2021

ALGORITHM SELECTION FOR THE CAPACITATED VEHICLE ROUTING PROBLEM USING MACHINE LEARNING CLASSIFIERS

Justin C. Fellers
University of Rhode Island, justin.c.fellers@gmail.com

ALGORITHM SELECTION FOR THE CAPACITATED VEHICLE ROUTING PROBLEM USING MACHINE LEARNING CLASSIFIERS

BY

JUSTIN C. FELLERS

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

IN

SYSTEMS ENGINEERING

UNIVERSITY OF RHODE ISLAND

2021

MASTER OF SCIENCE THESIS

OF

JUSTIN C. FELLERS

APPROVED:

Thesis Committee:

Major Professor Manbir Sodhi

Thomas Wettergren

Jason Parent

Brenton DeBoef

DEAN OF THE GRADUATE SCHOOL

UNIVERSITY OF RHODE ISLAND

2021

ABSTRACT

The Vehicle Routing Problem (VRP) is a widely known NP-Hard operations research problem endowed with a wide range of heuristic algorithms generated from decades of global research. These heuristics provide solutions within a reasonable run time, but at some expense to optimality. The literature further suggests heuristic performance in one class of problems comes at a cost in performance to other classes. This study aimed to develop methods for selecting the best heuristic from a defined set to solve an arbitrary VRP instance.

Known as the algorithm selection problem, this study implemented supervised machine learning techniques to construct prediction models based upon instance characteristics. These models were evaluated by metrics commonly found in both algorithm selection and machine learning studies. Built from a set of 23 features and a portfolio of four varied heuristics, the leading model correctly predicted the best algorithm with 79% accuracy despite the single best heuristic occurring only 49% of the time.

Models were constructed using a custom problem space of 5,000 VRP instances developed organically by novel methods adapted from the literature. Adequacy of the problem space, regarding its range of difficulties and sufficiency of size, was also explored. The results indicate the problem space was appropriately diverse and the prediction models, which were developed by learning algorithms using provided data, are unlikely to improve accuracy if given more data.

ACKNOWLEDGMENTS

I could have never accomplished this research without the love and advocacy of my family. My wife Nicole and both of my boys gave me more patience and support than I could have asked for (or expected to need) with returning to the studies and embarking on my first significant research project.

Dr. Sodhi, my major advisor, was the linchpin in my accomplishing not only this work but my graduate studies overall. For two years he answered all my questions with grace, and encouraged my often self-questioning decision to return to school after more than a decade. My other committee members, Dr. Parent and Dr. Wettergren, also gave me the right skills, direction, and feedback to pursue my goals. Though not a formal committee member, Dr. Steinhaus of the U.S. Coast Guard Academy provided the inspiration for this project and remained engaged throughout its duration. The privilege was mine to continue some of her previous work. My peers, notably Marwan and Jose, motivated me to think critically and remain focused. For all of these experience and many more, I am grateful for my time at URI.

Lastly, I must thank the U.S. Coast Guard as an organization for reinvesting in me through my education. The service granted me the time and funding to improve myself, and I am appreciative for the opportunity.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

Introduction

This research investigates the use of supervised machine learning techniques to successfully implement the Algorithm Selection Problem on a portfolio of evolutionary and classical capacitated vehicle routing problem heuristics. By generating a sizeable problem space comprised of diverse instances, this study evaluates different methods to reduce classification loss and seeks to increase knowledge of the algorithm portfolio by analyzing the features impactful to performance.

The subsections of this chapter provide disparate, high level descriptions of the core techniques and theoretical principles applied in the study. This includes formal definitions and problem formulations for the algorithm selection problem, vehicle routing problem, and supervised machine learning techniques. The final section synthesizes these concepts as they are applied in this study and provides a basic structure for the remainder of the report.

1.1 Algorithm Selection Problem

Developed by John Rice at Purdue University in 1975, the algorithm selection problem is the process of creating a model that uses features from known problem instances to select the best performing algorithm from a portfolio of candidate algorithms [1, 2]. Notwithstanding its creation over 40 years ago, this problem remains relevant today as it provides an avenue to leverage the best parts of existing algorithms, which are already developed and tested, for diverse applications.

Defined more precisely, the algorithm selection problem is formulated by using a feature space, $\boldsymbol{F}$, to achieve an optimal performance mapping from a problem space, $\boldsymbol{P}$, to a defined algorithm space, $\boldsymbol{A}$ [3, 4]. Figure 1 shows the key components of the algorithm selection problem.

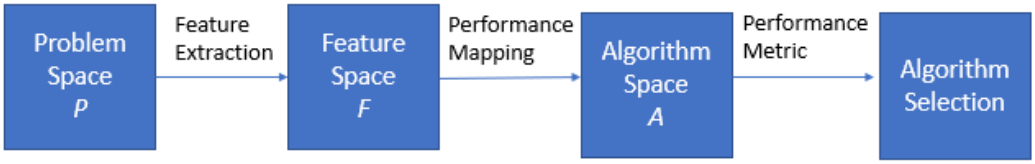

Figure 1. High level process flow for an algorithm selection problem.

Previous research documents the elemental requirements of each step in this process, which are summarized below from the work of Smith-Miles [4]:

1. Problem Space shall consist of instances of assorted difficulties;
2. Feature Space must suitably characterize instance properties;
3. Algorithm Space should contain diverse algorithms for solving instances;
4. The performance metric appropriately evaluates the algorithms' results.

For the practitioner, the first two requirements are perhaps the most challenging. Building a library of instances to constitute $\boldsymbol{P}$ requires a certain methodology to ensure it contains problems of assorted difficulties and properties. Randomly sampling instances without any structure can lead to a large dataset, but not necessarily one which is diverse. Furthermore, determining the features to form $\boldsymbol{F}$ indicates an expert-level of domain knowledge. Naively guessing features or constructing an excessively large set can not guarantee performance, and, in some cases, may degrade performance due to high dimensionality [5].

The third requirement ensures $\boldsymbol{A}$ is composed of algorithms with different solution methods. From the No Free Lunch (NFL) theorems presented by Wolpert and Macready, it is proven that "for any algorithm, any elevated performance over one class of problems is exactly paid for in performance over another class" [6, 4]. While certain algorithms can be generally inferior in all problem settings, the NFL

theorems suggest no single algorithm can be universally superior. This may be applied to heuristic solving algorithms, which often obtain a solution by making assumptions that exploit certain instance properties over others [2, 3]. The final requirement dictates the need for the performance metric to be appropriate for the desired algorithm selection. This is defined by the user and is goal-oriented. For this study, the objective is to select the algorithm which achieves the lowest cost solution. Other metrics, such as time to solution, are outside of the scope of this study.

This study implements the algorithm selection problem on the capacitated vehicle routing problem (CVRP). The next section discusses the CVRP and provides a purposeful framework for the study's importance.

1.2 Capacitated Vehicle Routing Problem

The vehicle routing problem (VRP) is one of the most highly studied operations research problems. Formulated by Dantzig and Ramser in 1959 as the Truck Dispatching Problem, the problem minimizes the cost incurred for a fleet of vehicles to service a set of spatially disparate customer demands [7]. The applicability of this problem cannot be overstated, and surely it remains one of the most important economic considerations for the distribution and service industries. Clear applications exist in the waste collection, package delivery, and road servicing sectors [8]. The VRP may also be used internally by organizations to manage operations, such as planning production lines or inventory systems with minimal waste.

Variations of this problem to satisfy a more specific goal can be found throughout the literature, and typically include additional constraints in the problem formulation. Examples may include the requirement for vehicles to service both pickup and delivery demands, known as the VRP with Pickup and Delivery, or for customer services to occur within a designated time period, known as the VRP

with Time Windows [8].

This study focuses on the symmetric variant of the Capacitated VRP (CVRP), which is largely the original problem presented by Dantzig and Ramser. Formal definitions are documented throughout the literature [3, 8, 9, 10, 11], and are presented here in similar form. Let $G = (V, A)$ be an undirected, two dimensional graph where V is equal to the set of nodes {0, 1, ..., n} and A is equal to the set of edges which connect any two nodes. Each node is located in a unique position indicated by $(x_i, y_i) \, \forall i \in V$. Each edge is represented by a cost, c_{ij}, to travel from node i to node j and is calculated as the Euclidean distance between the two nodes given by

$$c_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

Since this research considers the symmetric CVRP,

$$c_{ij} = c_{ji}$$

There exists one depot, located at node 0, and all remaining nodes represent a customer with a demand, d_i, that is greater than 0.

$$d_i > 0 \qquad \forall i \in V \setminus \{0\}$$

From the single depot, a fleet of K homogeneous vehicles, each with a service capacity of Q, travel along the edges $\in A$ to satisfy all customer demands and return to the depot position. The goal function is to minimize the fleet's cumulative incurred cost.

Mathematical formulations for the CVRP, and other variants of the VRP, are presented at length throughout the literature [3, 8, 10, 11, 12]. The CVRP formulation is as follows:

$$min\, z = \sum_{i \in V} \sum_{j \in V} c_{ij} * x_{ij} \tag{1}$$

subject to

$$\sum_{i \in V \setminus \{0\}} x_{ij} = 1 \tag{2}$$

$$\sum_{j \in V \setminus \{0\}} x_{ij} = 1 \tag{3}$$

$$\sum_{i \in V \setminus \{0\}} x_{i0} = K \tag{4}$$

$$\sum_{j \in V \setminus \{0\}} x_{0j} = K \tag{5}$$

$$\sum_{i \notin C} \sum_{j \in C} x_{ij} \geq \lambda(C) \qquad \forall C \subseteq V \setminus \{0\}, \text{where } C \neq \emptyset \tag{6}$$

$$\lambda(C) = \lceil \frac{\sum d_i}{Q} \rceil \qquad \forall i \in C, \forall C \subseteq V \tag{7}$$

$$x_{ij} \text{ binary} \qquad \forall i, j \in V \tag{8}$$

The goal function provided in equation 1 minimizes the cost to travel along the instance edges. By restricting the decision variables, x_{ij}, to binary values in equation 8, each edge is either selected or not selected in the final solution. The constraints in equations 2 and 3 ensure each customer is serviced with exactly one vehicle arrival and one vehicle departure. By setting the i and j values to 0 in equations 4 and 5, respectively, these constraints ensure that exactly K vehicles both depart from and return to the depot position. Equation 6 preserves vehicle capacity and eliminates the formation of sub tours. Where C is a subset of nodes, $\lambda(C)$ is the number of vehicles required to service the subset, and is calculated in equation 7. By requiring a greater than or equal to number of vehicles to enter the subset, it cannot become disjoint from $V \setminus \{C\}$ and is provided at least the minimum requirement of service capacity to satisfy all demands.

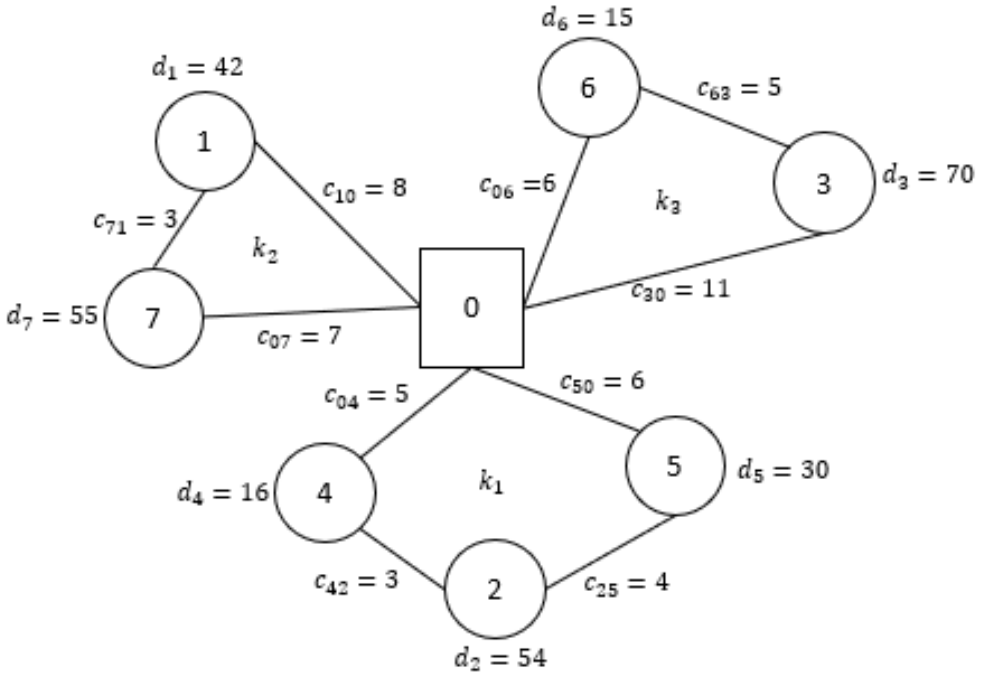

Figure 2. Graphical representation of CVRP.

Figure 2 illustrates an example CVRP with eight nodes and parameters $K = 3$, $Q = 100$. Node 0 corresponds to the depot and the remaining seven nodes represent customers with specified demand values. Each route, k_i, has a cumulative demand value ≤ 100, services its customers exactly once, and conducts a round trip from the depot. The final cost of this solution is equal to the total costs incurred by all routes, or 58. While feasible, the provided solution is not necessarily optimal.

Attaining an optimal solution to the CVRP through exact methods is an NP-Hard problem [10, 3, 8], which means no polynomial time algorithm exists for exactly solving all problems to optimality. Therefore, researchers have spent considerable resources developing heuristic algorithms to provide good solutions, sometimes producing optimal or nearly optimal results [11]. The historical record of creating algorithms for problems such as the CVRP is extensive, yet often a new algorithm's improvements can be isolated to certain instance types [2, 6, 3].

Despite its computational complexity, attaining the best solution to the CVRP remains significant for industry managers and resource planners. Applied to the CVRP, the algorithm selection problem will indicate which algorithm to use on a new instance based upon its feature values, preventing the need to compare solutions from multiple algorithms. This study considers a portfolio of four CVRP solver algorithms, two of which were designed and developed by doctoral candidates at the University of Rhode Island within the last five years, and two classical CVRP constructive heuristics. These algorithms are defined later in this report.

The algorithm selection problem as defined by Rice in 1975 has become nearly inseparable from the modern applications of machine learning [4]. The next section provides a background on the machine learning techniques used in this study.

1.3 Supervised Machine Learning

Supervised machine learning is a highly-regarded strategy for identifying patterns in data by relating instance features to instance labels. The process consists of inputting known instances to a learning algorithm for hypothesizing a feature-to-label relationship in the form of a prediction function f. Future instances are subsequently presented to f for prediction; therefore, the objective is to generate a prediction function capable of generalizing the data beyond those used in the learning algorithm [5]. Traditional computer analysis methods may include writing a program to generalize data, whereas machine learning uses data to generalize a program [13]. Though supervised machine learning can be used for regression, this study is strictly a classification problem.

The format for presenting data to a supervised learning algorithm is exact. Instances are represented by a vector of features $\vec{x}$ and a label y, and a data set D is the aggregation of n instances:

$$D = \{(\vec{x_1}, y_1), (\vec{x_2}, y_2), ..., (\vec{x_n}, y_n)\}$$

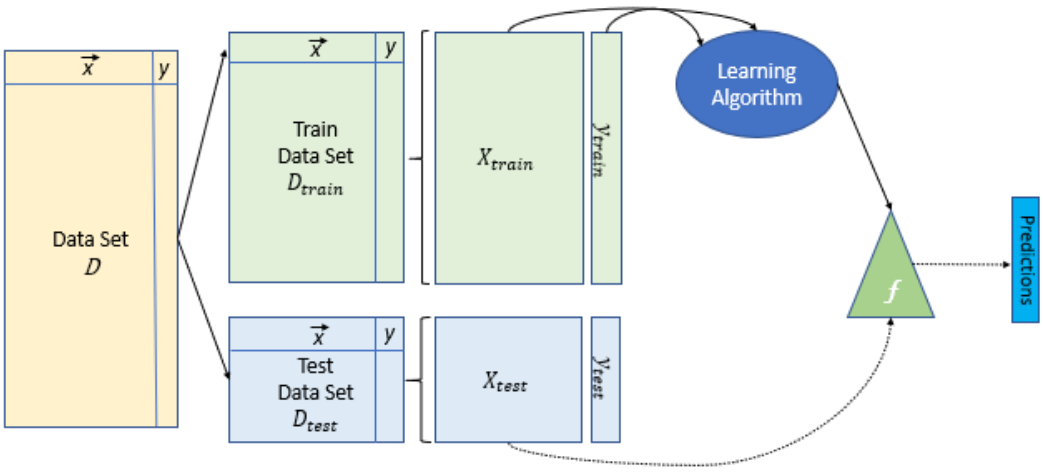

Figure 3. Data set implementation for supervised learner.

D must be organized to represent both the known instances used to train the learning algorithm and the future instances used to test the performance of f. This is achieved by dividing D into one subset for training and another for testing, and separating features from labels. This process creates the commonly named data structures of $[X_{train}, y_{train}]$ and $[X_{test}, y_{test}]$, where X is a $n \times d$ matrix of n instances with d features, and y is a column matrix of n labels [14]. This process implies two distinct partitions of D, where $[X_{train}, y_{train}] \Rightarrow D_{train}$, $[X_{test}, y_{test}] \Rightarrow D_{test}$, and $D_{train} + D_{test} \Rightarrow D$.

Figure 3 shows the process of a supervised learning technique. The learning algorithm only uses $[X_{train}, y_{train}]$ to construct f, which makes predictions on the $[X_{test}]$ instances. This diagram may be detailed further to show the iterative nature of optimizing f with respect to its loss function ℓ, achieved by modifying parameters of the learning algorithm. Designed to capture those features which are important to learn, ℓ is determined by the user based upon the learning objectives [13]. Since this study is a classification problem, the zero/one loss function is

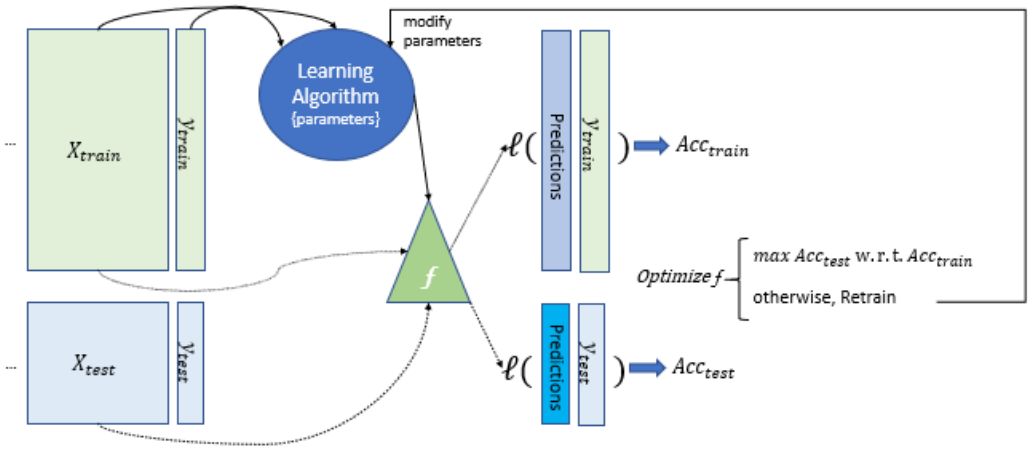

Figure 4. Classifier optimization using loss function.

appropriate and defined as follows. Where $\hat{y}$ represents a predicted y value:

$$\ell(y, \hat{y}) = \begin{cases} 0 & \text{if } y = \hat{y} \\ 1 & \text{otherwise} \end{cases}$$

Measuring the prediction accuracy Acc of f over a set of instances is equal to the average loss over the set [13]:

$$Acc_t = 1 - \frac{1}{n}\sum_{i=1}^{n}[\ell(y_i, f(x_i)] \qquad t \in \{D_{train}, D_{test}\}$$

Figure 4 shows how ℓ is used to optimize f, and leads to the introduction of overfitting and underfitting. Simply minimizing ℓ to 0 during training will likely overfit the data by memorizing D_{train}, preventing f from generalizing well [13]. Failing to minimize ℓ to any degree will likely under utilize a model's ability to learn, resulting in an underfit model. Optimality occurs where test set accuracy is maximized with respect to the accuracy of the training set. However, to preserve data integrity the test set should not be used to tune model parameters [13]. Rather, models are tuned using a subset of training data, known as a validation set, or by employing one of the various cross validation techniques.

The models used in this study are tuned using K-fold cross validation. This

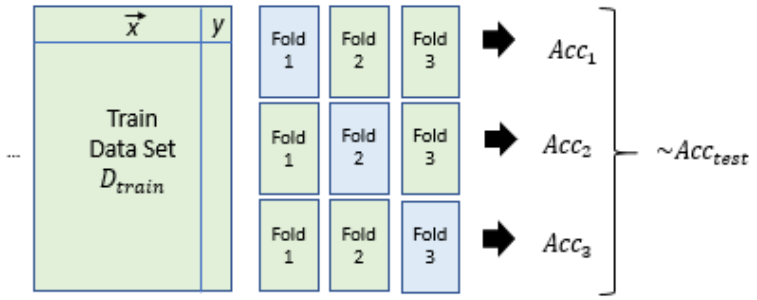

Figure 5. K-Fold cross validation with K=3.

method maximizes the number of training instances provided to the learning algorithm. Figure 5 presents a cross validation data structure with three folds. Given a learning algorithm with specified parameters, K-fold cross validation starts by partitioning D_{train} into K equally sized subsets, or folds. The learning algorithm is trained with K-1 folds and tested on the remaining fold. This process is conducted K times, with each fold serving as the test fold exactly once. The average accuracy of the test folds approximates the how well an f produced by the learning algorithm under the specified parameters will generalize new data [13]. This process is repeated for all parameter combinations in a user defined set. Using the best parameters (e.g. those which produced the highest average accuracy), the final f is produced by training a new model with all of D_{train} [13]. Final model evaluation occurs by presenting D_{test} to f for prediction.

Supervised machine learning, which optimizes a feature-to-label relationship within a given data set, draws a clear connection to the algorithm selection problem presented in 1.1. Applied to the CVRP, individual instances are represented as a vector of features and its best performing algorithm as the label. This research compares three different learning algorithms: Decision Tree, k-Nearest Neighbor, and Single-Layer Perceptron. These techniques were chosen because of their varied approaches to solve the classification problem, allowing unique insights to the algorithms' performance.

1.4 Final Model and Summary

As presented, this study implements the algorithm selection problem for the CVRP using three different supervised machine learning models. Assembling these components into a cohesive study required two primary phases of work, which are described below:

1. Phase 1
    (a) Create a problem space $\boldsymbol{P}$ of 5,000 CVRP instances;
    (b) Extract the feature space of instances $\boldsymbol{F}$ to serve as $\vec{x}$;
    (c) Map instances to algorithm solutions generated by $\boldsymbol{A}$;
    (d) Select best performing algorithm (e.g. lowest cost) to serve as y;
    (e) Constitute final data set D;
    (f) Conduct exploratory data analysis;
2. Phase 2
    (a) Construct machine learning models from D;
    (b) Evaluate each model and draw conclusions from observed performance.

Figure 6 displays these steps, along with their inputs and outputs, as completed by phase. Using a sample size of 5,000 instances for the problem space is a functional balance between the computational expense required to generate the labels, as well as the number of instance creation combinations presented in chapter 2. Examining its adequacy is explored in chapter 4 by evaluating model accuracy as a function of data set size.

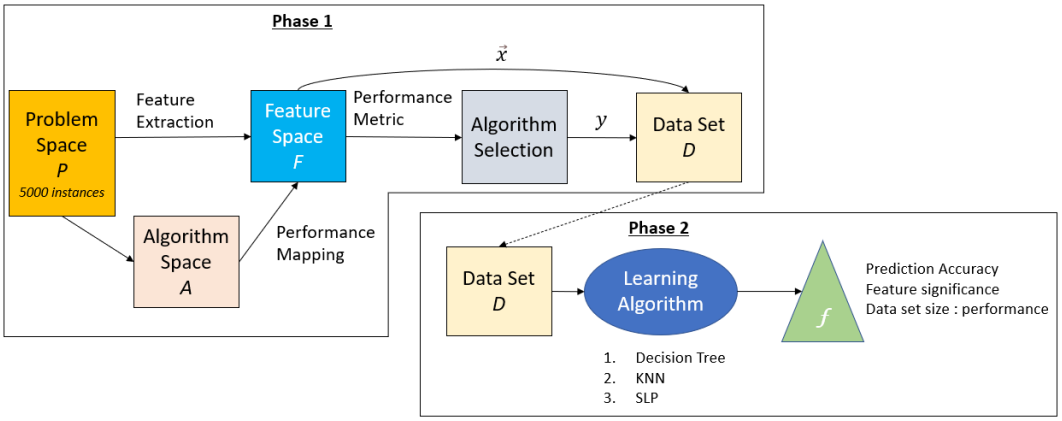

Figure 6. Research steps completed by phase.

List of References

[1] J. R. Rice, "The algorithm selection problem," Dept. of Computer Science, Purdue University, West Lafayette, IN, Tech. Rep. 75-152, 1975.

[2] L. Kothoff, "Algorithm selection for combinatorial search problems: A survey," *Artificial Intelligence Magazine*, vol. 35, pp. 48–60, 2014.

[3] M. K. Steinhaus, "The application of the self organizing map to the vehicle routing problem," Ph.D. dissertation, University of Rhode Island, 2015, paper 383.

[4] K. Smith-Miles, "Cross disciplinary perspectives on meta-learning for algorithm selection," *Association for Computing Machinery Computing Surveys*, vol. 41, pp. 6:1–25, 2008.

[5] P. Domingos, "A few useful things to know about machine learning," *Communications of the Association for Computing Machinery*, vol. 55, pp. 78–87, 2012.

[6] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computatation*, vol. 1, pp. 67–82, 1997.

[7] G. Dantzig and J. H. Ramser, "The truck dispatching problem," *Management Science*, vol. 6, pp. 80–91, 1959.

[8] L. C. Yeun, W. R. Ismail, K. Oman, and M. Zirour, "Vehicle routing problem: Models and solutions," *Journal of Quality Measurement and Analysis*, vol. 4, pp. 205–218, 2008.

[9] G. Clarke and J. Wright, "Scheduling of vehicles from a central depot to a number of delivery points," *Operations Research*, vol. 12, pp. 568–581, 1964.

[10] M. F. Abdelatti and M. S. Sodhi, "An improved gpu-accelerated heuristic technique applied to the capacitated vehicle routing problem," in *Genetic and Evolutionary Computation Conference*, July 2020.

[11] E. Uchoa, D. Pecin, A. Pessoa, A. S. Marcus Poggi, and T. Vidal, "New benchmark instances for the capacitated vehicle routing problem," *European Journal of Operational Research*, vol. 257, pp. 848–858, 2017.

[12] M. Fischetti, P. Toth, and D. Vigo, "A branch-and-bound algorithm for the capacitated vehicle routing problem on directed graphs," *Journal of Quality Measurement and Analysis*, vol. 42, pp. 846–859, 1994.

[13] H. Daume, "A course in machine learning," 2017, unpublished. [Online]. Available: http://ciml.info/

[14] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

CHAPTER 2

Algorithm Selection Problem Formulation

This chapter details the methodology for assembling the components of this study's algorithm selection problem. These elements collectively formulate the final data set explored in chapter 3 and classification models presented in chapter 4.

2.1 Problem Space

Creating the problem space occurred by constructing a library of CVRP instances which are later represented by features to train and test the classification models. For this study, the creation process was adapted from the article *New Benchmark Instances for the Capacitated Vehicle Routing Problem* by Eduardo Uchoa, et al. This paper presents methods for generating a new set of CVRP instances designed to provide a "comprehensive and balanced experimental setting" for evaluating algorithm performances [1]. Included in his work, Uchoa verified two algorithms performed differently over the team's newly created set. While this positively indicates instance diversity, this study explored a more exhaustive set of features over a larger data set, and considered a different algorithm portfolio. Due to the significance of diversity in the problem space as indicated in section 1.1, its creation is described in detail.

Originally presented on a (0,1000) $\times$ (1000,0) grid, Uchoa, et al. created a new benchmark set of 100 instances. To engender diversity into the set, the team established various sampling domains from which to select key components for any given instance. These included the depot and city positions, collectively referred to as nodes, as well as demand values, vehicle capacities, and average route lengths. Starting with 100 nodes in the first instance and gradually increasing to 1000 nodes

in the last instance, the first half of the set increased the number of nodes linearly by five, and the second half increased exponentially. Instance components were constructed by iterating random permutations of each sampling domain. Under this structure, the authors achieved a nearly equal application of instance types across a relatively small set of instances.

This study implemented the domains presented by Uchoa, et al.; however, it also introduced an additional city positioning function, modified the number of nodes per instance, reduced the grid size, and applied creation modules at random, a design more conducive for building a large set of instances. As stated, this study constructed a set of 5,000 instances, which lie in a (0,500) $\times$ (500,0) grid and are bound between 20 and 200 nodes.

Implemented in the Python programming language using the ArcGIS module *arcpy*, instances were generated in the 2-D Euclidean plane with CVRP parameters found throughout the literature and documented in section 1.2.

2.1.1 Creation Methodology

For each of the 5,000 instances, a random number between 20 and 200 (stepped by five) was chosen as n, the number of total nodes including the depot. Next, one equally likely function was selected to choose the depot position:

1. 'Random (R)': chooses any random point;
2. 'Eccentric (E)': chooses point (0,0);
3. 'Central (C)': chooses point (250,250).

Then, n - 1 points were selected as cities using one of the below functions:

1. 'Random (R)': chooses all points at random;

2. 'Cluster (C)': creates a set S of random length from a discrete uniform distribution U[3,8] and positions S random seed points throughout the grid. Where P contains all remaining points of the grid and is equal to length n-1-S, the elements of P are assigned a probability of selection based upon their proximity to the elements of S. Where $d(p, s)$ equals the Euclidean distance between point p and seed s, this probability is equal to:

$$\sum_{s \in S} exp^{\frac{-d(p,s)}{40}} \qquad \forall p \in P$$

Through experimentation, Uchoa et al. selected this equation to avoid clusters which are "bridged" together or extremely dense [1]. Evaluating other equation values was outside the scope of this study.

3. 'Random-Cluster (Rc)': first chooses half of the points at random, then the other half under the cluster method,

4. 'Equidistant (E)': divides the point grid into 5 rows $\times$ $n/5$ columns and chooses the centroid of each rectangle as the cities. The point closest to the depot drops from the instance to preserve a more uniform set of distances among locations. Not included in the work of Uchoa, et al., this module was designed to challenge algorithms with instances involving a lower fraction of distinct distances.

Demand values were then assigned to each city from one of seven functions:

1. 'Unitary (U)': assigns a value of 1 to all cities;

2. 'Small values, large variance (Svlr)': assigns demands from a uniform distribution U[1,10];

3. 'Small values, small variance (Svsr)': assigns from U[5,10];

4. 'Large values, large variance (Lvlr)': U[1, 100];

5. 'Large values, small variance (Lvsr)': U[50,100];

6. 'Quadrant (Q)':

   (a) if city is in Q2 or Q4 it receives demand from U[1,50],

   (b) if city is in Q1 or Q3 it receives demand from U[51,100];

7. 'Many small values, few large values (Msfl)': 70-95% of cities are assigned demand from U[1,10], the remaining are assigned from U[50,100].

Lastly, the instance's average route length, capacity, and minimum number of vehicles were calculated. Average route length, or the average number of cities serviced per vehicle, was determined by r, a random number from a continuous triangular distribution T[3,6,25]. To determine the vehicular capacity Q, the value r was multiplied by the instance's cumulative demand and divided by the number of cities, with the resultant rounded up to the nearest integer. Where the cities are represented by the set C, this operation was computed by

$$Q = \lceil \frac{r * \sum_{c \in C} d_c}{|C|} \rceil$$

The minimum number of vehicles $MinVeh$ was then calculated by dividing the cumulative demand by vehicle capacity, also rounded up to the nearest integer

$$MinVeh = \lceil \frac{\sum_{c \in C} d_c}{Q} \rceil$$

Each instance was written to its own text file, formatted similarly to existing CVRP benchmark instances, and captured all data necessary to be read-in and solved by the algorithm portfolio, including the node coordinates, demand values, and vehicular capacity. Instance files were titled with a unique identification number, as well as the number of nodes, minimum number of vehicles, and a sequential

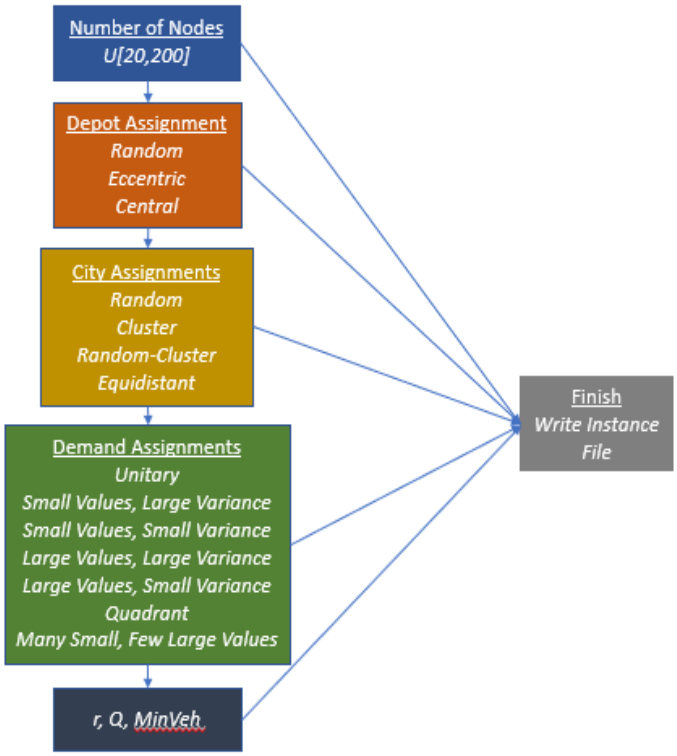

Figure 7. CVRP creation process.

concatenation of the depot/city/demand functions used in its creation. For example, consider the first instance created and its file name 'instance0-n75-k10-REQ'. This title readily shows the number of nodes as 75 and minimum number of vehicles as 10. It further indicates the use of the 'Random' depot, 'Equidistant' city, and 'Quadrant' demand functions.

Figure 7 shows the design of creating an instance. Results from each step influence the next to ensure feasibility of a given CVRP instance. For example, the depot position is first selected from the grid to prevent a city from occupying the same space. Similarly, demand assignments must precede the Q and $MinVeh$ computations.

These functions engineer unique instances from assorted domains that diversify spatial characteristics, demand requirements, and vehicular capacities. As presented, the structure affords 84 combinations of depot, city, and demand as-

signment modules, each containing further organic variability. Creating a comprehensive problem space representative of every combinatorial option presented by all parameters would be prohibitively immense [1]. Otherwise, to the purpose of this study, the need for machine learning classification could be replaced by exhaustive domain analysis, in which every combination is assigned its best performing algorithm. Instead, this study aims to generate a sufficiently sized problem space to provide the classification algorithms with adequate data for generalizing the domain. Evaluating the impact of the problem space size on classification accuracy is discussed in chapter 4.

The 84 combinations, or instance types, simply indicate the functions used to create a given instance by the three middle steps presented in Figure 7. For example, the instance type 'RCQ' signifies the depot is positioned by 'Random', cities are positioned by 'Cluster', and demands are assigned by 'Quadrant'. By creating 5,000 instances, the instance type is a discrete random variable and should be Uniformly distributed in the problem space with a mean of 60.

2.1.2 Results

Creation of the 5,000 instance problem space required approximately five days of processing time on a Dell XPS-13 laptop with 16 GB of RAM and two cores. By using *arcpy* to generate the instances as points of geometry with assigned attributes, results were immediately viewable in ArcMap. This study used ArcMap 10.7.1, which is the central desktop application used in ArcGIS to display and manipulate data in a visual format [2], and became an invaluable asset for visually inspecting module results and confirming accurate text file exportation.

Figure 8 shows an instance visualization made possible through ArcMap. By inspection it can be seen the depot is positioned at 'Random' (record 0 of the attribute table corresponds to node 0 of the instance), the cities appear to be spa-

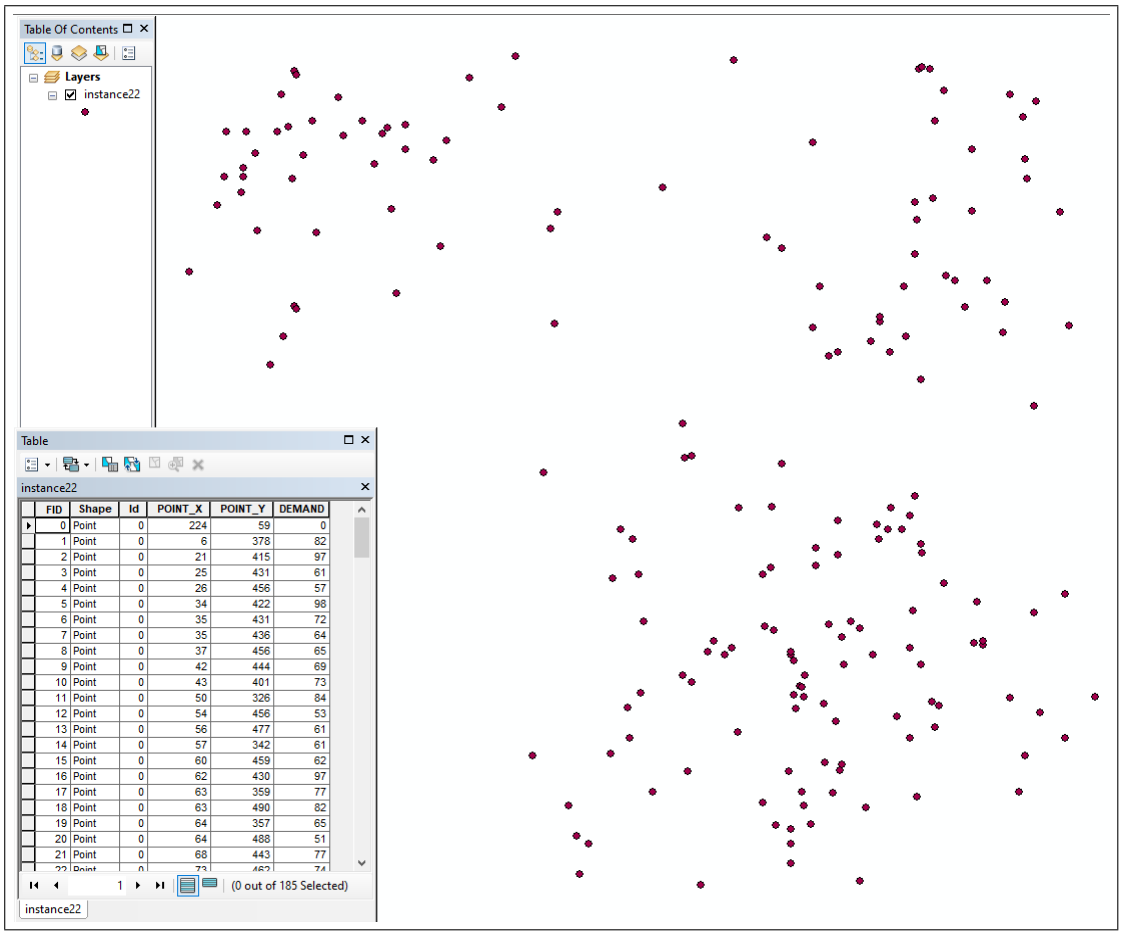

Figure 8. Instance 22: Created with 185 nodes, 'Random' depot, 'Cluster' cities, and 'Quadrant' demands.

tially positioned by 'Cluster', and, by considering city X/Y coordinates, demands are consistent with the 'Quadrant' module.

Beyond visual inspections, the creation methods were also verified with quantitative analyses. First, instances were investigated for feasibility despite the creation process' aggregate variation from pipelining modules. This included comparing each instance's computed capacity Q to its city demands d_i, with any instances containing an insufficient Q deemed infeasible. Next, the actual frequency of instance types and average route length values were compared to their expected values.

Comparing each instance's Q to its maximum city demand revealed 103 infea-

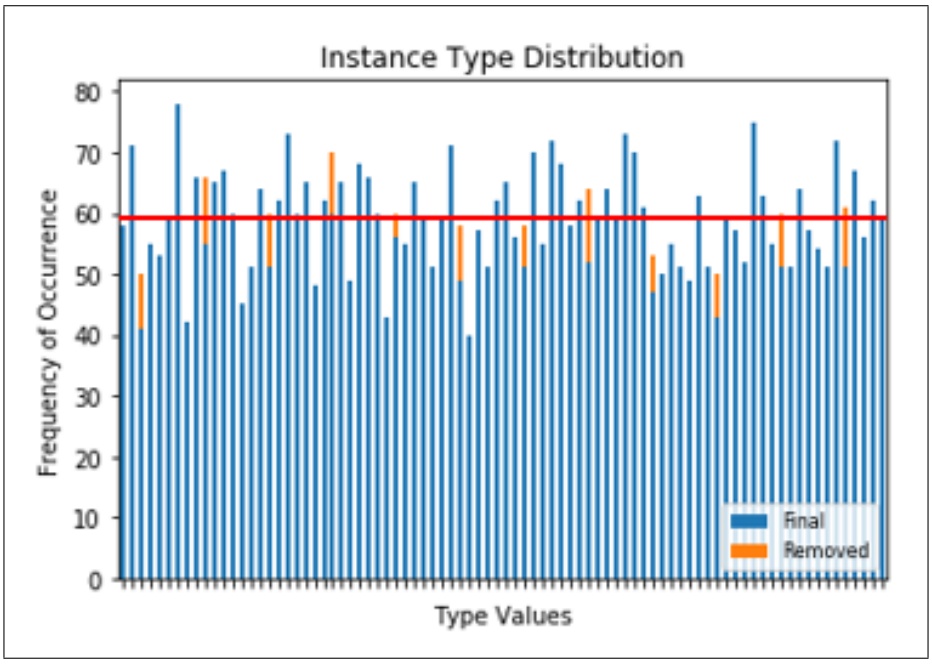

Figure 9. Bar plot of Instance Types

sible instance creations (e.g. where Q was less than the maximum demand value). Upon further investigation, all instances originated from the 'Many small values, few large values' (Msfl) demand module. Recall this module assigns 70-90% of the demands from U[50,100] and the remainder from U[1,10]. When coupled with an insufficiently small r value as presented in section 2.1.1, the calculated Q becomes less than the highest city demand. Consider the following example for an instance with 25 nodes. Under the 'Msfl' module, the largest single city demand maybe be 77 and the cumulative demand may be 240. If assigned an r value of 5.65, the resultant capacity Q is calculated to 57. Since the vast majority of instances with 'Msfl' demands were feasible (approximately seven to one), this study chose to discard the 103 infeasible instances and continue with the remaining 4,897 instances as the problem space.

The distribution of instance types found in the problem space can indicate if the creation process or removal of infeasible instances introduced a bias. Figure

9 shows the distribution of each instance type in the problem space, where the x-axis represents the 84 instance type values, and the y-axis indicates its frequency of occurrence. The blue bars show the count of each type in the final problem space, and the orange bars indicate those removed due to infeasibility. Though not precisely Uniform with an average value of 60, as shown by the red line, the distribution does reflect results consistent with random sampling. The identical spacing of the orange bars is due to alphabetically ordering the type values along the x-axis, and all infeasible instances contain the 'Msfl' identifier. While instance types with the 'Msfl' are indeed represented less frequently in the problem space, this is independent of the depot and city assignment modules. Despite removing the infeasible instances, the underlying distribution is reasonably preserved. By the Law of Large Numbers, this distribution would improve its uniformity as the number of instances increases; however, this may not be necessary to machine learning algorithms to generalize the instance domain [3]. The complete list of instance types and their values represented in Figure 9 is provided in the appendix.

Lastly, comparing the desired and actual average route lengths can further indicate if the creation process performed as designed. Sampled from a continuous triangular distribution T[3,6,25], the desired average route length (the r variable discussed in section 2.1.1) was thoughtfully designed by Uchoa, et al. to diversify the average number of cities served per vehicle in any given instance, independent of other characteristics [1]. Although, an instance's actual average route length is calculated by dividing the number of cities by the minimum number of vehicles. Therefore, interplay among the r, the assigned demand values, and the number of cities may result in the actual value deviating from the desired value. The decision to employ this specific distribution is not explicitly discussed by Uchoa, et al. However, the distribution of average route lengths from the original CVRP

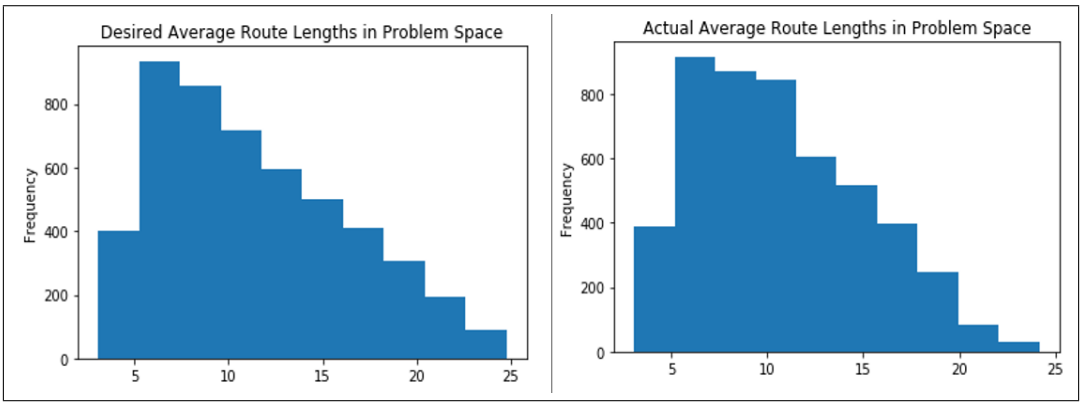

Figure 10. Desired vs. Actual r-values

benchmark problems was plotted by Steinhaus, and appears similar to the T[3,6,25] distribution [4]. Experimentation with other distributions was not included as a part of this study. Figure 10 shows side-by-side histograms of the desired r values on the left and the actual r on the right. Though the actual values do vary from the desired, the underlying shape of the distribution remains unchanged and suggests the creation process performed as designed.

The next section describes the feature space used to define the problem space in the classification models.

2.2 Feature Space

The instances' raw form, as described in section 2.1.1, is not conducive for classification. As discussed in chapter 1, each instance must be processed into a vector of features that suitably represents its unique characteristics. The features chosen to represent the problem space to the machine learning algorithms is defined as the feature space [5, 4].

Developing the feature space is arguably the most important factor to ensure the classification algorithms can actually learn from the data [3]. Additionally, instances should be characterized by the minimal amount of appropriate features. Constructing an excessively large feature space does not guarantee performance,

and, in some cases, may degrade performance due to "the curse of dimensionality", an idiom which indicates that machine learning becomes exponentially harder as the number of dimensions increases [3, 6].

While there are several approaches to developing a feature space for algorithm selection, this study considered a set of static features. In doing so, the machine learning algorithms will learn from previously solved instances to predict the best performing algorithm on future, or unseen, instances [7]. This work considered 23 features in the feature space found from the literature.

The feature space used in this research derives from multiple studies, including the work of Steinhaus, who researched the classification effectiveness of the Self-Organizing Map (SOM) on a set of 200 CVRP instances [8, 9, 10, 4]. These features largely describe the instance size, spatial attributes, and vehicle requirements, and are summarized in the following list along with their data type:

1. **Number of Cities**: Number of cities in the instance, not including the depot (integer value).

2. **Standard Deviation of Distance Matrix**: The standard deviation is extracted from the instance's distance matrix, which is symmetrical and indicates the Euclidean distance between every two nodes, including the depot (float value).

3. **X Coordinate of Instance Centroid**: X coordinate for the centroid of the rectangular plane generated by the city locations, excluding the depot (float value).

4. **Y Coordinate of Instance Centroid**: Y coordinate for the centroid of the rectangular plane generated by the city locations, excluding the depot (float value).

5. **Radius of Instance**: Average distance between city and instance centroid (float value).

6. **Fraction of Distinct Values in Distance Matrix**: Of all values in the distance matrix, including the depot, this is the percentage of those which are unique (float value).

7. **Standard Deviation of Nearest Neighbor (NN) Distances**: The standard deviation of the nodes' NN distances, including the depot (float value).

8. **Coefficient of Variation of the NN Distances**: The standard deviation of the NN distances divided by the mean of the NN distances, including the depot (float value).

9. **Ratio of Clusters to Cities**: The number of clusters divided by the number of cities. The number of clusters is solved using the DBSCAN algorithm and is described later in this section (float value).

10. **Ratio of Outlier Cities to Cities**: The number of cities not inclusive of a cluster divided by the number of cities (float value).

11. **Ratio of Edge Cities to Cities**: The number of cities on the boundary edge of a cluster divided by the number of cities (float value).

12. **Number of Clusters**: The number of clusters as indicated by DBSCAN, not including the depot (integer value).

13. **Mean Cluster Radius**: The average distance of each city from its assigned cluster centroid, divided by the number of clusters (float value).

14. **X Coordinate of Depot**: X coordinate of depot location (integer value).

15. **Y Coordinate of Depot**: Y coordinate of depot location (integer value).

16. **Standard Deviation of Demand**: The standard deviation of all city demands, represented as a proportion of the vehicle capacity (float value).

17. **Ratio of Total Demand to Total Capacity**: The cumulative demand of all cities divided by the total capacity of the fleet, assuming the minimum number of vehicles are used (float value).

18. **Ratio of Maximum Cluster Demand to Vehicle Capacity**: The largest cumulative cluster demand divided by vehicle capacity (float value).

19. **Ratio of Outlier Demand to Total Demand**: The cumulative demand of all outlier cities divided by the total instance demand (float value).

20. **Ratio of Maximum City Demand to Vehicle Capacity**: The largest city demand divided by the vehicle capacity (float value).

21. **Average Route Length**: The average number of cities serviced by a single vehicle (float value).

22. **Area of Instance**: The area of the rectangle in which all nodes are positioned, including the depot (float value).

23. **Minimum Number of Vehicles Required**: Ceiling of the instance's cumulative demand divided by vehicular capacity (integer value).

2.2.1 Clustering Methodology

Determining how to solve for the number clusters in a data set is inherently subjective, and requires two major considerations. First, the best algorithm to conduct the clustering must be chosen. This is derived from understanding the spatial domain of the data and evaluating the research goals (e.g. cluster outliers vs. do not cluster outliers). Once chosen, the algorithm's parameters must be

properly tuned to ensure insightful results. This may be conducted through experimentation with different settings and/or reviewing the literature of previous applications.

This study performed clustering with the Density-based Spatial Clustering with Applications and Noise (DBSCAN) algorithm. This algorithm is one of the few that performs well on clusters of various shapes/sizes and does not cluster outliers [11]. As implemented through the Python library *sklearn*, the algorithm requires two user-defined parameters: epsilon (*eps*) and minimum samples (*min_samples*). *eps* is defined as the minimum distance between points to be considered in the same cluster, and *min_samples* prescribes the minimum number of points required to account for a cluster [11].

The algorithm produces two outputs: the first is a list assigning each point as an outlier or to a cluster, and the other is a list of all "core samples", or those points containing *min_samples* points within a distance of *eps* [11, 12, 4]. Those points not identified as an outlier or a core sample are considered border, or edge, points. Collectively, these outputs enable the inclusion of features 9, 10, 11, 12, 13, 18, and 19 in the feature space. With a problem space of 4,987 instances, this study consulted the DBSCAN literature to establish methdods for automatically selecting the best parameters for each CVRP instance.

The algorithm's authors provide insightful guidance how to tune DBSCAN for points in the two-dimensional plane. The developers' inaugural experiments indicated the performance of their algorithm did not improve with a *min_samples* greater than four [12]. Values between one and three may be considered, though a value of one would produce trivial results as all points would constitute a unique cluster. Accordingly, this study set the $min\_samples = 4$ for all instances. The authors recommend selecting the last parameter, *eps*, by visually inspecting the

instance's "sorted k-dist graph", which is a plot sorted in descending order by the points' distance to its k^{th} neighbor [12, 4]. That value corresponding to the first elbow in the graph may serve as the instance's *eps*. While effective for a limited number of instances, this study instead turned to the novel approach for automation presented by Steinhaus, who experimented with four different procedures on a CVRP benchmark set of 102 instances [4]. This study utilized the automation method deemed most appropriate for the VRP, and consisted of the following steps [4]:

1. Generate the range $[median, 85^{th}\text{percentile}]$ of all k-distance values for k=4;
2. Discretize the range into λ equally spaced values where $\lambda = 10$, resulting in the list V;
3. Run DBSCAN with parameters $min\_samples = 4$ and $eps = v$, $\forall\, v \in V$, and record the number of clusters found to list C;
4. Find mode of C, if there is more than one mode return to step 2 and discretize into 2λ values;
5. Calculate the median of all v resulting in the mode value of C, and run final DBSCAN using this value as *eps* and $min\_samples = 4$.

By automating these steps across the problem space, each instance configured its own density threshold to identify clusters, a practice consistent with the purpose of DBSCAN [12, 4]. Alternative methods, such as applying identical *eps* values across all instances or considering a range of *eps* values around the mean k-distance value, degrade performance by limiting instance-specific density measures or increasing vulnerability to extreme outlier points [4]. Figure 11 shows an arbitrary instance created by this study with its DBSCAN generated clusters viewed through ArcMap.

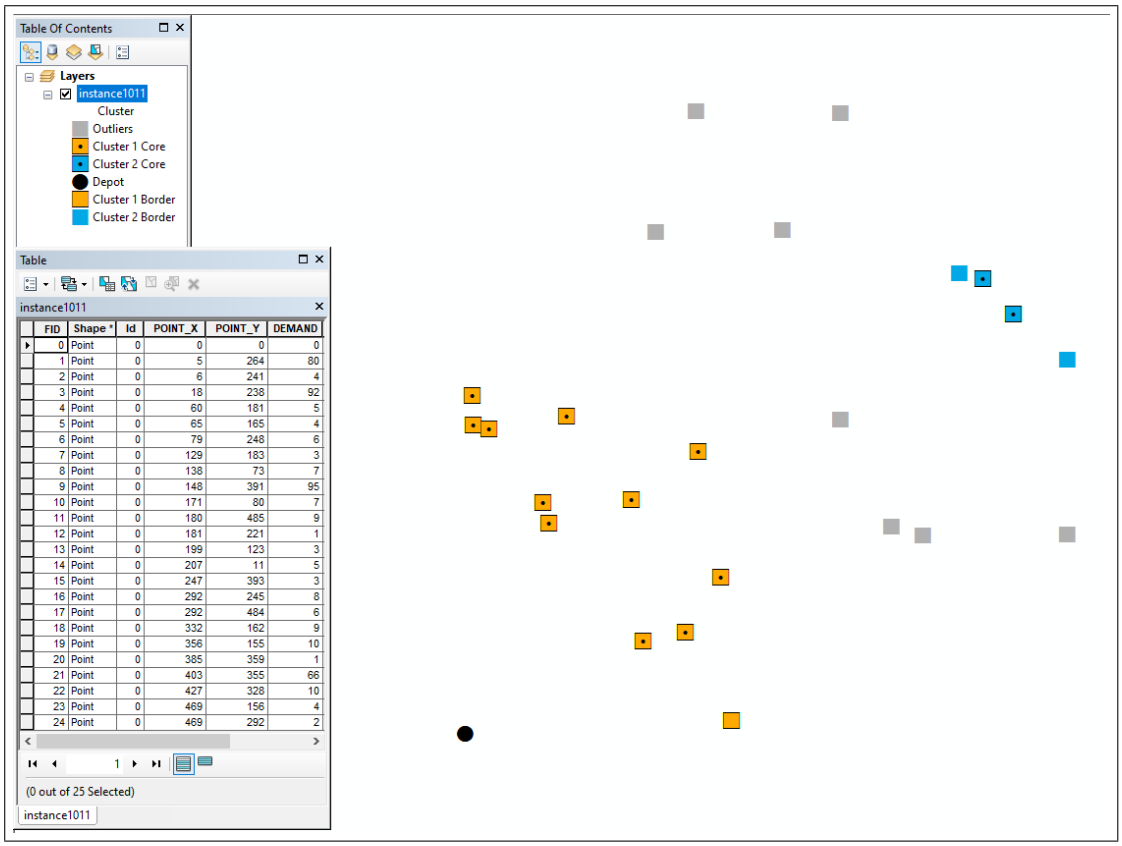

Figure 11. Instance1101 cluster results from DBSCAN. The orange and blue points represent the cluster identifications of this instance's cities, while the gray points indicate outliers and the black circle corresponds to the depot position.

Consistent with Figure 6, the 23 features discussed above were extracted from all instances in the problem space to constitute their $\vec{x}$ in the study's data set. The aggregation of all instance vectors resulted in matrix X of dimension 4897×23.

2.3 Algorithm Space

Heuristic solving algorithms for the CVRP, as well as other combinatorial search problems, often obtain a solution by making assumptions that exploit certain instance properties over others [7]. This is known formally from the No Free Lunch theorems presented by Wolpert and Macready, which prove an algorithm's performance in one class of problems comes at an expense in those of another [13, 5, 7, 4, 3]. To produce a meaningful algorithm selection problem, the heuris-

tics which constitute the algorithm space must generate solutions with dissimilar methods.

This study's algorithm space contained a static portfolio comprised of four CVRP solution heuristics. As a static portfolio, its assembly occurred prior to solving any of the instances and its composition remained unchanged throughout the study, including the values of any applicable initialization parameters [7]. All CVRP instances were solved with each algorithm.

2.3.1 Algorithm Portfolio

The algorithm portfolio for this study consisted of two classical heuristics and two evolutionary algorithms, or metaheuristics. Classical heuristics, such as the ones considered in this study, are known to produce quality solutions quickly despite considering a relatively small amount of the solution space [14]. Metaheuristics explore the solution space at much greater depth than classical heuristics; however, they are more difficult to implement, demand greater run time, and require parametric tuning [14]. Modestly put, classical heuristics are simple yet effective and metaheuristics are powerful yet complicated.

The classical heuristics used in this study are the Sweep with 2-Opt and the Clarke and Wright Savings algorithms. The evolutionary algorithms include novel implementations of a GPU genetic algorithm and a SOM. These algorithms were implemented in the Python programming language by past and present doctoral researchers at the University of Rhode Island [15, 4]. The following sections discuss the algorithms' histories and solution methods.

2.3.2 Classical Heuristics

The classical heuristics used in this study are deterministic. Since there are no parameters in the solution methods, these algorithms would, if tasked to resolve

the problem space, provide the same solutions as the ones recorded by this study. These algorithms provide reliable solutions with ordinary computing hardware, and contain historical merit with CVRP benchmark problems [4]. Though run time was not formally considered by this study, both the Clarke and Wright and Sweep algorithms took approximately three days of computational time on a Dell XPS-13 with 16GB of RAM and two cores.

Sweep Algorithm with 2-Opt Improvement Heuristic

Introduced by Gillet and Miller in 1974, the Sweep algorithm is classified as a two-phase heuristic. First, the algorithm formulates cities into clusters based upon a greedy distance-based approach of "rotating a ray centered at the depot" [14]. Each cluster must be serviceable by a single route. Then, the algorithm optimizes routes as individual Travelling Salesman Problems (TSP), using either exact or approximate methods [16, 14, 4, 17]. The 2-Opt improvement heuristic is a search method to improve routes by removing two edges of the solution and evaluating the efficacy of alternate connections [18, 14]. For this study, the best improvement was implemented to the solution. While variants of this algorithm are present throughout the literature, the solution procedures used by this study are outlined by the following steps [14, 4, 17]:

1. Scale all city (x,y) coordinates with respect to the depot;

2. Convert city coordinates from Cartesian (x,y) to polar (θ, r) by the following equations:

$$r = \sqrt{x^2 + y^2} \qquad \theta = \arctan(\tfrac{y}{x})$$

3. Order cities in a list by increasing θ values;

4. Starting with the city at the top of the list, construct K routes:

(a) Add city to route k if capacity constraint is not violated. Otherwise, continue to next city in the list,

(b) Once no more cities can be added to route k, generate new route $k+1$ and repeat;

5. Perform 2-Opt improvement heuristic on all K routes and record final solution to list S;

6. Return to step 4 and restart with next city in the list. Repeat until all cities have served as the starting city;

7. Report the best solution of list S.

Those instances where the Sweep algorithm with 2-Opt improvement produced the best solution from the algorithm space received the categorical label SP.

Clarke and Wright Savings Algorithm

The Clarke and Wright Savings algorithm is a well known constructive heuristic developed in 1964 by its eponymous authors [19, 14, 4]. This study used the parallel version of the algorithm, which combines and replaces locally optimum routes without violating the instance's capacity constraint. The solution procedures used in this study are outlined by the following steps [14, 4]:

1. Where n = number of cities in the instance, generate a list of n feasible routes by connecting every city to the depot in the form:

$$(0, i, 0) \qquad \forall\, i \in n$$

2. For any two routes i and j compute the savings, s_{ij}, recovered if the routes can be merged, and record to a list in decreasing order:

$$s_{ij} = c_{i0} + c_{0j} - c_{ij}$$

3. Starting with the savings at the top of the savings list:

   (a) If the addition of demand j does not violate the vehicle capacity, merge routes $(0, i, 0)$ and $(0, j, 0)$ into route $(0, i, j, 0)$ and remove routes $(0, i, 0)$ and $(0, j, 0)$. Continue to next s_{ij} in list;

   (b) If the addition of demand j violates the vehicle capacity, skip to next s_{ij} in list;

   (c) The route is finalized once no feasible merges remain due to the vehicle constraint;

4. Return to step 3 and restart with the next s_{ij} in the list until all cities cannot be feasibly merged to the current route.

5. Report the solution.

Those instances where the Clarke and Wright algorithm resulted in the best solution from the algorithm space received the categorical label CW.

2.3.3 Metaheuristics

The metaheuristic algorithms used in this study are inherently more complex than those methods described in the previous section. Both the genetic algorithm and the SOM contain an account of parameters. This research applied each algorithms' recommended settings from the literature based upon previous performance with CVRP benchmark instances [15, 4]. Select parameters, namely those governing the number of search iterations per instance, were chosen to compliment the portfolio holistically. Unlike the classical heuristics, these algorithms may provide different solutions to the problem space if tasked to resolve all instances, irregardless of whether the parameters are altered.

Given their advanced search methods, these algorithms utilize considerably more computational resources. The genetic algorithm applied in this study was

implemented by Abdelatti on a NVIDIA 2080 Ti GPU with 11GB of global memory and 4,352 CUDA cores. Solving the problem space consumed approximately 12 weeks of computational time. The SOM implemented by Steinhaus took approximately four weeks of computing time on a Dell PowerEdge R7425 with 512 GB of RAM and 128 cores.

Self-Organizing Map

Originally formulated by Teuvo Kohonen in 1982, the SOM was created as an artificial neural network to detect geometric patterns in data [4, 20, 21]. The algorithm behaved similarly to the Elastic Net algorithm, which was used to solve the TSP by initializing a band of neurons, called petaloids, that compete for city assignment through the reduction of some loss function [22, 4]. Later, the SOM was applied to the CVRP by incorporating the vehicle capacity and city demand constraints [23, 24, 25, 26, 27, 28]. This study applied the novel implementation by Steinhaus that incorporates a revised bias term and automates parameter control across the problem space through Fuzzy Logic [4, 29]. A complete review of the algorithm used by this study is available in [4], though the major steps with this study's defined parameters are as follows:

1. Where D = total instance demand and Q = instance vehicle capacity, randomly initialize k petaloids in the instance space, where $k = \lceil \frac{D}{Q} \rceil$;

    (a) Order all nodes (cities and depot) randomly, letting i serve as the index of length N;

    (b) Present node X_i to the network. All neurons not assigned to a city compete to win node X_i based on a combination of Euclidean distance and the bias term for vehicle capacity. If X_i is the depot, a winning neuron on each petaloid is assigned;

(c) Update petaloid structures commensurate with winning neuron assignment and return to step 1a until all nodes $i \in N$ are assigned;

(d) Record solution to list S;

2. Repeat step 1 for R replications. Report the best solution of list S;

(a) If there are no feasible solutions in S due to violated capacity constraints, return to step 1 and set $k = k + 1$.

All instances were initialized with identical parameter values and this study set $R = 50$. In Steinhaus's earlier work $R = 100$; however, this led to the SOM dominating a set of 200 CVRP instances by achieving the best solution nearly 75% of the time when compared to the CW and SP algorithms.

While appropriate in a competition-based setting where the goal is to prove one algorithm over another, tuning a metaheuristic into dominant form is self-defeating for the formulation of an algorithm selection problem. Instead, the number of replications selected for this study was designed for the algorithms to complement one another across the problem space, a practice consistent with the literature [30]. Those instances where the SOM algorithm resulted in the best solution received the categorical label SOM.

Accelerated Genetic Algorithm

Made famous by John Holland in 1975, the Genetic Algorithm (GA) uses nature-inspired evolutionary procedures to find good solutions by evolving new solutions from a population of candidate solutions [15, 4, 31]. The algorithm's evolutionary procedures are referred to as *selection*, *crossover*, and *mutation*, the number of candidate solutions is called population size (or $popSize$), and the number of evolutionary processes is called *generations* [15].

The GA is a computationally expensive algorithm typically consisting of long

run times; however, this study applied the novel GPU implementation by Abdellati, which executes operations in parallel extremely quickly [15]. The work presented in this report constitutes the largest algorithm selection problem for the CVRP inclusive of a GA. A complete review of the GA used in this study is available in [15], with the major steps as follows:

1. Initialize random *popSize* feasible solutions with respect to the vehicle capacity constraint;
2. For G generations:
    (a) Conduct *selection* of parent solutions (the two lowest-cost solutions from a random sample of four candidate solutions);
    (b) Perform 1-point *crossover* from the two parent solutions to create two child solutions;
    (c) Apply *mutation* to child solutions by swapping two random node locations with some probability and readjust depot positioning to maintain feasibility;
    (d) Conduct 2-opt heuristic to children solutions and recalculate final cost of children solutions;
    (e) If a child solution is lower than a parent solution, child solution enters population of candidate solutions
    (f) Through "adopted elitism", bear the top 5% of solutions from the old population to the new population
    (g) Replace duplicate solutions with previous parent and child solutions
    (h) Record the best solution from the generation to list
3. Report the best solution from all generations

All instances were initialized with identical parameter values and this study set G=5000 and $popSize$=100. Abdellati's earlier work achieved impressive results on benchmark problems ranging from 15-75 cities, though had yet to be directly compared to other heuristics on larger problems. Selecting G and $popSize$ for this study was an element of research: larger values may lead to performance domination and lower values may result in non-competitive results. Those instances where the GA algorithm resulted in the best solution received the categorical label GA.

2.4 Performance Metric and Algorithm Mapping

The final component of the algorithm selection problem requires an appropriate performance metric to discern the best performance. This study selects the algorithm achieving the lowest solution to the CVRP cost function, or total distance travelled by all vehicles. Other metrics, such as run time, were not considered in this study. As discussed in section 2.3, of the four algorithm solutions attained for each instance, that which produced the lowest solution earned the instance's label. Consistent with Figure 6, an instance is mapped to its best performing algorithm in the form of its y label in the final data set. The aggregation of all instance labels resulted in a column matrix y of dimension 4897×1. Of note, only 16 instances had multiple best performing algorithms.

List of References

[1] E. Uchoa, D. Pecin, A. Pessoa, A. S. Marcus Poggi, and T. Vidal, "New benchmark instances for the capacitated vehicle routing problem," *European Journal of Operational Research*, vol. 257, pp. 848–858, 2017.

[2] *What is ArcMap?*, Environmental Science Research Institute, jan 2016.

[3] P. Domingos, "A few useful things to know about machine learning," *Communications of the Association for Computing Machinery*, vol. 55, pp. 78–87, 2012.

[4] M. K. Steinhaus, "The application of the self organizing map to the vehicle routing problem," Ph.D. dissertation, University of Rhode Island, 2015, paper 383.

[5] K. Smith-Miles, "Cross disciplinary perspectives on meta-learning for algorithm selection," *Association for Computing Machinery Computing Surveys*, vol. 41, pp. 6:1–25, 2008.

[6] H. Daume, "A course in machine learning," 2017, unpublished. [Online]. Available: http://ciml.info/

[7] L. Kothoff, "Algorithm selection for combinatorial search problems: A survey," *Artificial Intelligence Magazine*, vol. 35, pp. 48–60, 2014.

[8] L. I. B. Dilek Tuzun, Michael A. Magent, "Selection of vehicle routing heuristic using neural networks," *International Transactions in Operational Research*, vol. 4, pp. 211–221, 1997.

[9] N. K. Kendall E. Nygard, Paul Juell, "Neural networks for selective vehicle routing heuristics," *ORSA Journal on Computing*, vol. 2, pp. 353–364, 1990.

[10] K. Smith-Miles, J. van Hemert, and X. Y. Lim, "Understanding tsp difficulty by learning from evolved instances," *Learning and Intelligent Optimization*, pp. 266–280, 2010.

[11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[12] J. S. Martin Ester, Hans-Peter Kriegel and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," *KDD*, vol. 96, pp. 226–231, 1996.

[13] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computatation*, vol. 1, pp. 67–82, 1997.

[14] G. Laporte, M. Gendreau, J.-Y. Potvin, and F. Semet, "Classical and modern heuristics for the vehicle routing problem," *International transactions in operational research*, vol. 7, no. 4-5, pp. 285–300, 2000.

[15] M. F. Abdelatti and M. S. Sodhi, "An improved gpu-accelerated heuristic technique applied to the capacitated vehicle routing problem," in *Genetic and Evolutionary Computation Conference*, July 2020.

[16] B. E. Gillett and L. R. Miller, "A heuristic algorithm for the vehicle-dispatch problem," *Operations research*, vol. 22, no. 2, pp. 340–349, 1974.

[17] N. Suthikarnnarunai, "A sweep algorithm for the mix fleet vehicle routing problem," *International MultiConference of Engineers and Computer Scientists*, vol. 2, pp. 1914–1919, 2008.

[18] S. Lin and B. Kernighan, "An effective heuristic algorithm for the traveling salesman problem," *INFORMS*, vol. 22, no. 2, p. 498–516, 1973.

[19] G. Clarke and J. Wright, "Scheduling of vehicles from a central depot to a number of delivery points," *Operations Research*, vol. 12, pp. 568–581, 1964.

[20] T. Kohonen, "Self-organized formation of topologically correct feature maps," *Biological Cybernetics*, vol. 43, no. 1, pp. 59–69, 1982.

[21] T. Kohonen, "Clustering, taxonomy, and topological maps of patterns," in *Proceedings of the Sixth International Conference on Pattern Recognition*. Silver Spring, MD: IEEE Computer Society Press, 1982, pp. 114–128.

[22] R. Durbin and D. Willshaw, "An analogue approach to the travelling salesman problem using an elastic net method," *Nature*, vol. 326, no. 6114, pp. 689–691, 1982.

[23] H. Ghaziri, "Supervision in the self-organizing feature map: Application to the vehicle routing problem," in *Meta-Heuristics*. Springer, 1996, pp. 651–660.

[24] H. E. Ghaziri, "Solving routing problems by a self-organizing map," in *Artificial Neural Networks, 1991, International Conference on Artificial Neural Networks*. ICANN, 1991, pp. 829–834.

[25] H. Ghaziri and I. H. Osman, "Self-organizing feature maps for the vehicle routing problem with backhauls," *Journal of Scheduling*, vol. 9, no. 2, pp. 97–114, 2006.

[26] Y. Matsuyama, "Self-organization via competition, cooperation and categorization applied to extended vehicle routing problems," in *Neural Networks, 1991., IJCNN-91-Seattle International Joint Conference on*, vol. 1. IEEE, 1991, pp. 385–390.

[27] A. Modares, S. Somhom, and T. Enkawa, "A self-organizing neural network approach for multiple traveling salesman and vehicle routing problems," *International Transactions in Operational Research*, vol. 6, no. 6, pp. 591–606, 1999.

[28] M. Schwardt and J. Dethloff, "Solving a continuous location-routing problem by use of a self-organizing map," *International Journal of Physical Distribution & Logistics Management*, vol. 35, no. 6, pp. 390–408, 2005.

[29] M. Steinhaus, A. N. Shirazi, and M. Sodhi, "Modified self organizing neural network algorithm for solving the vehicle routing problem," in *2015 IEEE 18th International Conference on Computational Science and Engineering*, 2015, pp. 246–252.

[30] L. Kothoff, "Algorithm selection in practice," *Artificial Intelligence and Simulation of Behaviour Quarterly*, vol. 138, pp. 4–8, 2014.

[31] J. Holland, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence.* Ann Arbor, Michigan: University of Michigan Press, 1975.

CHAPTER 3

Data Exploration

This chapter presents Exploratory Data Analysis (EDA) of the study's data set D generated by the methodology discussed in chapter 2. The results from feature extraction and algorithm mapping, when considered together, may reveal patterns useful in understanding D holistically. The purpose of this chapter is to gain insight to the characteristics of D before modeling the algorithm selection. Unless stated otherwise, these analyses occurred in the R programming language.

3.1 Label Analysis

This section provides a bar chart of the label distribution in D. This visualization summarizes algorithm performance across the problem space and implies how well the machine learning techniques may classify each label. The frequency of label occurrence in the final data set is shown in Figure 12.

This study concludes no single CVRP solver heuristic dominated the algorithm portfolio; however, the CW and SOM earned significantly more labels than the GA and SP. Consequently, the machine learning algorithms will be provided a greater number of training examples for these two heuristics, which may allow for simpler prediction of their labels in the test data. However, a disparity in predicted label accuracy is not guaranteed [1].

For the purposes of algorithm selection, one metric presented in the literature for gauging model performance is comparing it to a simulated model which universally applies the most frequently occurring label to all instances. This is called the Single-Best-Solver model, and, in this study, would yield 48.9% accuracy with the CW algorithm [2]. This method of model evaluation, among others, is discussed in chapter 4. Later, this chapter explores feature-to-label relationships that yield

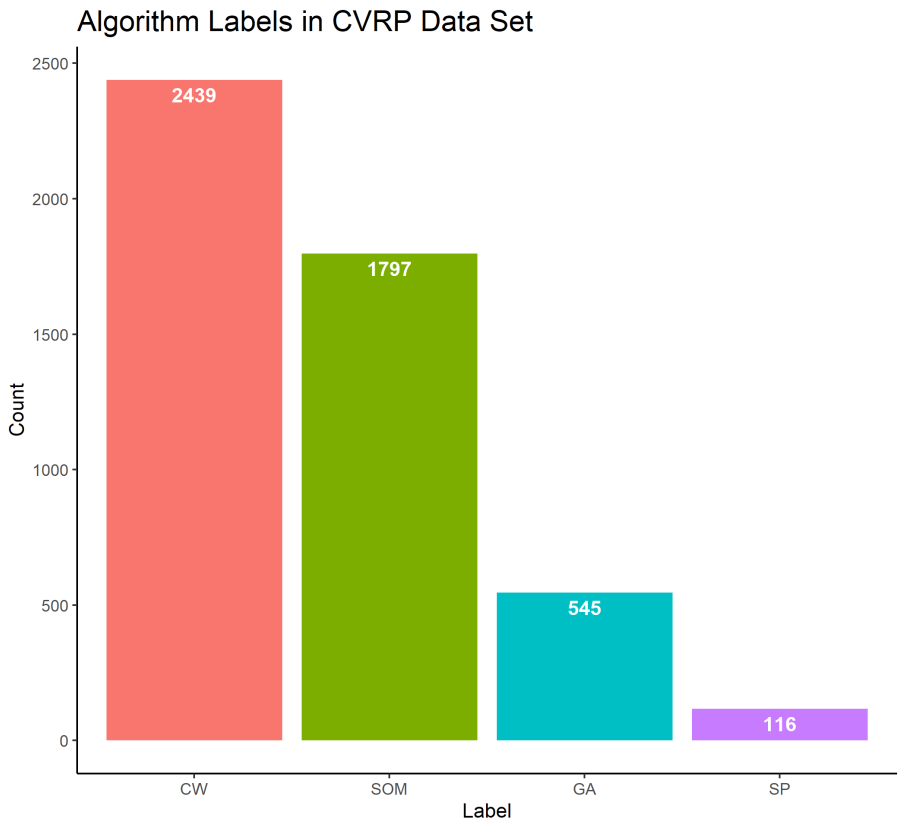

Figure 12. Plot of algorithm labels in the problem space resultant to the performance mapping discussed in section 2.3.

greater intuition on how the algorithms performed with respect to feature values.

3.2 Feature Analysis

This section explores the data distributions and descriptive statistics of the 23 features. First, histograms are introduced for all features to assess the range and frequency of their domain values. These plots are used to evaluate the diversity of the problem space and identify patterns imprinted on the features from the instance creation process. Next, the coefficient of variation for all features is used to meaningfully compare feature variability, which is a practical measure of determining those features containing the most information in D [3].

3.2.1 Distributions

This section presents histograms for all 23 features in the data set. The features are arranged sequentially in groups of four and are short-titled in accordance with section 2.2. All features have independent axes to eliminate shape misrepresentation otherwise inflicted by the scales of neighboring features.

The first tranche of plots displayed in Figure 13 shows a highly diverse range of values for the feature "1.NumberCities". This is to be expected, since all values between 19 and 199 (stepped by five) are equally likely in the creation process outlined in section 2.1. The other three features, especially "3.XCent" and "4.YCent" are more limited.

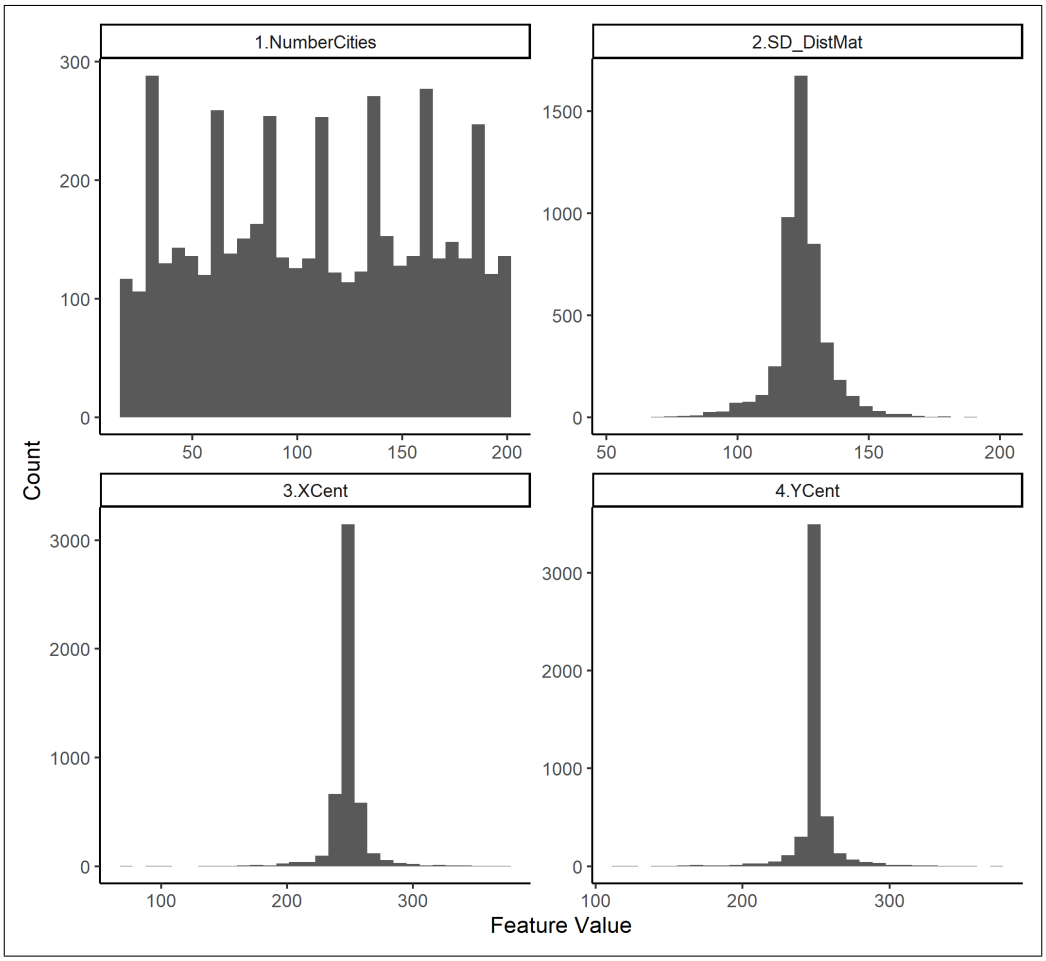

Figure 13. Histograms of features 1-4.

The second set of plots shown in Figure 14 exhibit greater diversity across the problem space. Of note is feature "6.FracDistinctDist", with an apparently abnormal amount of lower end values. However, this feature was targeted by the novel 'Equidistant' city positioning module presented in section 2.1. Recall this module positions cities at equidistant, or nearly equidistant, locations from one another. Consequently, it minimizes the percentage of unique values in the instance's distance matrix.

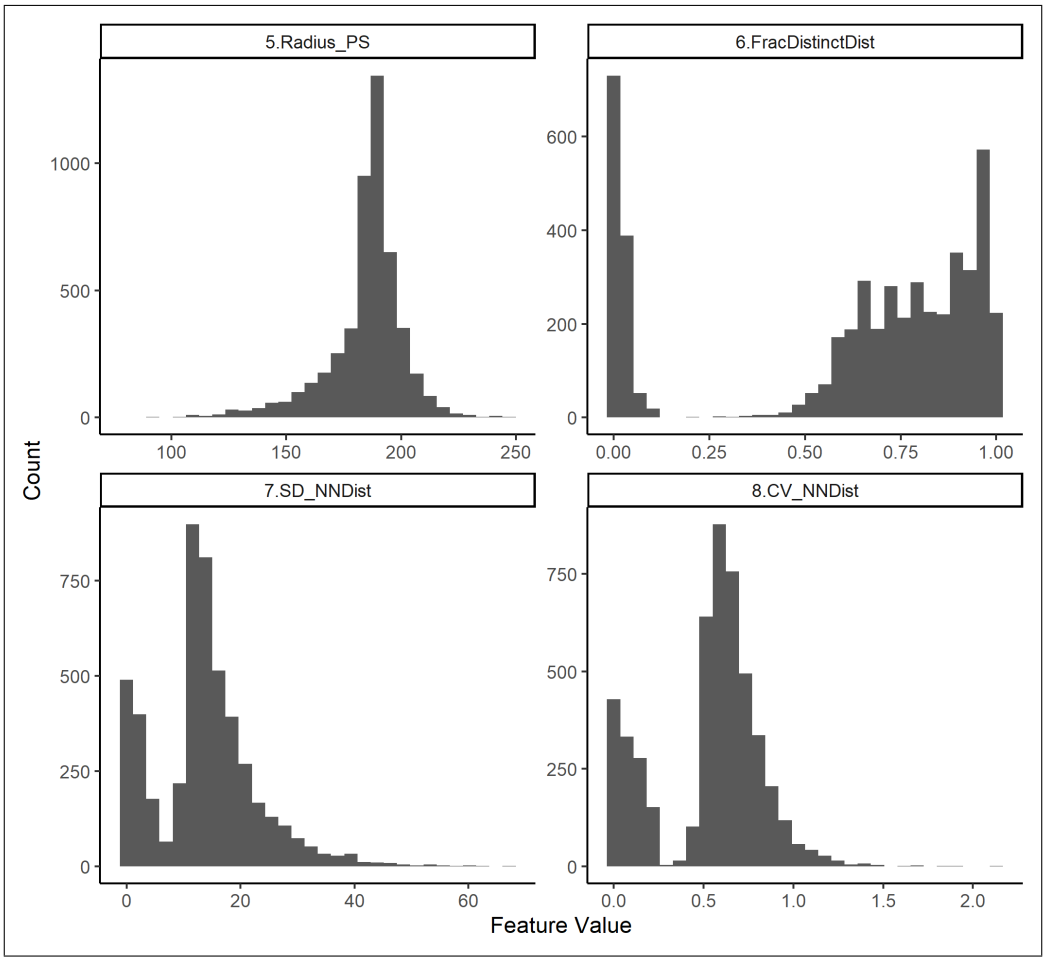

Figure 14. Histograms of features 5-8.

The next batch of plots rendered in Figure 15 further support the problem space containing instances with diverse features. These features all derive from the automated city clustering technique discussed in section 2.2.1 using the DBSCAN

algorithm. Of interest is the apparently high value of five clusters in "12.NumberClusters". Further analysis also revealed its connection to the 'Equidistant' city positioning module. Since this module divides the point grid into 5 rows $\times$ $n/5$ columns and chooses the centroid of each rectangle as the cities, an apparent spill-over occurs in the DBSCAN k-distance measurements when the number of nodes, n, exceeds 50. Indeed, all instances created with the 'Equidistant' city module contain one cluster when $n < 50$ and five when $n \geq 55$. This behavior is consistent with the DBSCAN documentation [4].

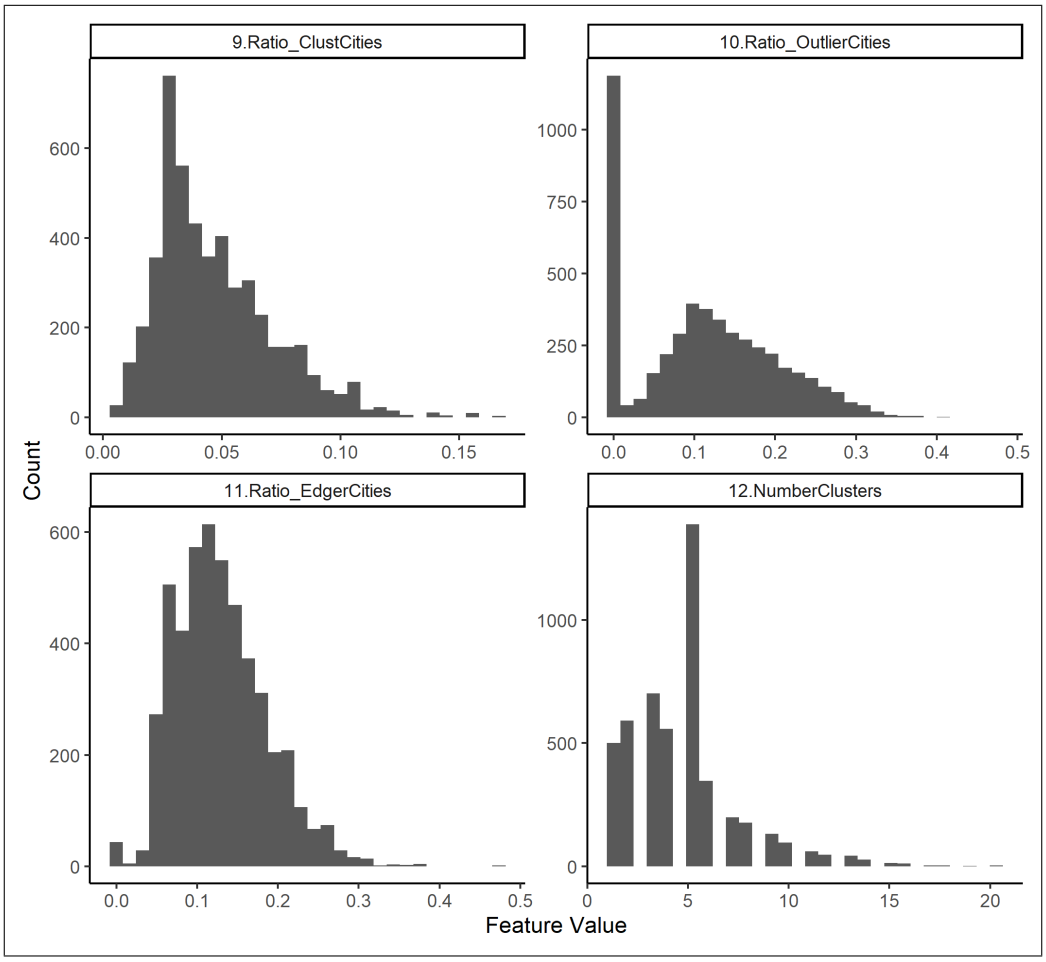

Figure 15. Histograms of features 9-12.

The subsequent set of plots displayed in Figure 16 continue to reflect diversity in the problem space. The distributions of "14.XDepot" and "15.YDepot" are as

expected from the depot positioning modules discussed in section 2.1. It is readily apparent these features capture the depot position at either the origin (0,0), the center (250,250), or some random point. The other two features also capture a wide range of values.

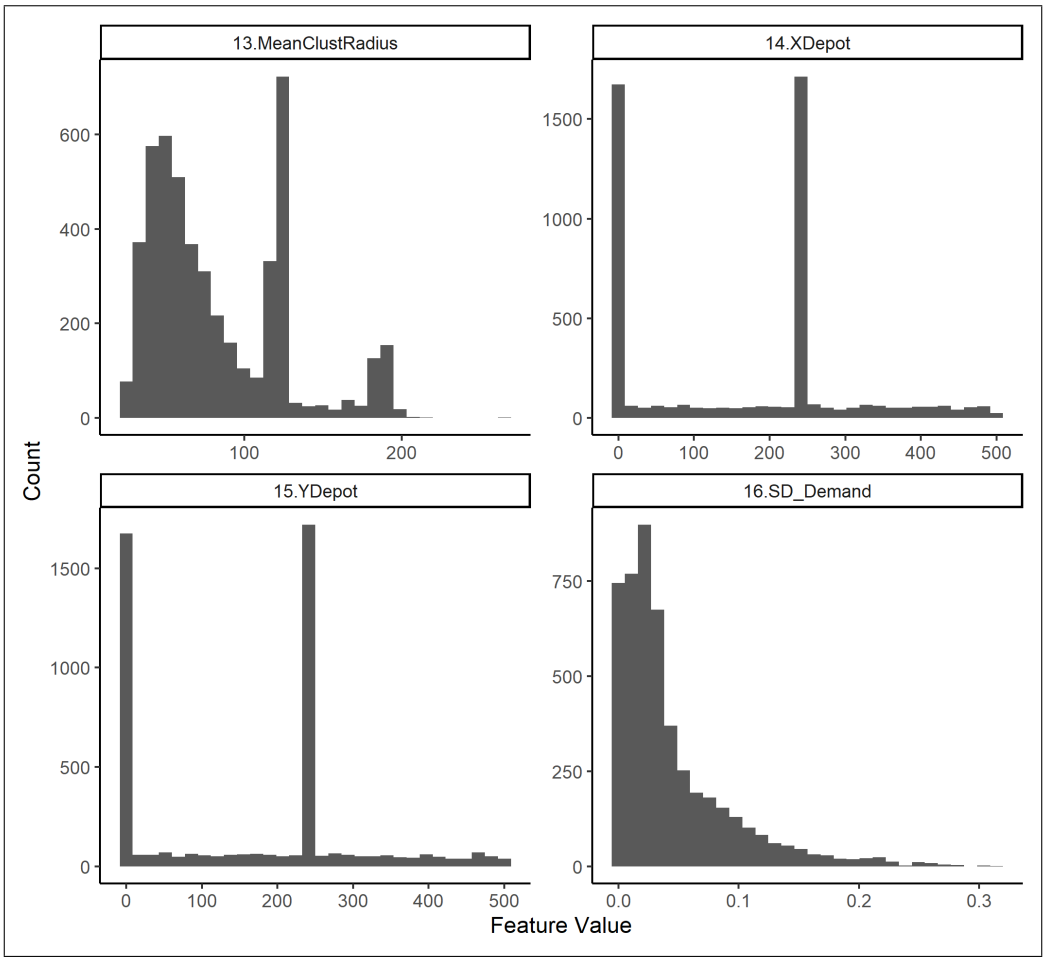

Figure 16. Histograms of features 13-16.

The next set of graphs, depicted in Figure 17, further reflect diversity in the problem space and behavior consistent with the instance creation process. In particular, "19.Ratio_OutlierdemDem" displays a high count where the ratio of outlier city demand to total city demand is equal to 0. This too should be expected from the 'Equidistant' city module. As discussed above and consistent with the DBSCAN literature, 'Equidistant' city instances generate no outliers due to the

hierarchical nature of the clustering algorithm [4, 5]. Rather, all cities are assigned to one cluster when $n \leq 50$ and five clusters otherwise. All remaining values in this feature come from the other city positioning modules.

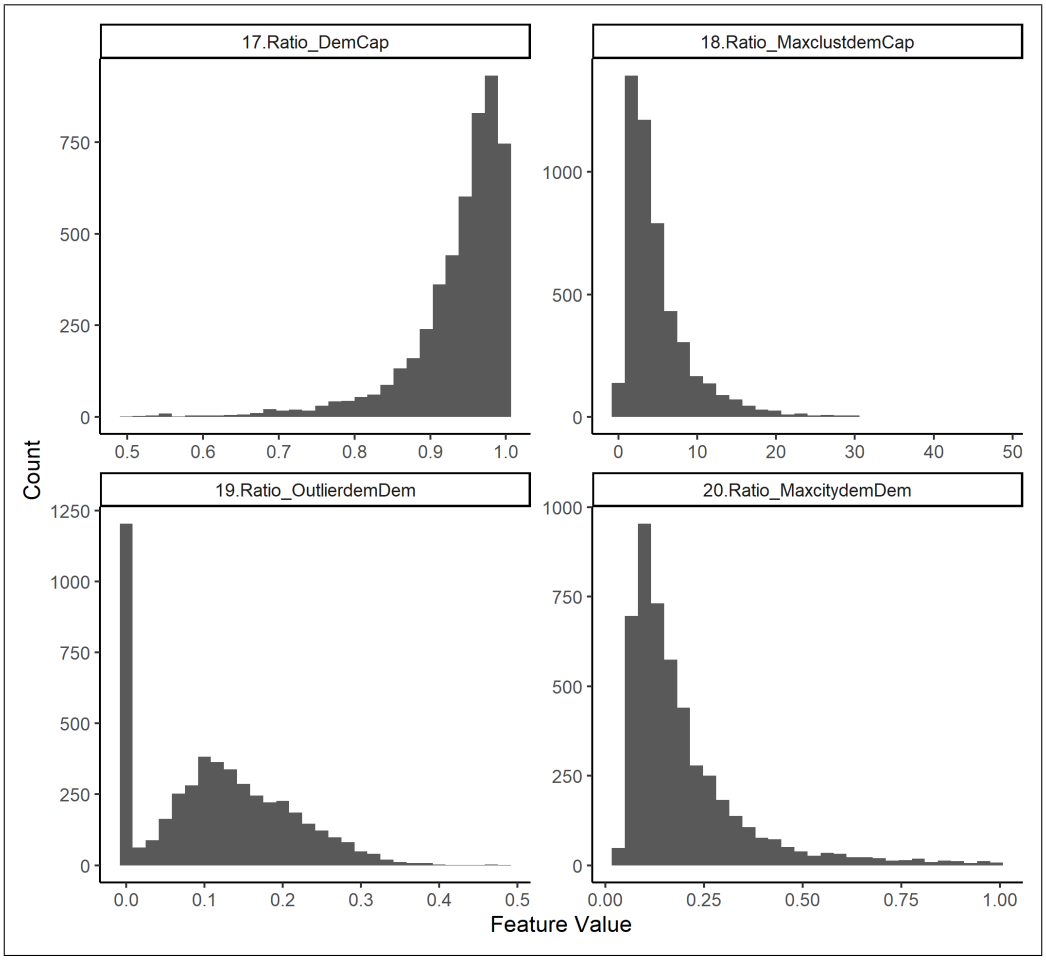

Figure 17. Histograms of features 17-20.

The last batch of plots displayed in Figure 18 showcase some of the most diverse features in the data set, particularly "21.AvgRouteLength" and "23.MinNumberTrucks". However, this is not surprising since these two features are linked in the creation process and directly engineered to engender diversity. The "21.AvgRouteLength" graph provides a more granular level of the plot presented earlier in Figure 10. Though not exact, one may observe its shape resemble the desired triangular distribution T[3,6,25] designed to diversify the problem space by Uchoa

et al. [6].

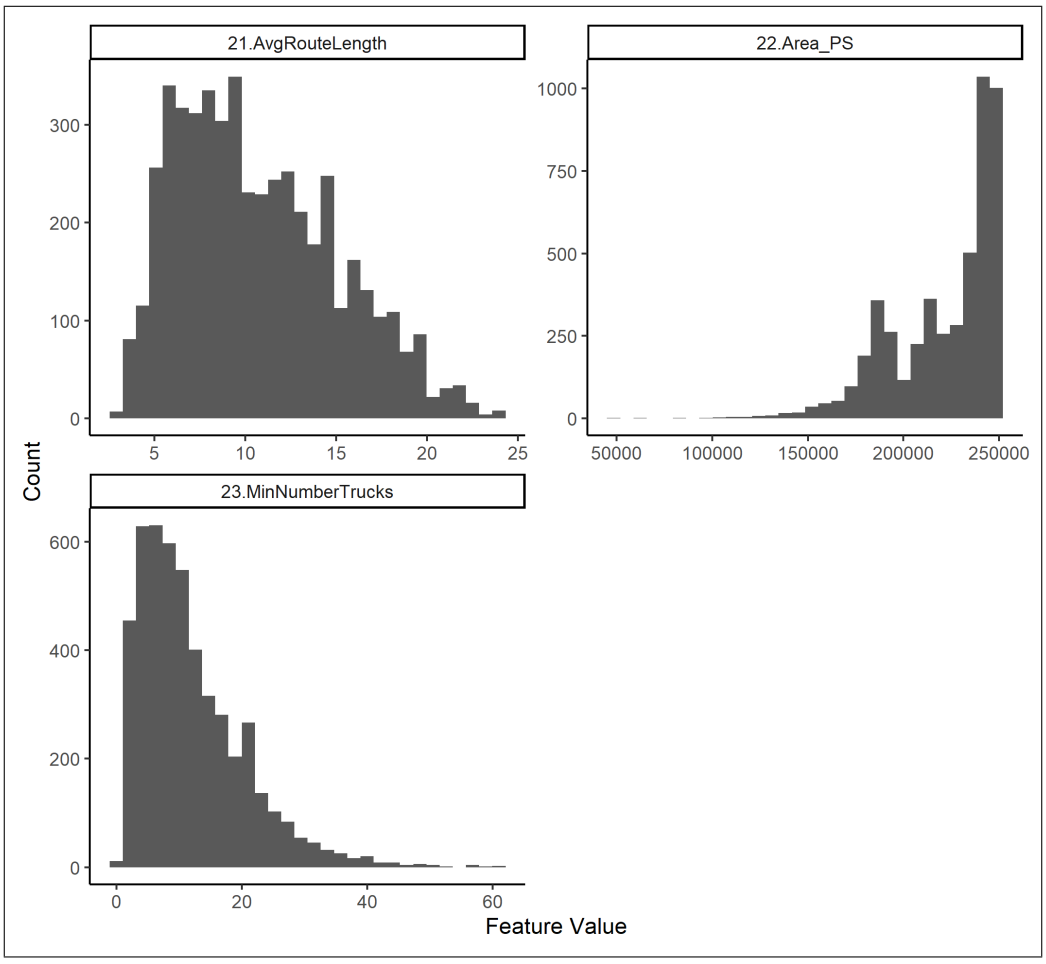

Figure 18. Histograms of features 21-23.

3.2.2 Descriptive Statistics

In addition to the histograms presented above, the statistics presented in table 1 provide an additional layer of exploratory analysis to the feature domains. Provided are the arithmetic mean, μ, and standard deviation, σ, of each feature, as well as the coefficient of variation (cv), where

$$cv = \frac{\sigma}{\mu}$$

Dividing μ by σ cancels feature units, resulting in a unit-less metric capable of comparing feature variability.

Feature	Mean	Std Deviation	Coefficient of Variation
1. Number of Cities	109.141	53.102	0.487
2. Std Deviation of Distance Matrix	124.531	10.731	0.086
3. X Coordinate of Instance Centroid	248.491	15.103	0.061
4. Y Coordinate of Instance Centroid	249.994	14.431	0.058
5. Radius of Instance	184.814	16.754	0.091
6. Fraction of Distinct Distance Matrix Values	0.613	0.359	0.586
7. Std Deviation of Nearest Neighbor Distance	13.683	8.906	0.651
8. Coefficient of Variation of Nearest Neighbor Distance	0.534	0.292	0.547
9. Ratio of Clusters to Cities	0.047	0.025	0.522
10. Ratio of Outlier Cities to Total Cities	0.113	0.088	0.782
11. Ratio of Edge Cities to Total Cities	0.129	0.056	0.437
12. Number of Clusters	4.637	2.76	0.595
13. Mean Cluster Radius	83.207	43.315	0.521
14. X Coordinate of Depot	165.426	143.707	0.869
15. Y Coordinate of Depot	164.14	143.113	0.872
16. Std Deviation of Demand	0.044	0.047	1.089
17. Ratio of Total Demand to Total Capacity	0.936	0.068	0.073
18. Ratio of Maximum Cluster Demand to Vehicle Capacity	5.003	4.428	0.885
19. Ratio of Outlier Demand to Total Demand	0.112	0.091	0.805
20. Ratio of Maximum City Demand to Vehicle Capacity	0.199	0.156	0.781
21. Average Route Length	10.693	4.39	0.411
22. Area of Instance	221529.3	27508.7	0.124
23. Min Number of Vehicles	12.048	8.375	0.695

Table 1. Feature space summary.

Measuring the variation of features is considered a "reasonable notion of importance [for machine learning algorithms], since this is the direction in which most information is encoded in the data" [3]. To fully appreciate this concept, consider an arbitrary feature with all instances having identical values. To a machine learning algorithm, this feature provides no useful information to discern one instance over another. However, this analysis alone is not absolute. Consider some other feature, perhaps a randomly generated identification number between three and six digits in length. Almost certainly this feature would contain a moderately sized cv, despite its irrelevance to classification. The cv can also be misleading if the feature domain contains positive and negative values. Notwithstanding, cv is an appropriate metric for this study because the features have documented relevance in the literature and, as seen in the previous section, D only holds positive values. The top five cv values reside with features 16, 18, 15, 14, and 19.

3.3 Feature-To-Label Analysis

This section explores the distribution of features by their algorithm label. Presented as Box-and-Whisker plots, these visualizations quickly indicate the range of feature values where algorithms performed best. These plots are relevant for intuition on the data set provided to the machine learning algorithms, which learn from previous feature-to-label relationships to predict the labels of new instances from its feature values. Figures are each presented with four feature plots.

The feature "1.NumberCities" in Figure 19 presents an interesting result. The median value for the GA does not intersect with the box of any other algorithm, indicating it may be grouped differently from the others. By examination, it appears the GA performed best on instances containing lower city counts.

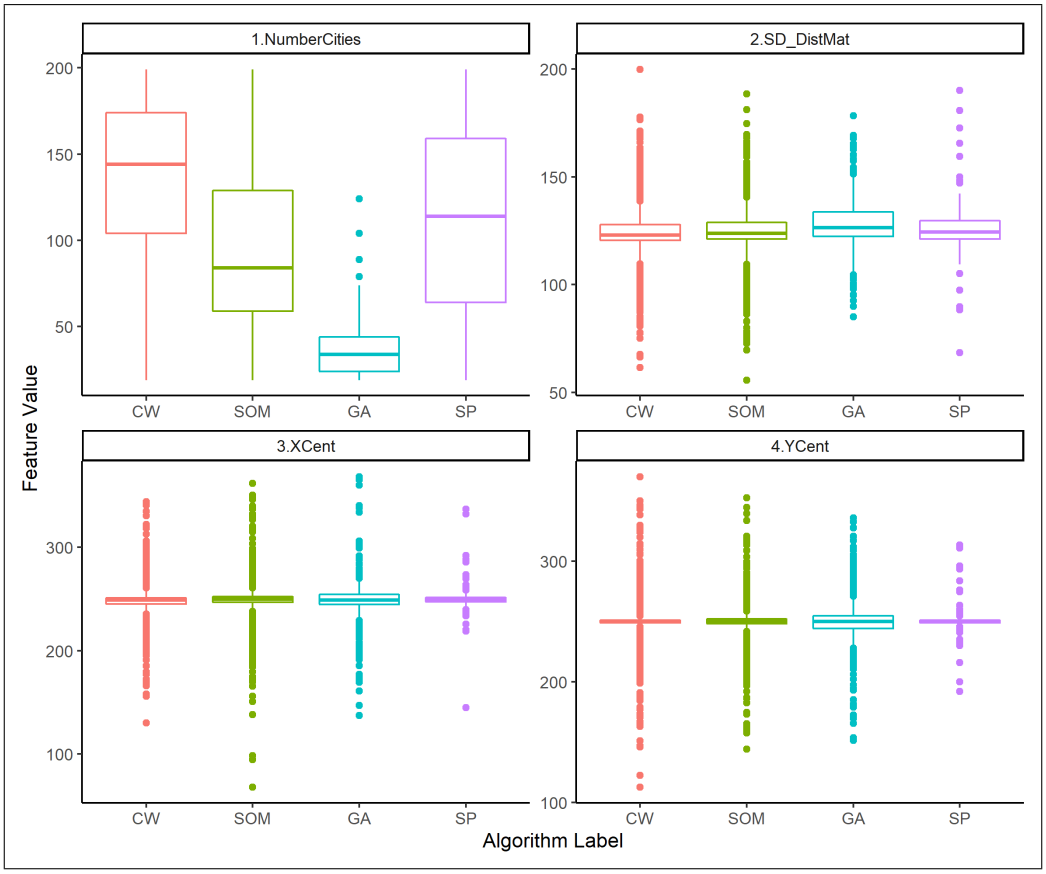

Figure 19. Box Plot of Features 1-4 according to algorithm label.

Figure 20 mostly shows overlapping algorithm performances, with some exceptions for the *GA*. In both "6.FracDistinctDist" and "7.SD_NNDist" the *GA* appears in its own group. To the delight of this study, these two features are consistent with the novel 'Equidistant' city positioning module. When an instance's cities are positioned in an equidistant manner, the fraction of distinct distances in its distance matrix are minimal, and so is the standard deviation of its nearest neighbor distance. From these plots, it is likely the *GA* performed poorly solving instances created under the 'Equidistant' module, and/or other instances containing similar spatial characteristics.

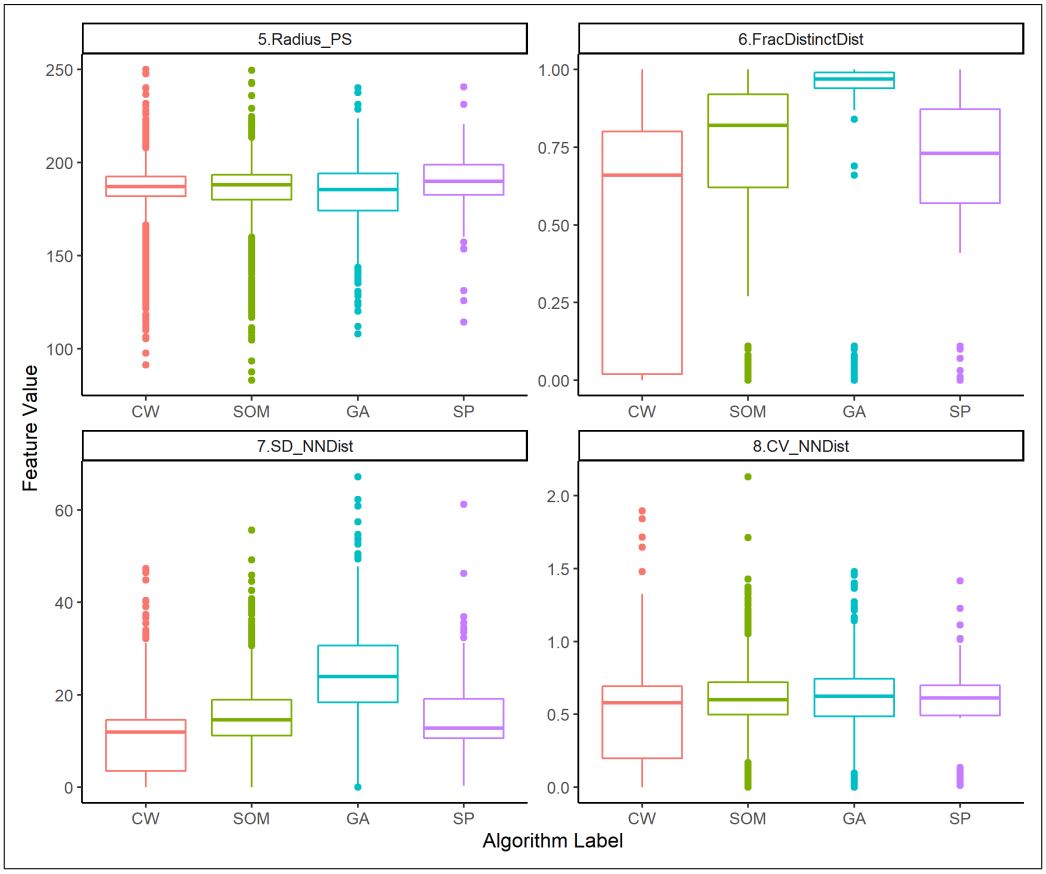

Figure 20. Box Plot of Features 5-8 according to algorithm label.

The set of plots shown in Figure 21 display similar grouping in the algorithm portfolio except "12.NumberClusters", which indicates another potential exception

for the GA. This feature is moderately linked to "1.NumberCities" due to the uniform application of $min\_samples = 4$ in the DBSCAN algorithm for all instances. While the clustering methodology outlined in section 2.2.1 allows for customized *eps* parameter selection, the number of clusters in an instance is inextricably linked to the number of cites it contains. For example, an instance containing 19 cities cannot exceed four clusters with $min\_samples = 4$. This plot, in conjunction with the "1.NumberCities" plot, builds a case for the GA performing best on instances comprised of lesser nodes.

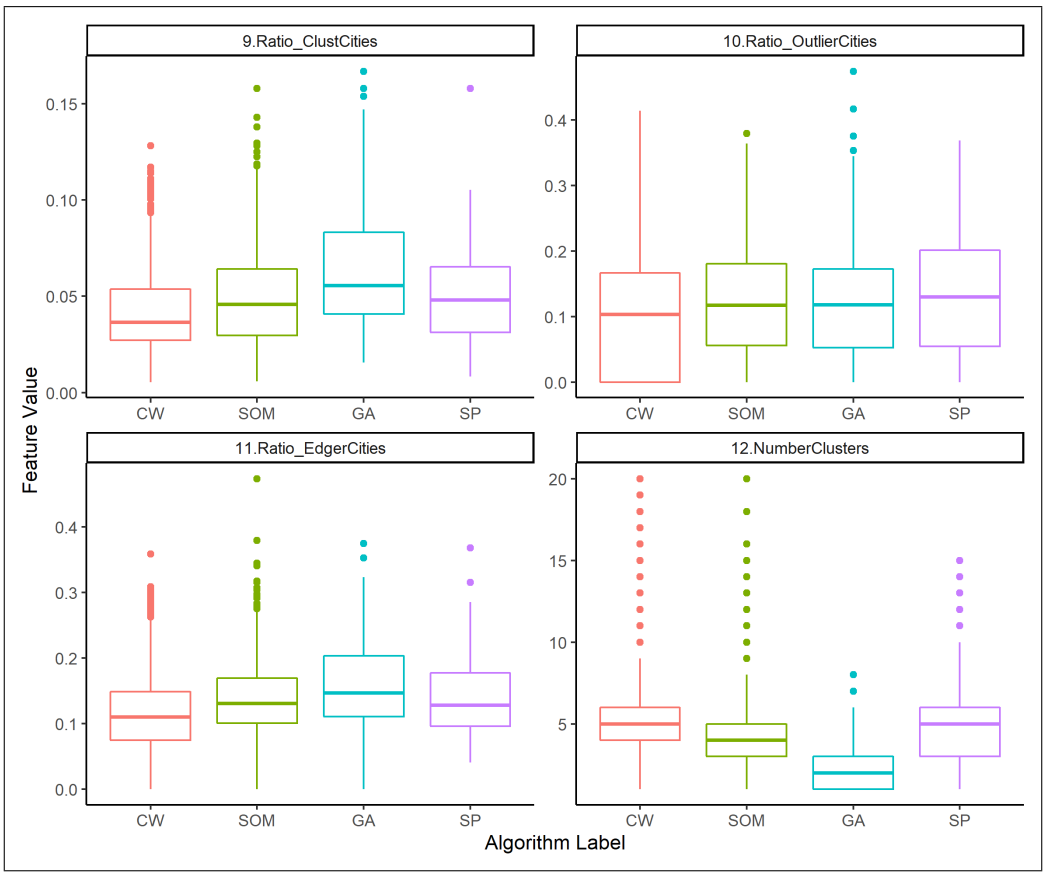

Figure 21. Box Plot of Features 9-12 according to algorithm label.

The next ensemble of plots, shown in Figure 22, display similar grouping in all features. The plots for "14.XDepot" and "15.YDepot" loosely suggest the SP performed best on instances with its depot located at the center of the grid;

however, the results are laden with outliers. The median *SOM* value appears nearly on top of its upper quartile, indicating this algorithm also favored instances with the depot near the center of the grid. Instances with a depot in the center correspond to those created under the 'Central' depot positioning module, or, more broadly, to those not constructed with the 'Eccentric' module.

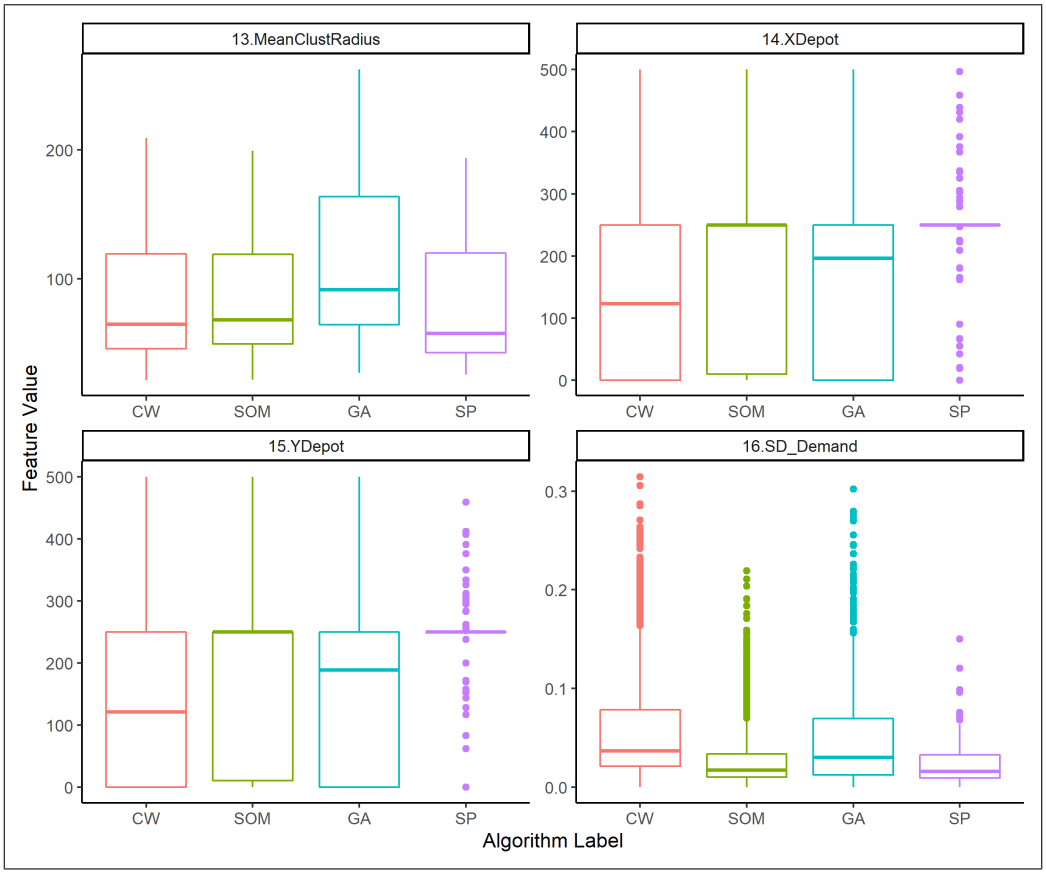

Figure 22. Box Plot of Features 13-16 according to algorithm label.

Figure 23 also shows similar or nearly similar grouping in the next set of features. However, the feature "17.Ratio_DemCap" may have the *CW* and *SP* algorithms, the two classical heuristics, in the same group. This feature introduces a possible pattern in performance based upon the tightness of an instance, or the ratio of instance total demand to total capacity [6]. Instance tightness in this study is a byproduct of its number of nodes, demand module, and randomly generated r

value discussed in section 2.1.1. In general, finding a quality solution to a very tight instance is difficult for heuristics [6] and, in this study, it appears the evolutionary algorithms produced poorer solutions as tightness increased.

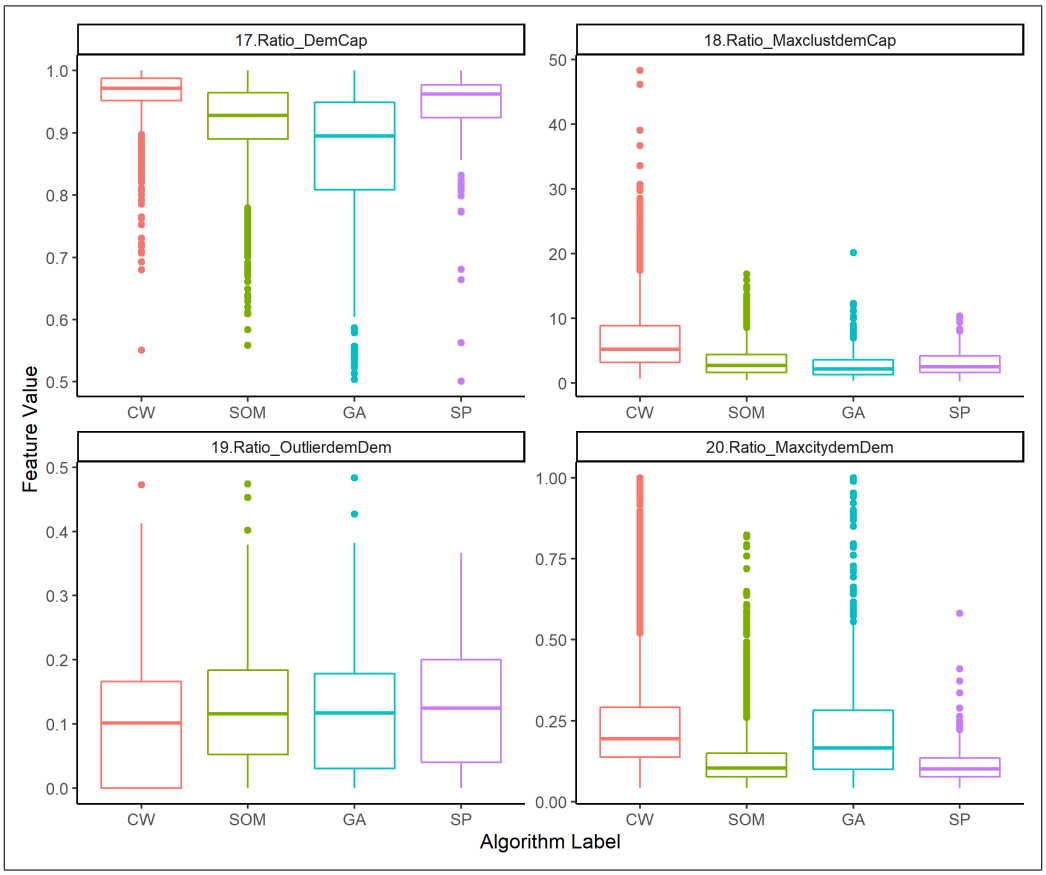

Figure 23. Box Plot of Features 17-20 according to algorithm label.

The final set of plots are shown in Figure 24. The features "21.AvgRouteLength" and "23.MinNumberTrucks" also reveal noteworthy algorithm behavior. Two groups possibly exist in "21.AvgRouteLength", one containing CW and GA and another with SOM and SP. This suggests one group performed better with more cities per route and another with less. Since the GA and SOM are both metaheuristics, perhaps increasing their search space in future studies would first lead to domination of its group. In "23.MinNumberTrucks" notice the CW in a clear group of its own, outperforming all other algorithms when the instance re-

quires a higher number of trucks, or routes. Indeed, with the exception of one *GA* outlier, the *CW* performed best on all instances requiring more than approximately 25 routes.

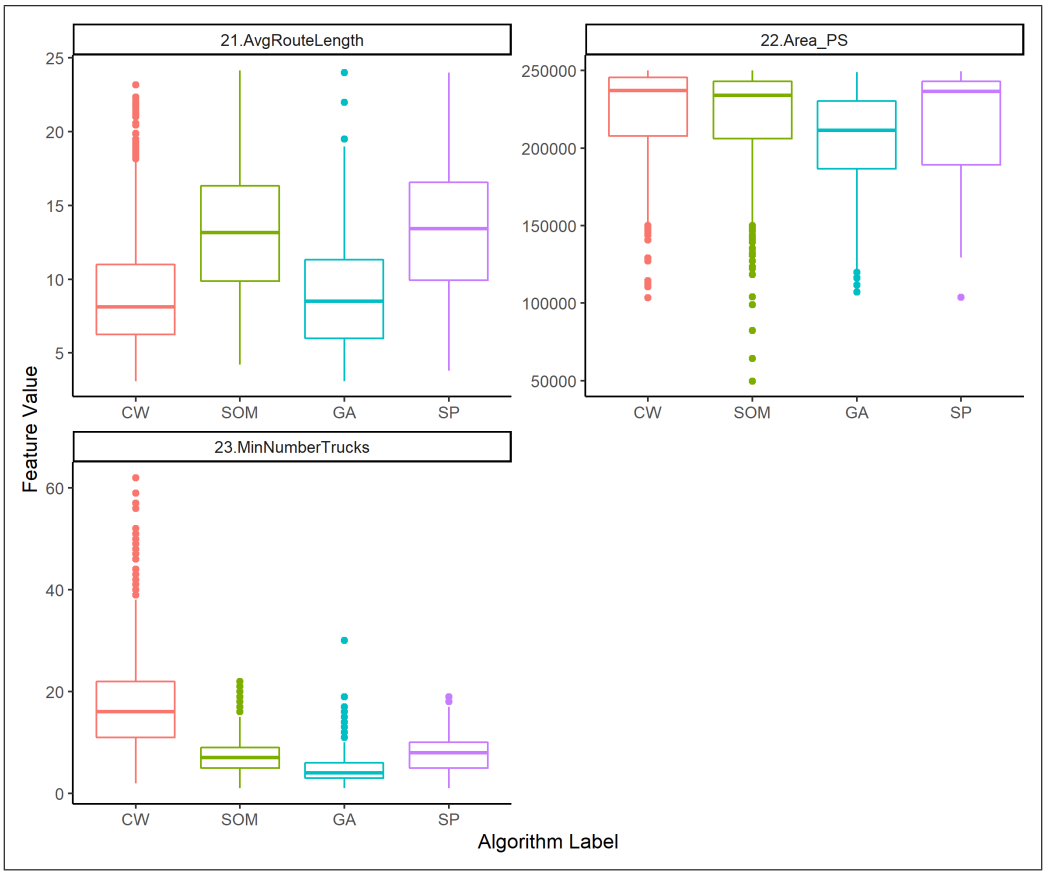

Figure 24. Box Plot of Features 21-23 according to algorithm label.

Identifying behaviors in feature-to-label relationships is both insightful and revealing to the data domain. However, those plots which indicate no apparent pattern at all showcase the importance of using machine learning to discriminate between features and labels. Should many features expressly indicate hard divides among algorithm performances, the need for machine learning would be reduced or eliminated altogether. Where no obvious pattern meets the human eye, machine learning algorithms operate in higher dimensions to consider boundaries formulated by the interaction of multiple features. As expressed by Domingos, "it has even

been said that if people could see in high dimensions machine learning would not be necessary" [1].

3.4 Principal Component Analysis

Principal Component Analysis (PCA) is an unsupervised machine learning technique used to consolidate information from a data set, expressed in the form of its variance, into lesser dimensions [3, 7, 8]. The output of PCA is the original data projected onto a new set of variables, known as Principal Components (PC). The PCs are constructed such that the first one captures the most variance, followed by the second, and onward. Consequently, most information in a data set may be captured within the first number of components [8].

PCA, a linear transformation method introduced by Karl Pearson in 1901 and made famous by Harold Hotelling 1933, is an effective instrument of compression because high dimensional data may inherently contain redundancies and/or correlations that indicate "an intrinsic lower-dimensional structure" [7]. Before applying PCA to this study's data, first consider the discoveries of the previous section. In particular, recall the box plots presented in Figure 20 and the apparent correlation between "6.FracDistinctDist" and "7.SD_NNDist". If an instance has all cities positioned at equal distances from one another, both the fraction of distinct values in its distance matrix and the standard deviation of the nearest neighbor distance will be minimized. Also discussed in the previous section is the inseparable structural limitation of "12.NumberClusters" conferred from "1.NumberCities". These limited examples indicate that PCA may have a wholesale effect on the data set due to a surplus of information gleaned through the 23 features.

The remainder of this section presents PCA methodology and computational results, including a visualization of D presented through the first few PCs. All analyses conducted in this section occurred in the Python programming language

using the *sklearn*, *pandas*, and *numpy* libraries.

As an unsupervised machine learning technique, PCA does not consider instance labels. Therefore, this study's PCA analysis used only the feature values provided matrix by X, of dimension 4897×23. The steps for PCA include:

1. Standardize all features to remove scale and unit of measure disparities, resulting in scaled matrix Z where all values lie in the range (0,1);

2. Compute the covariance matrix for Z^T, where columns represent instances and rows represent features. Features become the variables of interest by transposing Z, and the covariance matrix reflects the measurement of variability between features. The resulting matrix is C and is of dimension 23×23;

3. Extract the eigenvectors and eigenvalues of C and sort them in descending order. Since C is symmetrical, eigenvectors are orthonormal and the largest eigenvalues indicate which direction contains the most variance represented by a linear combination of features [9, 7]. The sorted eigenvector matrix E is of dimension 23×23;

4. Project Z (dimension 4897×23) onto E (dimension 23×23) through matrix multiplication, where $Z' = ZE$ and is of dimension 4897×23.

One product of PCA includes documenting the percentage of total variance explained by the PCs [4]. Presented cumulatively in Figure 25 is the amount of information captured from the study's feature data by the first d PCs. Notice how the first 10 PCs capture approximately 95% of the total variance, which indicates only 5% of the original signal is lost despite reducing dimensionality by more than half.

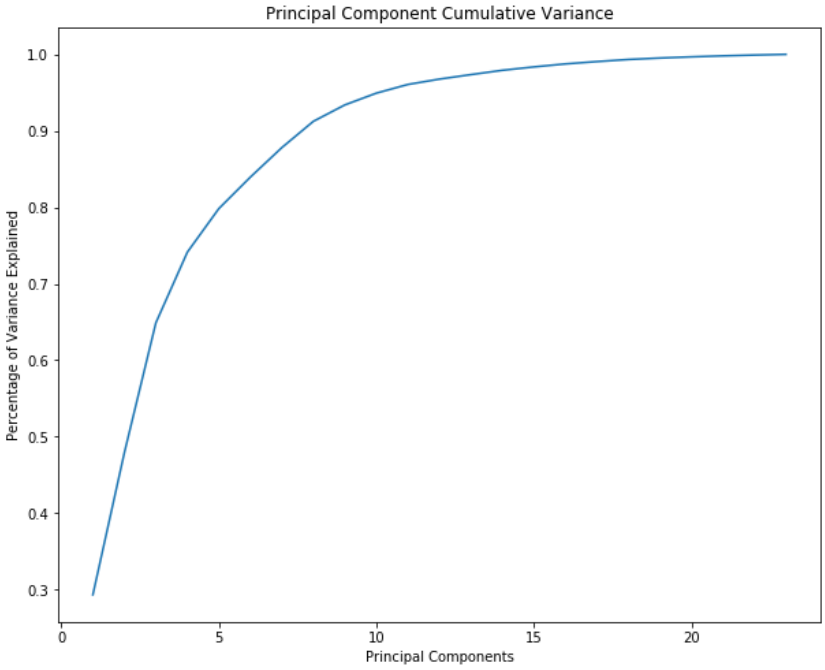

Figure 25. Total information in X explained by d principal components.

PCA also enables visualization of higher dimensional data. From Figure 25, the first three PCs encapsulate approximately 65% of the variance; therefore, plotting the instances by just these components provides a reasonable proxy for exploring boundaries in the data. Figure 26 presents this projection. Note each point represents one of the CVRP instances and is colored accordingly by its labeled algorithm. Coloring was made possible by concatenating labels to the Z' matrix. Observe how the CW algorithm gravitates towards the plane comprised of lower PC values, while the GA largely resides with the higher values. The SOM mostly lay between CW and GA and the SP appears mostly at random, yet is difficult to determine due to its infrequent occurrence. Albeit complex, a structure among labels and features clearly exists. The instances may also be shown in the two dimensional plane while capturing approximately 50% of the variance. This visualization is presented in Figure 27 and displays similar behavior from all algorithm labels.

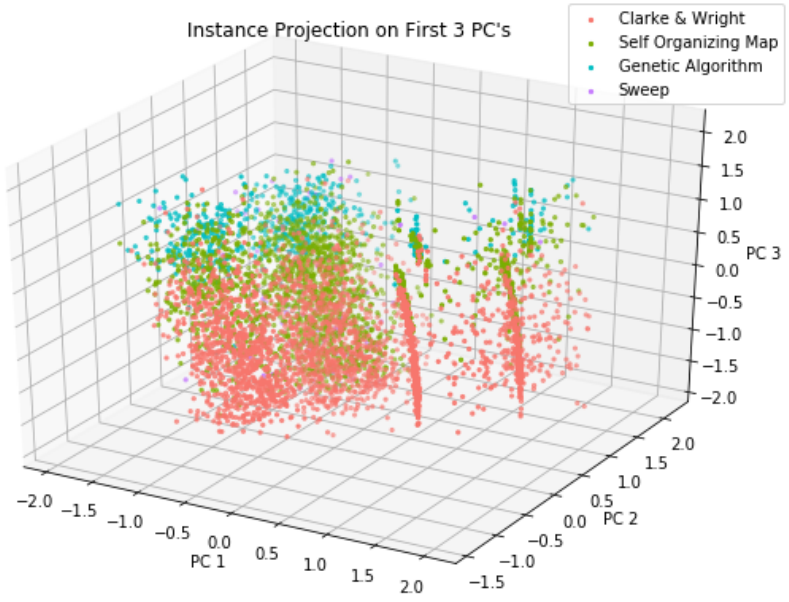

Figure 26. Problem space instances represented by their first three principal components, colored by algorithm label.

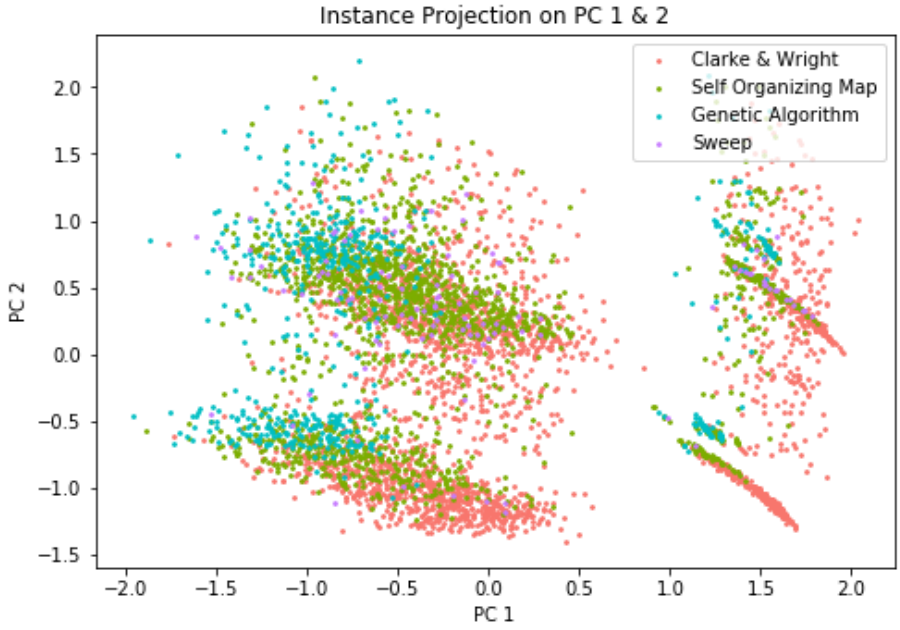

Figure 27. Instance visualization in the two dimensional plane using the first two principal components.

The problem space certainly contains boundaries of separation that suggest the features can indicate algorithm performance. While these boundaries are complicated, straightforward boundaries that readily and universally separate labels would make for a trivial machine learning problem. Machine learners thrive through induction and generalization to model complex data boundaries in the form of a prediction function [1]. This function later predicts the classification of new instances by identifying where it lay with respect to the label boundaries. Figures 26 and 27 are presented in this report as visual media to support the need for machine learning to generalize these boundaries and predict algorithm performance from feature values.

List of References

[1] P. Domingos, "A few useful things to know about machine learning," *Communications of the Association for Computing Machinery*, vol. 55, pp. 78–87, 2012.

[2] J. Beel and L. Kotthoff, "Proposal for the 1st Interdisciplinary Workshop on Algorithm Selection and Meta-Learning in Information Retrieval (AMIR)," in *Advances in Information Retrieval*, L. Azzopardi, B. Stein, N. Fuhr, P. Mayr, C. Hauff, and D. Hiemstra, Eds. Cham: Springer International Publishing, 2019, pp. 383–388.

[3] H. Daume, "A course in machine learning," 2017, unpublished. [Online]. Available: http://ciml.info/

[4] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[5] M. K. Steinhaus, "The application of the self organizing map to the vehicle routing problem," Ph.D. dissertation, University of Rhode Island, 2015, paper 383.

[6] E. Uchoa, D. Pecin, A. Pessoa, A. S. Marcus Poggi, and T. Vidal, "New benchmark instances for the capacitated vehicle routing problem," *European Journal of Operational Research*, vol. 257, pp. 848–858, 2017.

[7] M. P. Deisenroth, A. A. Faisal, and C. S. Ong, *Mathematics for Machine Learning*. University College, Lond: Cambridge Press, 2020.

[8] I. T. Jolliffe, *Principal Component Analysis*. King's College, Aberdeen, United Kingdom: Springer, 1986.

[9] H. Anton and C. Rorres, *Elementary Linear Algebra*. Drexel University: John Wiley and Sons, 2000.

CHAPTER 4

Machine Learning Methodology

This chapter describes the supervised machine learning methods used for executing the algorithm selection problem on data set D as a classification problem. The work presented in this chapter corresponds to the study's final steps as shown by Figure 6. All models and analyses occurred in the Python programming language using common machine learning and data analysis libraries, including *sklearn*, *pandas*, and *numpy*.

This chapter is organized into three sections. The first section defines the machine learning algorithms, or "learners", which generate the classification models and discusses the model selection techniques employed for hyper-parameter tuning. The second section evaluates the models by metrics consistent with machine learning and algorithm selection literature. The final section explores the sufficiency hypothesis of 5000 CVRP instances enabling adequate learner generalization of the data domain.

4.1 Classification Models

For this study, an arbitrary CVRP instance i was represented in the $(\vec{x_i}, y_i)$ form, where $\vec{x_i}$ is a 23 dimensional vector comprised of those features outlined in section 2.2 and y_i is the best performing algorithm from section 2.4. Consistent with the data structure outlined in chapter 1, the shape of D was 4897×24.

This study implemented the algorithm selection, or label classification, with three independent learners: Decision Tree (DT), k-Nearest Neighbors (KNN), and Single-Layer Perceptron (SLP). These learners utilize distinct methods to perform classification tasks, providing varied insight to the data domain. Model selection for each learner occurred through a grid search of its hyper-parameters, which are

the user-defined parameters that influence model performance and cannot be tuned exclusively on training data [1]. The DT and SLP models were optimized using 5-Fold Cross Validation (CV), and KNN with Leave-One-Out Cross Validation (LOOCV).

This research included a pilot study to determine a conservative range of values for the learners' hyper-parameters, which are explained in the subsequent sections. The pilot study also explored if learner performance, evaluated by prediction accuracy, would improve if provided a lower dimensional data set comprised of the first d principal components. This appeared promising with only with KNN learner; therefore, it was omitted from the DT and SLP learners for the purposes of this report.

All learners utilized an 80/20 split for training and testing, respectively. This method provided learners with 80% of the instances for training and model selection. Then the final model, newly fit with all training data, was tested with the remaining 20% of instances. The percentage of correct classifications on the test set is the model's reported accuracy.

4.1.1 Decision Tree

The DT learner classifies with a top-down approach, where one feature at a time is split based upon some criterion to minimize label impurities as the tree grows [1]. It does not directly consider interaction among features but instead indicates those features which singularly perform best for making predictions. The DT algorithm is inherently greedy and possesses inductive bias by always splitting on locally optimal feature values; however, it is remarkably intuitive and interpretable [1]. The model, or prediction function f, generated by the learner is piece-wise constant [2]. The algorithm's learning methods and hyper-parameters are explained below.

The DT learner constructs a classification model by using a purity function to calculate partitions in the data that homogenize label values in subsets of data [3]. The tree grows deeper as the splits compound, and the subsets increase purification. The first split is commonly referred to as the root, subsequent splits are called nodes, and the final subsets of data are considered the leaves [1]. The purity function is a hyper-parameter called *criterion* and is of prospective value 'entropy' or 'gini'. The entropy function, if evaluating K possible label values in a set of data S, is given by the equation [3]

$$E(S) = -\sum_{k=1}^{K} p_k \log_2 p_k$$

where p_k is the probability of a label occurring in the set

$$p_k = \frac{|S_k|}{|S|}$$

The expected reduction in entropy after the split, or information gain G, is calculated by parent node entropy less weighted entropy values of children nodes following the split [3, 4]

$$G(S, K) = E(S) - \sum_{k}^{K} \frac{|S_k|}{|S|} E(S_k)$$

As seen in the above equation, information gain increases for low children entropy values. Finding the best feature value on which to split is solved by exhaustively searching all split values in every dimension and selecting that which results in the largest information gain [2]. The gini is given by

$$E(S) = 1 - \sum_{i=1}^{K} p_k^2$$

In general, gini is simply a more computationally efficient metric than entropy, though it may produce different results [2]. Both entropy and gini were explored.

The learner's other hyper-parameter, maximum depth (max_depth), limits tree growth. If the learner is left to grow a tree to absolute purity, the resulting

model would likely be overfit from simply memorizing all training instances [1]. Instead, *max_depth* is set to stop the algorithm early, capturing only those nodes which provide the most information gain to generalize D. Consequently, impurities may exist in some or all of the leaves. In this case, the most frequent label in the leaf is the predicted value of a new instance [1, 2]. This study considered a *max_depth* range of values from 1 to 50.

This study tuned both the *criterion* and *max_depth* hyper-parameters through 5-Fold CV. The combination of parameters resulting in the highest mean accuracy consisted of *criterion*='entropy' and *max_depth*=6. With these parameter values, a final model fit with all training data produced a test accuracy of 76.4%. The model's confusion matrix, which presents the probability of correct and incorrect predictions for each label in the test set, is presented in Figure 28. Note the SP algorithm is never predicted as the best performing algorithm, which could be expected due to its infrequency of occurrence. The CW algorithm is correctly predicted 87% of the time, though is misclassified as GA or SOM with 2% and 11% probability, respectively.

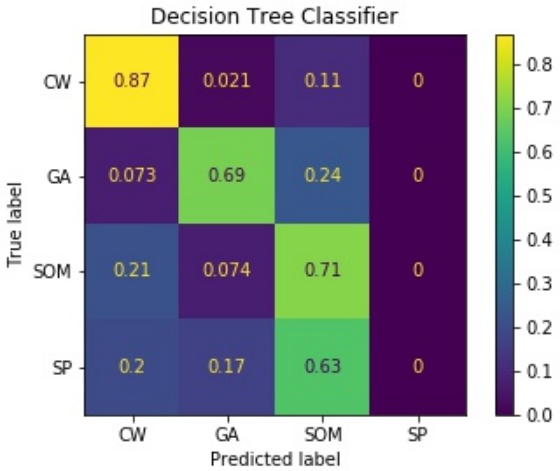

Figure 28. Decision Tree Model Confusion Matrix

The model also revealed the following features as the top three most impact-

ful to classification: *23.Minimum Number of Vehicles*, *1.Number of Cities*, and *20.Ratio of Maximum Demand to Vehicle Capacity*. On the contrary, the three least impactful to classification included: *3.X Coordinate of Instance Centroid*, *10.Ratio of Outlier Cities to Cities*, and *12.Number of Clusters*. For visualization purposes, the top two splits of the DT model are presented in Figure 29. The full model built to a depth of six splits is difficult to view in this report. This graphic represents the piece-wise prediction function f, where feature values of a test instance lie within boundaries created by the root and branches, ultimately allowing classification by the leaf.

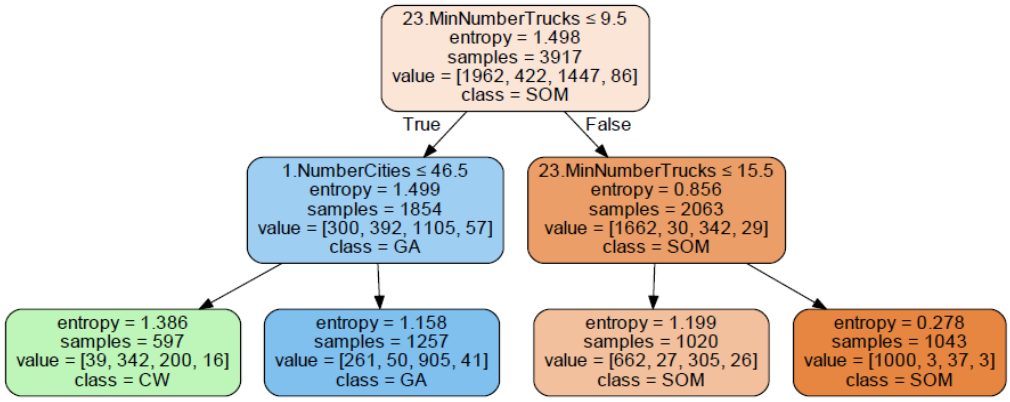

Figure 29. Top splits of DT model.

4.1.2 K-Nearest Neighbors

Contrary to the DT, KNN is a form of instance-based learning, one in which no explicit prediction function f is learned [5]. The algorithm simply makes predictions from the geometric relationship among instances in the data domain [1]. New instance labels are predicted by their proximity to the k nearest known instances, or neighbors. The class represented most frequently by its neighbors becomes the predicted label for the new instance. Referred to as a "lazy learner", training consists of merely storing the training instances [5]. However, predictions are com-

putationally expensive as the distance to every training instance must be measured for every test instance.

Since KNN models treat all features with equal value it can indicate similarities in the domain; however, its performance is vulnerable to inappropriate features and the curse of dimensionality [6]. As a result, this study explored if the performance of the KNN learner would improve if it considered less features, expressed in the form of principal components. Principal components are generated from Principal Component Analysis, an unsupervised machine learning technique explained in section 3.4. Data sets comprised of up to the first 15 principal components were considered. The learner's hyper-parameters included the distance *metric* ('euclidean' vs. 'manhattan'), *weights* ('uniform' vs. 'distance'), and number of nearest neighbors k. These hyper-parameters are explained below.

The *metric* prescribes the calculation method for measuring the distance between instances. The euclidean setting programs according to the Euclidean distance equation, given for two instances i and j in an M dimensional space

$$d_{i,j} = \sqrt{\sum_{m=1}^{M} (i_m - j_m)^2}$$

Manhattan corresponds to the "city block" measurement of absolute distance, given for two instances i and j in an M dimensional space

$$d_{i,j} = \sum_{m=1}^{M} |i_m - j_m|$$

The *weights* hyper-parameter assign an optional significance value for closer instances. The 'uniform' setting treats all instances in the space equally whereas 'distance' weights instances inversely to their distance from the test instance [2]. The final hyper-parameter, k, is the number of nearest neighbors considered for each test instance. This study explored a range of values from 1 to 75 for k.

As mentioned earlier, this study tuned the KNN hyper-parameters through LOOCV. Different from k-Fold CV explained in chapter 1, this method loops through all training examples and predicts its classification based upon the combination of hyper-parameters [1]. While this is an expensive operation for other learners, computations with KNN only occur at prediction and therefore requires "only as much computation as computing the k nearest neighbors for the highest value of k" [1].

The data set comprised of the first nine principal components resulted in the best performing KNN model. Figure 25 shows between 92-95% of the data set's information is compressed to the first nine principal components. Despite eliminating more than half of the total dimensions, less than 10% of the original signal is lost. This trade off proved beneficial to the distance-based KNN learner. The combination of parameters resulting in the highest mean accuracy consisted of *metric*='euclidean' and *weights*='distance', and k=43. With these parameters, a final model fit with all training instances achieved an accuracy of 75.1%. This model's confusion matrix is presented in Figure 30.

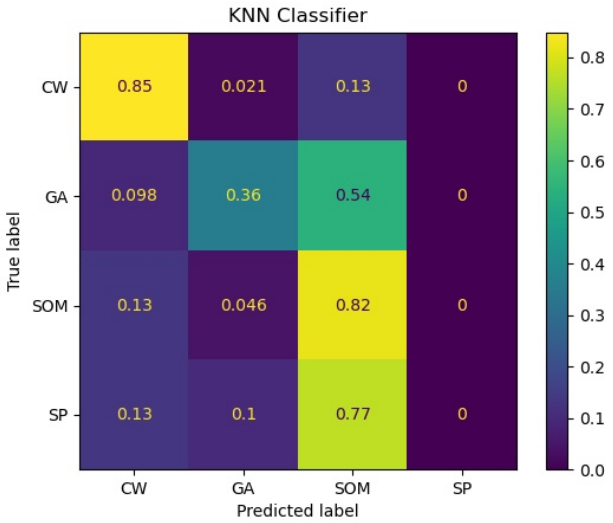

Figure 30. KNN Model Confusion Matrix.

Note the KNN model, just like the DT, also never predicts the SP algorithm as the best performing algorithm. The SOM algorithm is correctly predicted 82% of the time, approximately 10% better than the DT model. However, KNN is more than 30% less likely to correctly predict GA instances. The significance of model comparisons such as these are captured by their F-values, which are discussed in section 4.2.

4.1.3 Single-Layer Perceptron

The SLP is the final and most complicated learner considered in this study. The SLP is a form of neural network and classifies data with distinctly different methods than the DT and KNN learners. Rather than examining features independently or treating them all equally, the SLP models instance labels as a nonlinear regression function of linear feature combinations, allowing weighted interaction among the features in the model's prediction function f [7]. In general, this learner classifies a new instance based upon label probabilities formulated from a nonlinear regression model built from training data. The literature suggests these models typically outperform simple regression-based classifiers, but at the expense of being "over parameterized" to the point where intermediary information is "uninterpretable" [7].

The SLP title is a misnomer since the learner actually contains three layers: an input layer, a hidden layer, and an output layer. The "Single Layer" refers to the learner's sole hidden layer, whereas learners with more than one hidden layer are referred to as a Multi-Layer Perceptron [7]. There are two main steps to the SLP learner, the first is the forward pass and the other is the backward pass. These steps, along with the description and function of networks layers, are described below.

The forward pass is the learner's inference phase where it makes a prediction

for an arbitrary instance [1]. The input layer consists of vector $\vec{x_i}$, which contains raw feature values of length d for instance i. The hidden layer H is comprised of J user-defined nodes, referred to as h_j. A weight matrix W is used to perform a weighted linear summation of $\vec{x_i}$ to h_j, allowing interaction between every feature and every node [2]. Where b is the bias term, x_i is the feature value, and w_i is the feature weight, this transformation occurs through the following equation [2]

$$h_j = \sum_{i \in d} w_i x_i + b \quad \forall j \in J$$

The above transformation results in the first value of the hidden node computations. To allow the model to capture nonlinear relationships, these values are then passed through some nonlinear activation function $\sigma(h_j)$ [1]. This study used the rectified linear activation function (reLu) as it is recommended for most problems, though others such as the sigmoid or hyperbolic tangent function may also be used [1, 8, 9]

$$\sigma(h_j) = \max(0, h_j)$$

The h values are then connected to the output layer by another weight matrix. For classification problems the output layer consists of the Softmax activation function, so the shape of this weight matrix depends on the number of classes in the data set [2, 10]. The output layer receives one final linear summation that transforms the hidden layer results, $\sigma(h_j)$, to raw prediction scores s_k for all K classes. These scores are then passed through the Softmax function to be interpreted as probabilities and the most likely class earns the instance's predicted label $\hat{y}$ [2, 10]

$$\hat{y} = \arg\max_k P(y = k|s_k)$$

where the probabilities are calculated by [8]

$$P(y = k|s_k) = \frac{e^{s_k}}{\sum_{k \in K} e^{s_k}}$$

A SLP network is initialized with random weight values; therefore, the first forward pass possesses no knowledge of the data set. The method by which the network learns is through its backward pass, or back propagation [2, 9]. For a classification problem, back propagation makes use of a cross entropy loss function to tune weight values [2, 8, 9, 10]. This cross entropy loss function is applied to the Softmax probability of the actual label y_i and is defined as the loss for instance L_i [8, 10]

$$L_i = -ln\left(\frac{e^{s_{y_i}}}{\sum_{k \in K} e^{s_k}}\right)$$

If the Softmax probability for y_i is equal to 0.15, for example, then L_i is equal to 1.89. If equal to 1, its loss is 0. In the end, the goal of back propagation is to increase the Softmax probability of y_i. This is achieved by adjusting the network weight matrices, backwards through the network, with the nonlinear optimization method of gradient descent.

The gradient of a function, in this case the cross entropy loss function L, is a vector of partial derivatives that measures the steepness of L at some point [11]. The function is minimized by walking iteratively in the opposite direction of the gradient. A generalized update rule for the gradient descent method where $\vec{w}$ is a vector from the weight matrix, t is the epoch, η is the learning rate, and $\nabla \vec{w} L(w^t)$ is the weight vector gradient is given by [11]

$$\vec{w}^{(t+1)} = \vec{w}^t - \eta \nabla \vec{w} L(\vec{w}^t)$$

The computer SLP implementation performs back propagation automatically and invisibly to the user; however, the learning rate η is a hyper-parameter controlled by two variables. The suite of hyper-parameters considered for the SLP learner consisted of the number of hidden layer nodes, the back propagation solver, the learning rate initialization and method of decay, and the regularization parameter.

The *hidden_layer_sizes* parameter sets the number of nodes implemented in the network's hidden layer. This study explored a range from three to 30 hidden layer nodes. As discussed, this study used the reLu activation function to perform nonlinear transformations in all hidden layer nodes. Other activation functions were not considered by this study.

The back propagation method used to train the network is set in the *solver* parameter. Three values were explored: 'sgd', 'adam', and 'lbfgs'. Stochastic gradient descent, or sgd, is an optimization method where the prediction function f is computed using a small subset of instances at a time [11]. While this method introduces some variation into the model's direction of convergence, overall progress is achieved quicker so long the data is representative of the actual domains [3]. The adam solver is a separate sgd-based optimization method reported to work well on larger data sets with "several thousand instances" [2]. However, it has less user defined control than sgd in the *sklearn* implementation. The last solver, lbfgs, is a memory-efficient optimization solver which uses a greater number training examples than sgd solvers and is claimed to produce better results on smaller data sets [2]. Determining if a data set is large or small as it pertains to learner performance and computational hardware requires some subjectivity, therefore all solvers were evaluated in this study.

The next two hyper-parameters control the model's learning rate η. The first variable, *learning_rate_init* initializes the parameter value. With a default value of 0.001, this study explored an inclusive range above and below the default: (0.0001, 0.0005, 0.001, 0.005, 0.01). The *learning_rate* variable contains three possible values: 'constant', 'invscaling', and 'adaptive', and is only considered when *solver* is sgd [2]. A constant *learning_rate* leaves the initialized value unchanged throughout all gradient steps. The invscaling method calculates an effective learning rate

η' at each epoch t by the following method [2]

$$\eta' = \frac{\eta}{t^{0.5}}$$

The adaptive *learning_rate* maintains a constant η until two consecutive epochs are unable to decrease the loss function by .0001, at which point the most recent learning rate is reduced by 80% [2].

The last hyper-parameter considered for the SLP was α, the regularization term for the model prediction function f. Also called the L2 penalty, this parameter serves to penalize excessively large weight values in f that may lead to an unnecessarily complex model, or one that is over fit [12, 2]. This is achieved by adding the regularization term to the model, which consists of multiplying the sum of all weights in f by α. Accordingly, lower α values permit a more complex model to combat under fitting. With a default α value of 0.001, this study explored an inclusive range above and below the default: (0.0001, 0.0005, 0.001, 0.005, 0.01).

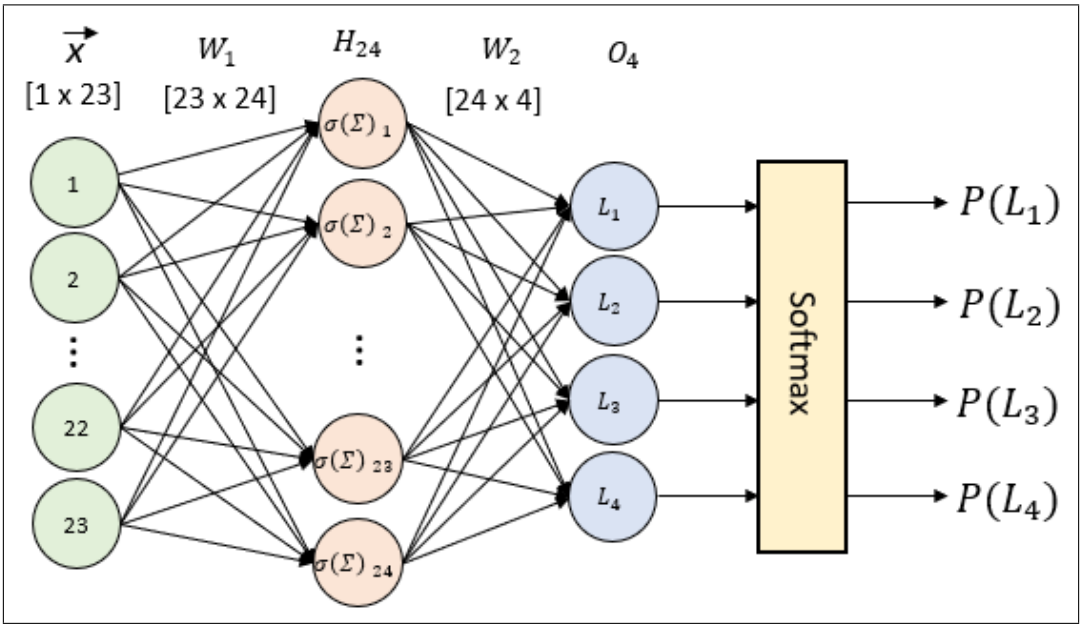

Figure 31. SLP network visualization.

This study tuned the SLP hyper-parameters through 5-Fold CV. The

combination of parameters resulting in the highest mean accuracy consisted of *hidden_layer_sizes*=24, *solver*='adam', *learning_rate_init*=0.005, *learning_rate*='constant', and α=0.0005. With these parameters, a final model fit with all training data produced a test accuracy of 79.4%. The model's confusion matrix is presented in Figure 32. This learner also never predicts the SP algorithm as the best performing algorithm. The SLP appears to contain the best results from both earlier models. It performs comparably well to the DT model in CW and GA predictions, and to the KNN model with SOM predictions.

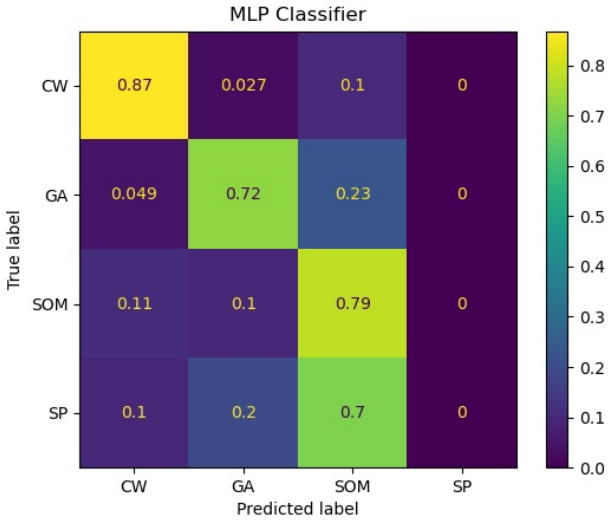

Figure 32. Single Layer Perceptron Model Confusion Matrix

Figure 31 presents a visualization of this study's SLP network. Instances, $\vec{x}$, are presented to the network by their 23 feature values. Then the hidden layer H, with 24 nodes, processes the weighted sums of all input features through W_1 with the reLu activation function. The results are forward fed through W_2 to the output layer O, where every label L in the portfolio is afforded its own node. Finally, these raw scores are fed into the Softmax function to be transformed into a probability distribution, whereby the algorithm with the greatest probability earns the predicted label.

4.2 Model Evaluation

This section reviews the classification models discussed in section 4.1 by metrics consistent with the machine learning and algorithm selection problem literature. These measures include evaluating F-scores, comparing Single-Best-Solver (SBS) and Virtual-Best-Solver (VBS) models, and examining predicted portfolio savings to the CVRP objective function. For convenience, the classification models are summarized in table 2.

Model	**Hyper-parameters**	**Accuracy**
Decision Tree	criterion = entropy, max_depth = 6 Model Selection: 5-Fold CV	76.4%
KNN	$D = 9$ PCs, n_neighbors = 43, weights = distance metric=euclidean Model Selection: LOOCV	75.1%
SLP	hidden_layers_sizes = (24), solver = adam, learning_rate_init= 0.005, learning_rate = constant, α=0.0005 Model Selection: 5-Fold CV	79.4%

Table 2. Final classification models: hyper-parameters and cross validation method.

4.2.1 F-Scores

A model's F-score, or harmonic mean of its precision and recall metrics, is a common and convenient measure of model performance quality [1, 2]. For each label, precision is the ratio of true classifications (TC) over the sum of true classifications and false classifications (FC). Recall is the ratio of true classifications over the sum of true classifications and false negative classifications (FNC).

$$\text{precision} = \frac{TC}{TC + FC}$$

$$\text{recall} = \frac{TC}{TC + FNC}$$

This study compares *weighted* F-scores, which weights label F-scores by their frequency of occurrence in the test set and better represents data with imbalanced

labels [2]. In doing so, the evaluation metric accounts for differences in model performance by class, such as the earlier discussed 30% disparity in GA predictions between the DT and KNN models. F-scores for model labels are given by

$$\text{F-score} = 2\left(\frac{\text{precision} * \text{recall}}{\text{precision} + \text{recall}}\right)$$

Where a value of 1 indicates perfection classification of all instances, the DT F-score is 75.1, the KNN is 73.3, and the SLP is 78.4. By this metric, the DT clearly outperforms the KNN despite their similarities in raw accuracy score. Furthermore, the SLP is considered the best model among all classifiers.

4.2.2 SBS and VBS Performance

The model comparisons described above provide insight to how well each model classified the test set in a pure machine learning context. This section details the models' accuracies compared to SBS and VBS models, which are a pair of metrics presented in the algorithm selection problem literature. The SBS is a simulated model in which the most frequently occurring label in the data set is predicted for all test instances [13, 14]. The VBS, also referred to as oracle performance, is an artificial model that correctly predicts all test instances [13, 14]. These metrics have become standard in modern algorithm selection problems, commonly appearing in competitions with the 'ASlib' algorithm selection library and other emerging fields of study [13, 15].

This study's SBS and VBS would achieve accuracies of 49.8% and 100%, respectively. The SBS would universally predict the CW algorithm in the test set. By this measure, all classification models generated in this study performed well, boosting accuracy between 25% and 30% beyond the naive SBS. This indicates all models learned effectively from the data before making predictions. As all models fall short of the VBS, it remains possible some improvements exist. However, a gap

between learned models and the VBS alone does not ensure significant improvements can still be made. Learners aim to generalize data domains by reducing loss, not eliminating loss altogether. As presented in section 3.4, the decision boundaries are quite complex for this study's data set and seeking a perfect classification in practice may be fleeting.

4.2.3 CVRP Cost Savings

This section details how each of the classifiers performed in terms of CVRP cost savings. For each of the five models (DT, KNN, SLP, SBS, and VBS), the total cost (i.e. fleet distance travelled) incurred from predictions in the test set was calculated. This resulted in a single real number for every model. Then, these sums were compared to one another as ratios. These ratios are presented below in table 3, where a value greater than one indicates the model in the column produced a lower total cost.

Ratio/Model	DT	KNN	SLP	SBS	VBS
DT:Model	1	1.00011	1.000769	0.990262	1.003432
KNN:Model	0.99989	1	1.000659	0.990154	1.003322
SLP:Model	0.999232	0.999341	1	0.989501	1.002661
SBS:Model	1.009834	1.009944	1.01061	1	1.013299
VBS:Model	0.99658	0.996689	0.997346	0.986875	1

Table 3. Summary of model CVRP cost ratios.

The margins separating models by CVRP cost savings are remarkably thin, which should be expected from a competitive algorithm portfolio. This metric provides a different means of evaluation than those presented earlier. For example, as one may recall from table 2, KNN produced the lowest accuracy (and F-score) of all three classifiers. However, its overall cost to the test instances is better (i.e. lower) than the DT. This suggests KNN more accurately classified instances of greater magnitude than the DT despite its lower accuracy/F-score. By this

metric, KNN outperformed the DT.

Also of note, all learned models improved beyond the SBS model. This further suggests the learners exploited patterns in the data before making predictions. In summary, the SLP out performs all others with the obvious exception of the VBS. Notwithstanding, the SLP is within a mere 0.33% of the VBS model despite 20% less accurate predictions.

4.3 Problem Space Adequacy

Since machine learning algorithms are customarily applied to observational data, such as the data used in this study, the natural question arises regarding the adequacy of its size to enable generalization [6]. Machine learning algorithms are inductive learners, so increasing the data set size can improve classification only to a point seeing as "fixed-size learners can only take advantage of so much data" [6]. Accordingly, one should expect the models produced by a learning algorithm to have an upper bound on accuracy such that providing even more data to the learner does not appreciably improve results.

This study hypothesized 5000 CVRP instances would suffice for learner generalization of the data domain encoded by the creation process outlined in section 2.1.1. Evaluating this element of the study is of particular interest given the time complexity and computational expense of generating the instance labels. Exploring this aspect of the study could be beneficial for identifying learner limitations and/or recognizing the need to increase the problem space for future studies.

To evaluate the learners' performance as a function of data size, a subset of instances D_i were sampled from the data set, where i started at 25 and increased to 4897 over 500 equally spaced intervals. For each D_i the learner was tuned using cross validation to identify its best hyper-parameters. Then, a full model was fit with all training data using the hyper-parameters resulting in the highest

average cross validation accuracy and evaluated on the withheld test data. These models also employed an 80/20 split for training and testing, respectively. This process occurred for each learner, with the results presented below. This method is inherently greedy and limited by cross validation; however, it should facilitate useful insight for how and when the learners respond to the addition of new data.

This element of the research was made possible through high performance computing. While the DT model only took about four hours of processing time on a Dell XPS-13 with 16 GB of RAM and two cores, the KNN and SLP models required approximately 96 hours and 192 hours on a Dell PowerEdge R7425 with 512 GB of RAM and 128 cores.

4.3.1 Decision Tree Generalization

The decision tree learner, shown by Figure 33, produced interesting results. The smallest data sets actually resulted in the highest accuracy, though with the greatest amount of variation. While this came as a surprise, it can be justified. This graph suggests there are more factors to consider than simply the size of the data to influence the prediction accuracy of its best performing model. Likely, these factors include the distribution of feature values and labels in D_i. Since the decision tree is a greedy algorithm, always choosing the locally optimal split, it may perform exceedingly well on a small, easy-to-classify subset of data. The decision boundaries may be quite wide and, unless one of the more challenging instances close to the actual boundary happen to be in D_i, the classification can be perfect. The likelihood of this occurring rapidly depreciates as the number of instances increase because the range of feature values grow more diverse.

The learner begins to smooth around 3200 instances, beyond which test accuracy does not significantly increase but variation does reduce. After 4000 instances the variation reduces even further as the graph appears to center around a mean

accuracy of 76%. While this is not a textbook-level quality of an upper bound, this learner appears to respond to the information provided by new instances and generalizes well. It is the opinion of this study that adding more instances may further smooth the learner's performance, but likely not improve its accuracy.

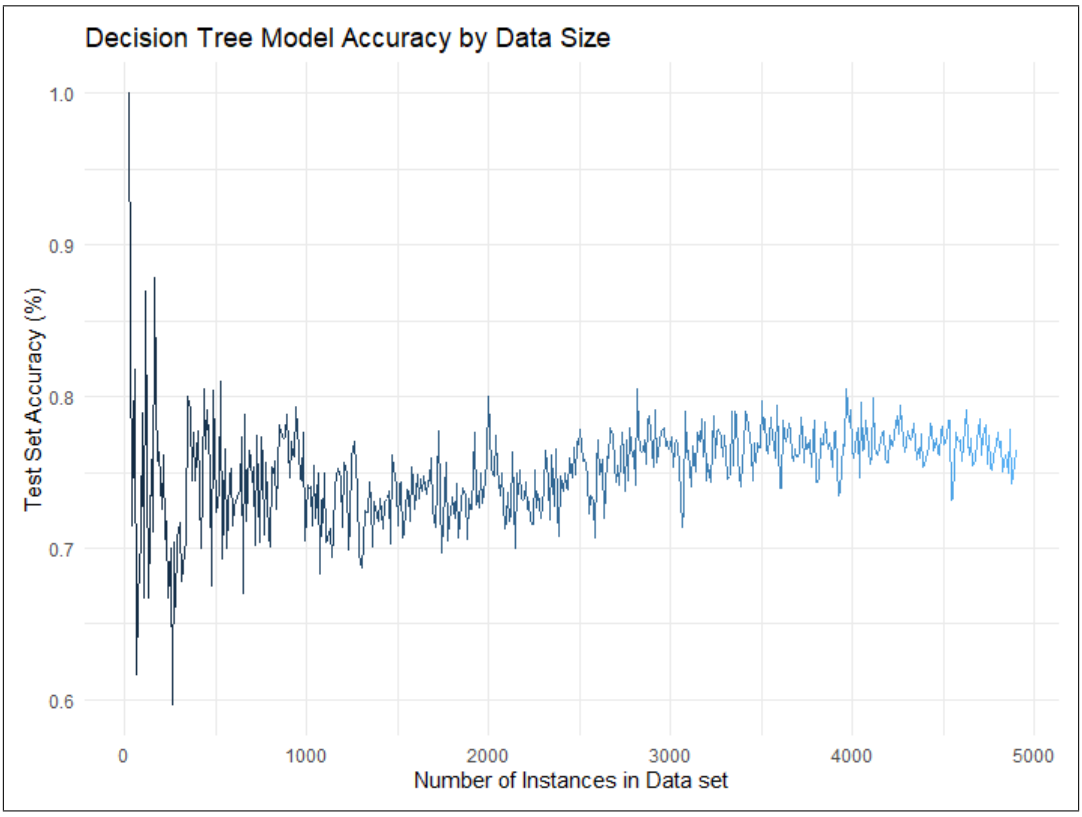

Figure 33. Evaluation of Decision Tree accuracy by the number of instances provided to train and test.

4.3.2 K-Nearest Neighbor Generalization

The KNN learner, shown in Figure 34, produced contrasting results to the decision tree in smaller data sets. The shape of this graph is akin to the machine learning literature which suggests that larger data sets can improve prediction accuracy [6]. The largest variation in the graph still exists in the smaller D_i, with test accuracy ranging from below 40% to nearly 90%. This too should be expected since KNN is an instance-based learner that simply predicts the most frequently

occurring label in the k closest training instances. The smallest data sets are the most sensitive to the addition of new instances.

KNN begins to smooth earlier than the decision tree at around 2800 instances. Beyond this point the test accuracy is not significantly increased and the variation appears to stabilize in the remainder of samples. Like the decision tree, this learner makes good use of additional data and appears to center around 76% accuracy. KNN may also be smoothed by additional instances beyond those provided in this study, though likely not experience a boost in accuracy.

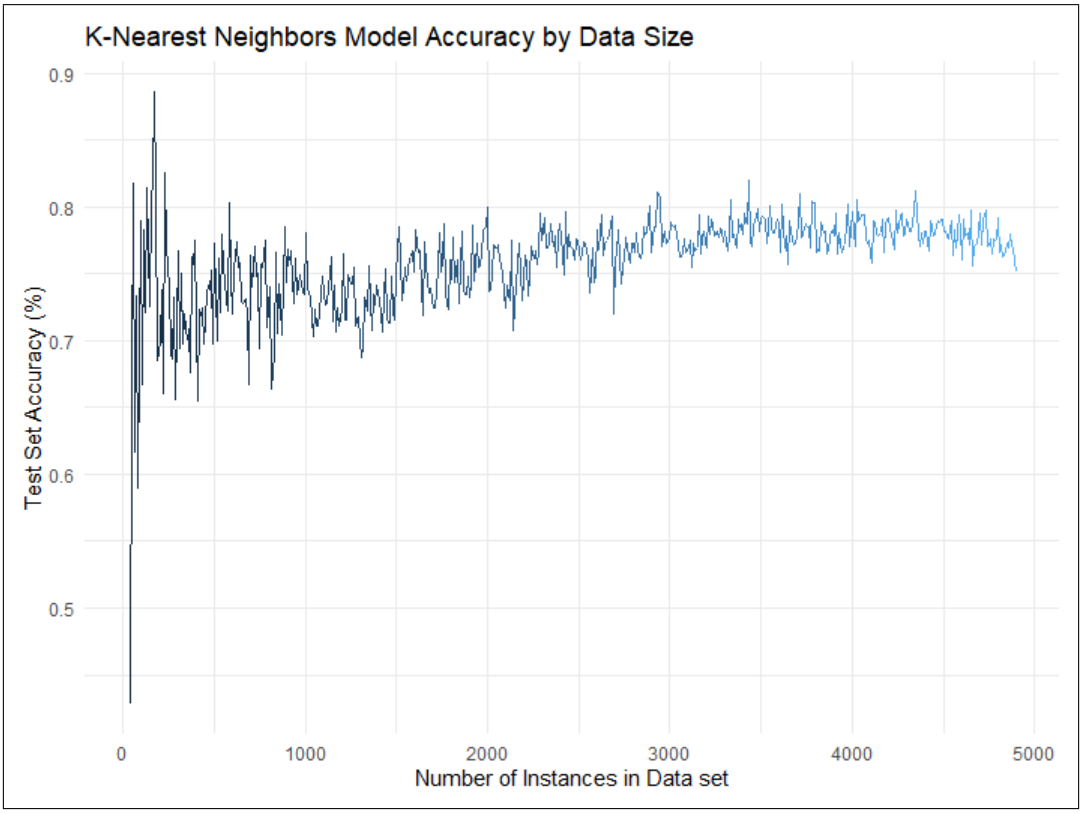

Figure 34. Evaluation of KNN accuracy by the number of instances provided to train and test.

4.3.3 Single-Layer Perceptron Generalization

The SLP, like both the DT and KNN, experiences the majority of variation in the smaller sets of D_i. However, the SLP may generalize sooner than the other

two learners. As the graph approaches 2200 instances the resulting models appear within 5% of the accuracy produced from the full D. By contrast, this same threshold requires approximately 3500 instances for the DT and KNN learners. This difference could be explained by the learner's parametric methods that allow for complex nonlinear decision boundaries.

The variation in the SLP graph reduces nicely beyond 4000 instances. Here, the models perform within 2.5% of the full model accuracy. Like the other two learners, the SLP certainly adapts to new instances to improve generalization. Based on the shape of this curve, this study finds it unlikely that additional instances would notably improve accuracy; however, the variation between models may be reduced. This learner appears to center around 80% accuracy.

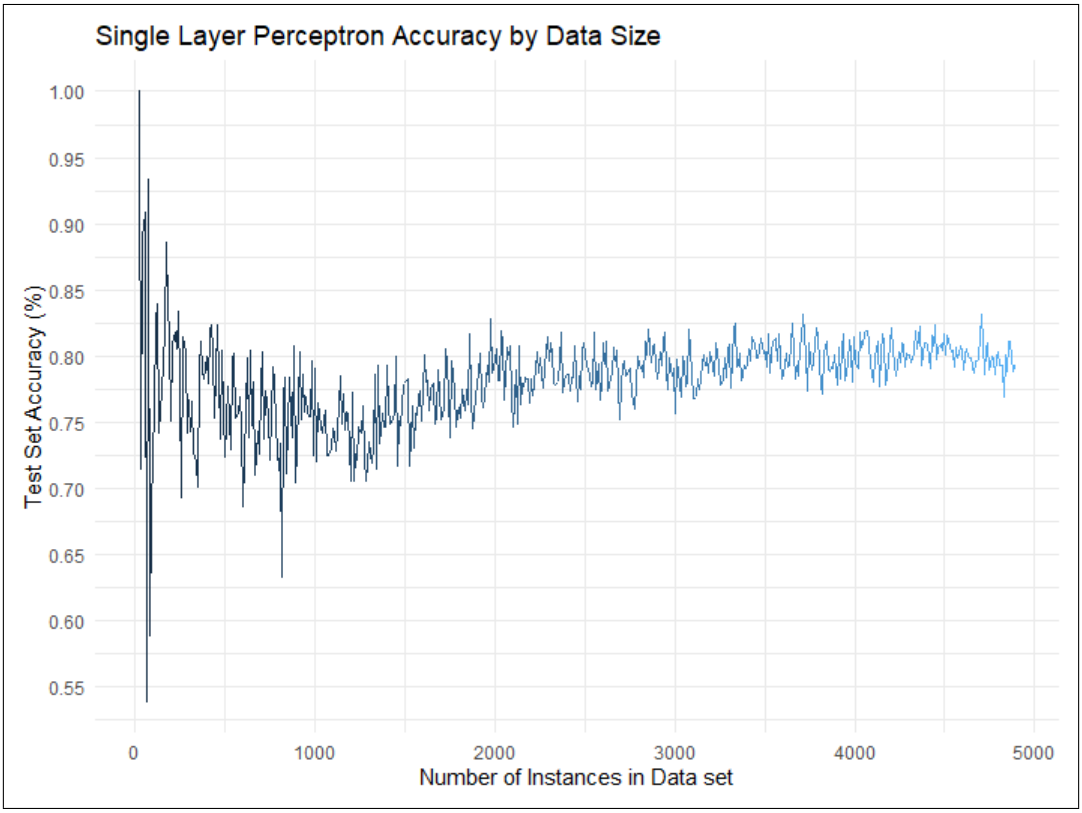

Figure 35. Evaluation of SLP accuracy by the number of instances provided to train and test.

List of References

[1] H. Daume, "A course in machine learning," 2017, unpublished. [Online]. Available: http://ciml.info/

[2] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[3] M. Alvarez, "Decision trees," September 2020.

[4] T. Mitchell, *Machine Learning*. New York: McGraw Hill, 1997.

[5] M. Alvarez, "k-nearest-neighbors," October 2020.

[6] P. Domingos, "A few useful things to know about machine learning," *Communications of the Association for Computing Machinery*, vol. 55, pp. 78–87, 2012.

[7] M. E. Kutner, C. J. Nachtsheim, J. Neter, and W. Li, *Applied Linear Statistical Models*. New York: McGraw Hill, 2005.

[8] F.-F. Li, J. Johnson, and S. Yeung, "Lecture 3: Loss functions and optimization," April 2019.

[9] R. Zemel, R. Urtasun, and S. Fidler, "Csc 411: Lecture 10: Neural networks i," November 2020.

[10] M. Alvarez, "Multinomial logistics regression," November 2020.

[11] M. Alvarez, "Gradient descent," October 2020.

[12] M. Alvarez, "Linear regression," November 2020.

[13] M. Lindauer, J. N. v. Rijn, and L. Kotthoff, "The algorithm selection competitions 2015 and 2017," *Artificial Intelligence*, vol. 272, pp. 86–100, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S000437021830198X

[14] L. Kotthoff, P. Kerschke, H. Hoos, and H. Trautmann, "Improving the State of the Art in Inexact TSP Solving using Per-Instance Algorithm Selection," in *LION 9*, 2015, pp. 202–217.

[15] J. Beel and L. Kotthoff, "Proposal for the 1st Interdisciplinary Workshop on Algorithm Selection and Meta-Learning in Information Retrieval (AMIR)," in *Advances in Information Retrieval*, L. Azzopardi, B. Stein, N. Fuhr, P. Mayr, C. Hauff, and D. Hiemstra, Eds. Cham: Springer International Publishing, 2019, pp. 383–388.

CHAPTER 5

Conclusion and Future Work

This chapter concludes the current research and recommends areas to focus future work. In summary, this research formulated and solved the algorithm selection problem for the CVRP as a machine learning classification problem. Given the computational complexity of the CVRP, and global research investment to produce well performing heuristics, the significance of this study lay within the methods used to produce well performing algorithm selection models.

The observations presented here primarily address the essential algorithm selection problem ingredients presented in chapter 1. For convenience, this list is provided below:

1. Problem Space shall consist of instances of assorted difficulties;
2. Feature Space must suitably characterize instance properties;
3. Algorithm Space should contain diverse algorithms for solving instances;
4. The performance metric appropriately evaluates the algorithms' results.

The instance creation process outlined in section 2.1.1 targeted the first requirement. Exploring the problem space in section 3.2 through a series of histograms and descriptive statistics implies this element was satisfied. However, there is room for improvement and expansion. The inclusion of additional modules may further diversify the problem space and present new challenges to heuristic solvers. An example may include a new city positioning module whereby cities are split between the 'Equidistant' and the 'Random' and/or 'Random-Cluster' module. This may allow for greater diversity in features such as '6.FracDistinctDist' and '7.SD_NNDist'.

This study pursued the second requirement by using the 23 features discussed in section 2.2. These features, both in raw and compressed forms, proved capable of representing instances in vector form to machine learning algorithms. Supported by feature-to-label plots in section 3, some of these features can even indicate patterns in algorithm performance before any modeling is approached. Made possible through PCA, the features also allow limited visualization of the complex decision boundaries among instances and algorithms. Dimensionality reduction methods through compression techniques such as PCA or Linear Discriminant Analysis should remain a factor in future studies. This is especially true if any distance-based algorithms, whether supervised or unsupervised, are used for modeling.

The algorithm portfolio, composed of two classical heuristics and two evolutionary algorithms, was designed to solve instances with a diverse array of methods. The modeling results, punctuated by a 30% boost in accuracy between the best performing model and a simulated Single-Best-Solver model, suggest the existing portfolio fulfilled this requirement. However, the SP algorithm rarely appeared as the best performing algorithm in the entire data set and was never predicted correctly by any model. This component of the study likely has the most immediate room for future work.

For the classical heuristics, the SP may be improved. As discussed in section 2.3.1, the algorithm's first phase clusters cities by the relation of their coordinates in polar form, notwithstanding violations of vehicular capacity. Then, its second phase solves all clusters as individual Travelling Salesman Problems (TSP) using the 2-Opt heuristic. In the future, this algorithm may be improved in its second phase by substituting or incorporating the 2-Opt with some other approximate or exact method.

The evolutionary algorithms may also be tuned for higher performance, albeit

at the expense of greater run time. For the GA, users may increase the number of generations and/or population size. The number of replications may be increased for the SOM. There are multiple ways to estimate how altering these user-defined parameters may influence performance in the problem space. One method is to simply re-run the entire set of instances. However, this may prove computationally burdensome. Alternatively, a limited set of problems comprised of those instances where the algorithms performed poorly in this study can be explored. For the GA this may include instances where the number of nodes exceeds 50 or the minimum number of trucks is greater than 10, for example. Inferences can be made from the data analysis presented in chapter 3 for this approach. Another method is to use the best cost solution from the algorithm portfolio generated in this study to index a termination condition on an arbitrary instance, and then apply the discovered parameters to a broader set of instances.

The performance metric used in this study, CVRP solution quality, served its purpose well for implementing the algorithm selection problem. Other metrics, such as run time, may be considered in the future; however, run time is impacted by more than just instance properties. Other factors include the manner and language in which the heuristic is programmed and the hardware used for processing instances. Streamlining the algorithm portfolio to a centralized processor could be an important undertaking if reporting on run time is desired in the future.

While the three learners used this study produced good results for the classification problem, there may be other learners capable of improving accuracy. Lastly, using CVRP cost savings as a performance metric can lead to new classification methods by weighting those instances with a greater savings as more important to classify.

Appendix A: Instance Types in Problem Space

	Inst_Type	Final	Removed		Inst_Type	Final	Removed
0	CCLvlr	58	0	**42**	ERLvlr	65	0
1	CCLvsr	71	0	**43**	ERLvsr	56	0
2	CCMsfl	41	9	**44**	ERMsfl	51	7
3	CCQ	55	0	**45**	ERQ	70	0
4	CCSvlr	53	0	**46**	ERSvlr	55	0
5	CCSvsr	59	0	**47**	ERSvsr	72	0
6	CCU	78	0	**48**	ERU	68	0
7	CELvlr	42	0	**49**	ERcLvlr	58	0
8	CELvsr	66	0	**50**	ERcLvsr	62	0
9	CEMsfl	55	11	**51**	ERcMsfl	52	12
10	CEQ	65	0	**52**	ERcQ	59	0
11	CESvlr	67	0	**53**	ERcSvlr	64	0
12	CESvsr	60	0	**54**	ERcSvsr	59	0
13	CEU	45	0	**55**	ERcU	73	0
14	CRLvlr	51	0	**56**	RCLvlr	70	0
15	CRLvsr	64	0	**57**	RCLvsr	61	0
16	CRMsfl	51	9	**58**	RCMsfl	47	6
17	CRQ	62	0	**59**	RCQ	50	0
18	CRSvlr	73	0	**60**	RCSvlr	55	0
19	CRSvsr	60	0	**61**	RCSvsr	51	0
20	CRU	65	0	**62**	RCU	49	0
21	CRcLvlr	48	0	**63**	RELvlr	63	0
22	CRcLvsr	62	0	**64**	RELvsr	51	0
23	CRcMsfl	60	10	**65**	REMsfl	43	7
24	CRcQ	65	0	**66**	REQ	59	0
25	CRcSvlr	49	0	**67**	RESvlr	57	0
26	CRcSvsr	68	0	**68**	RESvsr	52	0
27	CRcU	66	0	**69**	REU	75	0
28	ECLvlr	60	0	**70**	RRLvlr	63	0
29	ECLvsr	43	0	**71**	RRLvsr	55	0
30	ECMsfl	56	4	**72**	RRMsfl	51	9
31	ECQ	55	0	**73**	RRQ	51	0
32	ECSvlr	65	0	**74**	RRSvlr	64	0
33	ECSvsr	59	0	**75**	RRSvsr	57	0
34	ECU	51	0	**76**	RRU	54	0
35	EELvlr	59	0	**77**	RRcLvlr	51	0
36	EELvsr	71	0	**78**	RRcLvsr	72	0
37	EEMsfl	49	9	**79**	RRcMsfl	51	10
38	EEQ	40	0	**80**	RRcQ	67	0
39	EESvlr	57	0	**81**	RRcSvlr	56	0
40	EESvsr	51	0	**82**	RRcSvsr	62	0
41	EEU	62	0	**83**	RRcU	59	0

BIBLIOGRAPHY

Abdelatti, M. F. and Sodhi, M. S., "An improved gpu-accelerated heuristic technique applied to the capacitated vehicle routing problem," in *Genetic and Evolutionary Computation Conference*, July 2020.

Alvarez, M., "Decision trees," September 2020.

Alvarez, M., "Gradient descent," October 2020.

Alvarez, M., "k-nearest-neighbors," October 2020.

Alvarez, M., "Linear regression," November 2020.

Alvarez, M., "Multinomial logistics regression," November 2020.

Anton, H. and Rorres, C., *Elementary Linear Algebra*. Drexel University: John Wiley and Sons, 2000.

Beel, J. and Kotthoff, L., "Proposal for the 1st Interdisciplinary Workshop on Algorithm Selection and Meta-Learning in Information Retrieval (AMIR)," in *Advances in Information Retrieval*, Azzopardi, L., Stein, B., Fuhr, N., Mayr, P., Hauff, C., and Hiemstra, D., Eds. Cham: Springer International Publishing, 2019, pp. 383–388.

Clarke, G. and Wright, J., "Scheduling of vehicles from a central depot to a number of delivery points," *Operations Research*, vol. 12, pp. 568–581, 1964.

Dantzig, G. and Ramser, J. H., "The truck dispatching problem," *Management Science*, vol. 6, pp. 80–91, 1959.

Daume, H., "A course in machine learning," 2017, unpublished. [Online]. Available: http://ciml.info/

Deisenroth, M. P., Faisal, A. A., and Ong, C. S., *Mathematics for Machine Learning*. University College, Lond: Cambridge Press, 2020.

Dilek Tuzun, Michael A. Magent, L. I. B., "Selection of vehicle routing heuristic using neural networks," *International Transactions in Operational Research*, vol. 4, pp. 211–221, 1997.

Domingos, P., "A few useful things to know about machine learning," *Communications of the Association for Computing Machinery*, vol. 55, pp. 78–87, 2012.

Durbin, R. and Willshaw, D., "An analogue approach to the travelling salesman problem using an elastic net method," *Nature*, vol. 326, no. 6114, pp. 689–691, 1982.

What is ArcMap?, Environmental Science Research Institute, jan 2016.

Fischetti, M., Toth, P., and Vigo, D., "A branch-and-bound algorithm for the capacitated vehicle routing problem on directed graphs," *Journal of Quality Measurement and Analysis*, vol. 42, pp. 846–859, 1994.

Ghaziri, H., "Supervision in the self-organizing feature map: Application to the vehicle routing problem," in *Meta-Heuristics*. Springer, 1996, pp. 651–660.

Ghaziri, H. and Osman, I. H., "Self-organizing feature maps for the vehicle routing problem with backhauls," *Journal of Scheduling*, vol. 9, no. 2, pp. 97–114, 2006.

Ghaziri, H. E., "Solving routing problems by a self-organizing map," in *Artificial Neural Networks, 1991, International Conference on Artificial Neural Networks*. ICANN, 1991, pp. 829–834.

Gillett, B. E. and Miller, L. R., "A heuristic algorithm for the vehicle-dispatch problem," *Operations research*, vol. 22, no. 2, pp. 340–349, 1974.

Holland, J., *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. Ann Arbor, Michigan: University of Michigan Press, 1975.

Jolliffe, I. T., *Principal Component Analysis*. King's College, Aberdeen, United Kingdom: Springer, 1986.

Kendall E. Nygard, Paul Juell, N. K., "Neural networks for selective vehicle routing heuristics," *ORSA Journal on Computing*, vol. 2, pp. 353–364, 1990.

Kohonen, T., "Clustering, taxonomy, and topological maps of patterns," in *Proceedings of the Sixth International Conference on Pattern Recognition*. Silver Spring, MD: IEEE Computer Society Press, 1982, pp. 114–128.

Kohonen, T., "Self-organized formation of topologically correct feature maps," *Biological Cybernetics*, vol. 43, no. 1, pp. 59–69, 1982.

Kothoff, L., "Algorithm selection for combinatorial search problems: A survey," *Artificial Intelligence Magazine*, vol. 35, pp. 48–60, 2014.

Kothoff, L., "Algorithm selection in practice," *Artificial Intelligence and Simulation of Behaviour Quarterly*, vol. 138, pp. 4–8, 2014.

Kotthoff, L., Kerschke, P., Hoos, H., and Trautmann, H., "Improving the State of the Art in Inexact TSP Solving using Per-Instance Algorithm Selection," in *LION 9*, 2015, pp. 202–217.

Kutner, M. E., Nachtsheim, C. J., Neter, J., and Li, W., *Applied Linear Statistical Models*. New York: McGraw Hill, 2005.

Laporte, G., Gendreau, M., Potvin, J.-Y., and Semet, F., "Classical and modern heuristics for the vehicle routing problem," *International transactions in operational research*, vol. 7, no. 4-5, pp. 285–300, 2000.

Li, F.-F., Johnson, J., and Yeung, S., "Lecture 3: Loss functions and optimization," April 2019.

Lin, S. and Kernighan, B., "An effective heuristic algorithm for the traveling salesman problem," *INFORMS*, vol. 22, no. 2, p. 498–516, 1973.

Lindauer, M., Rijn, J. N. v., and Kotthoff, L., "The algorithm selection competitions 2015 and 2017," *Artificial Intelligence*, vol. 272, pp. 86–100, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S000437021830198X

Martin Ester, Hans-Peter Kriegel, J. S. and Xu, X., "A density-based algorithm for discovering clusters in large spatial databases with noise," *KDD*, vol. 96, pp. 226–231, 1996.

Matsuyama, Y., "Self-organization via competition, cooperation and categorization applied to extended vehicle routing problems," in *Neural Networks, 1991., IJCNN-91-Seattle International Joint Conference on*, vol. 1. IEEE, 1991, pp. 385–390.

Mitchell, T., *Machine Learning*. New York: McGraw Hill, 1997.

Modares, A., Somhom, S., and Enkawa, T., "A self-organizing neural network approach for multiple traveling salesman and vehicle routing problems," *International Transactions in Operational Research*, vol. 6, no. 6, pp. 591–606, 1999.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E., "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

Rice, J. R., "The algorithm selection problem," Dept. of Computer Science, Purdue University, West Lafayette, IN, Tech. Rep. 75-152, 1975.

Schwardt, M. and Dethloff, J., "Solving a continuous location-routing problem by use of a self-organizing map," *International Journal of Physical Distribution & Logistics Management*, vol. 35, no. 6, pp. 390–408, 2005.

Smith-Miles, K., "Cross disciplinary perspectives on meta-learning for algorithm selection," *Association for Computing Machinery Computing Surveys*, vol. 41, pp. 6:1–25, 2008.

Smith-Miles, K., van Hemert, J., and Lim, X. Y., "Understanding tsp difficulty by learning from evolved instances," *Learning and Intelligent Optimization*, pp. 266–280, 2010.

Steinhaus, M., Shirazi, A. N., and Sodhi, M., "Modified self organizing neural network algorithm for solving the vehicle routing problem," in *2015 IEEE 18th International Conference on Computational Science and Engineering*, 2015, pp. 246–252.

Steinhaus, M. K., "The application of the self organizing map to the vehicle routing problem," Ph.D. dissertation, University of Rhode Island, 2015, paper 383.

Suthikarnnarunai, N., "A sweep algorithm for the mix fleet vehicle routing problem," *International MultiConference of Engineers and Computer Scientists*, vol. 2, pp. 1914–1919, 2008.

Uchoa, E., Pecin, D., Pessoa, A., Marcus Poggi, A. S., and Vidal, T., "New benchmark instances for the capacitated vehicle routing problem," *European Journal of Operational Research*, vol. 257, pp. 848–858, 2017.

Wolpert, D. H. and Macready, W. G., "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computatation*, vol. 1, pp. 67–82, 1997.

Yeun, L. C., Ismail, W. R., Oman, K., and Zirour, M., "Vehicle routing problem: Models and solutions," *Journal of Quality Measurement and Analysis*, vol. 4, pp. 205–218, 2008.

Zemel, R., Urtasun, R., and Fidler, S., "Csc 411: Lecture 10: Neural networks i," November 2020.