

**USING ARTIFICIAL INTELLIGENCE AND MACHINE
LEARNING TO SOLVE NP-COMPLETE GRAPH THEORY
PROBLEMS**

MUSTAFA KEMAL BİNLİ

A Thesis Submitted to
The Graduate School of Izmir University of Economics
Master Program in Computer Engineering

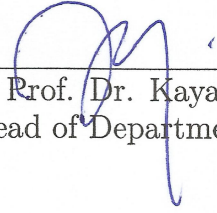
İzmir
2020

Approval of the Graduate School



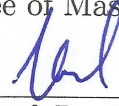
Prof. Dr. Mehmet Efe Biresselioğlu
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.



Asst. Prof. Dr. Kaya Oğuz
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.



Asst. Prof. Dr. Kutluhan Erol
Advisor

Examining Committee Members

Asst. Prof. Dr. Kutluhan Erol, Advisor



Prof. Dr. Cem Evrendilek, Member



Assoc. Prof. Dr. Ayşegül Alaybeyoğlu Soy, Member



ABSTRACT

USING ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING TO SOLVE NP-COMPLETE GRAPH THEORY PROBLEMS

Binli, Mustafa Kemal

Computer Engineering Master Program

Advisor: Asst. Prof. Dr. Kutluhan Erol

February, 2020

Computer science literature is abound with NP-complete problems with numerous practical applications ranging from logistics to information security. This thesis attempts to address such problems by drawing inspiration from heuristic search literature and machine learning, striving for optimality while improving computation time. Our approach involves utilizing A* Algorithm in conjunction with Linear Programming (LP) approximations as a heuristic function and proceeds to train a Neural Network to improve on the LP heuristic with respect to both computation overhead and reduction in search space size. We demonstrate our approach in the context of All Colors Shortest Path (ACSP), a rich graph theory problem that extends the Travelling Salesperson Problem (TSP). Our results indicate that using LP as a heuristic function reduces search space by half, while preserving optimality. Artificial Neural Network (ANN) train to replace LP as a heuristic does a much better job reducing the search space by six fold at a fraction of the computational overhead. While it is not guaranteed to be admissible, empirically it produces mostly optimal or almost optimal solutions. Note that our results are based on small problem sizes due to computational resource availability, but we believe they are sufficient to establish the viability of our approach for future studies of larger scale.

Keywords: NP-complete, ACSP, TSP, Machine Learning, ANN, A* Search Algorithm.

ÖZET

NP-TAM ÇİZGE TEORİ PROBLEMLERİNİN ÇÖZÜMÜ İÇİN YAPAY ZEKA VE MAKİNE ÖĞRENMESİ KULLANIMI

Binli, Mustafa Kemal

Bilgisayar Mühendisliği Yüksek Lisans Programı

Tez Danışmanı: Dr. Öğr. Üyesi Kutluhan Erol

Şubat, 2020

Bilgisayar bilimi literatürü, lojistikten bilgi güvenliğine kadar bir çok alanda polinom zamanda çözümü bilinmeyen NP-tam problemlerle doludur. Bu tez, sezgisel arama literatüründen ve optimallik için çabalayan makine öğrenmesinden ilham alarak bu tür problemlerin hesaplama süresini iyileştirmeye çalışır. Yöntemimiz, sezgisel bir fonksiyon olarak Doğrusal Programlama (DP) yaklaşıklıkla birlikte A* Algoritmasının kullanılmasını içerir ve hem hesaplama yükü hem de arama alanı boyutunu azaltmak açısından DP sezgiselliğini geliştirmek için bir Sinir Ağı eğitimine yönelir. Yaklaşımımızı Gezgin Satıcı Problemi (GSP) 'ni genişleten zengin bir çizge teori problemi olan All Colors Shortest Path (ACSP) bağlamında gösteriyoruz. Sonuçlarımız, sezgisel bir fonksiyon olarak DP kullanmanın, arama alanını yarı yarıya azaltırken, aynı zamanda da optimalliği koruduğunu göstermektedir. Bir sezgisel olarak DP'nin yerini alan Yapay Sinir Ağı'nı (YSA) eğitmek, hesaplama alanını altı kat azaltırken hesaplama yükünde yalnızca çok küçük bir oranda artış getirmektedir. A* algoritmasında kullanılan sezgisel fonksiyonun onanırılık özelliğini garanti edilmese de, deneysel olarak çoğunlukla optimal veya neredeyse optimal çözümler üretmektedir. Sonuçlarımız, hesaplama kaynaklarının kullanılabilirliği nedeniyle küçük problem boyutlarına dayanmakta olsa da gelecekteki daha büyük ölçekli çalışmalarda yaklaşımımızın uygulanabilirliğini belirlemek için yeterli olduğuna inanmaktayız.

Anahtar Kelimeler: NP-tam, ACSP, Makine Öğrenmesi, Yapay Sinir Ağları, A*.

ACKNOWLEDGEMENT

First and foremost, I would like to thank my academic advisor Dr. Kutluhan EROL for always believing in me and supporting me with his guidance and expertise. I consider myself very lucky that I was given an opportunity to work with him. In addition, I would like to thank Dr. Cem EVRENDİLEK, Dr. Hüseyin AKCAN, Dr. Belma ASLIM, Dr. Firdevs BİNLİ, Dr. Erkmen Giray ASLIM and Kemal DEMİR, who have contributed to the development of my thesis and encouraged me to thrive as an academic person.

From my hearth, I thank Eyüp ERTÜRK, my respected elder, who has given me endless support in every aspect possible, to ease my path on this journey.

Words are certainly not enough to express my heartfelt gratitude to my friends who never lost faith in me. I will forever be grateful to Tansu TAŞÇIOĞLU, Tarık TAŞKENT and Dünya KILAVUZ.

Last but not least, the biggest word of gratitude goes to my family for the countless opportunities they have provided to support me.

TABLE OF CONTENTS

ABSTRACT	iii
ÖZET	iv
ACKNOWLEDGEMENT	v
TABLE OF CONTENTS	vi
LIST OF FIGURES	vii
LIST OF TABLES	viii
CHAPTER 1: INTRODUCTION	1
CHAPTER 2: PRELIMINARIES	3
2.1 <i>A* and Linear Programming</i>	3
2.2 <i>Travelling Salesperson and All Colors Shortest Path Problems</i>	4
2.3 <i>Deep Learning and ANN structure</i>	5
CHAPTER 3: RELATED WORK	7
3.1 <i>Approaches for Solving NP-Complete Problems</i>	7
3.2 <i>Training ANNs</i>	8
CHAPTER 4: METHODOLOGY	11
4.1 <i>Roadmap</i>	11
4.2 <i>Some Colors Shortest Path (SCSP) and its ILP Formulation</i>	12
4.3 <i>Using A* for ACSP/SCSP</i>	14
4.3.1 <i>LP Approximation for the Heuristic Function</i>	15
4.3.2 <i>ANN Approximation for the Heuristic Function</i>	16
4.4 <i>Benchmark</i>	19
4.5 <i>Training an ANN as a heuristic function</i>	20
CHAPTER 5: EXPERIMENTAL RESULTS	25
5.1 <i>Training Results of ANN</i>	25
5.2 <i>Comparison of LP versus ANN as effective A* Heuristics</i>	27
CHAPTER 6: CONCLUSION	30
REFERENCES	35
APPENDICES	36
<i>Appendix A- *NN Estimate Results</i>	36
<i>Appendix B- A* Search Space Results</i>	40

LIST OF FIGURES

Figure 2.1.	Sample ACSP Graph	5
Figure 2.2.	Sample Multi-Layer Perceptron (MLP)	6
Figure 3.1.	Deep and Shallow Networks Comparison Using Different Neuron Size (Source : (Goodfellow et al., 2016))	9
Figure 4.1.	A*LP Search Demonstration	17
Figure 4.2.	SCSP Graph Data Representation	21
Figure 4.3.	Representation and solution of SCSP graph for 4 colors optimal path finding within a 5 colored vertex.	22
Figure 5.1.	Training Error Graph of ANN	25
Figure 5.2.	Solution Quality for ANN versus LP Estimate Ratios	27
Figure 5.3.	A* Search Space Comparison	28

LIST OF TABLES

Table 4.1. ANN Architecture for SCSP Problem Set 24

Table 5.1. Solution Quality Comparison results 26

Table 5.2. ANN versus LP CPU Time Comparison 27

Table 5.3. A* Search Space Comparison 28

Table 5.4. A*NN Optimal Cost Comparison 28

Table A.1. A*NN Estimate Results with Respect to Optimal Solution Cost of
ACSP Problem 36

Table B.1. A* Algorithm Search Space Comparison Results 40

CHAPTER 1 : INTRODUCTION

Computer science literature is abound with NP-complete problems with numerous practical applications ranging from logistics to information security. These problems remain intractable as best known algorithms perform in exponentially many steps in terms of input size, and we must be content with heuristic or approximation algorithms, which perform in reasonable time, albeit with inferior, suboptimal results. Some optimization problems are so hard that even their constant-factor approximations are NP-hard (Garey and Johnson, 1979).

In this study, we introduce a tiered approach to attacking such problems, drawing inspiration from heuristic search literature and machine learning, striving for optimality while improving computation time.

A* is a well-known AI search algorithm that can find optimal solutions when used in conjunction with an admissible heuristic function which always underestimates an optimal solution of the problem (Hart et al., 1968). As a base line, one can use a heuristic function of 0, which will find the optimal solution, at the cost of searching the entire space (Uniform Cost Search). Next, we utilize Integer Linear Programming (ILP) encodings of such problems, and use LP as a search heuristic algorithm. The approach we call A*LP combines the strengths of A* search, together with Linear Programming (LP) as an admissible heuristic function in order to optimally solve NP-complete problems. LP can be solved in polynomial time, and it provides a lower or an upper bound approximation to ILP, an NP-complete problem with well-known polynomial-time reductions from many other NP-complete problems. This method was carried out by examining the ILP model introduced for the All Colors Shortest Path (ACSP) problem in the study of (Bilge et al., 2015). While LP as an admissible heuristic guarantees optimality, and significantly reduces the search space per our experimental results, its computational overhead is too large to improve overall computation time.

Therefore, as the next step, we explore replacing LP using a machine learning approach based on neural networks, that can predict the optimal solution cost, at a fraction of LP computation time. The use of rapidly developing Artificial Neural Networks (ANN), which have proven themselves in many application areas, raises the following questions:

Can it produce an admissible heuristic for the A* algorithm? What are the optimal training structures or parameters? Can we make a good estimate of the solutions of NP-complete problems by considering ANNs? We explore the viability of our general approach in the context of graph theory problems, more specifically ACSP (All Colors Shortest Path), and our modification to it –Some Colors Shortest Path (SCSP)– with minor variety and complexity. Accordingly, we utilize manual feature map of graph properties as an input data to train Neural Network for the efficacy of our work in terms of solution quality. We compare the LP and ANN approaches for the A * algorithm. We show how these approaches affect the solution quality, the search space size and computation time of the A* algorithm.

The rest of this document is organized as follows: Chapter 2 presents preliminary information for the algorithms and methods used in this study. Chapter 3 provides a literature review of ANN's approach to problems and the A* algorithm. Chapter 4 describes the methods applied to the underlying NP-complete problem, and Chapter 5 analyses the results from the experimental data. Finally Chapter 6 summarizes the study and proposes directions for future research.

CHAPTER 2 : PRELIMINARIES

2.1 *A* and Linear Programming*

A* is an AI search algorithm, which utilizes a heuristic function $h(n)$ that given a search node n , estimates the cost of the optimum path from n to a goal node (Hart et al., 1968). Together with $g(n)$, which denotes the already established cost of reaching a node n from the starting node I , A* uses a ranking function $f(n) = g(n) + h(n)$ to explore the search space, always picking the node with the lowest $f()$ value from its search frontier, maintained in a data structure called “Open List”. A* is guaranteed to return the optimal solution, provided that the heuristic function $h()$ satisfies the admissibility constraint $0 \leq h(n) \leq h^*(n)$ for all nodes n , where $h^*(n)$ is the cost of the optimal path from n to a goal node (Hart et al., 1968).

Linear programming (LP) involves a set of real-valued variables, a set of linear constraints on those variables, and a linear objective function on those variables that we wish to optimize. In other words, it is a mathematical method used to optimize the linear relations of various problems by maximizing or minimizing a linear objective function. Several software libraries implement LP solutions in polynomial time (*CPLEX Optimizer*, 2019).

Integer Linear Programming (ILP) places the additional constraints that the variables must take integer values only. ILP is NP-complete, and thus all known algorithms for ILP runs in exponential time in the worst case (Khachiyan, 1979; Kearns et al., 1994). As ILP is NP-complete, any other NP problem can be transformed to ILP in polynomial time. There are well-known ILP encodings for many NP-complete problems, including TSP and ACSP (Bilge et al., 2015; Akcan and Evrendilek, 2017; Mohammad Reza Bonyadi and Azghadi, 2008) which will be discussed in Section 2.2.

Note that in case of a minimization problem, an LP solution is always less than equal to its ILP counterpart, as the ILP version has additional constraints. Thus, LP can be used as an admissible heuristic function for A* by encoding $h()$ for each node in the search space as a linear optimization problem. The encoding for a child node can be incrementally generated from the encoding of its parent by modifying variables and constraints.

2.2 Travelling Salesperson and All Colors Shortest Path Problems

Travelling Salesperson Problem (TSP) investigates the minimum cost of traversing the set of all cities arriving provided that a return to the origin city. The goal is to obtain a Hamiltonian cycle, which visits each vertex exactly once and having the least possible cost among all tours existing in the graph/map (Mohammad Reza Bonyadi and Azghadi, 2008). Although the name and the story behind it suggest a salesperson travelling to find the shortest path, TSP has a broader concept which can be applied in diverse range of areas such as genetics, manufacturing, telecommunications and many more (Applegate et al., 2006).

All Colors Shortest Path (ACSP), while being derived from TSP with the same core objective of finding the shortest path, has additional constraints and characteristics chasing the computational nature of the problem complexity : it may skip some vertices or it might visit some other vertices multiple times resulting in non-simple paths (Bilge et al., 2015). But note that no optimum path will traverse an edge more than once in any direction. An example ACSP graph and the optimal solution are shown in Figure 2.1. In this figure, vertices are designated by circles, and the pair of numbers in each circle denote vertex no and color no, respectively. Edges are designated by grey lines, and the rectangular box on each edge designate the weight, interpreted as the cost of traversing that edge. From the base station (vertex no : 1-3) which is starting point of the graph through the red-colored vertices, along with the bold edges mark the optimal path.

More formally, ACSP is defined as “Given an undirected graph $G(V, E)$ with a color drawn from a set C of colors assigned to each vertex, and a non-negative weight associated with each edge, ACSP is the problem of finding the shortest (possibly non-simple) path starting from a designated base vertex $s \in V$ such that every color occurs at least once on the path” (Bilge et al., 2015). Therefore, $edge(i, j) \in E$ represents the traversable undirected edge between vertex i and vertex j with $w_{i,j} = w_{j,i}$ as the cost of traversing this edge. The vertices are demonstrated as $V = \{1, \dots, n\}$ associated with the colors $C = \{1, \dots, k\}$ values. Bilge et. al. provides an ILP and LP relaxation model implemented to solve the ACSP problem (Bilge et al., 2015). General approach to solve the ILP

formulation for the graph G results in minimization of $\sum_{(i,j) \in E'} x_{i,j} * w_{i,j}$ ($x_{i,j}$ is the binary decision variable that assumes one when $edge(i,j)$ is the part of solution and zero otherwise)

In this thesis, the logical structure of the ILP model was modified in order to use it in ANN training, in data processing, and also in LP heuristic function of A* algorithm as explained in Chapter 4.

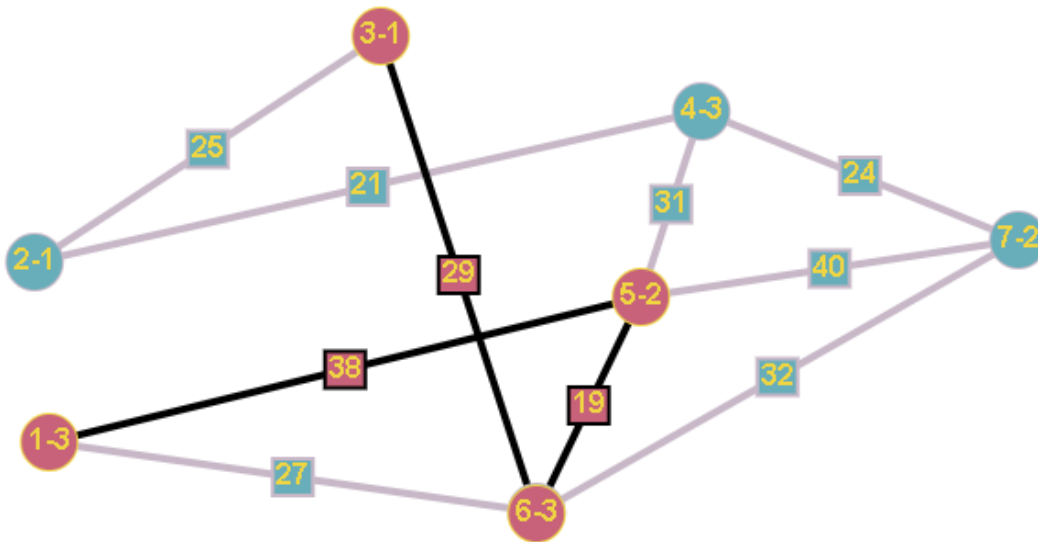


Figure 2.1. Sample ACSP Graph

2.3 Deep Learning and ANN structure

ANNs (Artificial Neural Networks) emulate the human brain in terms of structure and transfer of the information between neurons to the subsequent layer and utilize a gradient descent based feedback with the error obtained over time in the most efficient way to adapt itself to the desired structure (Jain et al., 1996). In other words, deep learning is biologically inspired by the transformation and conduction of traits that attempt to establish a relationship between the stimulants associated with the stimuli present in the brain. As a branch of Machine Learning, deep learning utilizes algorithms to simulate the brain learning process using multiple layers by providing nonlinearity via

activation functions for parallel multi-feature transformation, and to enable decision-making (Zurada, 1992). This structure also represents concepts in multiple hierarchical models that correspond to various levels of abstraction. The information passes through each layer and the output of the previous layer provides input in a dependent way for the next layer. The first layer in a network is called the input layer and the last layer is called as an output layer, while all layers between these two layers are called hidden layers. Each layer has a simple and standard algorithm that classically contains some kind of an activation function (LeCun et al., 2015). Lastly, with back-propagation algorithm by Yann Le Cun (LeCun et al., 1989), it adjusts the network parameters to minimize the error function. Basic structure of Multi-layer Perceptrons (MLP) is demonstrated in Figure 2.2.

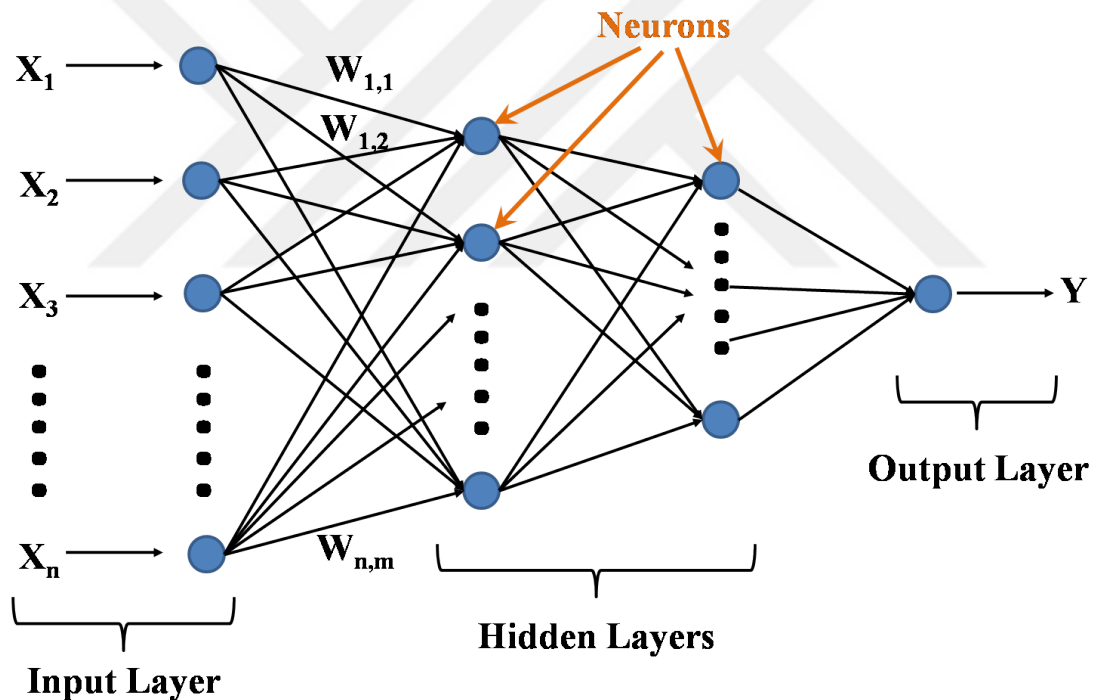


Figure 2.2. Sample Multi-Layer Perceptron (MLP)

CHAPTER 3 : RELATED WORK

NP-completeness has attracted great interest in computer science. There has been a significant range of work on ANN models for NP-complete problems (Smith, 1999; Smith and Potvin, 2001; Huang et al., 2019), spearheaded by Hopfield's work in combinatorial problems which has gained further momentum recently (Smith, 1999; Smith and Potvin, 2001). In addition, recent graph neural network studies are also promising for solving NP-Complete graph-theory problems(Huang et al., 2019; Scarselli et al., 2008).

ACSP-t -the tree version of ACSP- has an applicability to the real world practices (Akçay et al., 2018) which makes it interesting. Also, the ACSP problem has proven to be NP-complete and cannot be solved with an optimal constant factor. Thus, approximation algorithms and heuristic solutions have been developed for the graph and tree versions of ACSP (Bilge et al., 2015; Akçay et al., 2018). As such this problem is a good starting point to investigate whether we can train a new heuristic for A* to solve NP-complete problems.

3.1 Approaches for Solving NP-Complete Problems

In this section, we review approaches about ANN to solve NP-complete graph theory problems.

In 1985, Hopfield and Tank introduced Hopfield Neural Network (HNN) to tackle NP-hard problems, especially TSP (Hopfield and Tank, 1985). While their initial results seemed promising, lack of the reproducibility of their results in the study casted doubts on their effectiveness (Wilson and Pawley, 1988). Moreover, further researches limited the success with modifications to the H-T energy function (Yao and Mitra, 1988; Unaltuna and Pitchumani, 1994) and Hopfield Network to obtain that the stability by means of subspace approach and keeping all of the constraints under a single term (Gee, 1993; Aiyer et al., 1990).

Other approaches reaching from past to present are up to Kohonen's Self-Organizing Feature Map (Kohonen, 1982), elastic net (Durbin and Willshaw, 1987) and their combination. Application of these techniques to vehicle routing problems leveraging elastic band studies and Self-Organizing Feature Map do not produce feasible results

(Smith, 1999; Huang et al., 2019).

In recent times, deep learning has attracted great attention; especially Graph Neural Network (GNN). This method combines an n vertex and in a graph G in a multi-dimensional Euclidean space representing $\tau(G, n)$ function in a way that it can be implemented in compliance with all graph structures (Scarselli et al., 2008). Additionally, it is illustrated that the parameters of the graph are well learned by a supervised learning algorithm on the large dataset (Scarselli et al., 2008). Then, with the contribution of GNN, the solution of NP-complete problems became the focus. In one of these studies, it is aimed to learn how to solve the decision variables by considering the Travelling Salesperson Problem. It envisages a training effective message-passing algorithm to calculate whether the edges with weight values iteratively communicate with the vertices to determine if there is a better route than the cost determined (Prates et al., 2019). It is seen that the network trained with this method gives a promising 80% accuracy and generalizes the problem to some extent (Prates et al., 2019). In another project, it is seen that with small examples, GNN-based message passing neural network (MPNN) can be trained in combinatorial optimization problems and generalizes the problem with 85% accuracy (Selsam et al., 2018). Recently, Graph Convolution Network (GCN) has been used for the solution of NP-Hard problems by (Li et al., 2018). In this method, classical algorithms and deep learning methods are combined to focus on the solution of Maximal Independent set problem. Performance comparison shows that GCN strengthens both deep learning and classical heuristics according to other methods and approaches for solving NP-Hard problems (Li et al., 2018).

3.2 Training ANNs

In this section, we present prior research on training ANNs in order to establish a foundation to guide our approach to NP-complete problems. More specifically, we will examine the structure of our dataset for the training and development of an appropriate ANN in the literature that we will use with the A* algorithm.

To keep ANN with optimal capacity during the training phase, the training error must be as small as possible and the gap between the training error and the test error must be as diminutive as possible (Goodfellow et al., 2016). Therefore, instead of giving

direct input data to ANN, we chose to replace it with manual engineering (Goodfellow et al., 2016). Because of the importance of introducing nonlinearity to the input function, this process is crucial to reduce training error and to avoid underfitting. In addition, as depicted by (Bengio et al., 2007; Erhan et al., 2009; Goodfellow et al., 2013), the Figure 3.1 demonstrates that increasing the number of hidden layers decreases the training error by increasing the layer size in the neural network (Goodfellow et al., 2016). However, although increasing the input complexity reduces the underfitting, it is clear that the test error will move away from the training error. We know from the efforts to provide training generalization that increasing the number of training samples (Goodfellow et al., 2016) will reduce the overfitting gap between training and test errors.

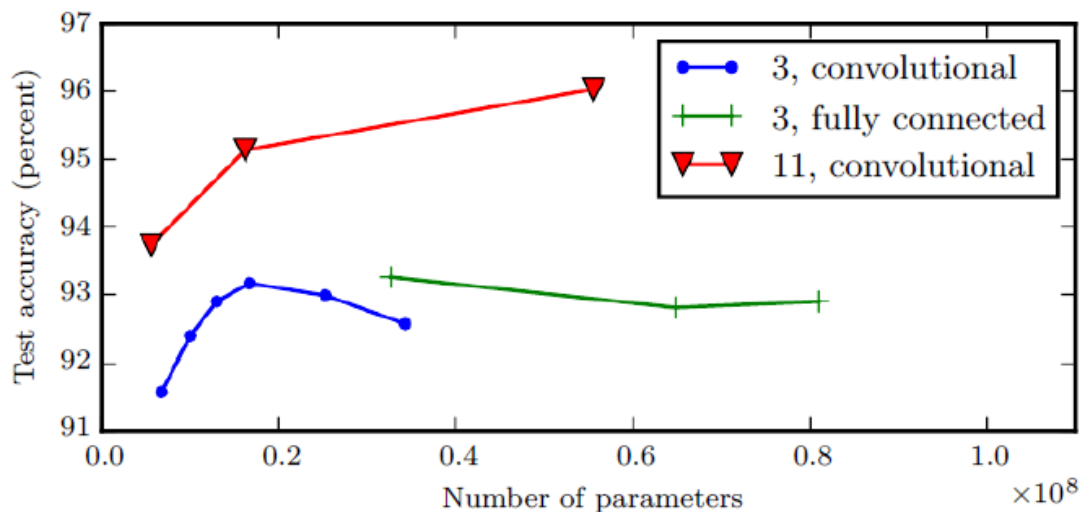


Figure 3.1. Deep and Shallow Networks Comparison Using Different Neuron Size (Source : (Goodfellow et al., 2016))

In the training phase, activation functions prevent linear transitions and ensure the stability of the neural network (Glorot and Bengio, 2010; Nair and Hinton, 2010; Jarrett et al., 2009; Glorot et al., 2011). In many aspects, it is recommended that the Rectified Linear Unit (ReLU), which is one of the activation functions in new generation approaches, is more useful in providing nonlinearity than other functions such as Sigmoid, Softsign, Tanh and Maxout (Nair and Hinton, 2010; Jarrett et al., 2009; Glorot et al., 2011). Our experiments show that using ReLU saturates training error and stabilize capacity.

Furthermore, Sigmoid or Tanh act functions could not exhibit good performance as much as ReLU.



CHAPTER 4 : METHODOLOGY

4.1 Roadmap

The goal of this thesis is to generate a heuristic function using Machine Learning as an alternative to LP approximation to drive A* search algorithm for ACSP.

In order to solve ACSP using A*, we need a heuristic function that can estimate the cost of completing the partial solution at any given node in the search space. In other words, we must estimate the cost of travelling through the remaining colors, not yet visited, on the way from the start vertex to the current vertex. In addition to the usual input for ACSP, we also provide an additional list of colors to be visited, not necessarily the entire set of colors on the graph. Thus we need to slightly modify and restate ACSP in order to address the needs of the heuristic function. We call this modified version SCSP (Some Colors Shortest Path) throughout the thesis and it should be noted that SCSP is actually equivalent to ACSP with the color of the vertices not in that list repainted to the color of the base.

Next, we modify the ILP formulation of ACSP in (Bilge et al., 2015) accordingly to handle SCSP by including in the specific colors only.

We must also evaluate how well LP approximates the ILP in our problem space, and see how much using LP as a heuristic reduces the A* search space in our problem space. With the comparison of $h() = 0$ (Uniform Cost Search) iteration quantity in the A* search algorithm, we measure how LP approximation affects the A* algorithm's search space.

Once we have shown the potential of our approach using LP, we proceed to address the significant computational overhead of using LP (not just once, but once at each search node explored by A*) by utilizing supervised machine learning to generate a much faster heuristic function. For generating the training set, we solve SCSP instances by generating the corresponding ILP encodings and solving them using CPLEX. In this way, we train an ANN to be utilized as a heuristic for the A* algorithm.

In order to generate the SCSP problem instances the previous step, we create a

benchmark of ACSP and SCSP problems, we conduct the experiments to measure the reductions in the search space when using $h() = 0$, $h() = LP$, and $h() = ANN$ respectively, both from search space and solution quality perspectives.

Finally, we proceed to present our experiment results, and their discussion in Chapter 5 and Chapter 6, respectively.

The rest of this section is organized as follows. Section 4.2 defines SCSP modification of ACSP, section 4.3 explains A* algorithm with utilizing LP function as $h(n)$ on A* and generating heuristic model of ANN, section 4.4. consists of benchmark of raw data, and lastly, section 4.5 demonstrates the training methodology of ANN in order to perform as a good heuristic model on A*.

4.2 Some Colors Shortest Path (SCSP) and its ILP Formulation

We make a minor modification to the ILP formulation given by (Bilge et al., 2015) to solve the ACSP problem with A*. In internal nodes of the search space, we must be able to estimate the cost of completing the path from the current node by traversing the remaining colors that has not been visited up to that point. Thus we must reformulate the ACSP problem (Bilge et al., 2015) in order to allow the specification of some colors required to be visited. Accordingly, we define the Some Colors Shortest Path (SCSB) problem as follows:

Given a set of colors $C = \{c_1, \dots, c_k\}$, a subset of colors $C_s \subseteq C$ and an undirected graph $G(V, E)$ with a non-negative weight value $w_{i,j}$ for each $edge(i, j) \in E$, a designated base vertex $s \in V = \{1, 2, \dots, n\}$, a color value $\kappa(v) \in C$ for each $v \in V$, SCSP is the problem of finding the shortest (possibly non-simple) path starting at s such that every color in C_s occurs at least once on the path.

For the ILP formulation(Bilge et al., 2015), we follow the approach described by Bilge et.al. and we transformed the undirected graph $G(V, E)$ to a directed graph $G'(V', E')$ by adding two new vertices with color label C_0 and zero weighted directed edges, $E' = \{(0, s)\} \cup \{(i, n + 1) | i = 1, 2, \dots, n\} \cup E$. The resulting ILP formulation is depicted in constraints (4.1) through (4.11) below. The objective function in (4.1) minimizes the

sum of the edges in the optimal solution. Constraint (4.2) guarantees that the solution always starts with a randomly selected base node $s \in V$. Constraints (4.3) ensures each distinct color in $C_s \in C$ is visited at least once and note that original formulation uses C and not C_s ! This is the only point where SCSP ILP model differs from that of ACSP. Constraint (4.4) specifies that the number of edges traversed by visiting any vertex in G is equal to the number of visited edges leaving it. This obviously applies to all nodes except for source and sink vertex in G' . Constraints 4.5, and 4.6 construct the rules related with the decision variable y_j for all $j \in V' \setminus \{0\}$. If node j navigates in a viable solution, the binary decision variable y_j takes the value of 1, zero otherwise. Constraint 4.5 assumes that visiting an edge (i, j) is a result of visiting vertex j , while Constraint 4.6 verifies the adverse. Constraint 4.7, along with 4.8, is used to eliminate any possible sub-tours, and to guarantee connectedness to the base. Constraint 4.7 ensures that the total flow in a vertex visited is equal to a value greater than the total flow from that vertex. Constraint 4.8 is responsible for the regulation of the flow values. The flow is associated with the edge only if that edge is part of a solution. Lastly, the constraints 4.9, 4.10, and 4.11 are the integrality constraints for the decision variables $x_{i,j}$, y_j , and $f_{i,j}$, respectively.

$$\text{Minimize} \quad \sum_{(i,j):(i,j) \in E'} x_{i,j} * w_{i,j} \quad (4.1)$$

$$\text{Subject to} \quad x_{0,s} = 1 \quad (4.2)$$

$$\sum_{\substack{(i,j):(i,j) \in E' \\ \wedge \kappa(j)=c}} x_{i,j} \geq 1, \forall c \in C_s \quad (4.3)$$

$$\sum_{j:(j,i) \in E'} x_{j,i} = \sum_{j:(i,j) \in E'} x_{i,j}, \forall i \in V \quad (4.4)$$

$$y_j \geq x_{i,j}, \forall (i,j) \in E' \quad (4.5)$$

$$\sum_{i:(i,j) \in E'} x_{i,j} \geq y_j, \forall j \in V' \setminus \{0\} \quad (4.6)$$

$$\sum_{j:(j,i) \in E'} f_{j,i} = y_i + \sum_{j:(i,j) \in E'} f_{i,j}, \forall i \in V \quad (4.7)$$

$$x_{i,j} \leq f_{i,j} \leq (n+1) * x_{i,j}, \forall (i,j) \in E' \quad (4.8)$$

$$x_{i,j} \in \{0, 1\}, \forall (i,j) \in E' \quad (4.9)$$

$$y_i \in \{0, 1\}, \forall i \in V' \setminus \{0\} \quad (4.10)$$

$$f_{i,j} \in \{0, 1, \dots, n+1\}, \forall (i,j) \in E' \quad (4.11)$$

4.3 Using A* for ACSP/SCSP

A graph object is created to contain all the information including vertex colors and edge weights for the ACSP / SCSP problem to be used in the A* algorithm. The graph object stores all properties, such as vertices change with their colors, and edges with the associated weight values. In order to conserve space, this graph object is shared by all the nodes in the search space, without replication. In addition, each node contains the indexes of the vertices on path from the start to the current vertex, along with the path cost $g(n)$ and completion estimate $h(n)$. Note that in order to estimate the $h(n)$ function for a child node n generated by A*, we formulate an SCSP problem where the start node is n and C_s is obtained by eliminating already seen colors from C . To describe another way, the child node generated by A* is taken as the starting point on the graph for every time and it is aimed to obtain heuristic predictions by giving information about both the remaining colors so far and the colors seen so far.

Goal criteria consists of verifying that the remaining set of colors to be visited is empty. The children of a given node is created by A* determined according to edge connections of graph node object. In the evaluation of $h() = 0$, it proceeds to the goal by calculating the $g()$ function according to the cost spent in travelling to each child node to which the nodes are connected according to the graph structure. The goal criterion of A* algorithm depends on the minimum cost of traversing all the colored nodes that are required to be circulated as a color set in the ACSP or SCSP problem. In this case, when the child node of each parent node is selected, it will check whether all colors are selected with the least cost. The A* algorithm is expected to achieve all colors at minimum cost, as long as the heuristic function underestimates the cost of reaching the actual goal at each step in the graph baseline.

4.3.1 LP Approximation for the Heuristic Function

Instead of calculating the optimal result, LP finds the decision variables ($x_{i,j}$, $y_{i,j}$ and $f_{i,j}$) of the probable paths shown as ILP formulation in (Bilge et al., 2015), and therefore the LP solution introduced to the ACSP problem is used as the heuristic estimation method to reach the optimal path of the A* algorithm.

ILP gives the optimal result in solving the problem. However, ILP determines decision variables as integer values and takes exponential time, which quickly becomes cost prohibitive for large data sets. On the other hand, it is important for us that LP underestimates ILP in a short time and to a good extent. In this case, we underestimate ILP in polynomial time utilizing LP. In other words, we aim to estimate the $h()$ function of the A* algorithm in polynomial time in less time than the time that ILP will lead us to the optimal result. At that point, when performing an LP solution, integer value generates the optimal or near-optimal estimates in polynomial time by giving real values between 0 and 1 rather than integer values of 0 or 1.

We utilize CPLEX Studio Version 12.8.0 to solve LP and ILP problems (*CPLEX Optimizer*, 2019). This library requires us to instantiate an LP model with the problem constraints as specified in Section 4.2 which covering the ILP Formulation of SCSP. Since it would be prohibitive to create such an object from scratch for every node in our search

space, we initialize it once for the root node, and for each subsequent child, we add additional constraints to estimate the $h()$ value, and then retract those additional constraints back.

For LP, when the A* algorithm needs to calculate the LP solution for each control of the child's nodes through the successors, making sure that the path traversed from the start vertex to the current vertex is included as part of the LP solution. Thus, the value of the $x_{i,j}$ decision variable in the LP mathematical formulation of the problem is forced to be 1 for each edge already traversed. Since the forced $x_{i,j}$ decision values (to be $x_{i,j} = 1$) are systematically calculated inside of LP in advance as a real distance travelled, for A* algorithm, estimate that is produced by LP, always consists of $f()$ values. In other words, there is no need to separately calculate $g()$ and $h()$, since LP solution already provides the $f()$ value. The first created model for the LP solution is always stored permanently so that, by reaching the child nodes prediction values, it can be directed in different ways to achieve an optimal solution. The sample usage of A* with LP approximation for heuristic is depicted on Figure 4.1.

While computing the $f()$ for internal nodes in the search space, as an alternative to the method of assigning 1 to edge variables for already traversed edges, another possible approach is using the SCSP formulation directly, which eliminates the constraints for colors that have already been visited. In this manner, to compute the $h()$ function of each selected node by the A* algorithm, the colors seen so far is subtracted from the color set and the selected node is designated as the start vertex by utilizing the SCSP. For this case, the SCSP generates an estimate of the remaining colors from the initial vertex to produce solutions with minimal cost. While we have utilized SCSP for training our ANN as described in Section 4.3.2, we have not fully explored it in the context of A*LP yet; we believe that using SCSP might provide some computational performance improvements to A*LP.

4.3.2 ANN Approximation for the Heuristic Function

The general working principle of A* relates to $f() = g() + h()$ as mentioned in Chapter 2. Therefore, for providing significant progress on computation performance of A*, it requires a heuristic function that makes zippy and reliable predictions.

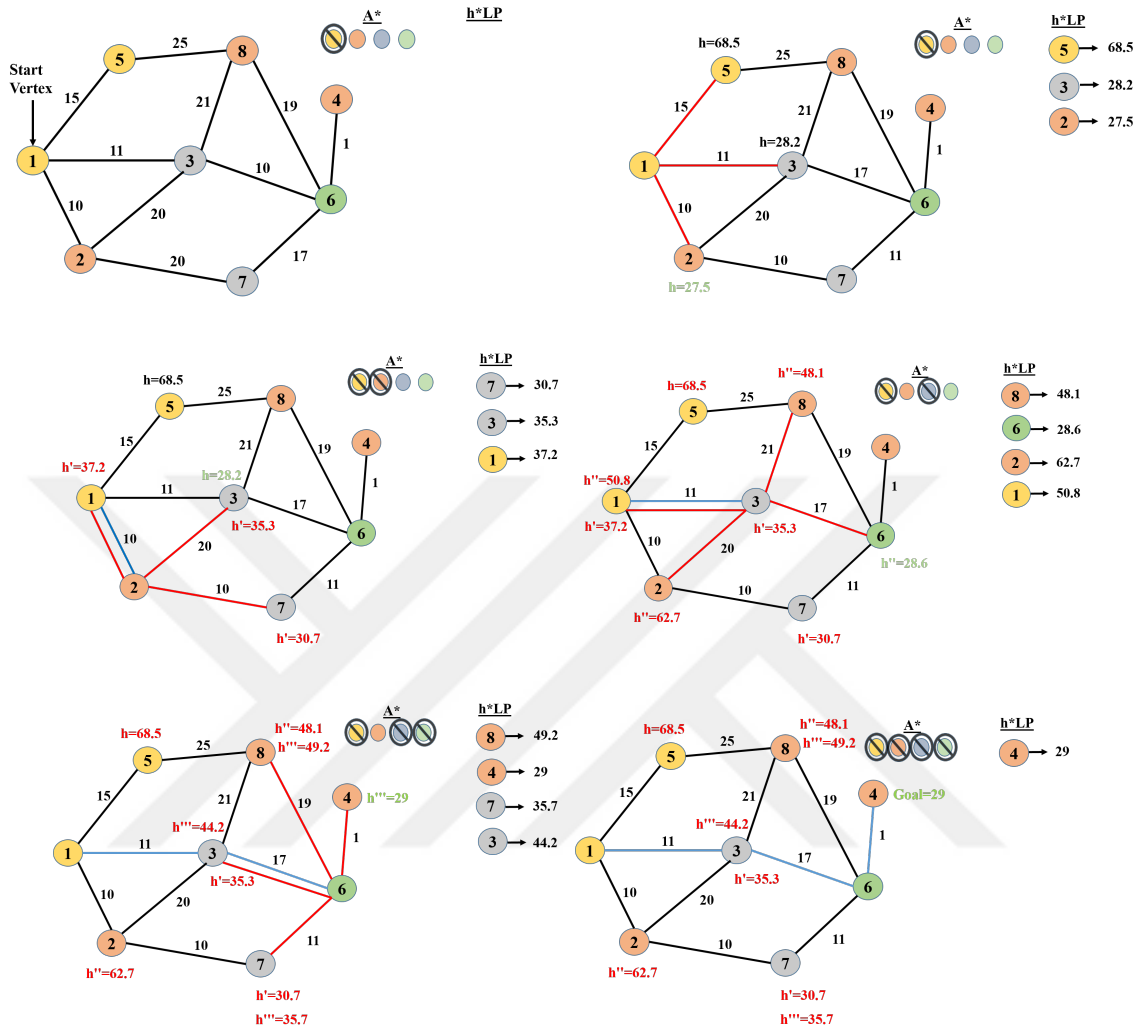


Figure 4.1. A*LP Search Demonstration

The purpose of the ANN in this problem is to estimate the $h()$ function; in other words, given the graph, edge weights, vertex colors, current vertex, and list of colors to be visited, it must predict the cost of the optimal solution traversing those colors, starting at the current vertex. Along with this manner, ANN layers in the system are regressed by back-propagating in order to suit the properties of the problem structure and as a heuristic tool in A*, it estimates solutions with shrinking the complex featured input size layer by layer to only one output. In this case, compared to LP, the good side of ANN would be that it can produce close and optimal predictions if it is well trained.

Step by step walking through the optimal solution of A* is decided with state of start node. When A* choose the child node as an optimal way of heuristic result, child node becomes start node and all of the node color values that it has travelled so far are shown as 0 in the color vector and the child in which it is found is evaluated as the starting node. In other words, the colors are examined so far will be 0, while the colors that remain to be visited will be indicated by the value 1. In this case, ANN's prediction will be in the direction of which colors we will search for from the starting node by figuring out the binary color values. To illustrate this, initially, all color vector values will show 1, while the first node will be the starting node, and we expect that the value it generates will be close to the answer we want to optimally reach at the end of our problem. In this way, ANN will generate heuristic estimates as a result of changing color values and initial node values each time A* progresses. In this manner, we call interoperability of A* algorithm and ANN as A*NN and we explain that A*NN possesses two sides:

- NN training side of the problem consists of SCSP solutions which includes random starting vertex with the value of $V = \{1, 2, \dots, n\}$, seen colors so far with the binary value of 1 and remaining colors not seen so far with the binary value of 0. In other words, according to SCSP, the subset of all colors set will be depicted as 1 and SCSP will answer the question of "From start vertex s , for obtaining colors subset C_s with the binary value 1, what is the optimal and shortest path cost?" for training.
- A* side of the problem shows opposite manner of the training. The child node selected by A* substitutes of start vertex and the colors are seen so far will be of 0 and while remaining ones will be 1. Therefore, A* algorithm's question should be as follows : "Let the child node be starting point, what is the estimated optimal path cost of finding remaining color or colors with the value 1?"

As in the data representation shown in Figure 4.1, each child node selection by A* modifies the color vector with respect to the diagonal color vertex representation. In other words, the selected child node color affects the color vector specified in the second part and switches the binary color representation of input data. This process works oppositely for both training side of ANN and prediction side of A*NN based on the colors seen and remaining.

4.4 Benchmark

Graph Dataset was generated by using C++ Boost Library (*Boost C Libraries*, 2019) for ease of display of graphs. In order to compare the values of the data and effectively train the Neural Network given the availability of computational resources at this stage of our research, all graphs contain 10 vertices. Once we demonstrate the efficacy of our approach, we intend to expand our work to larger problem sizes in the future. The vertices are randomly connected with each other utilizing 30 random weighted edges whose weights are between 10 and 40. In addition, color values are randomly distributed to all vertices to provide 5 different colors for different data representation. One hundred thousand different graphs that formed raw graph data were created using this approach. Moreover, for comparison purpose, one hundred test sets were produced that are totally independent of raw graph dataset.

In addition to the graph dataset, we generated SCSP problem sets that specify the start vertex, and list of colors to be visited, in order to train our ANN. So, processed raw data forms our manual feature map to ANN and consists of whole graph features and solutions along with partial ones of each.

By using ILP, the optimal shortest path to the SCSP problem of each graph can be computed. So as to generate SCSP problem dataset, we partition the problem which colors we should choose and by the way of this which vertices we need to examine through the color combinations we choose as $C_s \in C$. Before traversing the graph, we create a starting point for graph by means of a randomly selected start vertex. In this way, partial solutions are generated on graphs by modifying ACSP in terms of SCSP. In other words, for all $V = \{1, \dots, n\}$ and start vertex $s \in V$, we always choose random vertex s , by the way of this, the decision variable of SCSP will be $X_{0,s} = 1$. For all $C = \{1, \dots, k\}$, selected and to be assumed visited colors $C_s \in C$, we randomly choose subset color as $r = \{1, \dots, k\}$ and $C_s = \text{Combinations}(C, r)$.

The data files for SCSP problem set and raw dataset can be accessed via google drive link (https://drive.google.com/drive/folders/1ePDtGMafMrNcKvU1EnRk0fS9qtnJ_ue?usp=sharing). Accordingly, one sample graph with the encoding on this thesis is

illustrated in Figure 4.2 and Figure 4.3. Figure 4.2(a) shows the raw graph data produced by C++ Boost Library, Figure 4.2(b) illustrates the color vector of input data along with the starting vertex and optimal value of graph for training ANN. Notice that in Figure 4.2(b), selected colors are demonstrated as a 1 whereas remainings are 0. Figure 4.2(c) depicts the adjacency matrix of the graph and the matrix diagonally consists of the color values of vertices. Moreover, note that Figure 4.2(b) and Figure 4.2(c) together form processed graph data for generating SCSP/ACSP problem data so as to train ANN.

In Figure 4.3, the vertices contain two values: the first one is the vertex number and the other one is the color value. Between the connected vertices, each edge carries a cost (weight) value. In accordance with Figure 4.2 (b), the optimal SCSP solution starting at vertex 1 and visiting four colors (1, 2, 3, and 4) from the entire color wset is presented with red vertices and a black colored path. Finally, we can see that the optimal solution path cost shown by Figure 3 is equal to the optimal value in Figure 4.2.(b).

4.5 Training an ANN as a heuristic function

We generated the input data for the three parts of the graph's raw features by combining them. The first part is the adjacency matrix; the second part relates to the color values and the last one consists of the numerical value of the start vertex.

The primary way to prepare the training dataset is to create a 10x10 adjacency matrix from the raw data. The diagonal of the matrix takes a color value specifically between 1 and 5 for this problem. The additional five terms represent a binary value (0 or 1) which indicates that the vertex has previously been selected (or not intended for) with respect to the color values. This information is crucial to determine choosing optimal path or not optimal one. Therefore, the network will figure out which vertex is selected via lights on or off with respect to binary color value. Lastly, it will also be added to the training set by procreating start vertex showing the beginning of traverse that is chosen in decimal.

The target values calculated by ILP will be given to the input layer together with the adjacency weight matrix and color matrix before ANN is trained, as it forms the basis of regression problems.

The input layer of the network includes the raw state of the graph information, the color

Graph G->Color C= {1 [c=1]; 2 [c=3]; 3 [c=4]; 4 [c=4]; 5 [c=2]; 6 [c=4]; 7 [c=1]; 8 [c=3]; 9 [c=5]; 10 [c=5]}
Edge E->Weight W= {1--4 [w=29]; 2--4 [w=20]; 3--7 [w=30]; 4--6 [w=33]; 5--4 [w=21]; 6--7 [w=40]; 7--8 [w=10]; 8--5 [w=21]; 7--2 [w=17]; 6--2 [w=23]; 9--6 [w=10]; 9--10 [w=16]; 10--3[w=15]; 10--7 [w=29]}

(a) Raw graph data information

Binary Color Values					Start Node	Optimal Value
C1	C2	C3	C4	C5		
1	1	1	1	0	1	71

(b) Binary color vector

Nodes	1	2	3	4	5	6	7	8	9	10
1	<u>1</u>	0	0	29	0	0	0	0	0	0
2	0	<u>3</u>	0	20	0	23	17	0	0	0
3	0	0	<u>4</u>	0	0	0	30	0	0	15
4	29	20	0	<u>4</u>	21	33	0	0	0	0
5	0	0	0	21	<u>2</u>	0	0	21	0	0
6	0	23	0	33	0	<u>4</u>	40	0	10	0
7	0	17	30	0	0	40	<u>1</u>	10	0	29
8	0	0	0	0	21	0	10	<u>3</u>	0	0
9	0	0	0	0	0	10	0	0	<u>5</u>	16
10	0	0	15	0	0	0	29	0	16	<u>5</u>

(c) Processed data adjacency matrix of raw graph data

Figure 4.2. SCSP Graph Data Representation

values indicating the state of selection of the vertices, and the start vertex with decimal value, respectively. Thus, the conditions in which A* chooses each node and picks their children are determined in the training dataset in order to illustrate the preferred pathways.

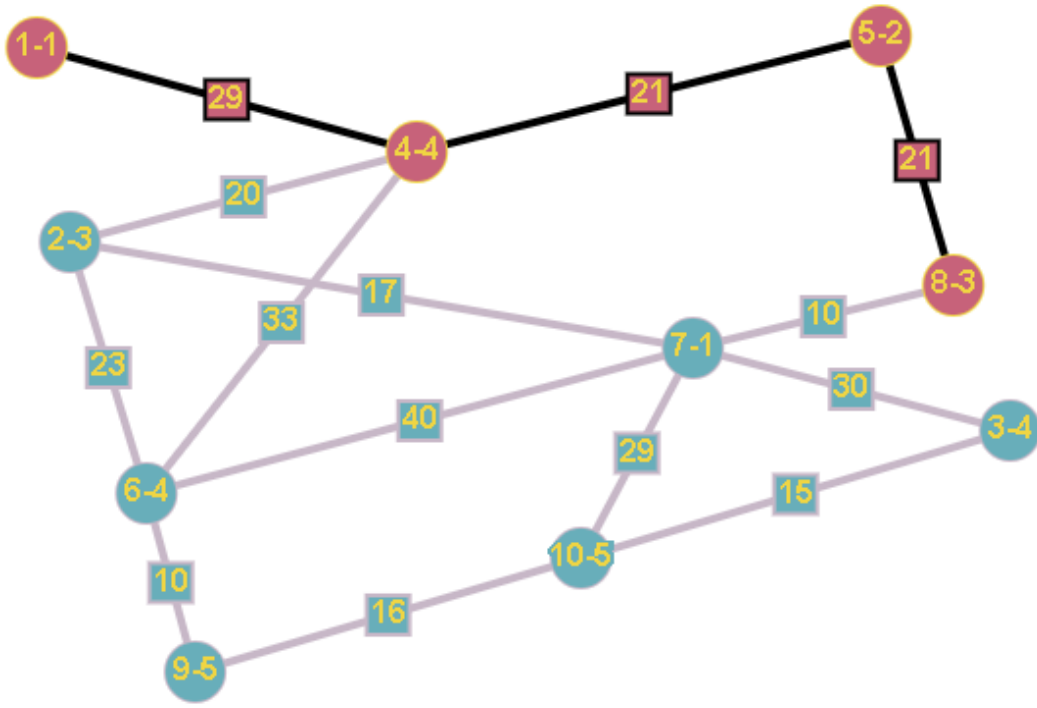


Figure 4.3. Representation and solution of SCSP graph for 4 colors optimal path finding within a 5 colored vertex.

According to this, we produce a predicting label as the cost of the subpaths likely to be favored by the A* algorithm. Raw graphs previously created are computed in accordance with the changes in the ILP formulation to the values given to the $x_{i,j}$ decision variables and the color variables (depicted in Section 4.2), and to the node selection status of each of the subpaths forming the non-simple path. In addition, two different non-optimal solutions are created from raw data to show the preferences of inconsistent but conclusive non-optimal paths in the training dataset and to stabilize the network. Non-optimal solutions are found by solving instances of SCSP where the base and color subset are randomly selected.

Lastly, because of the complexity of problem representation and for avoiding the overfitting configs, it is necessary to utilize an augmented size of data. By means of the raw graph data processing method using SCSP formulation, we have created approximately one million data samples for the purpose of training ANN. The data file for SCSP problem set can be accessed via google drive link (<https://drive.google.com/drive/folders/1eIg63wvEx0YSAEQChnGH-o4EiL1Ivj7R?usp=sharing>).

In this thesis, Multilayer Perceptron (MTL) is used as an ANN to learn and predict the graph structured data. For training, Python Tensorflow library (Abadi et al., 2015) simplifies an efficient area with the structure of its interface which has an user-friendliness, however usage with A* brings about performance issues to solve SCSP problem. Therefore, we utilized Python TensorFlow library (Abadi et al., 2015) for training and by extracting calculated weights and biases' values, wrote own interoperable code with A* in C++. In addition to this, all experiments including training of ANN were executed on machine with the properties of 2.50Ghz Intel(R) Core i7 CPU, 16Gb RAM and 8Gb NVIDIA GeForce GPU.

Being successful in ANN training comes in the form of trial testing. Among the theoretical knowledge of ANN, the excess number of data is crucial to achieve efficiency in the use of complex data to avoid overfitting situation. In addition, it has been showed in the studies that increasing both the depth and the neuron size of the network gives great results about obtaining an efficient information from ANN (Bengio et al., 2007; Erhan et al., 2009; Goodfellow et al., 2013).

In this study, when we evaluate according to mean absolute error and mean square error, we found the best network structure as 106 x 340 x 170 x 85 x 44 x 22 x 11 x 5 x 1 (106 with input layer, different sizes of fully connected 7 hidden layer and 1 output layer for regression). Weight initialization for each dense layer is evaluated as a normal distribution. In addition, batch normalization, as conducted by (Ioffe and Szegedy, 2015), is used shortly after the activation function for all hidden layers in the study to saturate nonlinearity for each layer of the neural network. By the way of this, we reached efficient performance on training error, while the other regularization methods such as L1/L2 Normalization and Dropout did not meet expectations. Accordingly, the obtained ANN structure is shown in Table 4.1.

Table 4.1. ANN Architecture for SCSP Problem Set

Type	Output Size	Param #
Input Layer	340	36380
Batch Norm	340	1360
Hidden Layer	170	57970
Batch Norm	170	680
Hidden Layer	85	14535
Batch Norm	85	340
Hidden Layer	44	3784
Batch Norm	44	176
Hidden Layer	22	990
Batch Norm	22	88
Hidden Layer	11	253
Batch Norm	11	44
Hidden Layer	5	60
Batch Norm	5	20
Output Layer	1	6

CHAPTER 5 : EXPERIMENTAL RESULTS

5.1 Training Results of ANN

The ANN model, which was developed according to the characteristics of the SCSP problem, was trained using 750 epoch and a batch size as 70. Mean Absolute Error (MAE) was used in the comparison of the training set and the validation set. 80% of the dataset was used for training and the remaining 20% for validation purposes. Figure 5.1 shows the training graph. In the graph, we see that the MAE decreases to 7 in the direction of the training set while the validation set observed is at 7.2. The current values have been effective in our study. However, in future studies, it is considered that better results can be obtained by increasing the number of the training sets and the epochs on machines with more computing power and capacity.

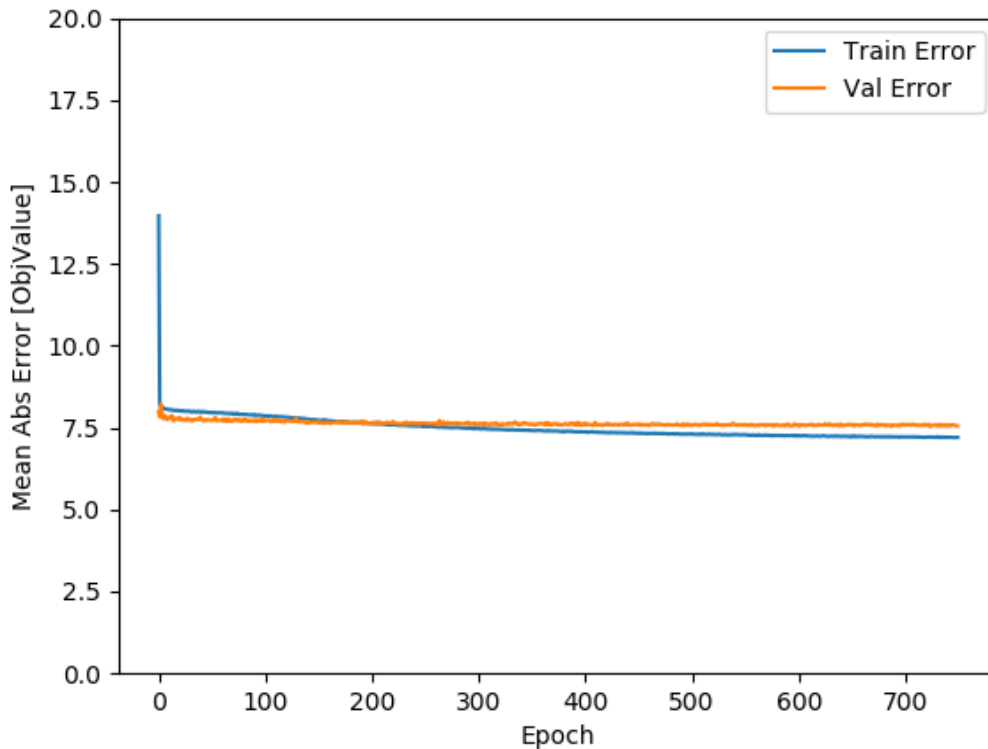


Figure 5.1. Training Error Graph of ANN

Table 5.1. Solution Quality Comparison results

	Max	Min	Mean	Std
ILP	103.00	43.00	70.37	11.17
LP	87.05	41.36	60.78	10.63
LP/ILP	1.00	0.67	0.86	0.07
ANN	102.19	52.62	68.43	8.21
ANN/ILP	1.53	0.56	0.99	0.16

In this section, we also compared ANN and LP approximation qualities in order to measure the closeness to the optimal solution of the problem. As previously mentioned, the ACSP problem can be optimally solved using the ILP formulation. Thus, using a problem set of one hundred samples, we measure the ratio of the LP estimate to the optimal solution cost (as computed by ILP), as well as the ratio of the ANN estimate to the optimal solution cost. These results are summarized in Figure 5.2 and Table 5.1. When we consider these results, it is seen that LP always underestimates ILP as expected; while ANN overestimates on occasion. However, the overestimates are limited in magnitude and frequency: 75% of the samples are less than 1.07 times the cost of the optimal solution. While it is possible to make adjustments to ANN so that it overestimates less often, this might adversely impact the savings in the search space size for the A* algorithm. As we will see in the next section, using ANN as a heuristic yields very close to optimal results, so such an adjustment is not warranted in our view. Except for the maximum and minimum values in Figure 5.2, the values for ANN are very akin to the optimum. Even if the results do not always underestimate ILP, it will give us hints that if it works with A*, it will always give very close results to the optimal, to solve notwithstanding that not always with optimal results. Table 5.1 contains the values showing the closeness of the results obtained by LP and ANN to the optimal results with ILP for 100 graph test data. If Figure 5.2 and Table 5.1 are interpreted together; on average, it can be said that ANN's estimates are almost always close to optimal values compared to LP. According to Table 5.1, ANN's proximity to ILP is 0.99, whereas LP remains 0.86.

Maximum results of Table 5.1 demonstrates that LP never overestimates the ILP solutions.

In the same way, we observe the computation times of LP and ANN by looking at

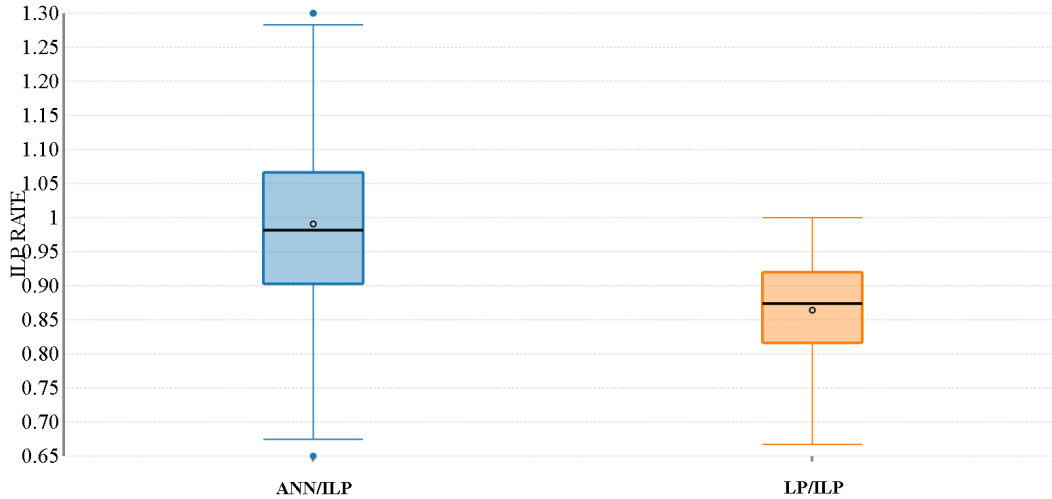


Figure 5.2. Solution Quality for ANN versus LP Estimate Ratios

Table 5.2. ANN versus LP CPU Time Comparison

	Means	Max	Min	Median	Std
LP CPU TIME (ms)	29.3422	40.0000	13.5940	28.4380	5.6831
ANN CPU TIME (ms)	1.0731	1.4220	0.7500	1.0780	0.1317

the respective CPU utilizations. Accordingly, Table 5.2 shows us the average CPU time statistics that ANN and LP spend for a single forecast. It is seen that ANN runs 30 times faster than LP.

5.2 Comparison of LP versus ANN as effective A* Heuristics

In this section, we consider the number of nodes expanded by the A* algorithm, as a metric to evaluate the search space size. We take the uninformed heuristic $h()=0$ as a baseline, and measure the ratio of search space for $h() = 0$ to that of $h() = LP$ and $h() = ANN$, respectively. For comparison, we use one hundred problem test set to analyse the study. Accordingly, A*LP and A*NN search space reduction analysis with respect to $h() = 0$ of A* is depicted in Figure 5.3.

Figure 5.3 shows the comparison of the two methods used in A*. According to the results of 100 graph data, it is understood that A*NN reduces search space better than A*LP.

Table 5.3. A* Search Space Comparison

	Max	Min	Mean	Std
A*NN	66.46	2.22	5.87	6.63
A*LP	10.81	1.02	2.24	1.88

Table 5.4. A*NN Optimal Cost Comparison

	Max	Min	Mean	Median	Std
A*NN to Optimal Ratio	1.1451	1.0000	1.0016	1.0000	0.0146

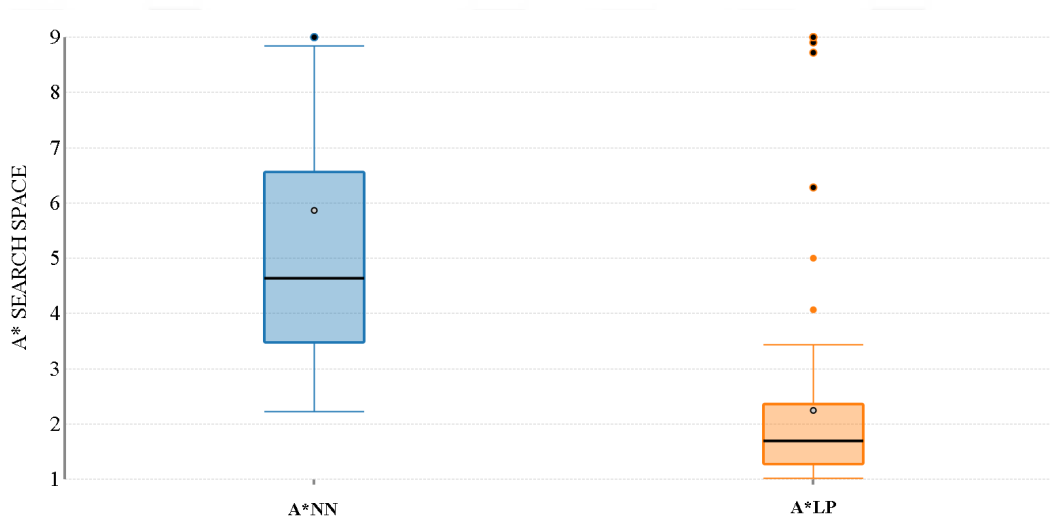


Figure 5.3. A* Search Space Comparison

Moreover, if Figure 5.3 and Table 5.3 are considered together, A*NN can reduce search space of A* algorithm by the effect of 2 to 66 and on the average of 5.85, while A*LP can force the pace about 11 times in the best case. Accordingly, by looking at the comparison graph in Figure 5.3, we can say that A*NN intensely decreases the search space of A* at an average rate between 5 and 6 times faster.

Furthermore, Table 5.4 involves the relationship between A* algorithm and ILP. In this table, we compare the solution values of $h() = 0$, $h() = LP$ and $h() = NN$, the heuristic function of the A* algorithm, with the optimal solutions of the ILP. Accordingly, since $h() = 0$ and $h = LP$ constantly underestimate solutions which are always optimal, we do not insert the results into the Table 5.4. However, although A*NN in some cases overestimates

solutions resulting from near optimal values, it can be said to be almost optimal with an average value of 1.0016. The detailed results about A*NN Estimates and A* Search Space Comparisons are in the appendix of this thesis.



CHAPTER 6 : CONCLUSION

This thesis combines ANN with the traditional artificial intelligence search algorithm A* utilizing the capacity of machine learning in the context of the ACSP problem and our minor modification to it, called SCSP. We picked ACSP, because it is more complex than the TSP and it is an NP-Complete graph theory problem in the direction to new interest in the literature (Bilge et al., 2015; Akcan and Evrendilek, 2017; Akçay et al., 2018). Customized to the ACSP problem, we produced efficiency results obtained by combining the A* algorithm with LP approximation and ANN as heuristic functions.

The two methods obtained in this study -A*NN and A*LP- require partial solutions. At this point, ACSP has been modified to SCSP so as to produce solutions for subsets of the overall color set.

We first evaluated LP in order to figure out its approximation to the ILP in our problem space, and demonstrated LP as a reasonable heuristic for reduction of the A* search space in our problem space. By the way of comparison of $h() = 0$ in the A* algorithm, we measured effects of LP approximation to the A* algorithm's search space. Secondly we trained ANN to be used as a heuristic function as an alternative to LP. Our results show that ANN reduces the search space by six fold whereas LP reduces it by half. Furthermore, computation time for ANN is 1.0731 milliseconds, whereas LP computation time is 29.3422 millisecond of cpu time. While A*NN is not guaranteed to produce optimal results as it might overestimate the solution cost on occasion, our empirical results show that it produces optimal or almost optimal solutions most of the time.

One limitation of our study is that our experiments are confined to small size graph with ten vertices and 30 edges due to computational resource availability. However, we believe A*NN will perform more favourably compare to ILP as the problem size increases. We will undertake this as a topic of future research. Other future directions include attacking NP-complete problems that are different from ACSP. We envision that studies aimed at optimal solution utilizing A*NN may shed light on problems such as TSP, Clique Decision Problem etcetera. Therefore, we consider that using ANN as a heuristic tool for such different NP-complete problems can produce interesting outcomes.

As another way; we can address larger graph problems using the partial solutions we have achieved. The most straightforward way to do this is to apply the Divide and Conquer principles.

The last method would be to train ANNs so as to predict the optimal path of graph theory problems directly without utilizing A*. This is particularly challenging because without A*, the solution will not be guaranteed to be a continuous path that visits all required colors.



REFERENCES

- Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y. and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software Available at tensorflow.org (Accessed 9 January 2020).
- Aiyer, S. V., Niranjan, M. and Fallside, F. (1990). *A theoretical investigation into the performance of the Hopfield model*, IEEE transactions on neural networks, Vol. 1, pp. 204–215.
- Akcan, H. and Evrendilek, C. (2017). *Complexity of energy efficient localization with the aid of a mobile beacon*, IEEE Communications Letters, Vol. 22, pp. 392–395.
- Akçay, M. B., Akcan, H. and Evrendilek, C. (2018). *All colors shortest path problem on trees*, Journal of Heuristics, Vol. 24, pp. 617–644.
- Applegate, D. L., Bixby, R. E., Chvatal, V. and Cook, W. J. (2006). *The traveling salesman problem: a computational study*, Princeton University press, New Jersey.
- Bengio, Y., Lamblin, P., Popovici, D. and Larochelle, H. (2007). *Greedy layer-wise training of deep networks*, Advances in neural information processing systems, pp. 153–160.
- Bilge, Y. C., Çağatay, D., Genç, B., Sarı, M., Akcan, H. and Evrendilek, C. (2015). *All colors shortest path problem*, arXiv preprint arXiv:1507.06865.
- Boost C Libraries* (2019). Boost C Libraries Available at www.boost.org (Accessed 9 January 2020).
- CPLEX Optimizer* (2019). Software Available at www.ibm.com (Accessed 9 January 2020).
- Durbin, R. and Willshaw, D. (1987). *An analogue approach to the travelling salesman problem using an elastic net method*, Nature, Vol. 326, pp. 689–691.

- Erhan, D., Manzagol, P.-A., Bengio, Y., Bengio, S. and Vincent, P. (2009). *The difficulty of training deep architectures and the effect of unsupervised pre-training*, Artificial Intelligence and Statistics, pp. 153–160.
- Garey, M. R. and Johnson, D. S. (1979). *Computers and intractability*, Vol. 174, Freeman, San Francisco.
- Gee, A. H. (1993). *Problem solving with optimization networks*, PhD thesis, University of Cambridge.
- Glorot, X. and Bengio, Y. (2010). *Understanding the difficulty of training deep feedforward neural networks*, Proceedings of the thirteenth international conference on artificial intelligence and statistics, pp. 249–256.
- Glorot, X., Bordes, A. and Bengio, Y. (2011). *Deep sparse rectifier neural networks*, Proceedings of the fourteenth international conference on artificial intelligence and statistics, pp. 315–323.
- Goodfellow, I., Bengio, Y. and Courville, A. (2016). *Deep Learning*, MIT Press. Available at <http://www.deeplearningbook.org> (Accessed 12 September 2019).
- Goodfellow, I. J., Bulatov, Y., Ibarz, J., Arnoud, S. and Shet, V. (2013). *Multi-digit number recognition from street view imagery using deep convolutional neural networks*, arXiv preprint arXiv:1312.6082.
- Hart, P. E., Nilsson, N. J. and Raphael, B. (1968). *A formal basis for the heuristic determination of minimum cost paths*, IEEE transactions on Systems Science and Cybernetics, Vol. 4-, pp. 100–107.
- Hopfield, J. J. and Tank, D. W. (1985). *“Neural” computation of decisions in optimization problems*, Biological cybernetics, Vol. 52, pp. 141–152.
- Huang, T., Ma, Y., Zhou, Y., Huang, H., Chen, D., Gong, Z. and Liu, Y. (2019). *A Review of combinatorial optimization with graph neural networks*, 2019 5th International Conference on Big Data and Information Analytics (BigDIA), pp. 72–77.
- Ioffe, S. and Szegedy, C. (2015). *Batch normalization: Accelerating deep network training by reducing internal covariate shift*, arXiv preprint arXiv:1502.03167.

- Jain, A. K., Mao, J. and Mohiuddin, K. M. (1996). *Artificial neural networks: A tutorial*, Computer, Vol. 29, pp. 31–44.
- Jarrett, K., Kavukcuoglu, K., Ranzato, M. and LeCun, Y. (2009). *What is the best multi-stage architecture for object recognition?*, 2009 IEEE 12th international conference on computer vision, pp. 2146–2153.
- Kearns, M. J., Vazirani, U. V. and Vazirani, U. (1994). *An introduction to Computational Learning Theory*, MIT press, Cambridge.
- Khachiyan, L. G. (1979). *A polynomial algorithm in linear programming*, Doklady Akademii Nauk, Vol. 244, pp. 1093–1096.
- Kohonen, T. (1982). *Self-organized formation of topologically correct feature maps*, Biological cybernetics, Vol. 43, pp. 59–69.
- LeCun, Y., Bengio, Y. and Hinton, G. (2015). *Deep learning*, Nature, Vol. 521, pp. 436–444.
- LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. and Jackel, L. D. (1989). *Backpropagation applied to handwritten zip code recognition*, Neural computation, Vol. 1, pp. 541–551.
- Li, Z., Chen, Q. and Koltun, V. (2018). *Combinatorial optimization with graph convolutional networks and guided tree search*, Advances in Neural Information Processing Systems, pp. 539–548.
- Mohammad Reza Bonyadi, H. S.-H. and Azghadi, M. R. (2008). *Population-Based Optimization Algorithms for Solving the Travelling Salesman Problem*.
- Nair, V. and Hinton, G. E. (2010). *Rectified linear units improve restricted boltzmann machines*, Proceedings of the 27th international conference on machine learning (ICML-10), pp. 807–814.
- Prates, M., Avelar, P. H., Lemos, H., Lamb, L. C. and Vardi, M. Y. (2019). *Learning to solve NP-complete problems: A graph neural network for decision TSP*, Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 33, pp. 4731–4738.

- Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M. and Monfardini, G. (2008). *The graph neural network model*, IEEE Transactions on Neural Networks, Vol. 20, pp. 61–80.
- Selsam, D., Lamm, M., Bünz, B., Liang, P., de Moura, L. and Dill, D. L. (2018). *Learning a SAT solver from single-bit supervision*, arXiv preprint arXiv:1802.03685.
- Smith, K. A. (1999). *Neural networks for combinatorial optimization: a review of more than a decade of research*, INFORMS Journal on Computing, Vol. 11, pp. 15–34.
- Smith, K. and Potvin, J.-Y. (2001). *Artificial Neural Networks for Combinatorial Optimization*.
- Unaltuna, M. K. and Pitchumani, V. (1994). *A stochastic reward and punishment neural network algorithm for circuit bipartitioning*, Proceedings of IEEE International Symposium on Circuits and Systems-ISCAS'94, Vol. 1, pp. 181–184.
- Wilson, G. and Pawley, G. (1988). *On the stability of the travelling salesman problem algorithm of Hopfield and Tank*, Biological Cybernetics, Vol. 58, pp. 63–70.
- Yao, Wang, B. L. and Mitra (1988). *Alternative networks for solving the traveling salesman problem and the list-matching problem*, IEEE 1988 International Conference on Neural Networks, pp. 333–340.
- Zurada, J. M. (1992). *Introduction to artificial neural systems*, Vol. 8, West Publishing Co, West St. Paul.

APPENDICES

*Appendix A-*NN Estimate Results*

Table A.1. A*NN Estimate Results with Respect to Optimal Solution Cost of ACSP Problem

Sample No.	A*NN	Optimal Solution (ILP-LP-UCS)	A*NN/ILP
1	60	60	1.0000
2	75	75	1.0000
3	56	56	1.0000
4	63	63	1.0000
5	68	68	1.0000
6	72	72	1.0000
7	59	59	1.0000
8	79	79	1.0000
9	60	60	1.0000
10	53	53	1.0000
11	78	78	1.0000
12	75	75	1.0000
13	78	78	1.0000
14	51	51	1.0000
15	73	73	1.0000
16	84	84	1.0000
17	68	68	1.0000
18	58	58	1.0000
19	68	68	1.0000
20	66	66	1.0000
21	76	76	1.0000
22	69	69	1.0000
23	103	103	1.0000
24	68	68	1.0000

Table A.1 continued from previous page

Sample No.	A*NN	Optimal Solution (ILP-LP-UCS)	A*NN/ILP
25	72	72	1.0000
26	80	80	1.0000
27	65	65	1.0000
28	75	75	1.0000
29	73	73	1.0000
30	85	85	1.0000
31	61	61	1.0000
32	72	72	1.0000
33	74	74	1.0000
34	61	61	1.0000
35	78	78	1.0000
36	81	81	1.0000
37	44	44	1.0000
38	86	86	1.0000
39	50	50	1.0000
40	84	84	1.0000
41	73	73	1.0000
42	94	94	1.0000
43	63	63	1.0000
44	61	61	1.0000
45	62	62	1.0000
46	59	59	1.0000
47	73	73	1.0000
48	59	59	1.0000
49	75	75	1.0000
50	65	65	1.0000
51	62	62	1.0000
52	53	53	1.0000

Table A.1 continued from previous page

Sample No.	A*NN	Optimal Solution (ILP-LP-UCS)	A*NN/ILP
53	77	77	1.0000
54	71	71	1.0000
55	72	72	1.0000
56	64	64	1.0000
57	56	56	1.0000
58	77	77	1.0000
59	68	68	1.0000
60	66	66	1.0000
61	72	72	1.0000
62	66	66	1.0000
63	77	77	1.0000
64	72	72	1.0000
65	73	73	1.0000
66	78	78	1.0000
67	43	43	1.0000
68	58	58	1.0000
69	82	81	1.0123
70	63	63	1.0000
71	86	86	1.0000
72	55	55	1.0000
73	89	89	1.0000
74	58	58	1.0000
75	85	85	1.0000
76	64	64	1.0000
77	71	62	1.1452
78	87	87	1.0000
79	81	81	1.0000
80	65	65	1.0000

Table A.1 continued from previous page

Sample No.	A*NN	Optimal Solution (ILP-LP-UCS)	A*NN/ILP
81	66	66	1.0000
82	74	74	1.0000
83	68	68	1.0000
84	85	85	1.0000
85	90	90	1.0000
86	64	64	1.0000
87	79	79	1.0000
88	65	65	1.0000
89	73	73	1.0000
90	86	86	1.0000
91	73	73	1.0000
92	58	58	1.0000
93	67	67	1.0000
94	73	73	1.0000
95	63	63	1.0000
96	89	89	1.0000
97	69	69	1.0000
98	64	64	1.0000
99	96	96	1.0000
100	69	69	1.0000

Appendix B-A* Search Space Results

Table B.1. A* Algorithm Search Space Comparison Results

Sample No.	UCS (h() $=$ 0)	A*LP	A*NN
1	724	324	246
2	3087	2962	938
3	577	63	74
4	796	476	58
5	616	395	210
6	2067	656	280
7	739	289	210
8	1050	368	287
9	803	649	293
10	449	242	53
11	3390	1591	1135
12	593	68	143
13	8277	4876	2812
14	1059	905	299
15	1805	608	546
16	4006	3351	1180
17	1038	495	422
18	868	717	113
19	710	315	249
20	1312	489	204
21	3481	2519	1504
22	767	278	301
23	2601	1419	390
24	491	270	130
25	1325	704	407
26	2247	2211	612
27	609	225	129
28	1104	847	323

Table B.1 continued from previous page

Sample No.	UCS (h(=0)	A*LP	A*NN
29	432	367	132
30	3909	2792	812
31	997	100	261
32	329	138	148
33	1654	1318	452
34	866	477	285
35	3059	2811	799
36	2221	1466	508
37	281	26	33
38	6649	3955	2233
39	245	49	40
40	2833	1111	381
41	1426	1222	344
42	9359	5325	1353
43	969	848	121
44	981	769	128
45	1087	558	234
46	720	429	185
47	858	454	226
48	1463	1250	200
49	2902	2541	627
50	724	370	272
51	891	688	237
52	409	121	80
53	1861	209	28
54	3062	2693	498
55	1584	938	321
56	1069	436	178
57	954	625	222
58	1658	955	128

Table B.1 continued from previous page

Sample No.	UCS (h(=0)	A*LP	A*NN
59	936	398	248
60	745	561	191
61	1881	1311	538
62	2000	1716	631
63	919	226	366
64	572	481	85
65	1963	1085	228
66	1709	1190	226
67	510	235	28
68	471	75	74
69	2281	2199	546
70	557	317	63
71	4700	3593	850
72	400	145	75
73	1510	1315	575
74	987	594	182
75	2354	1504	590
76	444	296	131
77	852	439	75
78	4069	2279	646
79	2656	1762	629
80	817	238	197
81	869	672	143
82	3125	2887	544
83	2221	1260	429
84	4031	3337	1206
85	2625	2118	362
86	499	299	75
87	1906	1811	347
88	436	342	157

Table B.1 continued from previous page

Sample No.	UCS (h)=0	A*LP	A*NN
89	1704	738	237
90	6044	3865	1144
91	3108	1098	537
92	1576	1098	254
93	429	349	75
94	842	448	129
95	1561	1437	371
96	2986	1080	408
97	1555	1205	317
98	317	178	44
99	7995	3132	1815
100	466	184	85