NEURAL NETWORK ESTIMATORS FOR OPTIMAL TOUR LENGTHS OF TSP INSTANCES WITH ARBITRARY NODE DISTRIBUTIONS

A Thesis

by

Taha Varol

Submitted to the
Graduate School of Sciences and Engineering
In Partial Fulfillment of the Requirements for
the Degree of

Master's Degree

in the
Department of Data Science

Özyeğin University
August 2022

Copyright © 2022 by Taha Varol

NEURAL NETWORK ESTIMATORS FOR OPTIMAL TOUR LENGTHS OF TSP INSTANCES WITH ARBITRARY NODE DISTRIBUTIONS

Approved by:

Professor Okan Örsan Özener, Advisor
Department of Industrial Engineering
Özyeğin University

Assist. Professor İhsan Yanıkoğlu
Department of Industrial Engineering
Özyeğin University

Assist. Professor Erinç Albey,
Co-Advisor
Department of Industrial Engineering
Özyeğin University

Professor Mehmet Güray Güler
Department of Industrial Engineering
Yıldız Technical University

Date Approved: 11 August 2022

Professor Ali Ekici
Department of Industrial Engineering
Özyeğin University

To all humanity

ABSTRACT

To achieve operational efficiency in logistics, we need to solve complex routing problems. Due to their complexity, these problems are often solved sequentially, i.e., using cluster-first route-second (CFRS) type frameworks. However, such two-phase frameworks generally suffer from sub-optimality arising from the first phase. To mitigate this sub-optimality, information about optimal tour lengths of potential clusters can be exploited first, thereby transforming this two-phase approach into a less myopic solution framework. In that aspect, a quick and highly accurate Traveling Salesperson Problem (TSP) tour length estimator can be utilized for searching high-quality clusters. Motivated by this, we propose novel and computationally efficient neural network-based optimal TSP tour length estimators. Our approach uses an entirely new feature set consisting of node level, instance level, and solution level features by combining the power of artificial neural networks and theoretical knowledge in the routing domain. This data and knowledge hybridization enables us to achieve predictions with less than 0.7 percent deviation (on average) from the optimality. Unlike previous studies, we design and use new instances mimicking real-life logistics networks and morphologies. These instance characteristics introduce a substantial computational cost, making our instances harder to solve. To cope with these pathologies, we devise a new and efficient way of finding lower bounds and partial solutions to TSP later to be used as solution-level predictors. We also conduct a computational study where we produce up to 100 times lower prediction error on out-of-distribution test instances. Finally, we develop an enumeration-like mechanism by incorporating proposed machine learning models and metaheuristics to solve massive-scale routing problems efficiently. We significantly outperform the state-of-the-art solver in

terms of solution time and quality, demonstrating the potential of our models and the proposed method.

ÖZETÇE

Lojistikte operasyonel verimlilik için, karmaşık rotalama problemlerini çözmeye ihtiyaç duymaktayız. Bu problemlerin karmaşıklığından kaynaklı olarak, genelde önce-kümele sonra-rotala (ÖKSR) benzeri sıralı çözüm yöntemleri kullanılmaktadır. Fakat, bu tip iki fazlı çözüm yöntemlerinden elde edilen sonuçlar genelde alt-optimallikten muzdarip olmaktadır. Buradaki alt-optimalliği azaltmak adına, yaratılan kümelere ait optimal tur uzunlukları bilgilerinden faydalanılabilir. Bu sayede, bahsedilen iki fazlı çözüm yöntemi daha az miyopik bir çözüm yöntemine dönüştürülmüş olur. Bu bakımdan, çabuk ve yüksek isabetli bir Gezgin Satıcı Problemi (GSP) tur uzunluğu tahminleyicisi yüksek kaliteli kümeleri aramak amacıyla kullanılabilir. Buradan yola çıkarak, yeni ve hesaplama bakımından verimli, yapay sinir ağı temelli optimal GSP tur uzunluğu tahminleyicileri öneriyoruz. Yaklaşımımız, yapay sinir ağlarının gücünü rotalama alanındaki teorik bilgi ile birleştirerek düğüm seviyesi, örnek seviyesi ve çözüm seviyesi özniteliklerinden oluşan tamamen yeni bir öznitelik seti kullanmaktadır. Bu veri ve alan bilgisi hibridizasyonu, yüzde 0.7'den (ortalamada) daha az sapma ile en iyi tur uzunluğu tahminlemesine olanak sağlamaktadır. Önceki çalışmalardan farklı olarak, gerçek dünya lojistik ağ ve morfolojilerini örnek alan yeni tip örnekler tasarlayıp kullanıyoruz. Bu yeni tip örneklerin bazı karakteristik özellikleri hesaplama bakımından ciddi maliyetleri beraberinde getirerek optimal çözümlerin elde edilmesini zorlaştırmaktadır. Bu tip problem patolojileri ile başa çıkmak adına optimal GSP çözümüne ait daha sonra çözüm seviyesinde öznitelikler olarak kullanmak üzere alt sınır ve kısmi çözümler bulan yeni ve verimli bir yöntem geliştiriyoruz. Ek olarak, en iyi makine öğrenmesi modellerine göre dağılım dışı problem örneklerinde 100 kata kadar daha az tahminsel hata yaptığımızı gösteren bir hesaplamalı çalışma yapıyoruz.

Son olarak, önerilen makine öğrenmesi modellerini metasezgisel yöntemlerle birleştirerek çok büyük rotalama problemlerini etkili bir şekilde numaralandırma benzeri bir mekanizma ile çözülmesini sağlayan yöntem geliştiriyoruz. Geliştirilen yöntem en gelişmiş çözücüye kıyasla hem çözüm kalitesi hem de çözüm süresi bakımından ciddi şekilde daha iyi performans göstererek önerilen modellerin ve önerilen yöntemlerin potansiyellerini ortaya koymaktadır.

ACKNOWLEDGEMENTS

First and foremost, I would like to pay my sincerest gratitude to my advisor Okan Örsan Özener for his valuable time, guidance, patience, and mentorship during our countless research and thesis meetings. He has become a real role model for me since I started my academic journey. Second, I would like to thank my co-advisor Erinç Albey for his endless support and recommendations that make this thesis possible. It has been a great pleasure and experience working under his supervision.

I am thankful to my committee members Ali Ekici, İhsan Yanıkoğlu, and Mehmet Güray Güler for agreeing and reviewing this thesis. My special thanks go to Mehmet Önal and Hasan Sözer for giving me opportunities for doing research projects with them.

I would also thank my graduate student colleagues and future research collaborators Milad Elyasi and Taha Huzeyfe Aktaş for their support during my studies.

I want to thank every member of the Varol family for believing in and motivating me in every moment of my life.

Lastly, I am thankful for "BİDEB 2210-A Genel Yurtiçi Lisansüstü Burs Programı" and "ARDEB-1001 119M278 and 120E488" for financially supporting me during my graduate studies.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Traveling Salesperson Problem (TSP) is one of the most widely studied problems in computer science, operations research, and applied mathematics due to its vast application areas, such as gene mapping [1, 2], protein function prediction [3], chip design [4], 3D printing [5], and logistics [6, 7]. TSP aims to find the shortest closed tour that visits each node of a given network only once. Despite its easy and straightforward definition, TSP is an NP-Complete problem [8], and there is no known solution method to solve a TSP instance in polynomial-time. TSP could simply be formulated using Integer Programming (IP) and solved by exact methods such as branch-and-cut algorithm [9]. Unfortunately, exact methods require an extended computation time as the size of the TSP instance increases.

Problems that require solutions to batches of TSPs such as Vehicle Routing Problem (VRP) and Inventory Routing Problem (IRP) could be so complex that it may not be possible to solve these problems optimally even when these problems are small in size. Usually, a sequential method such as cluster-first route-second is used to obtain good solutions. In cluster-first route-second framework, the first phase entails determining "good clusters" and the second stage entails identifying the best route for each of the generated cluster. Unsurprisingly, this framework suffers from sub-optimality due to its myopic nature in the first phase, generating/selecting the clusters while not knowing their final routing cost/time. To mitigate this sub-optimality, information/estimation about the optimal tour lengths of the potential clusters can be exploited in the first phase, transforming this two-phase approach into a less myopic solution framework. In that aspect, a TSP tour length estimator with low prediction

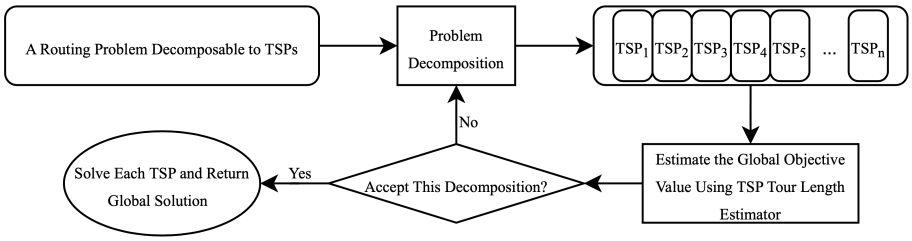


Figure 1: An Example Solution Framework for Cluster-First Route-Second Heuristic with TSP Tour Length Estimator Backbone

error and fast execution time can be utilized to search for high-quality clusters. The backbone of such solution framework could be fast and accurate estimation method that provides valuable insights into the objective function values of TSPs without solving them at all. Such estimation based approach is used as a global problem objective value estimator through the estimation of sub problems (See Figure 1). Considering the computational cost of solving large TSP instances, VRP and IRP with larger TSPs could be a good fit for cluster-first route-second framework with optimal TSP tour length estimator backbone.

In general, when finding an optimal solution is not possible under a time limit, or there is a need to solve routing-based problems repeatedly, optimal tour length estimators could dramatically reduce computational costs. There are many applications where TSP tour length estimators have the potential to increase computational efficiency. For example, in Mobile Facility Routing Problem (MFRP), which is used for humanitarian relief logistics and cellular communications, there is a need for continuous relocation of facilities and solving the resulting routing problems [10]. Another example is Location Routing Problem (LRP), which has many real-world applications in designing large distribution systems, such as newspaper distribution systems [11] and petroleum distribution systems [12]. In LRP, there is a simultaneous search for the best locations for plants, the optimal assignment of customers to plants, and the related minimum cost routes to serve customers. In both MFRP and LRP (or similar

problems), our proposed framework using effective tour length estimations in the clustering phase could provide good clusters, resulting in high-quality solutions. Thus, using such estimations in the clustering phase provides a more integrated approach for solving such problems.

To solve different routing problems, several studies use estimation of TSP tour lengths. For example, [13] used a tour building model for service regions of different shapes to solve vehicle routing problems for a newspaper distribution system. [14] used optimal tour length estimation to assess different distribution strategies under varying demand and node distributions. [15] utilized tour length approximation to improve the solution quality of LRP. [16] suggested using tour length estimation models with heuristic methods to solve LRPs. [17] showed that using optimal tour length estimation models to solve LRPs could reduce the computation time and find reasonable solutions to LRPs. Unlike routing problems, [18] used tour length estimators to solve the order assignment problem for last-mile delivery demonstrating the vastness of possible use cases for tour length estimators. Though different problem domains leverage different TSP tour length estimation methods, the motivation is the same: gaining a significant amount of insight into objective function values of sub-problems by significantly reducing the computation time.

The common approach is using fast and linear estimators or simple tour constructing heuristics to approximate the optimal tour length and reduce the run time. Contrary to these approaches, finding a general functional form of solution to TSP is a challenging task due to the combinatorial character of the problem. Therefore, using simple linear models that mostly use instance statistics, which are highly dependent on underlying distribution function of nodes, cannot be generalized for arbitrary TSP instances. In addition, [19] proved the asymptotical formulation of the shortest tour length visiting the points of known probability distributions. This shows the simplicity of optimal tour length estimation of large instances with known node distribution

functions. Thus, it is expected that simpler models fail in smaller instances mainly because instance statistics computed for smaller instances are less representative than the statistics computed for larger ones. Given the limitations of different methods, we aim to develop a method suitable for the complex structure of the TSP and robust in terms of size of the instances. In that aspect, we design a machine learning-based mechanism using artificial neural networks as the backbone structure to capture the complexity of the task and features exploiting the domain knowledge in routing to obtain high quality solutions even in the most challenging cases.

Another shortcoming of the studies in the literature is that they focus on instances with simple instance morphologies like uniform node distribution. Instead, we design and use new instances mimicking real-life logistics networks and morphologies (e.g., presence of varying sized clusters and arbitrary service region shapes). These instance characteristics introduce a substantial computational cost to the problem, making our instances harder. We also consider instances with varying size in order to prove the robustness of our approach. We show in Section 5 that our algorithm significantly outperforms the best models on test instances with arbitrary node distributions.

On the other hand, designing a learning process that enables models to generalize well even on arbitrary node distributions requires tackling several challenges. First of all, design of training instances is the most crucial part of any data-driven approach. Instead of learning only from instances with specific node distributions like uniform distribution, we consider instances with arbitrary node distributions that are shown to be more challenging than those used in the literature. We demonstrate the difficulty of instance types used in the literature along with our instances in terms of solution time in Section 5. The second challenge is instance representations and features. We engineer node coordinates and generate more meaningful and representative features. Also, we devise a new and efficient way of computing lower bounds to the optimal tours. The third challenge is the selection of models. Learning to solve TSP requires

highly nonlinear machine learning approaches as demonstrated in several studies. Therefore, we use neural networks with nonlinear layers to match the model and the problem. The last challenge is input normalization. We develop our own method and make input signals come from the same interval to eliminate scale difference within data. By crafting TSP specific domain knowledge with the power neural networks, our approach achieves less than 0.7% deviation from the optimal tour length on average.

Our contribution to the literature has five aspects. First, estimating the optimal TSP tour length requires a nonlinear estimator considering its structure and definition; thus, we propose a estimation method that captures this characteristics of the problem. Second, we propose a quite flexible instance generation method that allows generating hard-to-solve instances with pathological morphologies [20, 21]. Then, we show that training models on harder instances generalizes significantly better than models trained on easy and simple morphologies. Third, we develop a novel dendrogram-based heuristic to find partial solutions (control zones and moats) and high quality lower bound values by solving the moat packing problem [22]. Fourth, instead of using instance statistics as features, we use a mixture of delicately designed instance-level (height and width of tightest rectangle encapsulating all the nodes of an instance), node-level (normalized coordinate values and K-nearest Euclidean distances), and solution level (control zones, moats, and lower bound values) features. Lastly, we successfully implement our proposed mechanism (See Figure 1) to solve extremely large Generalized Traveling Salesperson Problems (GTSP). We significantly outperform the state-of-the-art solver [23] in terms of run time (up to 27 times faster) and solution quality (up to 5% better) using our proposed mechanism. To our knowledge, our implementation is the first machine learning based implementation for solving GTSP instances at this scale.

The rest of this thesis is organized as follows. In Section 2, we present comprehensive literature review. In Section 3, we describe our instance generation method.

In Section 4, we explain our feature engineering approach. In Section 5, we introduce our models and conduct a computational study with the best models in the literature. In Section 6, we present a powerful application of our model and proposed solution framework on large-scale routing problems. In Section 7, we discuss our conclusions, and the potential research directions and opportunities.

CHAPTER II

PREVIOUS WORKS

This section provides a multifaceted literature review on optimal TSP tour length estimation models. We review the related works under three categories. The first one is asymptotic estimation models, the second one is direct tour length estimation models, and the last one is the optimal tour length estimation through optimal tour construction models and heuristics.

The earliest tour length estimation models were asymptotical and based mainly on node distribution functions and statistics. [24] proposed the minimum journey time estimation method to survey the jute agriculture. [19] proved that optimal tour length L_n through n points distributed with an integrable probability distribution $f(x)$ over d-dimensional Euclidean space is asymptotically equal to right hand side of the equation (1) with probability 1. β_2 value, a β coefficient specific to 2-dimensional Euclidean space, was estimated between $[0.66, 0.84]$. Many studies focused on refining β_2 coefficient [25, 26, 27, 28, 29, 30, 31, 32].

$$\lim_{n\to\infty} L_n = n^{(d-1)/d}\beta_d d^{1/2} \int f(x)^{(d-1)/d} dx \tag{1}$$

Aside from asymptotic approaches, there are several direct tour estimation models in the literature. We classify direct tour length estimation models into two categories, namely empirical models and machine learning based models. [33], proposed a direct empirical estimation model (2) (where N, A and C represent number of locations, service region area and capacity, respectively) to estimate the total cost L of CVRP. [34] proposed that tour lengths L obtained by the Clark-Wright algorithm defines a 3-parameter Weibull distribution, and the location parameter of this distribution is

the optimal TSP tour length.

$$L \cong 0.6885NA^{0.5}\left[\frac{1}{C} + \frac{1}{\sqrt{N}}\right] \tag{2}$$

There are several successful implementations of machine learning approaches to estimating the optimal tour length. [35] trained multiple models and reported that the best model could estimate the tour length within 7% to the optimality. [36] developed an artificial neural network model to estimate the optimal tour length using instance statistics. [37] trained a regression model using number of nodes n, the standard deviation and the mean of distances of nodes to center point along axes ($cstdev_x$, $cstdev_y$, $\bar{c}_x$, $\bar{c}_y$), the standard deviation of nodes along axes ($stdev_x$, $stdev_y$) and the area of the service region (A) (3) to estimate the optimal tour lengths of distribution and shape free TSP instances within 1 percent to optimal. Their model outperformed the best of the existing estimation models on distribution-free test settings. However, for instances smaller than 1000, they proposed a correction model (where E is the corrected prediction) (4).

$$L \cong 2.791\sqrt{n\left(cstdev_x cstdev_y\right)} + 0.2669\sqrt{n\left(stdev_x stdev_y\right)\frac{A}{\bar{c}_x\bar{c}_y}} \tag{3}$$

$$\frac{E}{L} \cong 0.9325e^{0.00005298n} - 0.2972e^{-0.1452n} \tag{4}$$

In addition to the direct estimation models, very efficient and quick tour constructing models and heuristics have been proposed. [38], [39], and [40] have shown that k-opt moves can be used to approximate the optimal solution fast. Therefore, such heuristics can be used to estimate the optimal objective value due to their speed. [13] developed a constructive heuristic that produces an expected tour length $L \cong 0.9\sqrt{Size \times Area}$ for Euclidean TSP instances. [41] developed the first neural network architecture to solve TSP instances up to 30 nodes. Analogous with the spin-glass systems, the proposed network had an entirely interconnected neuron design. These neurons were trained to minimize the internal energy function, which is a quadratic function of the

total number of neurons in the network. However, the level of connectionism in the architecture makes it costly to use in larger instances. [42] reduced full connectionism to reduce training complexity; however, their network could not guarantee the optimality. [43] combined the elastic band method [44] with a self-organizing map [45] to solve the TSP instance with a competitive learning approach [46].

Recently with the rise of deep learning, some successful tour constructing models have been developed. [47] proposed a pointer network architecture that learns to permutate the input. [48] used reinforcement learning (RL) to learn the decision policies in combinatorial optimization problems. [49] used RL to learn optimal policies for combinatorial optimization problems over graphs. [50] and [51] trained RL models on uniform and unit square instances. Greedy solutions were further improved by sampling and 2-opt heuristic in return for a significant amount of additional computational cost and time. [52] trained Graph Neural Networks (GNN) to predict the optimality of edges.

CHAPTER III

INSTANCE GENERATION STRATEGY

We generate our training instances in the first place. Note that the size of training data is not the only concern since quality of the training data outweighs the quantity of the training data on hand. To obtain a model that is able to generalize the learned properties of TSP to wide range of instance types, training data must be generated accordingly (i.e., training data that does solely composed of similar instances) [53].

Generalization ability of a model can be defined as the model's performance on unseen test data. Suppose there is a significant difference between training data and test data performances (i.e., higher performance on training data than test data). In that case, the model overfits the training data. Clearly, a model learning from instances with the node distribution like uniform node distribution may perform well on test instances with the same distribution. This might seem a desirable setting to learn from the same type of instances. However, as we show later in Section 5, such models do not perform well on test instances from different node distributions, and this clearly limits their performance in practice. Enriching the training data with as diverse families of node distributions as possible might be a remedy for this issue; however, in that case, it is considerably more challenging to construct a well-performing learning mechanism as the training data is larger and more diverse. To this end, we design a data generation process such that each training instance is drawn from an arbitrary node distribution function. The process allows us to generate a diverse set of instances. Also, it leads us to develop a novel complex learning mechanism to learn from such varied data to yield high-quality solutions on numerous instance types.

Algorithm 1: Generating Training Instances in 2-D Euclidean Space

set *size*;
$x_{upper\ left} \leftarrow U(x_{lower\ limit}, x_{upper\ limit}, 1)$;
$y_{upper\ left} \leftarrow U(y_{lower\ limit}, y_{upper\ limit}, 1)$;
$x_{bottom\ right} \leftarrow U(x_{upper\ left}, x_{upper\ limit}, 1)$;
$y_{bottom\ right} \leftarrow U(y_{lower\ limit}, y_{upper\ left}, 1)$;
$height \leftarrow y_{upper\ left} - y_{bottom\ right}$;
$width \leftarrow x_{bottom\ right} - y_{upper\ left}$;
$c \leftarrow$ uniformly sample a value from $C = \{1, 2, 3, \ldots, n_{\text{max cluster}}\}$;
$n_c \leftarrow$ **round**$(size \times U(clustered\ points\ ratio_{\min}, clustered\ points\ ratio_{\max}, 1)$;
$n_c{}' = size - n_c$;
$s \leftarrow U(scaling\ factor_{\min}, scaling\ factor_{\max}, 1)$;
counter $\leftarrow 0$;
if $c = 1$ **then**
    $x_{cluster\ center} \leftarrow (x_{upper\ left} + x_{bottom\ right})\ /\ 2$;
    $y_{cluster\ center} \leftarrow (y_{upper\ left} + y_{bottom\ right})\ /\ 2$;
    while counter $< n_c$ **do**
        $x_{\text{point}} \leftarrow x_{\text{cluster center}} + \mathcal{N}(\mu, \sigma^2, 1) \times width \times s$;
        $y_{\text{point}} \leftarrow y_{\text{cluster center}} + \mathcal{N}(\mu, \sigma^2, 1) \times height \times s$;

        if (x_{point}, y_{point}) *is inside the frame* **then**
            add (x_{point}, y_{point}) to instance ;
            counter++;
        end
    end
else
    $x_{cluster\ centers} \leftarrow U(x_{upper\ left}, x_{bottom\ right}, c)$;
    $y_{cluster\ centers} \leftarrow U(y_{bottom\ right}, y_{upper\ left}, c)$;
    while counter $< n_c$ **do**
        $i \leftarrow$ uniformly sample a cluster from $\{1, 2, 3, \ldots, c\}$;
        pick i^{th} cluster;
        $x_{\text{point}} \leftarrow x_{\text{i}^{\text{th}}\text{cluster center}} + \mathcal{N}(\mu, \sigma^2, 1) \times width \times s$;
        $y_{\text{point}} \leftarrow y_{\text{i}^{\text{th}}\text{cluster center}} + \mathcal{N}(\mu, \sigma^2, 1) \times height \times s$;
        if (x_{point}, y_{point}) *is inside the frame* **then**
            add (x_{point}, y_{point}) to instance ;
            counter++;
        end
    end
end
$x_{noncluster\ points} \leftarrow U(x_{upper\ left}, x_{bottom\ right}, n_c{}')$;
$y_{noncluster\ points} \leftarrow U(y_{bottom\ right}, y_{upper\ left}, n_c{}')$;

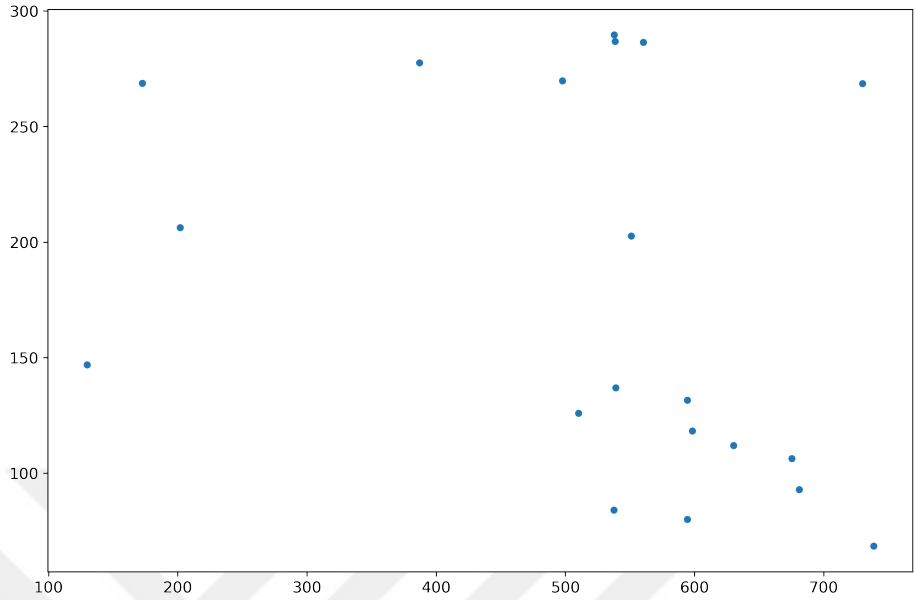


Figure 2: TSP instance with 20 nodes

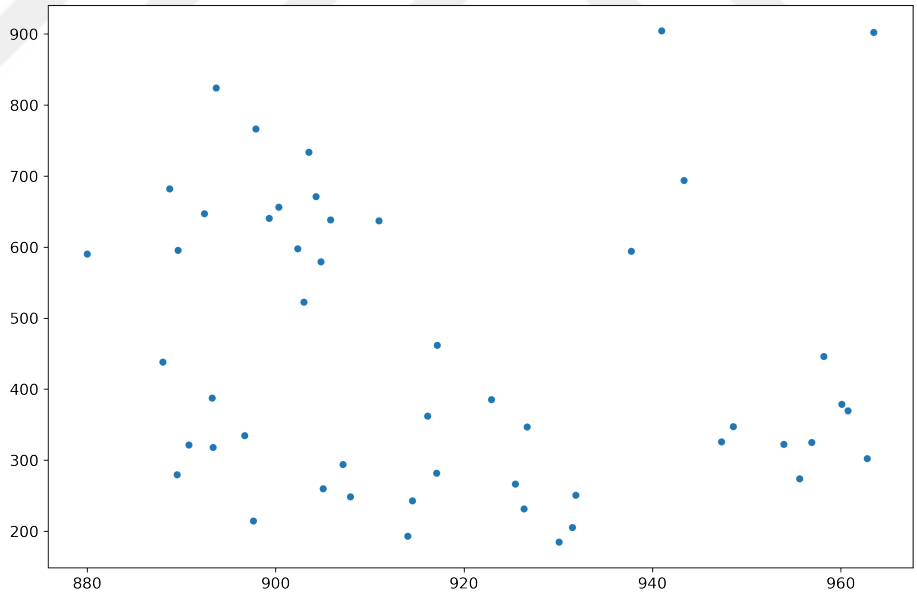


Figure 3: TSP instance with 50 nodes

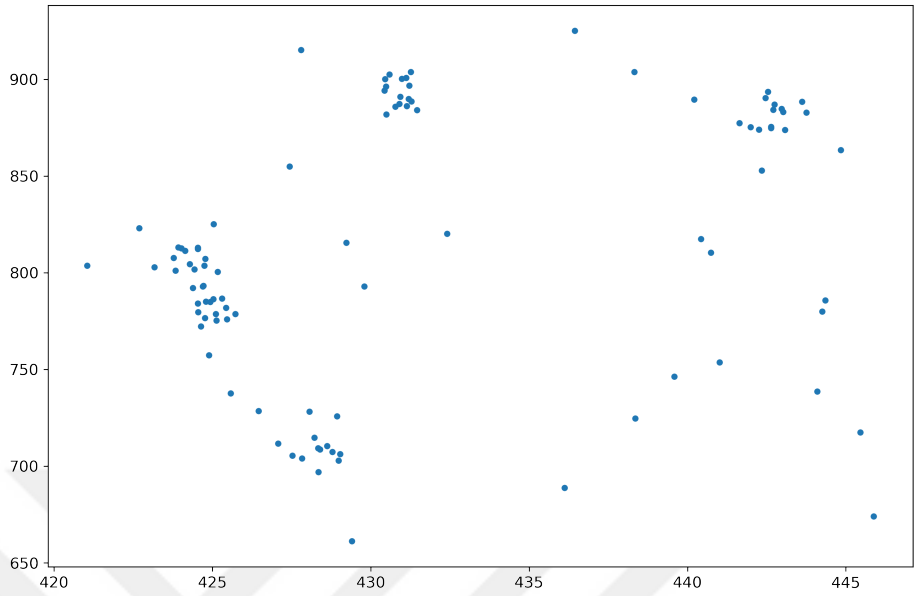


Figure 4: TSP instance with 100 nodes

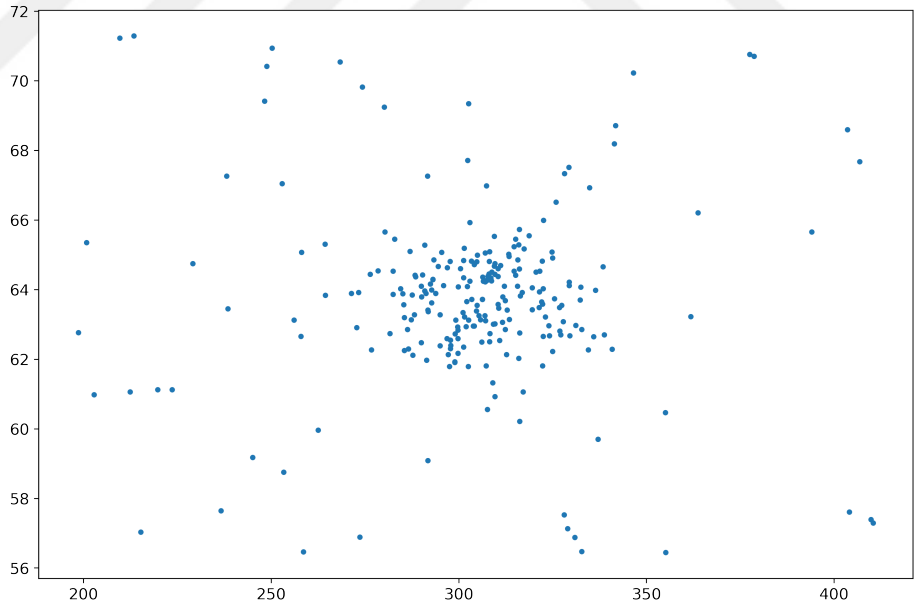


Figure 5: TSP instance with 250 nodes

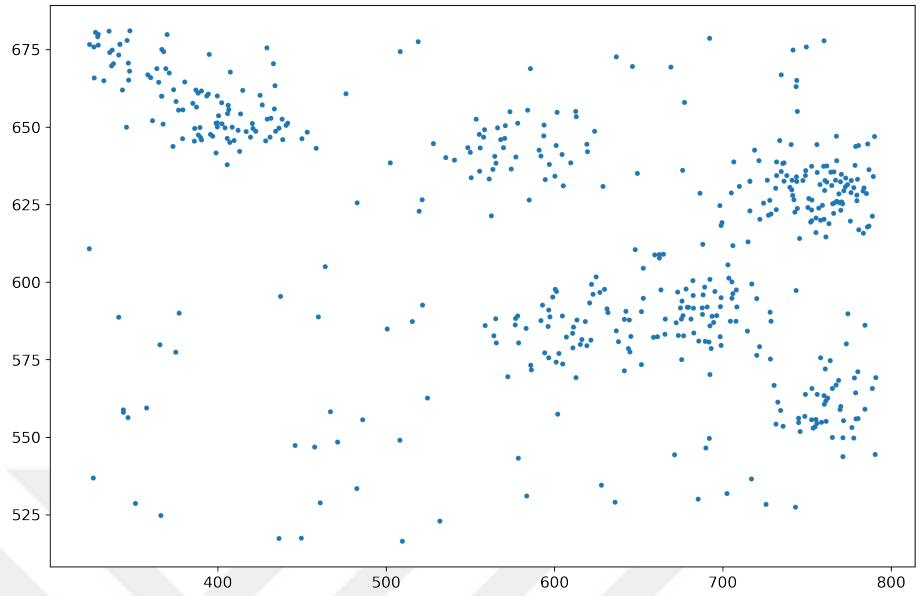


Figure 6: TSP instance with 500 nodes

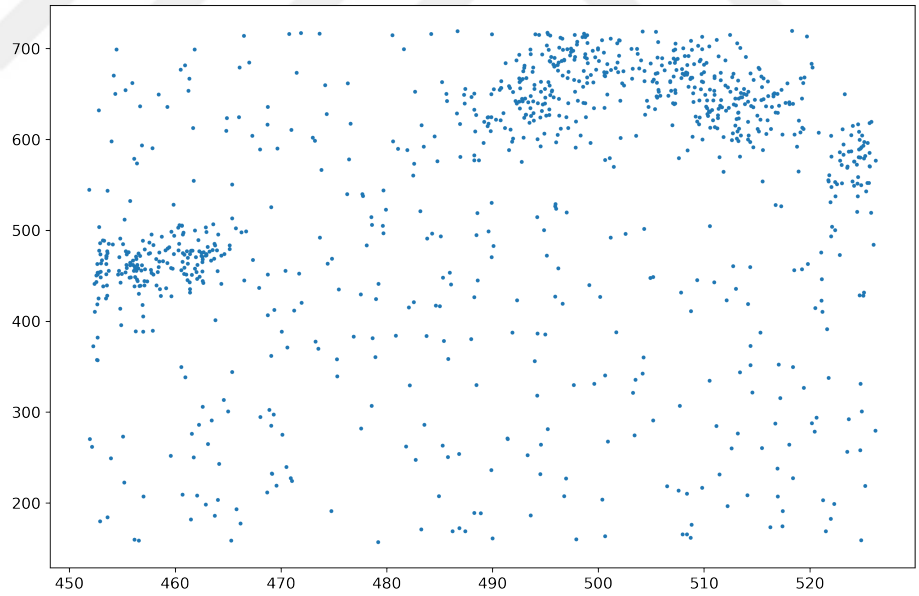


Figure 7: TSP instance with 1000 nodes

To create instances using arbitrary node distribution functions, we incorporate two key elements. First, we generate instances with aspect ratios (i.e., the ratio of height to width of tightest rectangle enclosing all nodes) varying between 0 and ∞. Clearly,

some of these instances may not be encountered (e.g., rod-shaped and extremely thin or long) in real life and may have a trivial solution. However, to generalize the learned properties of TSP instances to unseen test instances, we should include such instances even if they are trivial to estimate. Second,we generate instances mimicking real-life logistics and distribution networks. These instances include dense clusters (representing metropolitan areas) and sparse regions (representing rural areas) to increase our approach's applicability for real-life problems. Also, these instances present the utmost challenge in estimation as morphological characteristics of instances display high territorial variability. To illustrate this, consider an instance with a large cluster in the middle and some scatter points around that cluster (e.g., Figure 5). For the points outside the cluster, features such as the closest k points and the corresponding distances might be enough to obtain the final high-quality estimation to the instance; however, those same features might be insufficient for the points within the cluster as there are many other alternatives to consider. As can be seen from Figures 2-7, we create instances with no obvious sign of common morphological pattern. All the details regarding our instance generation methodology is provided in Algorithm 1.

CHAPTER IV

FEATURE ENGINEERING

Representing a combinatorial optimization problem instance is the most important yet also the most challenging step in designing a learning mechanism. Thus, the problem of input representation must be crafted carefully. As for TSP, there is a very complex relationship between the node (relative) locations and the optimal solutions, and the optimal tour length might change dramatically by merely moving a single, arbitrary node from its current position to another position. Hence, it is necessary to consider such problems as a whole when representing them as learnable vectors.

Many tour length estimation models in the literature [37, 33, 36, 35] use instance level statistics combined with different instance-level features such as a service region area and the number of nodes per unit area. Tour generating heuristics and models take only node-level features (node coordinates) as an input [40, 49, 51] and use hand crafted methods or learn decision policies to construct a tour. However, these approaches require a tremendous amount of computation since they build their output from atomic level information. On the other hand, our instances are significantly different from previous studies since we generate them using arbitrary and unique distribution functions. In order to keep the computational and time efficiency high and to represent each instance without losing their essence and structural information, we define a highly representative feature space that consists of instance-level features, node-level features, and solution-level features. In addition, this feature space contains semantically distinct feature subsets such that each of these subsets represents an instance from a separate point of view. In the following subsections, we refer to these subsets as *feature blocks* and explain each of them thoroughly.

4.1 Instance Frame Block

The area is one of the most commonly used feature in tour length estimation literature. Contrarily, we do not directly use the area; instead, we use orthogonal components of the area, namely the height and width of the tightest instance frame. Preserving shape information of the frame is critical for the optimal tour length estimation since it provides more realistic information about the tour length than the area does. It is a 2-dimensional vector, and it stores the height and the width of the tightest rectangular frame enclosing all nodes. Let X and Y denote the sets of x-coordinates and y-coordinates of all the nodes in an instance, respectively. We represent this block with $(width,\ height)$ tuple and compute it by using the coordinates of the extreme nodes (5).

$$(width,\ height) = (\max\{X\} - \min\{X\},\ \max\{Y\} - \min\{Y\}) \tag{5}$$

4.2 Normalized Coordinates Block

Using raw coordinates as features to estimate the optimal tour length via optimal tour building must be coupled with complex model designs and outrageously more computational effort. To reduce the amount of computational cost and to keep model complexity small, we convert raw coordinates to meaningful features without compromising underlying semantics of these features by embedding our domain knowledge into these features.

We utilize a method converting raw node coordinate features to more sophisticated features representing the absolute distances from a reference. Given a node $(x,\ y)$ in a 2-D Euclidean space, we compute normalized coordinates of that node with respect to a reference node $(x_r,\ y_r)$ as $(|x - x_r|,\ |y - y_r|)$. The motivation behind this feature block is that orthogonal components of all arcs in the optimal tour can be represented by taking the absolute difference of distances from two successive node to a reference node. Let a reference point r be $(x_r,\ y_r)$ and two nodes $p_t,\ p_{t+1}$ that

are successive in the optimal tour be $(x_r + \varepsilon_t, \quad y_r + \delta_t)$ and $(x_r + \varepsilon_{t+1}, \quad y_r + \delta_{t+1})$ respectively, where $\forall \varepsilon, \delta \in (-\infty, \ +\infty)$. Then, let $\mathcal{D}$ denote a function that gives the absolute distance between two points along each axis such that $\mathcal{D}((x_1, \ y_1), \ (x_2, \ y_2)) = (|x_1 - \ x_2|, \ |y_1 - \ y_2|)$. It is clear that $\mathcal{D}(\mathcal{D}(r, \ p_t), \ \mathcal{D}(r, \ p_{t+1})) = \mathcal{D}(p_t, \ p_{t+1})$ which is equal to $(|\varepsilon_t - \varepsilon_{t+1}|, \ |\delta_t - \delta_{t+1}|)$, and this equality can be written for any two successive nodes in the optimal tour. As a result, we represent normalized coordinates feature block with $2n$-dimensional vector whose first n-dimension represents the normalized x-coordinate values, and the rest represents the normalized y-coordinate values.

On the other hand, there is a critical decision to make for performing this normalization. The decision of selection method of a reference point is a great deal of importance. It is possible to select a reference point either randomly or deterministically. In the random method, a node from an instance is sampled uniformly with probability of being selected $\frac{1}{n}$ where n is the number of nodes. Then, the rest of the nodes are normalized accordingly. Thus, it is possible to create a feature vector in n unique ways for the same instance of size n meaning that the same instance is represented in n different ways. However, this leads to n possibly unique optimal tour length value predictions for the same instance. More importantly, it is not possible to tell which of these n values are the closest to the optimal tour length value. Instead of selecting the reference node randomly, we select it rather systematically. We sweep through the instance from left to right, then the first point encountered is selected as a reference point. In our experiments, we do not notice any difference between random and deterministic node selection methods. However, consistency purposes, we adopt the systematic selection method for this study and use in all our models.

4.3 K-Nearest Euclidean Distances Block

In any optimal TSP tour, the next node to visit from any node is generally one of the closest ones and sometimes the closest one. Therefore, it is fair to say that in a

2-D Euclidean TSP instance, distances of each node to the closest k nodes provide a reasonable amount of information about the optimal tour length. We call parameter k as a sight parameter. If $k = 1$, the information about the optimal tour length is collected myopically from only the nearest neighbors, and greedy methods like the nearest neighbor heuristic generally produces low-quality tour length values. If k is set to be a larger value, it increases the dimensionality of input and model complexity but it also makes the feature vector richer. Different instance sizes and instance types may require a different level of sight. For example, it may be more suitable to use different k values for TSP instances of sizes 20 and 100, or it may be needed to use different k values for TSP instances of the same size but have different node dispersion types. We consider the selection process of k as preliminary optimization and keep selection of k out of the scope of this study. However, in order to make k value consistent across all models, we use $k = 5$ for all models presented.

4.4 Partial Solutions Block

We engineer our feature blocks so that instance representation vectors become as flexible as possible since each instance is created with an arbitrary and unique probability distribution function. However, in model development, we find out that using previously defined blocks leads to poor prediction quality on some instances. The main reason for this is that these instances are quite underrepresented in training data and they have very unusual structures. Most of these instances have extremely dense clusters, very unusual aspect ratios of service region, or the combination of both. On the other hand, the model with the same three features work quite good on instances with simple and easy morphologies and balanced service region aspect ratios. To deal with this, we develop new feature block that consists of solution level features.

The bound values provide information about the optimal objective value depending on how strong they are. However, extended time of computation is required to obtain stronger bounds. Greedy methods such as 1-Tree relaxation of TSP and Nearest Neighborhood Heuristic are two commonly used efficient methods. However, as we test these methods, we realize that bound values obtained by using these methods are not tight enough to provide a good amount of additional information, especially in our pathological instances to improve overall prediction quality. Expectedly, lower bound values obtained from these methods fail to improve the performance of our models on pathological instances. Motivated by this, we devise our own method for finding higher quality lower bounds. In order to find high quality TSP lower bounds, we *partially* solve the *moat packing problem* [22] formulated as a linear programming (LP) problem (6,7,8).

$$\max \quad \sum_{S,\overline{S}\in\mathcal{M}} 2w_{S,\overline{S}} \tag{6}$$

$$\text{s.t.} \quad \sum_{|\{i,j\}\cap S|\,=\,1} w_{S,\overline{S}} \leq g_{ij},\ \forall i,j \in N\cup\{0\} \tag{7}$$

$$w_{S,\bar{S}} \geq 0 \qquad\qquad ,\ \forall\left\{S,\bar{S}\right\}\in\mathcal{M} \tag{8}$$

Depending on the cardinality of disjoint sets S and $\bar{S}$ we name decision variables $w_{S,\bar{S}}$ differently. For instance, if $\left|\bar{S}\right| = 1$, we name related $w_{S,\bar{S}}$ decision variables as *singular moats* or *control zones*. Control zones are perfectly circular regions with radius r, and each node is located at the center of it. In any optimal tour, the salesperson has to travel at least $2r$ units to pass through each node where the r unit is traveled for arrival and the remaining r unit is traveled for departure. Conceivably, the downside is that solving the LP only with control zones generally produce low-quality lower bound values, especially when the clusters are present [9]. Hence, to find stronger lower bound values, moats with varying sizes must be regarded in the

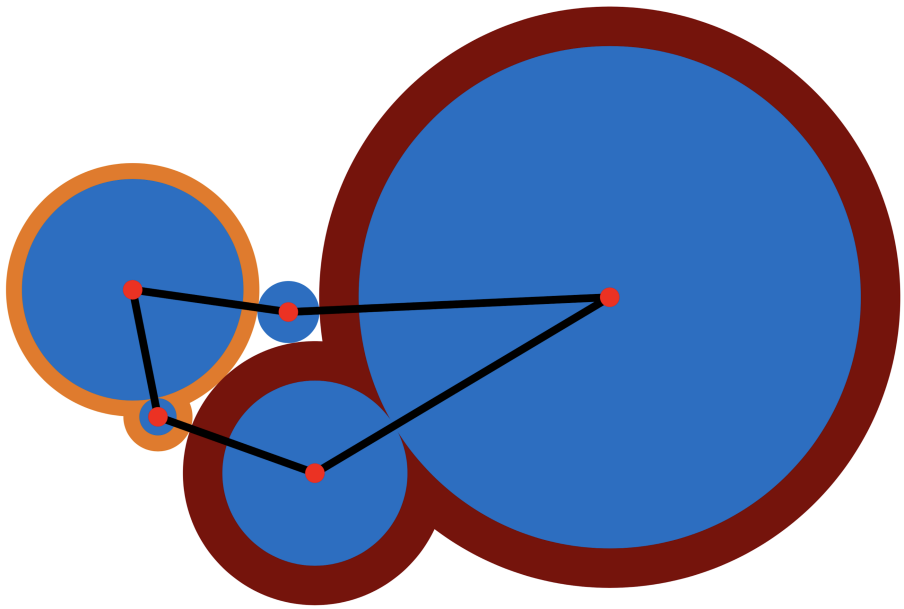


Figure 8: Moat configuration of a randomly generated TSP instance
As seen, nodes (red dots) under the same color pack (moat) are traveled successively in the optimal tour (black lines). The innermost blue circles within moats are control zones. Thicknesses of color packs are the decision variables $w_{S,\bar{S}}$ in the LP (6,7,8). (Best seen in color).

LP.

An n-city TSP gives possibility to definition of moats with at least 1 node and at most $n-1$ nodes. This means that finding a complete solution to the moat packing problem requires optimizing $2^n - 2$ decision variables. Finding this complete solution to moat packing problem is equivalent to finding the optimal solution. However, to find an acceptable lower bound, it is an overkill to solve the complete moat packing problem since the decision vector would be tremendously sparse. However, we need to consider a good subset of moats such that most of these moats have non-zero widths ($w_{S,\bar{S}}$) (i.e., denser decision vector). The importance of having mostly nonzero $w_{S,\bar{S}}$ is that the nodes belonging to corresponding moats are traveled together in the optimal tour, and nonzero moat widths define different parts of the optimal solution. Catching moats with non-zero widths leads to tighter bound values and better partial solutions, consequently, a better predictions.

Instead of optimizing $2^n - 2$ decision variables, we try different approaches to shrink the size of the decision vector and aimed to achieve better lower bound values. We first optimize the set of singular moats (control zones) and use it as a $n + 1$ dimensional feature block (n control zones plus the lower bound value), but could not improve the model performance. Then, we consider only control zones and binary moats (i.e., $0 < \left|\bar{S}\right| < 3$). In this case, for any TSP instance, at most $\binom{n}{1} + \binom{n}{2}$ or $\frac{n^2+n}{2}$ unique moats can be defined. Then we partially solve the moat packing problem by considering only $\frac{n^2+n}{2}$ decision variables. Although the obtained lower bound value and model performances is better compared to considering the control zones only, sparser and larger solution vector (n vs. $\frac{n^2+n}{2}$) make model training and computation time extremely inefficient.

To increase the quality of lower bound, efficiency of feature computation and model training, we devise a very efficient *dendrogram-based* heuristic method to determine moats that possibly host a partial solution (i.e., have non-zero moat width). This heuristic method reaches to moats with up to $n - 1$ nodes with a significantly lower computational cost compared to previous techniques. This novel procedure, begins with hierarchically clustering nodes with respect to their coordinate values using *Ward's minimum variance clustering method* [54]. We find Ward's method particularly useful for TSP instances since the method reduces the number of clusters by one at each merging point which resembles to a tour construction. Furthermore, the method minimizes the variance within clusters, which is the driving force for putting closer points into the same moats by using $\sqrt{\frac{2|A||B|}{|A|+|B|}}\,\|\vec{c_A} - \vec{c_B}\|_2$ as dissimilarity metric between clusters A and B, enabling to capture partial solutions within instance pathologies like clusters with excessive density.

At the bottom, the dendrogram leaves are singletons (clusters with only one node), and these singletons are analogous to control zones in the moat packing problem. At each merging point on the dendrogram, we merge two existing subclusters to create

Table 1: Mean Percentage Gap Between Lower Bounds and Optimality Obtained by Different Methods on Different Instance Size and Morphologies)

Morphology Type	**Simple**			**Pathological**		
Instance Size	**20**	**50**	**100**	**20**	**50**	**100**
1-Tree Relaxation	14.01	12.84	11.65	25.42	21.30	18.20
Dendrogram Method	1.83	2.29	2.15	8.90	8.07	6.74

a new supercluster. Each of these superclusters corresponds to a unique moat. After forming a new cluster, we compute new S and $\overline{S}$ sets and create corresponding decision variables $w_{S,\bar{S}}$. The global cluster (the cluster that contains all nodes) represents a trivial moat and has a moat width $w_{S,\bar{S}}$ equal to zero. Therefore, moving from bottom to the top of a dendrogram, we define n singular moats (control zones) and $n-2$ non-singular moats for the LP (6,7,8). In total, we create $2n-2$ unique clusters; then, we partially solve the moat packing problem by considering only $2n-2$ decision variables via Gurobi Solver [55]. Our method collects closer points under the same cluster. Thanks to the cluster formation technique we use, it is highly likely that nodes under the same cluster are traveled together in the optimal tour. Hence, using dendrogram heuristic, we dramatically reduce the size, sparsity, and computation time of the solution vector while significantly improving the lower bound. We compare our dendrogram heuristic and 1-tree relaxation in terms of lower bound values' mean percentage gap to optimal tour length in Table 1. We present a complete overview of performance comparison of the two methods in Figure 9.

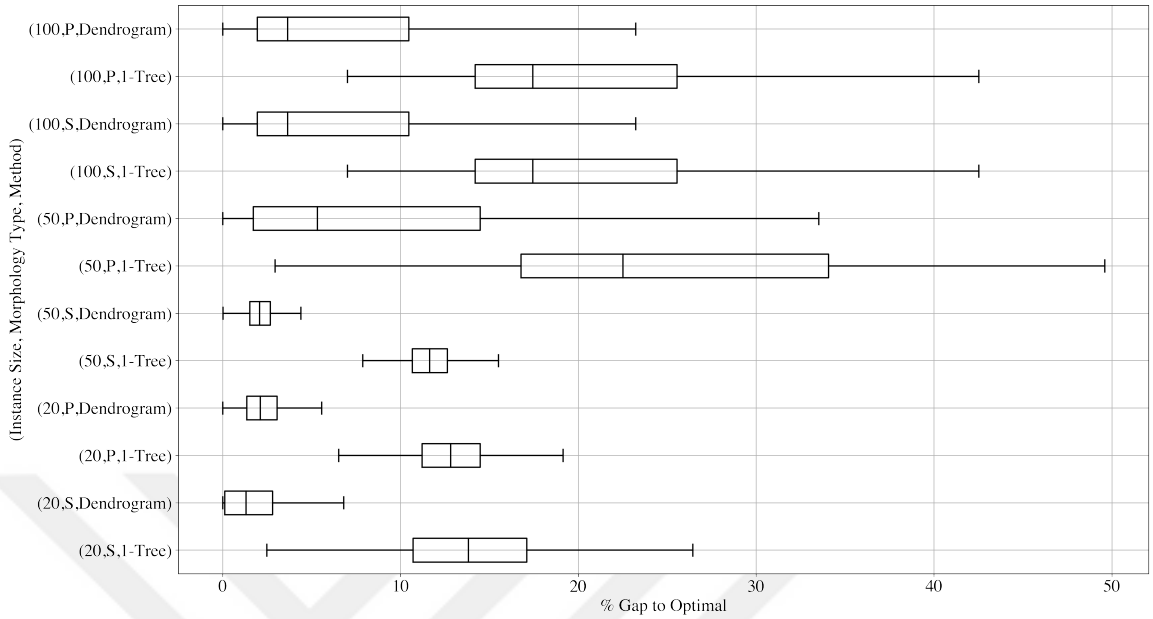


Figure 9: A Complete Overview on Lower Bound Performances of Methods

In the figure, the boxplots demonstrate the distribution of percentage (%) optimality gap of lower bounds (horizontal axis) under different test settings (vertical axis). Each label in the vertical axis is represented by a triple of *(Instance Size, Morphology Type, Method)*. We use 1000 test instances for each *(Instance Size, Morphology Type)* combination. As seen from Table 1 and Figure 9, our dendrogram-based method shows significantly better performance in terms of mean percentage gap and median percentage gap compared to 1-Tree relaxation. It should also be noted that, performance of both methods deteriorate on pathological morphologies. Despite the drop in the quality of the bound, our dendrogram method significantly improves the models' performances on our pathological instances. We demonstrate the contribution of this feature block on model performances in Section 5.

4.5 Input Standardization

Fitting a good model is demanding task when there is a considerable variation in the measure of training data. In neural networks, backpropagating the errors with

significantly different scales causes the different magnitude of gradient updates in the model parameters, primarily when mean squared error (MSE) loss is used as a loss function. In our data generation process, using the floating rectangular instance frame makes any two instances differ in their optimal tour length by a magnitude of ρ where $\rho \in (0, \infty)$. When making predictions with the models, we inspect that our models predict large-in-measure instances better than small-in-measure instances. In other words, we obtain larger average percentage deviations from the optimal tour length when making predictions for small-in-measure instances. To solve this problem, we first test min-max scaling and standard normal transformation methods. However, we find out that the model performances got even worse. The problem with these two methods or other variable-centric input transformation methods is that they treat each feature independently. However, most of the features are meaningful only with the other features of the same semantics in our case. For example, the partial solution block does not store tabular data. Although it is a $2n - 1$ dimensional vector, each dimension does not necessarily represent the same moat or control zone. In addition, it is clear that the first n dimension represents control zones; however, value of the k^{th} dimension where $1 \leq k \leq n$ is the control zone of an arbitrary node, not a control zone of a particular node. Similarly, the normalized coordinates block and Euclidean distances block are computed based on graphical structure of TSP. Taking those into consideration, we develop our own input standardization method. In our normalization method, we cast each instance into a unit square.

Let h_f and w_f denote the height and width of the tightest rectangular frame defined in Subsection 4.1. After computing all feature blocks, we move the instance frame to the origin without rotating it such that the bottom left corner of the frame overlaps with the origin $(0, 0)$. Then, we divide each feature and tour length value by $\max\{h_f, w_f\}$ so that each instance fits into a unit square. After applying this standardization, there are still scale differences among instances; however, the scale

difference is now negligible, since most input signals are coming from $[0, 1]$. The transformation method is universal because any instance can be casted into the same interval without compromising its structure, and it is trivially easy to revert them to their original state.

CHAPTER V

MODELS AND COMPUTATIONAL STUDY

In this section, we summarize our findings of an extensive computational study. First, we explain our models and training phase in detail. Then, we benchmark our models with each other and conduct benchmarking studies against the best tour length estimation model and the best TSP tour generation model separately. To present a fair comparison of the models, we compare them on exhaustive test settings. In the first setting, we benchmark models on our test instances. In the second setting, we benchmark models on their test instances. In the third and the last setting, we compare models on instances where none have been trained or tested before.

5.1 Training Data Generation

We create three datasets consisting of 80,000 TSP instances for each sizes 20, 50, and 100. In generating training instances, we use the Algorithm 1 with parameters given in the Table 2. We then solve each instance optimally by using Concorde TSP Solver. Furthermore, Concorde uses integer coordinate values and returns integer objective values. However, our instances contain float coordinate values. To prevent any precision problem, we multiply coordinate values in each instance by 10^4 before solving them. Then, we compute the optimal tour value with the optimal tour found by Concorde on original (unscaled) coordinate values.

Table 2: Parameter Set Used in Algorithm 1

$x_{lower\ limit}$	0	$n_{\max \text{ cluster}}$	10	$scaling\ factor_{\max}$	0.5
$x_{upper\ limit}$	1000	$clustered\ points\ ratio_{\min}$	0.5	μ	0
$y_{lower\ limit}$	0	$clustered\ points\ ratio_{\max}$	0.9	σ	1
$x_{upper\ limit}$	1000	$scaling\ factor_{\min}$	0.001	$size$	$\{20, 50, 100\}$

5.2 Neural Network Overview

Feed-Forward Neural Network (FFNN) models have become a staple for many real-world applications to map an input to a desired output. Multi-Layer Perceptron (MLP) models are the most used methods among FFNN [56]. MLP models are simply the chain of affine transformations over a representation vector h. In other words, each layer in an MLP performs a learned transformation sequentially. A layer blueprint is given in the Equation 9. In (9), h is an m-dimensional input vector, and h' is an n-dimensional output vector. w and b are called $m \times n$-dimensional layer weight matrix and n-dimensional bias vector, respectively. They are learned through optimizing problem-specific cost functions. g is a fixed function called activation, generally a nonlinear function. Let x be an input vector and $f^{(k)}$ be the k^{th} layer, a k-Layer MLP model $F(x;\theta)$ can be written as (10) where θ represents model parameters.

$$h' = g\left(w^T h + b\right) \tag{9}$$

$$F(x;\theta) = f^{(k)}\left(f^{(k-1)}\left(f^{(k-2)}\left(\ldots f^{(1)}(x)\ldots\right)\right)\right) \tag{10}$$

An MLP model is simply a chain of matrix multiplication and addition. Figure 10 demonstrates a 3-Layer MLP. In Figure 10, each node is a dimensional value from the representation vector space. Each directed connection represents information flow from current layer to the next layer proportional to its weight scalar. Let y be the target scalar that we aim to predict using input vector $x \in R^d$. Approximating the relationship between x and y is equivalent to finding parameter set θ minimizing the loss function. This loss function is generally the squared euclidean distance between the y and $F(x;\theta)$ for regression tasks (see rhs of Eq. 11).

$$F^*(x;\theta) = \underset{\theta}{\operatorname{argmin}} \left\|y - F(x;\theta)\right\|^2 \tag{11}$$

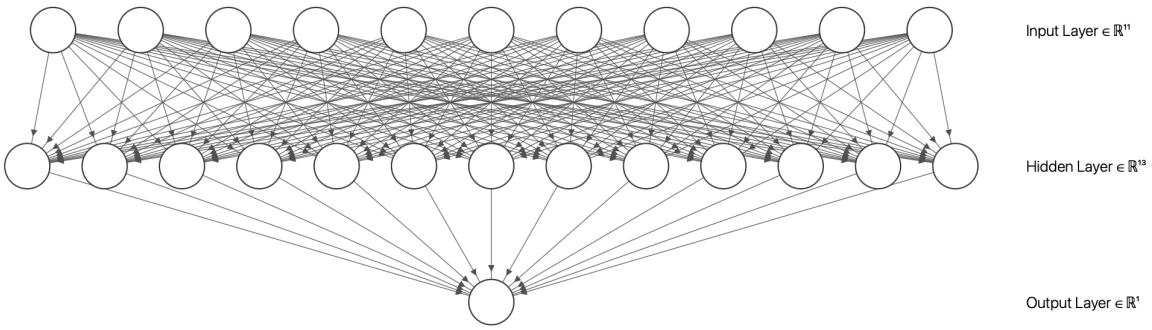


Figure 10: A visualization of a 3-Layer MLP

Neural network parameter optimization is done using gradient-based methods. In general, stochastic gradient descent (SGD) and variants are the methods of choice. The gradients for model parameters are computed by a procedure called backpropagation [57]. The critical thing is that the cost function should be differentiable so that error gradients can be calculated. In this method, neural network parameters converge to a better configuration step by step. A generic flow of neural network parameter optimization is provided in Algorithm 2.

Algorithm 2: A Generic SGD Optimization Routine for Neural Networks

```
1  require dataset;
2  require F(x; θ);
3  require L(y, F(x; θ)) ;
4  initialize θ;
5  set batch size, max epoch, step size;
6  max steps ← #dataset/batch size ;
7  step ← 0 ;
8  epoch ← 0 ;
9  while epoch < max epoch do
10 |   while step < max steps do
11 |   |   b ← sample a mini batch of batch size from dataset;
12 |   |   loss ← 1/(batch size) Σ_{i=1}^{batch size} L(y_i, F(b_i, θ));
13 |   |   error gradients_θ ← ∇_θ loss;
14 |   |   θ ← θ − (step size × error gradients_θ);
15 |   |   step++;
16 |   end
17 |   epoch++;
18 end
```

The training loop consists of two sequential operations, namely forward propagation (forward pass) and backpropagation. We perform a forward pass in steps 11 and 12 to compute the batch error. Then in steps 13 and 14, we backpropagate the errors

to compute error gradients for each parameter in the model and update the model parameters accordingly. On top of these, the dataset (x and y), model architecture ($F(x;\theta)$), and the loss function definition ($\mathcal{L}(y, F(x;\theta))$) are required.

5.3 Model Training

We design four different shallow neural network models with different connection type and input size. The first connection type one is a *full connection* between feature blocks (i.e., a standard densely connected MLP (See Fig. 10)), and the second one is a *partial connection* between feature blocks. Training with partial connection can be considered as parallel networks trained on each feature block separately and synchronously (See Figure 11). We aim to separate input signals of different feature blocks from the beginning of the networks by using partial connection between feature blocks. Therefore, hidden representations for each feature block can be learned disjointly in the input layer. Additionally, we use two different feature sets for each layer connection type. The computation of the partial solutions feature block costs the highest among all blocks in terms of time. However, it significantly improves the models' performances on not only highly pathological instances but also all instance type tested. Thus, we use two different feature set to investigate the computational cost in return for performance gain. The smaller feature set does not include the partial solutions block, while the larger feature set contains every block described previously, and everything else is kept the same. Let n denote the size of an instance. Then, dimensions block (X_D), normalized coordinates block (X_{NC}), k-NN Euclidean distances block (X_{KED}), and partial solutions block (X_{PS}) have 2, $2n$, $5n$, and $2n-1$ dimensions, respectively. Therefore, the small and large feature sets have $7n+2$ and $9n+1$ dimensions. In Table 3, we provide the model names along with their connection type and input structure. In addition, we provide a functional form for each models in Equation 12. $f^{(k)}$ represents a layer at depth k, $f_n^{(k)}$ represents a partial

Table 3: Models and Their Configurations

Model Name	Layer Connection Type	Input Structure
FCNNSmall	Fully Connected	Small: $concat(X_D, X_{NC}, X_{KED})$
FCNNLarge	Fully Connected	Large: $concat(X_D, X_{NC}, X_{KED}, X_{PS})$
PCNNSmall	Partially Connected	Small: (X_D, X_{NC}, X_{KED})
PCNNLarge	Partially Connected	Large: $(X_D, X_{NC}, X_{KED}, X_{PS})$

layer n at depth k, and *concat* represents a concatenation layer.

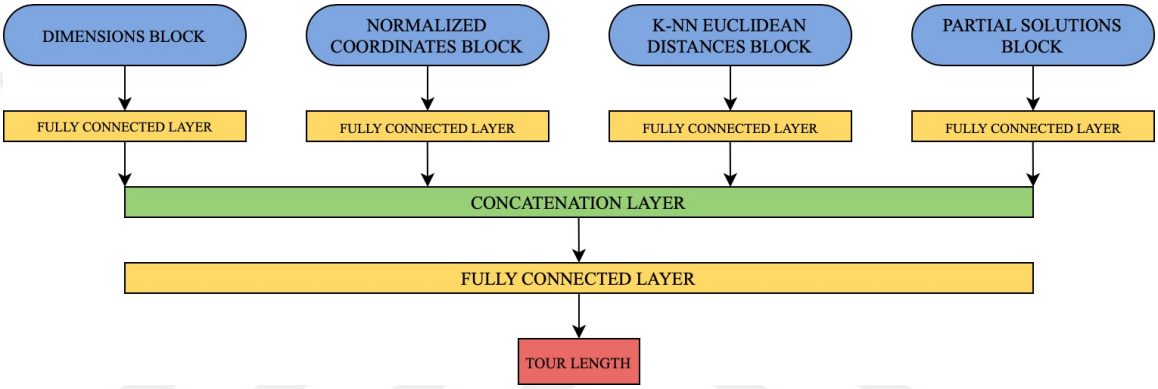


Figure 11: Model Architecture with Partially Connected Layers

$$FCNNSmall = f^{(3)}(f^{(2)}(f^{(1)}(concat(X_D, X_{NC}, X_{KED})))) \tag{12a}$$

$$FCNNLarge = f^{(3)}(f^{(2)}(f^{(1)}(concat(X_D, X_{NC}, X_{KED}, X_{PS})))) \tag{12b}$$

$$PCNNSmall = f^{(3)}(f^{(2)}(concat(f_1^{(1)}(X_D), f_2^{(1)}(X_{NC}), f_3^{(1)}(X_{KED})))) \tag{12c}$$

$$PCNNLarge = f^{(3)}(f^{(2)}(concat(f_1^{(1)}(X_D), f_2^{(1)}(X_{NC}), f_3^{(1)}(X_{KED}), f_4^{(1)}(X_{PS})))) \tag{12d}$$

We split 80% and 20% of total data as the training set and validation set, respectively, and we use the exact same data split for entire training pipeline. We train four models on TSP instances of sizes 20, 50, and 100 separately. In total, we train 12 different estimation models. Each model has only one hidden layer with 100 rectified linear units (ReLU)[58]. We train each model for 100 epochs with mini-batches of 128. We use MSE as a loss function, and we minimize the loss using Adam [59] optimizer.

After each training epoch, we store the best model parameter configuration that produces the lowest mean absolute percentage error (MAPE) score on the validation set. The reason preferring MAPE over MSE as a model selection criteria is that there is still a measure difference among instances despite casting instances in $[0, 1]$. In other words, the model that yields the smallest MSE on validation set produces predictions with higher MAPE score on small-in-measure instances. We train all models on a single machine with 2.2 GHz Intel Core i7 CPU with 6 Cores and 16 GB of RAM. Under these data, model and hardware configurations, each epoch took less than 1 second for each model.

5.4 Benchmarking the Models

We compare each model and Concorde based on criteria described in Table 4. However, benchmarking methods based on time basis is somewhat tricky due to the fact that methods utilize software and hardware differently. For example, Concorde runs ANSI C code to compute the results while models proposed by [51] and our models run Python code. The model (CS) proposed by [37] has a closed functional form with quite a few parameters; however, our models and models (KHW) proposed by [51] have thousands of parameters. In addition, KHW models are designed to exploit extreme parallelization ability of GPUs while some key components of our models (computation of X_{PS} block) and Concorde run only on CPU. To make a fair comparison between these methods especially on the run times, we comply the following procedures. First, we execute each method on the CPU solving each test instance sequentially, and we report the CPU run time based on sequential solution time. Secondly, if a method has GPU compatibility, we also report run times on the GPU. Our models do not utilize the GPU when computing the features. However, they can utilize GPU for making predictions. Therefore, to make a more reflective run time comparison with models which can run entirely on GPU, we parallelize the feature

Table 4: Benchmark Table Column Descriptions

Column Name	Description
Size	Size of benchmark TSP instances
MAPE	Mean Absolute Percentage Error between the optimal and estimated tour lengths
Mean E / O	Mean value of ratio of estimated tour length to optimal tour length
Std. Dev. E / O	Standard deviation of ratio of estimated tour length to optimal tour length
Min E / O	Minimum value of ratio of estimated tour length to optimal tour length
Max E / O	Maximum value of ratio of estimated tour length to optimal tour length
Time	CPU or GPU seconds required for computing the outputs for 1,000 TSP instances.

block computation pipeline over CPU cores such that we process 12 TSP instances simultaneously. Such parallelization is relatively insignificant compared with GPU parallelization. However, in terms of scalability, we show that our models are also significantly scalable over multiple processors when dealing with huge TSP batches. In addition, after computing the feature blocks, making inferences takes quite a short time due to the simplicity and efficient design of our neural networks. Thus, there is no run time difference between using CPU or GPU for making inference from our models.

We first benchmark our models against Concorde's run time on our test instances. We generate 1000 instances for each size. As stated earlier, we train models that produce high quality tour length estimates significantly quicker than Concorde in order to use these models as a backbone in heuristics such as cluster-first route-second. We do not compare tour length qualities since it is trivial that Concorde consistently outperforms our models in terms of tour length quality. We present the comparison of our models with Concorde on our test instances in Table 5. Run times provided in Table 5 are measured by sequentially solving 1000 of our test instances.

Our initial benchmarks on our test instances show that the worst outcome from our proposed models deviates from the optimal at most 2.3% on average. However, our models are significantly faster than Concorde in generating the tour length. Furthermore, Concorde, FCNNLarge, and PCNNLarge's run times take remarkably longer as the instance size increases. Nevertheless, considering the improvement on

Table 5: Performance Metrics of Methods on Our Test Instances

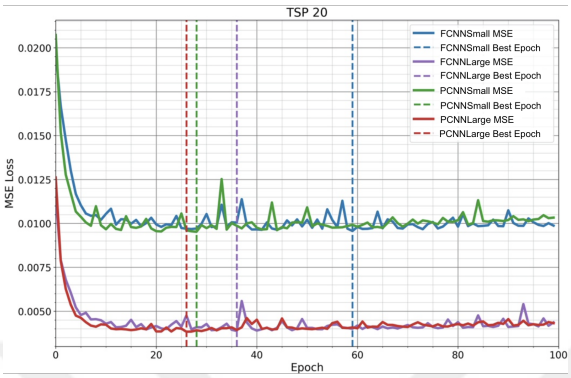
Method	**Size**	**MAPE**	**Mean** $E\,/\,O$	**Std. Dev.** $E\,/\,O$	**Min** $E\,/\,O$	**Max** $E\,/\,O$	**Time(s)[CPU]**
Concorde	20						34
FCNNSmall	20	2.33	1.00	0.03	0.85	1.17	0.3
FCNNLarge	20	1.63	1.00	0.02	0.89	1.09	4.5
PCNNSmall	20	2.33	1.00	0.03	0.83	1.15	0.3
PCNNLarge	20	1.59	1.00	0.02	0.87	1.08	4.5
Concorde	50						210
FCNNSmall	50	1.98	1.00	0.03	0.84	1.12	0.6
FCNNLarge	50	1.32	1.00	0.02	0.90	1.09	38
PCNNSmall	50	1.96	1.00	0.03	0.85	1.13	0.6
PCNNLarge	50	1.27	1.00	0.02	0.89	1.09	38
Concorde	100						495
FCNNSmall	100	1.56	1.00	0.02	0.83	1.17	1.2
FCNNLarge	100	1.00	1.00	0.01	0.89	1.11	260
PCNNSmall	100	1.50	1.00	0.02	0.84	1.18	1.2
PCNNLarge	100	0.95	1.00	0.01	0.91	1.10	260

MAPE scores brought by partial solutions features, increased time cost can be tolerated at these instance scales. Partial solutions significantly increase prediction quality in terms of error, standard deviation, and prediction range. Models with partial solutions features display more reliable prediction performance on pathological test instances. In addition, we test the solution times of the moat packing problem for larger TSP instances, and it provides good information about our models' scalability since our solution time for moat packing problems changes only when instance size changes. Contrarily, Concorde's solution time changes significantly depending on the pathology level of an instance (e.g., extreme values of the aspect ratio of instance frame or highly dense clusters). We observed that FCNNLarge and PCNNLarge run times could be more than 10x faster than Concorde on these highly pathological instances with size 100. In terms of unbiasness and reliability, our models do not display any bias in terms of expected $E\,/\,O$ ratios. This is a significant aspect of our models since many estimation models underestimate the tour length of relatively smaller TSP instances and severely suffer from this prediction bias.

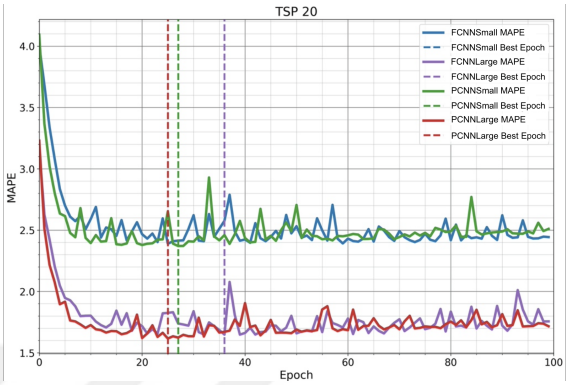
Models with smaller inputs (FCNNSmall and PCNNSmall) are the outperformers by a large margin in terms of run time. FCNNSmall and PCNNSmall provide speed up

against Concorde around 100x, 350x, and 400x on TSP instances of sizes 20, 50, and 100. Similarly, FCNNSmall and PCNNSmall provide speed up against FCNNLarge and PCNNLarge around 15x, 60x, and 200x on TSP instances of 20, 50, and 100. It is evident that larger instances make FCNNSmall and PCNNSmall more appealing to use against FCNNLarge, PCNNLarge, and Concorde, when algorithm run time and high prediction quality is the primary concern. It is also evident that there is diminishing MAPE trend as problems get larger, hinting that there is a high chance of having lower MAPE scores with any of the four models when they are used for larger instances ($n \gg 100$). More importantly, the effect of using partial solutions block becomes less noticeable as problem sizes increase since the MAPE gap between models with and without partial solutions features diminishes gradually. Therefore, such trend in MAPE also makes FCNNSmall and PCNNSmall more suitable for using in larger instances. However, models with partial solutions features are still more reliable for predicting optimal tour lengths despite their higher computational costs.

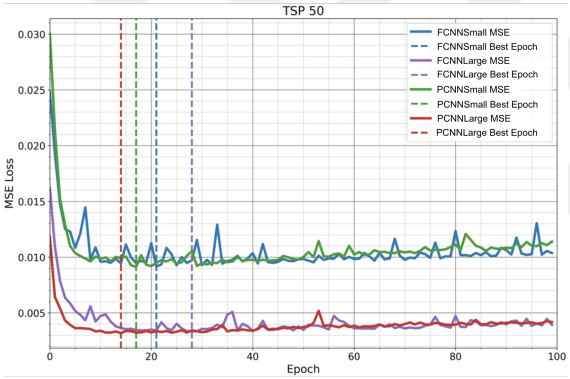
Although the impact of using partially connected layer architectures (PCNNSmall and PCNNLarge vs. FCNNSmall and FCNNLarge) is not as apparent as the advantage of using partial solutions block (FCNNLarge and PCNNLarge vs. FCNNSmall and PCNNSmall) on improving MAPE scores, the main advantage of separating the feature blocks is that it reduces the training time significantly. Our models are much more simpler and our training data are significantly fewer (64,000 vs. up to several millions of training instances) than that of recent neural network-based studies. As a rule of thumb, training larger models with larger training data can be used to further improve the model performance. Therefore, more and more computational effort is needed. Our findings indicate that the training time can be reduced dramatically if partially connected layers are preferred. Also, models trained on larger instances tend to converge earlier, indicating that less training steps are needed for larger models to



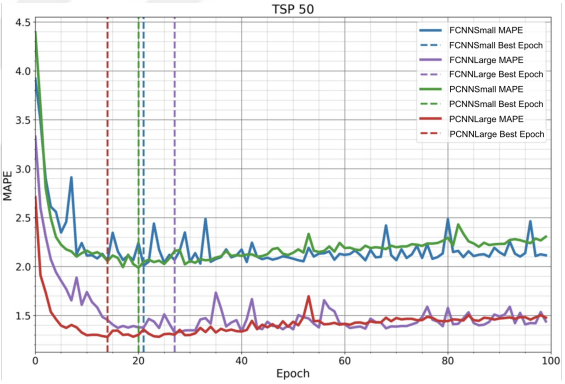
(a) MSE Loss for TSP 20 models



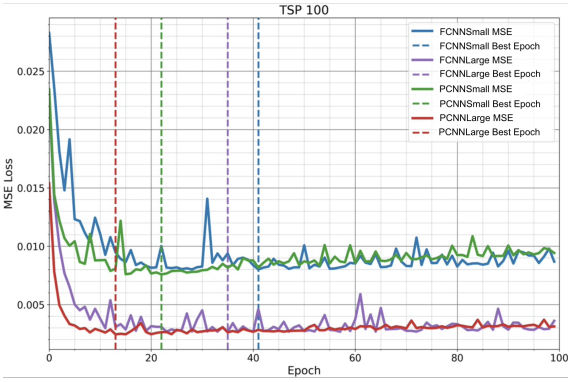
(b) MAPE for TSP 20 models



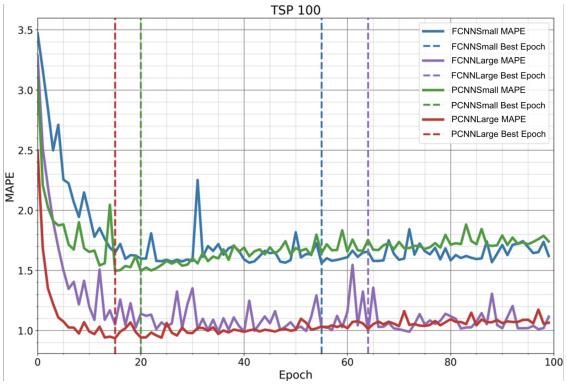
(c) MSE Loss for TSP 50 models



(d) MAPE for TSP 50 models



(e) MSE Loss for TSP 100 models



(f) MAPE for TSP 100 models

Figure 12: MSE and MAPE Progression of Models
In each plot, MSE Loss or MAPE metrics are plotted against epochs. Dashed vertical lines are the epochs when the best metric is achieved.

achieve at least the same loss metric as smaller models (See Figure 12).

In Figure 12, the learning curves of each model on different TSP sizes are presented. In order to show loss value horizon, we do not use early stopping condition when training the models.The dashed vertical lines represent the epochs where the smallest validation metrics (MAPE and MSE in our case) are achieved for each model. As it can be seen, PCNNLarge is the one that always converges first, and it is followed by PCNNSmall, and both of these models have partially connected layers. Depending on model size and the volume of training data, using partially connected layers can reduce the cost of training almost by half (i.e., it takes half the number of epochs to achieve the smallest loss value compared to fully connected layers). Such cost reduction is not meaningful for the models presented in this study since each epoch takes around a second or less. However, as a rule of thumb, combining millions of training data and models with lots of layers lead to better models. Considering such a scenario, reducing the training costs could be extremely valuable, and using partially connected layers is one of the options for this.

5.5 Benchmarking Against the Best Tour Length Estimation Model

We benchmark our models with the best tour length estimation model proposed by [37] (CS). CS outperforms similar approaches in terms of expected E / O ratio and standard deviation of E / O ratio on distribution-free test instances. In addition, CS is the first example model designed to work well even on distribution-free TSP instances. We compare our models with CS under three scenarios. We compare the models on our instances in the first scenario. Next, we compare models on their instances. Lastly, the comparison is made on instance types not included in both training data. We follow their instance generation procedure described in the original paper, but we limit the instance size to 20, 50, and 100. We refer the reader to the original text for more comprehensive descriptions of their instances.

Table 6: CPU Time(s) Required to Solve Each Instance Type Optimally

Size	**G1.1**	**G2.2**	**G3.1**	**G3.2**	**G3.3**	**G4**	**SG**	**US**	**UR**	**NS**	**NR(Ours)**
20	28	28	25	30	29	37	22	11	29	13	34
50	120	122	110	144	149	225	61	52	201	66	210
100	268	251	176	174	271	412	196	251	433	284	495

Values are Concorde's optimal solution time in terms of CPU seconds per 1000 instance. $G1.1$ represents CS's training instances.

We present comparison results on eleven different instance types. For some instance types, we use the same naming conventions used in the original paper. In addition, we also abbreviate some of the instance types that we use in our comparisons. SG, NR, NS, UR, US abbreviations stand for Small Graphs [37], Nonuniform Rectangular (our instances), Nonuniform Square, Uniform Rectangular and Uniform Square instances, respectively. Simply, the first letter in NR, NS, UR and US represents the node distribution function, and the second letter represents the service region shape. We create NS instances in a similar way to NR instances (our instances). The only difference is that we generate NS instances over a 1000×1000 square region. We generate UR and US test instances with uniform node distributions. Similar to NS instances, we crate US instances over a 1000×1000 square region. To demonstrate how hard each of these test instances to solve optimally, we measure the time required by Concorde to solve 1000 examples of each instance type, and we present these run times in Table 6.

In Table 7, we present the results of the first scenario. In addition, all of our models outperform the model of [37] on each instance size in terms of MAPE scores. However, their model runs slightly faster than our quickest models (FCNNSmall and PCNNSmall). The performance difference between our models and CS diminishes as the instance size increases. Also, their model has significantly higher variance and a considerably wider prediction range than even our worst-performing models (FCNNSmall and PCNNSmall). Regarding the fact that we use diversified instance

Table 7: Comparison with CS on Our Instances (Scenario #1)

Method	Size	MAPE	Mean E/O	Std. Dev. E/O	Min E/O	Max E/O	Time(s)[CPU]
CS	20	20.82	0.89	0.28	0.02	1.77	0.2
FCNNSmall	20	2.48	1.00	0.03	0.89	1.17	0.3
FCNNLarge	20	1.71	1.00	0.02	0.90	1.08	4.5
PCNNSmall	20	2.39	1.00	0.03	0.87	1.14	0.3
PCNNLarge	20	1.63	1.00	0.02	0.90	1.01	4.5
CS	50	16.81	0.97	0.25	0.04	1.73	0.2
FCNNSmall	50	1.95	1.00	0.03	0.87	1.13	0.6
FCNNLarge	50	1.24	1.00	0.02	0.94	1.05	38
PCNNSmall	50	1.93	1.00	0.03	0.91	1.11	0.6
PCNNLarge	50	1.22	1.00	0.02	0.95	1.05	38
CS	100	13.88	0.99	0.23	0.04	2.47	0.2
FCNNSmall	100	1.52	1.00	0.02	0.92	1.12	1.2
FCNNLarge	100	1.02	1.00	0.01	0.94	1.05	260
PCNNSmall	100	1.50	1.00	0.02	0.92	1.11	1.2
PCNNLarge	100	0.94	1.00	0.01	0.93	1.04	260

types in model training, it is not surprising to observe such performance differences between our models and CS.

In Table 8, we present the benchmarking results for the second and third scenarios together. For conciseness purposes, we trim some of the statistics previously provided in Table 5.5, and we present only MAPE scores of each model on each test set. As for G4 Graphs, we only use $m \in \{5, 10, 15\}$ where m is the number of corners in the convex hull of the service region.

In the second scenario, Table 8 shows that even our worst-performing models display better performance than CS on not only their training instances (i.e., G1.1 instances) but also each test instance set. However, the difference between model performances diminishes as instance sizes increase. The gap between our models and their model is the largest on G3.1 and G4 instances. In the original paper, the authors refer to G3.1 instances as the instances with cavity node dispersion, similar to cluster-like node accumulations around corners of a rectangular convex which is slightly similar to our instances morphologies. The main difference is that our clusters are not necessarily around the corners of a rectangular convex hull. G4 instances are instances with uniformly distributed nodes, and the specific property of these

Table 8: MAPE Scores of Models on Different Test Instance Sets (Scenarios #2, #3)

		Scenario #2							**Scenario #3**		
Method	**Size**	**G1.1**	**G2.2**	**G3.1**	**G3.2**	**G3.3**	**G4**	**SG**	**US**	**UR**	**NS**
CS	20	12.21	12.67	26.54	16.77	18.83	21.88	9.84	8.51	19.86	11.50
FCNNSmall	20	8.38	8.47	6.59	7.06	7.53	2.41	7.05	3.04	2.29	3.00
FCNNLarge	20	5.17	5.38	3.02	4.06	4.07	1.57	4.69	1.58	1.55	1.66
PCNNSmall	20	8.66	8.84	7.00	7.22	7.22	2.36	8.05	3.14	2.26	3.04
PCNNLarge	20	4.60	4.94	3.34	4.39	4.59	1.57	4.42	1.54	1.51	1.66
CS	50	5.17	5.51	13.01	7.78	9.02	15.88	3.43	8.31	14.15	12.99
FCNNSmall	50	4.89	4.78	4.08	3.55	4.02	2.10	3.62	1.89	1.87	2.20
FCNNLarge	50	2.19	2.18	1.48	1.92	2.18	1.32	1.54	1.03	1.19	1.14
PCNNSmall	50	5.17	5.08	3.89	3.72	4.10	2.12	4.17	1.83	1.84	2.23
PCNNLarge	50	2.28	2.30	1.56	2.16	2.45	1.36	1.69	1.00	1.14	1.17
CS	100	3.88	3.68	9.05	4.48	5.61	9.47	2.71	4.34	10.16	9.14
FCNNSmall	100	3.04	3.20	2.45	2.20	2.50	1.64	1.76	1.22	1.39	1.57
FCNNLarge	100	1.46	1.52	1.24	1.36	1.77	0.99	0.93	0.67	0.85	0.81
PCNNSmall	100	2.89	2.98	1.92	2.26	2.25	1.59	2.24	1.14	1.23	1.51
PCNNLarge	100	1.34	1.40	1.12	1.33	1.45	0.91	1.04	0.68	0.79	0.77

instances is that their service regions are irregular-shaped convex polygons. They use strictly rectangular service regions when generating their training instances; therefore, their model does not perform well on G4 graphs (i.e., out of the training distribution instances). Contrarily, we do not impose any service region shapes on our instances; thus, even our worst performing model significantly outperforms their model on these instances thanks to our instances, modeling approach, and well engineered features.

In the third scenario, we benchmark models on US, UR, and NS instances. Despite the fact that our training data contains none of these instance types, our models achieve under 0.7 MAPE on US instances, and similar performance on UR and US instances. Considering the easiness of solving US instances optimally (See Table 6), predicting the optimal tour lengths of these instances is relatively easier task. We do not include any US instances in our training data, and all models achieve the best MAPE score on US instances due to our more generalizable instance generation method and modeling approach. This also demonstrates that learning from challenging TSP instances could enable models to generalize well on easy to solve instances.

Similarly, our models achieve under 1 MAPE score on UR and NS instances since both types are more straightforward to solve than our NR instances. They do not include US or UR instances in their training data but include UR instances. However, their model predicts higher quality optimal tour length values on square instances (US and NS instances) than UR instances. This also shows that learning from rectangular instances has also positive effect on generalization over instances with square shaped service region.

5.6 Benchmarking Against the Best Tour Generating Models

We also benchmark our models against the recent deep learning-based TSP tour constructing models. There are several commonalities amongst these models. The first one is that all of them are trained on US instances. The second one is that these models require improvement steps to improve the initial solutions. After improvement, the run time of the algorithms significantly exceeds the run time of the solvers even when running the models on the GPUs, yet these models could not outperform solvers in terms of the solution qualities. Therefore, using those models for fast approximation of optimal tour length is a fruitless effort. To make a justifiable comparison with our models, we select the best model that generates the fastest initial solution without using any improvement step. The models proposed by [51] (KHW) produce the best solutions for US instances in terms of optimality gap and run time.

Technically, tour length estimations obtained from KHW cannot be smaller than optimal tour lengths. However, our estimations can be lower than optimal tour lengths; thus, instead of reporting the optimality gap, we report the MAPE scores to make a sensible comparison between these methods. In addition, their models run exceptionally fast on GPUs due to extremely parallelizable model infrastructure. Therefore, we run their models on both CPU and GPU and report their run times for each setting. On the other hand, our models are pretty lightweight, and they can run

Table 9: Comparison with KHW on Our Instances (Scenario #1)

Method	Size	MAPE	Mean E/O	Std. Dev. E/O	Min E/O	Max E/O	Time(s)[CPU]	Time(s)[GPU]
KHW	20	1.53	1.02	0.02	1.00	1.19	14	0.1
FCNNSmall	20	2.48	1.00	0.03	0.89	1.17	0.3	0.3
FCNNLarge	20	1.71	1.00	0.02	0.90	1.08	4.5	1.2
PCNNSmall	20	2.39	1.00	0.03	0.87	1.14	0.3	0.3
PCNNLarge	20	1.63	1.00	0.02	0.90	1.01	4.5	1.2
KHW	50	22.6	1.23	0.51	1.00	8.56	35	0.2
FCNNSmall	50	1.95	1.00	0.03	0.87	1.13	0.6	0.4
FCNNLarge	50	1.24	1.00	0.02	0.94	1.05	38	8
PCNNSmall	50	1.93	1.00	0.03	0.91	1.11	0.6	0.4
PCNNLarge	50	1.22	1.00	0.02	0.95	1.05	38	8
KHW	100	63.1	1.63	0.76	1.00	5.57	70	0.4
FCNNSmall	100	1.52	1.00	0.02	0.92	1.12	1.2	0.6
FCNNLarge	100	1.02	1.00	0.01	0.94	1.05	260	55
PCNNSmall	100	1.50	1.00	0.02	0.92	1.11	1.2	0.6
PCNNLarge	100	0.94	1.00	0.01	0.93	1.04	260	55

GPU run times of our models are computed by parallelizing feature computation pipeline on CPU and making estimations on GPU.

on CPU and GPU with almost the same efficiency as we observe in our experiments. As stated before, our features are crafted manually and then fed to models for prediction. Partial solution features require the highest computational effort among all feature blocks since it requires a Gurobi Solver call for each instance. It is impossible for us to compute those features on GPU for the time being since the feature block requires the solution of LPs. To increase the efficiency of our models, we parallelize the entire feature computation pipeline on CPU cores such that we process 12 TSP instances simultaneously. Then we compare our parallel CPU runtime with KHW's GPU runtime which is parallel in default. Unfortunately, parallelization with this magnitude makes our pipeline run only a few times faster than the sequential solution. However, it is important to demonstrate that having more CPU cores will make our pipeline finish faster and scale better just like how GPUs speed up processes. We also report the sequential run times of all models on the CPU.

We compare our models with KHW under the same scenarios that we conduct against CS. Table 9 presents the comparison under the first scenario. When the size is 20, their model performs slightly better than our models in terms of MAPE and GPU run time. Using only CPU, the fastest models are still our models (FCNNSmall

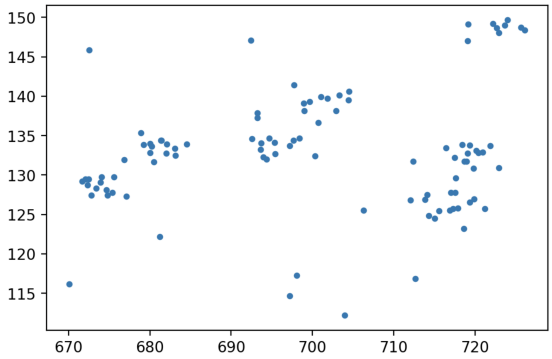


Figure 13: TSP with Dense Clusters

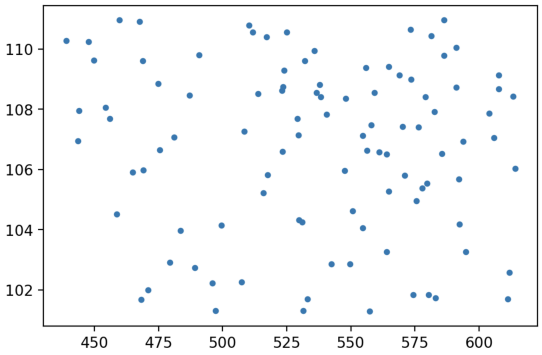


Figure 14: TSP with Extreme h/w Ratio

and PCNNSmall). As for larger TSPs, our models produce far superior optimal tour length estimations. Even though their models run fastest with GPU, the quality of their predictions gets worse, making them useless for estimating the optimal tour lengths. We also show that their models produce pretty low-quality tours and tour lengths under two pathological conditions. The first condition is the presence of dense clusters (See Figure 13). For the instance in Figure 13, their model estimates the optimal tour length 4.54 times the optimal tour length. However, FCNNSmall, FCNNLarge, PCNNSmall, and PCNNLarge estimate the 1.01, 1.00, 1.00, and 1.00 times the optimal tour length, respectively. The second condition is that value of $\max\left\{\frac{width}{height}, \frac{height}{width}\right\}$ significantly deviates from 1 (See Figure 14). For the instance in Figure 14, their model estimates the optimal tour length 4.71 times the optimal tour length. While the same models estimate the 1.03, 1.01, 1.02, and 1.01 times the optimal tour length, respectively.

In the second scenario, we compare all models on training instances of KHW. US instances are similar to their instances. The only difference is that US instances created over 1000×1000 square regions while they generate their instances over a unit square. Since their models only work with instances having node coordinates within $[0, 1]$ interval, we cast US instances into a unit square instead of simply dividing the coordinate values by 1000. This casting strategy also lead their models to achieve

Table 10: MAPE Scores of Models on Different Test Instance Sets (Scenario #2 and Scenario #3)

		Scenario #2	**Scenario #3**								
Method	**Size**	**US**	**G1.1**	**G2.2**	**G3.1**	**G3.2**	**G3.3**	**G4**	**SG**	**UR**	**NS**
KHW	20	0.24	1.57	1.61	0.97	1.94	1.90	2.46	0.74	1.49	0.47
FCNNSmall	20	3.04	8.38	8.47	6.59	7.06	7.53	2.41	7.05	2.29	3.00
FCNNLarge	20	1.58	5.17	5.38	3.02	4.06	4.07	1.57	4.69	1.55	1.66
PCNNSmall	20	3.14	8.66	8.84	7.00	7.22	7.22	2.36	8.05	2.26	3.04
PCNNLarge	20	1.54	4.60	4.94	3.34	4.39	4.59	1.57	4.42	1.51	1.66
KHW	50	1.62	26.14	29.25	16.34	21.96	21.84	11.65	4.73	16.29	2.32
FCNNSmall	50	1.89	4.89	4.78	4.08	3.55	4.02	2.10	3.62	1.87	2.20
FCNNLarge	50	1.03	2.19	2.18	1.48	1.92	2.18	1.32	1.54	1.19	1.14
PCNNSmall	50	1.83	5.17	5.08	3.89	3.72	4.10	2.12	4.17	1.84	2.23
PCNNLarge	50	1.00	2.28	2.30	1.56	2.16	2.45	1.36	1.69	1.14	1.17
KHW	100	4.32	67.41	72.68	53.36	74.46	78.44	89.2	101.46	64.5	6.09
FCNNSmall	100	1.22	3.04	3.20	2.45	2.20	2.50	1.64	1.76	1.39	1.57
FCNNLarge	100	0.67	1.46	1.52	1.24	1.36	1.77	0.99	0.93	0.85	0.81
PCNNSmall	100	1.14	2.89	2.98	1.92	2.26	2.25	1.59	2.24	1.23	1.51
PCNNLarge	100	0.68	1.34	1.40	1.12	1.33	1.45	0.91	1.04	0.79	0.77

slightly better scores than what they report in the original paper. In the third benchmarking scenario, we compare models on completely different test instances than instances that these models are trained on. Therefore, we use [37]'s instances along with UR and NS instances. Table 10 presents each MAPE scores of each method on each of these test instances. Table 10 shows that KHW models are robust to instance morphologies when the problem size is small. Therefore, their model produces competitive optimal tour length estimations. However, their models are vulnerable to unseen instance morphologies of larger instances. Unsurprisingly, our models show more consistent prediction performances across different test instance types. Using robust features such as partial solution blocks enables our models to generalize well even on completely new morphologies. Considering all results provided in Table 10, it is clear that our models are far more reliable for estimating the optimal tour length.

CHAPTER VI

AN APPLICATION

This section presents a powerful application where we integrate our objective value estimators in an efficiently implemented Genetic Algorithm (GA). To demonstrate how potent and efficient our proposed solution framework (Figure 1) and estimators are, we intentionally keep the implementation of our GA straightforward as we want our estimators to be the main player. We consider Equality Generalized Traveling Salesperson Problem (E-GTSP), in which the salesperson aims to visit only one city from n clusters, resulting in an n-city TSP problem. [60] has demonstrated the importance of the problem with various applications.

In our GA implementation, we represent chromosomes with a list of cities $\mathbf{C}$: $\{C_1, C_2, C_3, ..., C_n\}$ where each gene C_k, $k \in \{1, 2, 3, ..., n\}$, is a city from k^{th} cluster. It is important to note that these chromosomes do not represent tours; rather, they represent TSP instances which are basically what our neural network estimators expect to make predictions. Then, the fitness values of chromosomes are computed by making predictions with our neural network estimators. The lower the predicted optimal tour length value, the higher the fitness of a chromosome is. To create a mating pool, we rank the population based on fitness values and select the highest ranked e individuals as *elites*, and we copy them to the mating pool. In order to fill the pool, we apply a tournament selection with randomly generated leagues with l individuals. The fittest individuals from each league are selected and then added to the pool. In the breeding stage, we randomly sample 2 parents from the mating pool with replacement. When breeding, we utilize a crossover operator that exchanges genes from the exact chromosome locations between parents. Next, we mutate each

offspring with a probability of $\frac{1}{n}$ such that we replace the mutated gene (city) with another city from the same cluster. Furthermore, since our estimations have errors, we do not wholly rely on estimator predictions when determining the best tours. Instead, we collect the most promising individuals s from each generation. After the GA terminates, we solve each of these collected individuals (which are several n-city TSP instances) by using Concorde and pick the one yielding the shortest tour length as a GA solution. In this aspect, we leverage machine learning (ML) to search only a tiny fraction of the entire solution space and find high-quality solutions quickly.

6.1 Case Study on E-GTSP

We create random TSP instances of variable sizes (we refer total number of cities m), then generate n clusters from these m cities to define E-GTSP instances. In GTSPLIB, the ratio of $\lceil \frac{m}{n} \rceil$ is set to be 5; however, in our E-GTSP instances, we extend this ratio to extreme values such as 500, and 1000 to make these instances even harder. We generate benchmark E-GTSP instances with 100 clusters based on uniform DIMACS instances [61].

As far as E-GTSP is concerned, there are state-of-the-art E-GTSP solvers capable of providing a high-quality solution in a reasonable time. GK [62] and GLKH [23] have the same solution strategy that they first apply a global search (GS) routine, then refine the GS solution with a local search (LS) step. GLKH has a higher ratio of finding the optimal solutions than GK on GTSPLIB; therefore, we pick GLKH to benchmark our GA. When we compare our GA with GLKH, we run both algorithms with their default parameters. We provide our default parameters in Table 11. Moreover, our GA does not employ any post LS step; thus, we integrate GLKH's post optimization routine into our GA. We report both algorithms' solution performances based on their final solution and run time in Table 12.

Table 12 is generated by running each algorithm five times with different random

Table 11: Default Parameter Configuration of Our GA

Population Size	#Generations	Collection Size *(s)*	Mutation Rate	Elite Size *(e)*	League Size *(l)*
5000	150	100	0.01	1000	3

Table 12: Comparisons of Algorithms On Randomly Generated E-GTSP Instances with 100 Clusters and Varying Sizes

	GLKH			GA(Ours)				
Instance Name	Best Obj	Worst Obj	Avg Time(s)	Best Obj	Worst Obj	Avg Time(s)	Gap(%)	Speed Up
100VOA2500	**5827026**	5989189	1120	5928020	5994751	675	1.73	1.66
100VOA5000	5570392	5694197	1763	**5528488**	5627550	699	-0.75	2.52
100VOA10000	5579531	5899773	2987	**5525534**	5624343	690	-0.97	4.31
100VOA25000	5699373	5744016	7436	**5432869**	5515831	801	-4.68	9.28
100VOA50000	-	-	20845	**5313370**	5384423	993	-	21
100VOA100000	-	-	62400	**5335011**	5377796	2312	-	27

Best objective found for each instance is written in bold. "-" entries indicate that GLKH fails to find a feasible solution on that instance. Avg Time column is each method's run time under default parameters for each instance regardless of the feasibility of a solution.

seeds on an Ubuntu machine with AMD Ryzen Threadripper 3970X CPU and 64 GB of memory. In the table, the first numeric part in the instance names is the number of clusters, and the second is the number of cities. In that sense, our random instances have the total number of cities ranging from 2500 to 100000. The best solutions found for each instance are written in bold. Objective values denoted with "-" entries indicate that GLKH fails to find a feasible solution. The average solution times (Avg Time) are based on the GLKH's run time under default parameters, even if no feasible solution is found. The percentage gap is computed based on GLKH's best objective value such that negative values show that GA finds better solutions, and positive values show the other way round. Speed Up column is simply the ratio of GLKH's average runtime to GA's runtime under default parameters (regardless of whether a solution is feasible or not).

We consider our approach as *pseudo-enumeration* since we intelligently enumerate over a tiny collection of selected individuals by exploiting the efficiency of neural networks. Roughly speaking, there are around 1000^{100} unique TSP instances within *100VOA100000*. However, we consider only 15000 from the entire instance space.

This enumeration method works exceptionally well in larger instances (i.e., instances with more than 10000 cities) as GA reduces the run time dramatically (up to 27 times) and improves the solution significantly (See Table 12) compared to GLKH. Despite the apparent superiority of our method on massive instances, our method performs slightly worse in the smallest instance than GLKH. It performs better on the moderate-sized ones in terms of solution quality. More importantly, there is a great run time advantage of GA on all of these instances.

Considering the primary goal of our proposed framework (See Figure 1), which is finding high-quality solutions in a short period using machine learning, we show that our powerful method could easily outperform the state-of-the-art solver on massive optimization problems in terms of solution quality and run time. Moreover, better parameter configurations and better algorithm design are possible. For example, we start collecting promising individuals from the beginning; however, earlier generations have worse fitness values than the later generations. Therefore, the design of a more efficient individual collection procedure is possible. Also, we can achieve much better solutions than default parameters in given instances (up to %10 better solutions than GLKH), indicating that when using a machine learning backbone in a metaheuristic method, extensive parameter study is also required.

CHAPTER VII

CONCLUSION

This study develops multiple estimation models for estimating the optimal tour length of TSP instances with arbitrary node distributions. We fill certain gaps in the machine learning and TSP literature by showing that learning from combinatorial problems fails to generate valuable and generalizable results when the model training scope is limited with easy to solve and morphologically similar instances. In addition, we demonstrate the importance of a deep understanding of the problem and solution strategies to develop robust and efficient features and models. We also introduce a simple method by using one of our estimators to solve massive-scale E-GTSP instances that even the state-of-the-art solvers struggle with solving. We show that even with a simple metaheuristic design, our proposed method enables enumeration through a tiny fraction of an immense solutions space, yielding significantly higher-quality solutions and shorter run times.

Even though current machine learning models are still far from the performances of state-of-the-art solvers, they can be utilized for yielding fast and accurate estimations later to be integrated into metaheuristic frameworks, as demonstrated in this study. Compared to the best TSP tour length estimation models in the literature, we achieve up to 100 times reduction in error metric across different instance types while keeping the run time quite efficient. Moreover, when we use these estimators to solve extremely large optimization problems using the proposed solution framework, we can reduce the solution time dramatically while improving the quality of solutions significantly. Considering all, we show that our proposed models have a high potential for improving the run time and solution quality in cluster-first route-second type

approaches for solving massive routing problems.

REFERENCES

[1] R. Agarwala, D. L. Applegate, D. Maglott, G. D. Schuler, and A. A. Schäffer, "A fast and scalable radiation hybrid map construction and integration strategy," *Genome Research*, vol. 10, no. 3, pp. 350–364, 2000.

[2] C. Hitte, T. D. Lorentzen, R. Guyon, L. Kim, E. Cadieu, H. G. Parker, P. Quignon, J. K. Lowe, B. Gelfenbeyn, and C. Andre, "Comparison of MultiMap and TSP/CONCORDE for constructing radiation hybrid maps," *Journal of Heredity*, vol. 94, no. 1, pp. 9–13, 2003.

[3] O. Johnson and J. Liu, "A traveling salesman approach for predicting protein functions," *Source Code for Biology and Medicine*, vol. 1, no. 1, pp. 1–7, 2006.

[4] G. Reinelt, "TSPLIB—A traveling salesman problem library," *ORSA Journal on Computing*, vol. 3, no. 4, pp. 376–384, 1991.

[5] K.-Y. Fok, C.-T. Cheng, K. T. Chi, and N. Ganganath, "A relaxation scheme for tsp-based 3d printing path optimizer," in *2016 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC)*, pp. 382–385, IEEE, 2016.

[6] N. Christofides, A. Mingozzi, and P. Toth, "Exact algorithms for the vehicle routing problem, based on spanning tree and shortest path relaxations," *Mathematical Programming*, vol. 20, no. 1, pp. 255–282, 1981.

[7] G. Laporte, "The vehicle routing problem: An overview of exact and approximate algorithms," *European Journal of Operational Research*, vol. 59, no. 3, pp. 345–358, 1992.

[8] R. M. Karp, "Reducibility among combinatorial problems," in *Proceedings of a symposium on the Complexity of Computer Computations* (R. E. Miller and J. W. Thatcher, eds.), The IBM Research Symposia Series, pp. 85–103, Plenum Press, New York, 1972.

[9] D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook, *The traveling salesman problem: a computational study*. Princeton University Press, 2006.

[10] R. Halper and S. Raghavan, "The mobile facility routing problem," *Transportation Science*, vol. 45, no. 3, pp. 413–434, 2011.

[11] S. K. Jacobsen and O. B. G. Madsen, "A comparative study of heuristics for a two-level routing-location problem," *European Journal of Operational Research*, vol. 5, no. 6, pp. 378–387, 1980.

[12] C. F. M. Stokx and C. B. Tilanus, "Deriving route lengths from radial distances: Empirical evidence," *European Journal of Operational Research*, vol. 50, no. 1, pp. 22–26, 1991.

[13] C. F. Daganzo, "The distance traveled to visit N points with a maximum of C stops per vehicle: An analytic model and an application," *Transportation Science*, vol. 18, no. 4, pp. 331–350, 1984.

[14] D. E. Blumenfeld and M. J. Beckmann, "Use of continuous space modeling to estimate freight distribution costs," *Transportation Research Part A: General*, vol. 19, no. 2, pp. 173–187, 1985.

[15] T. W. Chien, "Heuristic procedures for practical-sized uncapacitated location-capacitated routing problems," *Decision Sciences*, vol. 24, no. 5, pp. 995–1021, 1993.

[16] G. Nagy and S. Salhi, "Nested heuristic methods for the location-routeing problem," *Journal of the Operational Research Society*, vol. 47, no. 9, pp. 1166–1174, 1996.

[17] G. Nagy and S. Salhi, "Location-routing: Issues, models and methods," *European Journal of Operational Research*, vol. 177, no. 2, pp. 649–672, 2007.

[18] S. Liu, L. He, and Z.-J. Max Shen, "On-time last-mile delivery: Order assignment with travel-time predictors," *Management Science*, vol. 67, no. 7, pp. 4095–4119, 2021.

[19] J. Beardwood, J. H. Halton, and J. M. Hammersley, "The shortest path through many points," *Proceedings of the Cambridge Philosophical Society*, vol. 55, p. 299, Jan. 1959.

[20] M. Bellmore and J. C. Malone, "Pathology of traveling-salesman subtour-elimination algorithms," *Operations Research*, vol. 19, no. 2, pp. 278–307, 1971.

[21] C. L. Valenzuela and A. J. Jones, "Estimating the Held-Karp lower bound for the geometric TSP," *European Journal of Operational Research*, vol. 102, no. 1, pp. 157–175, 1997.

[22] M. Jünger and W. R. Pulleyblank, "Geometric duality and combinatorial optimization.," in *Jahrbuck Überblicke Mathematik* (S. D. Chatterji, B. Fuchssteiner, U. Kluish, and R. Liedl, eds.), pp. 1–24, Brunschweig/Wiesbaden, Germany: Vieweg, 1993.

[23] K. Helsgaun, "Solving the equality generalized traveling salesman problem using the Lin–Kernighan–Helsgaun algorithm," *Math. Program. Comput.*, vol. 7, pp. 269–287, Sept. 2015.

[24] P. C. Mahalanobis, "A sample survey of the acreage under jute in Bengal," *Sankhyā: The Indian Journal of Statistics (1933-1960)*, vol. 4, no. 4, pp. 511–530, 1940.

[25] E. Bonomi and J.-L. Lutton, "The N-city travelling salesman problem: Statistical mechanics and the Metropolis algorithm," *SIAM Review*, vol. 26, no. 4, pp. 551–568, 1984.

[26] H. L. Ong and H. C. Huang, "Asymptotic expected performance of some TSP heuristics: an empirical evaluation," *European Journal of Operational Research*, vol. 43, no. 2, pp. 231–238, 1989.

[27] W. Krauth and M. Mézard, "The cavity method and the travelling-salesman problem," *EPL (Europhysics Letters)*, vol. 8, no. 3, p. 213, 1989.

[28] C.-N. Fiechter, "A parallel tabu search algorithm for large traveling salesman problems," *Discrete Applied Mathematics*, vol. 51, no. 3, pp. 243–267, 1994.

[29] J. Lee and M. Choi, "Optimization by multicanonical annealing and the traveling salesman problem," *Physical Review E*, vol. 50, no. 2, p. R651, 1994.

[30] M. G. Norman and P. Moscato, "The euclidean traveling salesman problem and a space-filling curve," *Chaos, Solitons & Fractals*, vol. 6, pp. 389–397, 1995.

[31] A. G. Percus and O. C. Martin, "Finite size and dimensional dependence in the Euclidean traveling salesman problem," *Physical Review Letters*, vol. 76, no. 8, p. 1188, 1996.

[32] D. S. Johnson, L. A. McGeoch, and E. E. Rothberg, "Asymptotic experimental analysis for the Held-Karp traveling salesman bound," in *Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 341–350, ACM Press San Francisco, 1996.

[33] S. Eilon, C. D. T. Watson-Gandy, and N. Christofides, *Distribution Management: Mathematical Modelling and Practical Analysis*. Griffin, London, 1971.

[34] B. L. Golden, "A statistical approach to the TSP," *Networks*, vol. 7, no. 3, pp. 209–225, 1977.

[35] T. W. Chien, "Operational estimators for the length of a traveling salesman tour," *Computers & Operations Research*, vol. 19, no. 6, pp. 469–478, 1992.

[36] O. Kwon, B. Golden, and E. Wasil, "Estimating the length of the optimal TSP tour: An empirical study using regression and neural networks," *Computers & Operations Research*, vol. 22, no. 10, pp. 1039–1046, 1995.

[37] B. Cavdar and J. Sokol, "A distribution-free TSP tour length estimation model for random graphs," *European Journal of Operational Research*, vol. 243, no. 2, pp. 588–598, 2015.

[38] S. Lin and B. W. Kernighan, "An effective heuristic algorithm for the traveling-salesman problem," *Operations Research*, vol. 21, no. 2, pp. 498–516, 1973.

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

[40] K. Helsgaun, "An effective implementation of the Lin–Kernighan traveling salesman heuristic," *European Journal of Operational Research*, vol. 126, no. 1, pp. 106–130, 2000.

[41] J. J. Hopfield and D. W. Tank, ""Neural" computation of decisions in optimization problems," *Biological Cybernetics*, vol. 52, no. 3, pp. 141–152, 1985.

[42] E. H. L. Aarts and J. H. M. Korst, "Boltzmann machines and their applications," in *International Conference on Parallel Architectures and Languages Europe*, pp. 34–50, Springer, 1987.

[43] B. Angeniol, G. D. L. C. Vaubois, and J.-Y. Le Texier, "Self-organizing feature maps and the travelling salesman problem," *Neural Networks*, vol. 1, no. 4, pp. 289–293, 1988.

[44] R. Durbin and D. Willshaw, "An analogue approach to the travelling salesman problem using an elastic net method," *Nature*, vol. 326, pp. 689–691, Apr 1987.

[45] T. Kohonen, *Self-Organization and Associative Memory: 3rd Edition*. Berlin, Heidelberg: Springer-Verlag, 1989.

[46] D. E. Rumelhart and D. Zipser, "Feature discovery by competitive learning," *Cognitive Science*, vol. 9, no. 1, pp. 75–112, 1985.

[47] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *Advances in Neural Information Processing Systems, NIPS' 15*, 2015.

[48] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," in *5th International Conference on Learning Representations, ICLR*, 2017.

[49] H. Dai, E. B. Khalil, Y. Zhang, B. Dilkina, and L. Song, "Learning combinatorial optimization algorithms over graphs," in *Proceedings of the 31st International Conference on Neural Information Processing Systems, NIPS'17*, p. 6351–6361, 2017.

[50] M. Deudon, P. Cournut, A. Lacoste, Y. Adulyasak, and L.-M. Rousseau, "Learning heuristics for the tsp by policy gradient," in *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pp. 170–181, Springer, 2018.

[51] W. Kool, H. van Hoof, and M. Welling, "Attention, learn to solve routing problems!," in *7th International Conference on Learning Representations, ICLR*, 2019.

[52] C. K. Joshi, T. Laurent, and X. Bresson, "An efficient graph convolutional network technique for the travelling salesman problem," *arXiv preprint arXiv:1906.01227v2*, 2019.

[53] Y. Bengio, A. Lodi, and A. Prouvost, "Machine learning for combinatorial optimization: A methodological tour d'horizon," *European Journal of Operational Research*, vol. 290, no. 2, pp. 405–421, 2021.

[54] J. H. Ward Jr, "Hierarchical grouping to optimize an objective function," *Journal of the American Statistical Association*, vol. 58, no. 301, pp. 236–244, 1963.

[55] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2021.

[56] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[57] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, Oct. 1986.

[58] V. Nair and G. E. Hinton, "Rectified linear units improve restricted Boltzmann machines.," in *International Conference on Machine Learning*, pp. 807–814, 2010.

[59] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015*, 2015.

[60] G. Laporte, A. Asef-Vaziri, and C. Sriskandarajah, "Some applications of the generalized travelling salesman problem," *Journal of the Operational Research Society*, vol. 47, no. 12, pp. 1461–1467, 1996.

[61] D. S. Johnson and L. A. McGeoch, *Experimental Analysis of Heuristics for the STSP*, pp. 369–443. Boston, MA: Springer US, 2007.

[62] G. Gutin and D. Karapetyan, "A memetic algorithm for the generalized traveling salesman problem," *Nat. Comput.*, vol. 9, pp. 47–60, Mar. 2010.

VITA

Taha Varol received his BS degree in Industrial Engineering from Boğaziçi University in 2018. He worked as a Data Scientist at Turkish Airlines between 2016-2019. He has been working as a Graduate Teaching and Research Assistant at Özyeğin University under the supervision of Prof. Okan Örsan Özener and Dr. Erinç Albey since 2019. His research focuses on machine learning applications in large-scale route planning and optimization.