro A Schäffer. A fast and scalable radiation hybrid map construction and integration strategy. *Genome Research*, 10(3):350–364, 2000.

[3] Marco Aliotta, Andrea Cannata, Carmelo Cassisi, Rosalba Giugno, Placido Montalto, and Alfredo Pulvirenti. Dbstrata: a system for density-based clustering and outlier detection based on stratification. In *Proceedings of the Fourth International Conference on SImilarity Search and APplications*, pages 107–108. ACM, 2011.

[4] Hyung-Chan An, Robert Kleinberg, and David B Shmoys. Improving christofides' algorithm for the st path tsp. *Journal of the ACM (JACM)*, 62(5):34, 2015.

[5] David L Applegate, Robert E Bixby, Vašek Chvátal, William Cook, Daniel G Espinoza, Marcos Goycoolea, and Keld Helsgaun. Certification of an optimal tsp tour through 85,900 cities. *Operations Research Letters*, 37(1):11–15, 2009.

[6] David L Applegate, Robert E Bixby, Vasek Chvatal, and William J Cook. *The traveling salesman problem: a computational study*. Princeton university press, 2011.

[7] Philip Avner, Thomas Bruls, Isabelle Poras, Lorraine Eley, Shahinaz Gas, Patricia Ruiz, Michael V Wiles, Rita Sousa-Nunes, Ross Kettleborough, Amer Rana, et al. A radiation hybrid transcript map of the mouse genome. *Nature genetics*, 29(2):194–200, 2001.

[8] Christopher Bailey, Timothy McLain, and Randal Beard. Fuel saving strategies for separated spacecraft interferometry. In *AIAA Guidance, Navigation, and Control Conference and Exhibit*, page 4441, 2000.

[9] LJ Bass and SR Schubert. On finding the disc of minimum radius containing a given set of points. *Mathematics of Computation*, 21(100):712–714, 1967.

[10] Mandell Bellmore and George L Nemhauser. The traveling salesman problem: a survey. *Operations Research*, 16(3):538–558, 1968.

[11] Jon Jouis Bentley. Fast algorithms for geometric traveling salesman problems. *ORSA Journal on computing*, 4(4):387–411, 1992.

[12] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B. Shah. Julia: A fresh approach to numerical computing. *SIAM Review*, 59(1):65–98, 2017.

[13] Norman Biggs, E Keith Lloyd, and Robin J Wilson. *Graph Theory, 1736-1936*. Oxford University Press, 1976.

[14] Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys (CSUR)*, 35(3):268–308, 2003.

[15] B Borah and DK Bhattacharyya. An improved sampling-based dbscan for large spatial databases. In *Intelligent Sensing and Information Processing, 2004. Proceedings of International Conference on*, pages 92–96. IEEE, 2004.

[16] Paul S Bradley, Usama M Fayyad, Cory Reina, et al. Scaling clustering algorithms to large databases. In *KDD*, pages 9–15, 1998.

[17] Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report, DTIC Document, 1976.

[18] William Cook. *In pursuit of the traveling salesman: mathematics at the limits of computation*. Princeton University Press, 2012.

[19] Georges A Croes. A method for solving traveling-salesman problems. *Operations research*, 6(6):791–812, 1958.

[20] George Dantzig, Ray Fulkerson, and Selmer Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the operations research society of America*, 2(4):393–410, 1954.

[21] George B Dantzig and John H Ramser. The truck dispatching problem. *Management science*, 6(1):80–91, 1959.

[22] Yong Deng, Yang Liu, and Deyun Zhou. An improved genetic algorithm with initial population strategy for symmetric tsp. *Mathematical Problems in Engineering*, 2015, 2015.

[23] Marco Dorigo and Gianni Di Caro. Ant colony optimization: a new meta-heuristic. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 2, pages 1470–1477. IEEE, 1999.

[24] Marco Dorigo and Luca Maria Gambardella. Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on evolutionary computation*, 1(1):53–66, 1997.

[25] Marco Dorigo and Thomas Stützle. Ant colony optimization: overview and recent advances. In *Handbook of metaheuristics*, pages 227–263. Springer, 2010.

[26] Delbert Dueck. *Affinity propagation: clustering data by passing messages.* PhD thesis, Citeseer, 2009.

[27] Delbert Dueck and Brendan J Frey. Non-metric affinity propagation for unsupervised image categorization. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8. IEEE, 2007.

[28] Ahmad Fouad El-Samak and Wesam Ashour. Optimization of traveling salesman problem using affinity propagation clustering and genetic algorithm. *Journal of Artificial Intelligence and Soft Computing Research*, 5(4):239–245, 2015.

[29] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. Clustering for mining in large spatial databases. *KI*, 12(1):18–24, 1998.

[30] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

[31] Leonhard Euler. Solution dune question curieuse qui ne paroit soumisea aucune analyse. *Mem. Acad. Sci. Berlin*, 15:310–337, 1759.

[32] Roger Fletcher. *Practical methods of optimization.* John Wiley & Sons, 2013.

[33] Merrill M Flood. The traveling-salesman problem. *Operations Research*, 4(1):61–75, 1956.

[34] Edward W Forgy. Cluster analysis of multivariate data: efficiency versus interpretability of classifications. *Biometrics*, 21:768–769, 1965.

[35] Brendan J Frey and Delbert Dueck. Clustering by passing messages between data points. *science*, 315(5814):972–976, 2007.

[36] MR Garey, D Johnson, and Hans Witsenhausen. The complexity of the generalized lloyd-max problem (corresp.). *IEEE Transactions on Information Theory*, 28(2):255–256, 1982.

[37] J William Gavett. Three heuristic rules for sequencing jobs to a single production facility. *Management Science*, 11(8):B–166, 1965.

[38] Bezalel Gavish and Stephen C Graves. The travelling salesman problem and related problems. 1978.

[39] Alan E Gelfand. Rapid seriation methods with archaeological applications. *Hodson, FR, Kendall, DG and Tautu (eds), Mathematics in the Archaeological and Historical Sciences. Edinburgh University Press, Edinburgh*, pages 186–201, 1971.

[40] Fred Glover. Future paths for integer programming and links to artificial intelligence. *Computers & operations research*, 13(5):533–549, 1986.

[41] Fred Glover. Tabu searchpart i. *ORSA Journal on computing*, 1(3):190–206, 1989.

[42] Fred Glover and Eric Taillard. A user's guide to tabu search. *Annals of operations research*, 41(1):1–28, 1993.

[43] Nicholas Gould. An introduction to algorithms for continuous optimization, 2006.

[44] Gunnar Gruvaeus and Howard Wainer. Two additions to hierarchical cluster analysis. *British Journal of Mathematical and Statistical Psychology*, 25(2):200–206, 1972.

[45] Renchu Guan, Xiaohu Shi, Maurizio Marchese, Chen Yang, and Yanchun Liang. Text clustering with seeds affinity propagation. *IEEE Transactions on Knowledge and Data Engineering*, 23(4):627–637, 2011.

[46] John A Hartigan and Manchek A Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979.

[47] Keld Helsgaun. An effective implementation of the lin–kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106–130, 2000.

[48] Mary Inaba, Naoki Katoh, and Hiroshi Imai. Applications of weighted voronoi diagrams and randomization to variance-based k-clustering. In *Proceedings of the tenth annual symposium on Computational geometry*, pages 332–339. ACM, 1994.

[49] Anil K Jain, M Narasimha Murty, and Patrick J Flynn. Data clustering: a review. *ACM computing surveys (CSUR)*, 31(3):264–323, 1999.

[50] David S Johnson and Lyle A McGeoch. The traveling salesman problem: A case study in local optimization. *Local search in combinatorial optimization*, 1:215–310, 1997.

[51] DS Johnson, G Gutin, LA McGeoch, A Yeo, W Zhang, and A Zverovitch. The traveling salesman problem and its variations. *Chapter: Experimental Analysis of Heuristics for the ATSP*, pages 445–487, 2002.

[52] DS Johnson and CH Papadimitriou. Performance guarantees for heuristics. *Lawler et al.(1985)*, pages 145–180, 1985.

[53] Michael Jünger, Gerhard Reinelt, and Giovanni Rinaldi. The traveling salesman problem. *Handbooks in operations research and management science*, 7:225–330, 1995.

[54] Robert L Karg and Gerald L Thompson. A heuristic approach to solving travelling salesman problems. *Management science*, 10(2):225–248, 1964.

[55] Scott Kirkpatrick. Optimization by simulated annealing: Quantitative studies. *Journal of statistical physics*, 34(5):975–986, 1984.

[56] Scott Kirkpatrick, C Daniel Gelatt, Mario P Vecchi, et al. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[57] Bernhard Korte, Jens Vygen, B Korte, and J Vygen. *Combinatorial optimization*, volume 2. Springer, 2012.

[58] Mary E Kurz. Heuristics for the traveling salesman problem. *Wiley Encyclopedia of Operations Research and Management Science*, 2011.

[59] Gilbert Laporte. The traveling salesman problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59(2):231–247, 1992.

[60] S Lau. Solving traveling salesman problems with heuristic learning approach. 2002.

[61] Shen Lin. Computer solutions of the traveling salesman problem. *The Bell System Technical Journal*, 44(10):2245–2269, 1965.

[62] Shen Lin and Brian W Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, 21(2):498–516, 1973.

[63] Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.

[64] Helena R Lourenço, Olivier C Martin, and Thomas Stützle. Iterated local search. In *Handbook of metaheuristics*, pages 320–353. Springer, 2003.

[65] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA., 1967.

[66] Alberto Colorni Marco Dorigo Vittorio Maniezzo. Distributed optimization by ant colonies. In *Toward a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life*, page 134. Mit Press, 1992.

[67] Clair E Miller, Albert W Tucker, and Richard A Zemlin. Integer programming formulation of traveling salesman problems. *Journal of the ACM (JACM)*, 7(4):326–329, 1960.

[68] Marcelo G Narciso and Luiz Antonio N Lorena. Lagrangean/surrogate relaxation for generalized assignment problems. *European Journal of Operational Research*, 114(1):165–177, 1999.

[69] Frank Neumann and Carsten Witt. Combinatorial optimization and computational complexity. *Bioinspired Computation in Combinatorial Optimization*, pages 9–19, 2010.

[70] Christian Nilsson. Heuristics for the traveling salesman problem. Technical report, Tech. Report, Linköping University, Sweden, 2003.

[71] Jorge Nocedal and Stephen J Wright. Numerical optimization, 2006.

[72] AJ Orman and H Paul Williams. A survey of different integer programming formulations of the travelling salesman problem. *Optimisation, Econometric and Financial Analysis*, 9:93–108, 2006.

[73] Ibrahim H Osman and James P Kelly. Meta-heuristics: an overview. In *Meta-heuristics*, pages 1–21. Springer, 1996.

[74] Christos H Papadimitriou and Kenneth Steiglitz. *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1982.

[75] Mostofa Ali Patwary, Diana Palsetia, Ankit Agrawal, Wei-keng Liao, Fredrik Manne, and Alok Choudhary. A new scalable parallel dbscan algo-

rithm using the disjoint-set data structure. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 62. IEEE Computer Society Press, 2012.

[76] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(Oct):2825–2830, 2011.

[77] Tanasanee Phienthrakul. Clustering evolutionary computation for solving travelling salesman problems. *International Journal of Advanced Computer Science and Information Technology (IJACSIT)*, 3(3):243–262, 2014.

[78] Robert Clay Prim. Shortest connection networks and some generalizations. *Bell Labs Technical Journal*, 36(6):1389–1401, 1957.

[79] César Rego, Dorabela Gamboa, Fred Glover, and Colin Osterman. Traveling salesman problem heuristics: leading methods, implementations and latest advances. *European Journal of Operational Research*, 211(3):427–441, 2011.

[80] Gerhard Reinelt. Tspliba traveling salesman problem library. *ORSA journal on computing*, 3(4):376–384, 1991.

[81] Gerhard Reinelt. *The Traveling Salesman: Computational Solutions for TSP Applications*. Springer-Verlag, Berlin, Heidelberg, 1994.

[82] Daniel J Rosenkrantz, Richard E Stearns, and Philip M Lewis, II. An analysis of several heuristics for the traveling salesman problem. *SIAM journal on computing*, 6(3):563–581, 1977.

[83] Martin WP Savelsbergh and Marc Sol. The general pickup and delivery problem. *Transportation science*, 29(1):17–29, 1995.

[84] Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer Science & Business Media, 2002.

[85] Alexander Schrijver. On the history of combinatorial optimization (till 1960). *Handbooks in operations research and management science*, 12:1–68, 2005.

[86] Thomas H Sloane, Frank Mann, and H Kaveh. Powering the last mile: an alternative to powering fitl. In *Telecommunications Energy Conference, 1997. INTELEC 97., 19th International*, pages 536–543. IEEE, 1997.

[87] Kate Smith-Miles, Jano van Hemert, and Xin Yu Lim. Understanding tsp difficulty by learning from evolved instances. In *International Conference on Learning and Intelligent Optimization*, pages 266–280. Springer, 2010.

[88] K Steiglitz and P Weiner. Some improved algorithms for computer solution of the traveling salesman problem. 1968.

[89] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. Data mining cluster analysis: basic concepts and algorithms. *Introduction to data mining*, 2013.

[90] Jano I van Hemert. Property analysis of symmetric travelling salesman problem instances acquired through evolution. In *European Conference on Evolutionary Computation in Combinatorial Optimization*, pages 122–131. Springer, 2005.

[91] David H Wolpert and William G Macready. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82, 1997.

[92] Aoying Zhou, Shuigeng Zhou, Jing Cao, Ye Fan, and Yunfa Hu. Approaches for scaling dbscan algorithm to large spatial databases. *Journal of computer science and technology*, 15(6):509–526, 2000.

[93] Hongfang Zhou, Peng Wang, and Hongyan Li. Research on adaptive parameters determination in dbscan algorithm. *JOURNAL OF INFORMATION &COMPUTATIONAL SCIENCE*, 9(7):1967–1973, 2012.