QUADRATIC ASSIGNMENT PROBLEM

A THESIS

submitted in partial fulfillment of the requirements

for the award of the dual degree of

Bachelor of Science-Master of Science

in

MATHEMATICS

by

GOKUL H

(16072)

DEPARTMENT OF MATHEMATICS
INDIAN INSTITUTE OF SCIENCE EDUCATION AND RESEARCH BHOPAL
BHOPAL - 462066

May 2021

भारतीय विज्ञान शिक्षा एवं अनुसंधान संस्थान भोपाल

Indian Institute of Science Education and Research Bhopal

(Estb. By MHRD, Govt. of India)

CERTIFICATE

This is to certify that Gokul H, BS-MS (Dual Degree) student in Department of Mathematics, has completed bonafide work on the thesis entitled **'QUADRATIC ASSIGNMENT PROBLEM'** under my supervision and guidance.

Dr. Kashyap Rajeevsarathy **Dr. Kushal Shah**

May 2021
IISER Bhopal

Committee Member	Signature	Date
Dr. Kashyap Rajeevsarathy		
Dr. Kushal Shah		
Dr. Ambuj Pandey		
Dr. Prahlad Vaidyanathan		

ACADEMIC INTEGRITY AND COPYRIGHT DISCLAIMER

May 2021 **Gokul H**
IISER Bhopal

ACKNOWLEDGEMENT

I heartily thank my supervisors Dr. Kashyap Rajeevsarathy and Dr. Kushal Shah for entrusting me this project. Moreover, I express my immense gratitude to them for the support and complete freedom that I have received during the entire project phase.

I thank my TEC members Dr. Prahlad Vaidyanathan and Dr. Ambuj Pandey for attending my presentations and for providing me valuable suggestions. I thank a few wonderful juniors at this juncture, namely Pritvi, Aman, Arman, Sahil, and Dipanshu for some productive discussions regarding my GNN model. Those discussions helped me in the smooth completion of this project work. I also thank all the wonderful faculties of the mathematics department for being supportive always.

I thank my best friend Neena for supporting me through thick and thin. Without her, my IISER life would have been incomplete. I am grateful to my friends Gokul, Asha, Aswin, Maneesha, Abhijit, Anjitha, Divyasree and Arathi for being with me when I needed them. A special mention to my dearest and most caring friend Srabana, without whom my final year of IISER life would have been nothing but a void. This acknowledgement will be incomplete if I do not mention my gang *Math πrates*, this friendship taught me that togetherness (as a team) could build castles.

I also thank all the people whom I have passed by during my five years of IISER life. Some made me happy during my tough days, and some others taught me some valuable life lessons. I am grateful for both.

Finally, I thank my parents for being with me in my hardest times and never letting me down.

Gokul H

ABSTRACT

Graph neural networks (GNNs) are deep-learning based methods that operate on the framework of graphs. In particular, tasks such as molecular fingerprinting, protein interfacing, and disease classification are modeled to learn from graph inputs. The central goal of this project is to use GNNs to build solutions to the decision version of the famous Traveling Salesperson Problem (TSP). Given a weighted graph $G = (V, E)$, and a number $C \in \mathbb{R}$, the decision TSP problem asks whether the graph admits a hamiltonian cycle with cost no larger than C. We begin this project by studying a more general class of combinatorial optimization problems known as the Quadratic Assignment Problems (QAPs). Then, we explore two different variants of TSP, namely the optimization variant and the decision variant. We also study the essential concepts of machine learning and see the different types of neural networks like Artificial Neural Networks (ANNs) and Recurrent Neural Networks (RNNs) which are relevant to this project. Following this, we establish a few important results from polyhedral theory, which will help us understand the Quadratic Assignment Polytope and see its properties. We then establish a key result which reveals that the Symmetric Traveling Salesperson Polytope is a projection of the Quadratic Assignment Polytope. Also, we use some concepts from graph theory to compute the dimension of the symmetric traveling salesperson polytope. Finally, we study the framework required to model the decision variant of the TSP as a GNN model and perform computational experiments on the same.

LIST OF FIGURES

CONTENTS

1. INTRODUCTION

1.1 Introduction

The Quadratic Assignment Problem (QAP) is a classic combinatorial optimization problem introduced in the year 1957 by Koopmans and Beckmann [17] to model a facility location problem. The problem has been continuously investigated till date by mathematicians and computer scientists around the globe. Many real-life problems can be mathematically modelled as a QAP. Exciting applications of QAP have been found in various domains such as placement problems, scheduling, VLSI design, statistical data analysis, parallel, distributed computing, etc. It is also interesting to note that many well-known optimization problems from combinatorics and graph-theory can be modelled as a QAP. Some examples are the traveling salesperson problem, the maximum clique problem, the graph partitioning problem, and the minimum feedback arc set problem. Also, from a computational point of view, the QAP is an extremely difficult problem, indeed, it is proven to be an NP-Hard problem.

In this chapter, we will formally introduce the problem statement of quadratic assignment problem and we will see a real-life application of the same, namely the facility location problem introduced by Koopmans and Beckmann. The references used for this section are [5, 7]. Following this, we explore well known Traveling Salesperson Problem (TSP) and the fact that the TSP can be modelled as a quadratic assignment problem. This section is based on [6, 13]. In the second half of this chapter, we will introduce the basics of machine learning and discuss in brief the different types of neural network topologies. This section is based on the results from [4, 9, 19, 22].

1.2 Quadratic Assignment Problem

Definition 1.1. Consider the set $\{1, 2, \ldots, n\}$ and three $n \times n$ coefficient matrices $A = (a_{ij})$, $B = (b_{ij})$, and $C = (c_{ij})$. The *quadratic assignment problem* denoted by $QAP(A, B, C)$ is as follows:-

$$\min_{\pi \in S_n} \Big(\sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} b_{\pi(i)\pi(j)} + \sum_{i=1}^{n} c_{i\pi(i)} \Big), \tag{1.1}$$

where S_n denotes the set of permutations of the set $\{1, 2, \ldots,\ n\}$.

Remark 1.2. The value of the above sum depends only on the matrices A, B, C, and on the permutation π. To mathematically formalize these dependencies, we write as follows:

$$K(A, B, C, \pi) = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} b_{\pi(i)\pi(j)} + \sum_{i=1}^{n} c_{i\pi(i)}.$$

The function $K(A, B, C, \pi)$ is referred to as the *objective function* of $QAP(A, B, C)$, and a permutation π_0 which minimizes it over all possible elements of S_n is termed as an *optimal solution* to $QAP(A, B, C)$. The corresponding value of the objective function, $K(A, B, C, \pi_0)$, is called the *optimal value* of $QAP(A, B, C)$.

We will now see the very first version of Quadratic Assignment Problem introduced by Koopmans and Beckmann [17], often termed as the facility location problem.

Example 1.3. *Facility location problem:* Suppose there are n facilities which are to be assigned to n different locations. The cost of this allocation process is proportional to material flow between the facilities multiplied by the distances between the locations plus, the initial costs for installing the facilities at their respective locations. The objective of this problem is to allocate each facility to a location such that the total cost involved is minimized.

This problem can be modelled as a QAP by defining the following three $n \times n$ coefficient matrices:

- $A = (a_{ik})$, where a_{ik} represents the flow from facility i to facility k.
- $B = (b_{jl})$, where b_{jl} is the distance from location j to location l.
- $C = (c_{ij})$, where c_{ij} represents the flow from facility i to facility j.

1.3 Traveling Salesperson Problem

The Traveling Salesperson Problem (TSP) is a well-studied classical problem from the field of combinatorics and graph theory. While the origin of this problem is unknown, it was formulated mathematically in the 1800s independently by Irish mathematician W.R.Hamilton [12], and by the British mathematician T.Kirkman [16]. In this section, we will explore two variants of the TSP.

1.3.1 Traveling Salesperson Problem - Optimization variant

Given n cities and the pairwise distances between all of them, this problem deals with the task of finding the minimal length tour for the salesperson covering each city exactly once and returning back to the initial city. Let us now formally state the traveling salesperson optimization problem.

Definition 1.4. Consider the set of integers $\{1, 2, \ldots, n\}$ representing the n cities, and consider a symmetric $n \times n$ matrix $D = (d_{ij})$ denoting the distance between the city i and the city j ($d_{ii} = 0$, for $1 \leq i \leq n$). Then, the *TSP* can be formally stated as follows:

$$\min_{\pi \in S_n} \Big(\sum_{i=1}^{n-1} d_{\pi(i)\pi(i+1)} + d_{\pi(n)\pi(1)} \Big). \tag{1.2}$$

It is interesting to note that the TSP can be formulated as a quadratic assignment problem with D as the distance matrix and the flow matrix taken to be the adjacency matrix of a cycle on n vertices. Note that, in the graph theory setting, the TSP deals with the task of finding the minimum cost hamiltonian cycle in a weighted undirected graph.

Definition 1.5. A *hamiltonian cycle* or a *spanning cycle* of a graph is a cycle that contains all the vertices of the graph.

See Figure 1.1 for an illustration of the hamiltonian cycle.

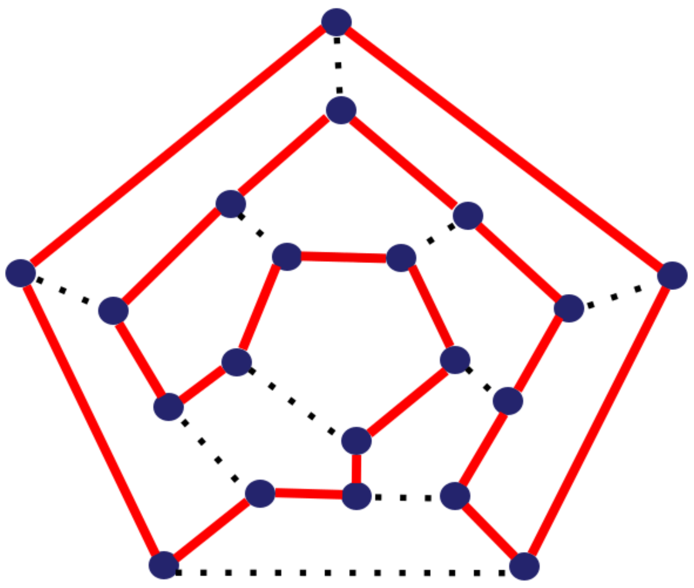

Fig. 1.1: A hamiltonian cycle.

1.3.2 Traveling Salesperson Problem - Decision variant

Unlike the above discussed optimization variant of TSP, this variant seeks a definitive answer. The decision variant of traveling salesperson problem can be stated as follows: Given a weighted graph $G = (V, E)$, and a number $C \in \mathbb{R}$, the problem asks whether the graph admits a hamiltonian cycle with cost no larger than C.

For the reminder of this thesis, we will use the acronym *TSP* to refer TSP-optimization problem and the acronym *TSP-Decide* to refer TSP-Decision problem.

1.4 Concepts of Machine Learning

Machine learning (ML), in simple words, can be explained as the branch of study that deals with developing computer programs capable of learning through experience and data. In recent times, with the advent of powerful hardware like Graphics Processing Unit (GPU) and the huge availability of big data, the world has started witnessing the presence of this field in various walks of life. Image recognition, product recommendations, speech recognition, self-driving cars, and medical diagnosis are a few fields where machine learning finds significant applications. The paradigms of machine learning can be broadly classified into three categories, namely:

1. **Supervised Learning:** Given a set of training data points, that is, a set of inputs and its corresponding outputs, the task of a supervised machine learning model is to learn the function that maps the data points. Thus after learning, if the user provides a new input, the model should correctly predict its output. It has numerous applications like image classification, spam mail detection, weather prediction, etc.

2. **Unsupervised Learning:** Unsupervised machine learning algorithms infer patterns from a dataset without reference to known or labeled outcomes. Unlike supervised machine learning, unsupervised machine learning methods are used when the values for the output data of the training set is unknown. Unsupervised learning is generally used to discover the underlying structure of the data. Applications include anomaly detection in bank transactions and clustering data into various groups based on similarities.

3. **Reinforcement Learning:** Reinforcement learning algorithms seek

to learn by interacting with the environment. Deepmind's Alpha Zero and Alpha Go are world-famous chess and go software that defeated the human world champions in their respective games.

In this project, we will be working exclusively with the supervised learning technique. We will now discuss about Artificial Neural Network (ANN), the most popular method for implementing supervised learning.

1.4.1 Artificial Neural Network (ANN)

Artificial Neural Network (ANN) is a programming paradigm inspired from neurons in the brain, though its actual task is not mimicking the brain activity. An artificial neural network typically has an input layer, an output layer, and a few hidden layers to connect the two. If the neural network has at least one hidden layer, then the network is called a *Multi-Layer Perceptron (MLP)*. The number of nodes and the number of layers are chosen for a problem by the programmer.

As mentioned earlier, ANN is the most popular method of implementing supervised learning. The reason for ANN to gain popularity among machine learning researchers is due to a celebrated result called the *universal approximation theorem* which roughly states that *neural networks with a single hidden layer can be used to approximate any continuous function to any desired precision.*

In Figure 1.2, a three-layered network with layer sizes (4, 6, 3) is shown.

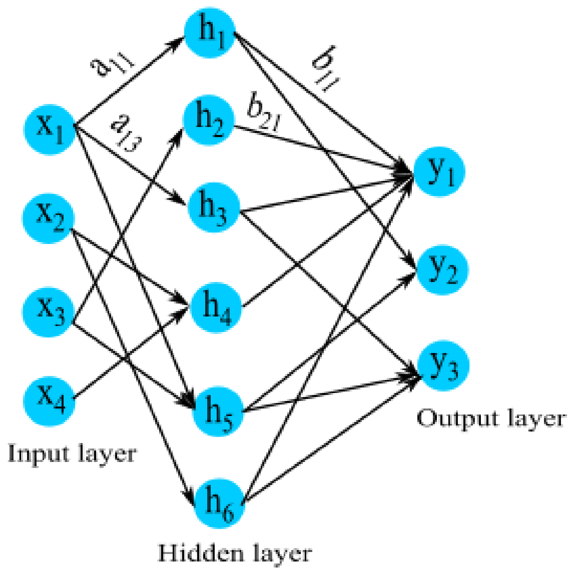

Fig. 1.2: An artificial neural network architecture.

The information passage from the input layer to the hidden layer results in the updation of the hidden layer parameter values in the following manner:

$$h_i = f_a\Big(\sum_j a_{ij}x_j\Big).$$

Here, the input values are made to undergo a linear transformation, following which it is passed through an activation function (denoted by f_a in the above expression), which is generally a non-linear function. The output of this activation function serves as the hidden layer parameter values. This is how the information flow happens over each subsequent layer. The parameter values for each layer depends on the input it receives from the previous layer. For instance, considering the above ANN architecture (Fig. 1.2), the output parameter values get updated in the following manner:

$$y_k = f_a\Big(\sum_i b_{ki}h_i\Big).$$

The non-linear activation functions are used in ANN to estimate non-linear relationship between input output. In its absence, the power of ANN will be limited to estimating only linear functions between the input and output, and the universal approximation theorem will fail in this case. The choice of activation function rests with the programmer. Generally, developers prefer to use Rectified Linear Unit (ReLU) function for hidden layers and the sigmoid function for the output layer. The most commonly used activation functions are:

- Linear function
  $f(x) = x$
- Sigmoid function
  $f(x) = \frac{1}{1+e^{(-x)}}$
- Rectified Linear Unit (ReLU) function
  $f(x) = \max\{0, x\}$
- Tan hyperbolic function
  $f(x) = tanh(x)$

These functions are illustrated in Figure 1.3 below.

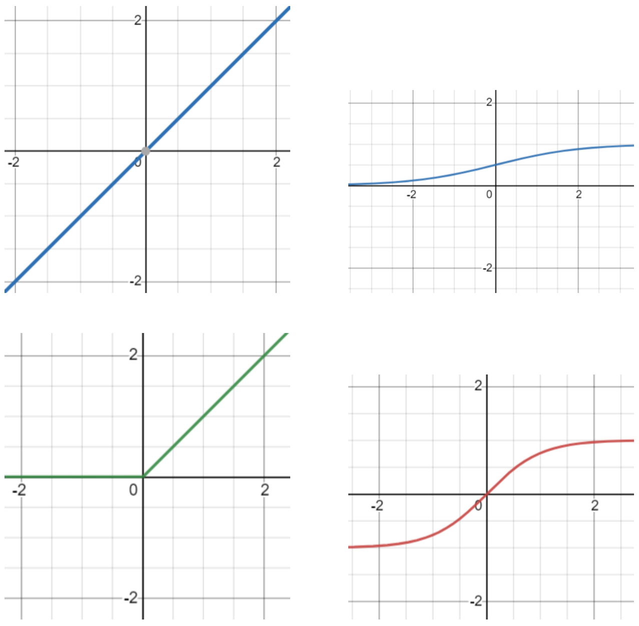

Fig. 1.3: Activation functions - Linear function (top left), Sigmoid function (top right), ReLU function (bottom left), and Tan hyperbolic function (bottom right).

The whole objective of ANN is to learn the model parameter values or weights (a_{ij} and b_{kl} in the above illustration). Then on obtaining a new input, the model could predict the output layer values with good accuracy. The learning of model parameters by an ANN model will be discussed in detail. Before we explain the learning process mathematically, we will first try to provide a rough intuition on the processes involved in the same.

As mentioned earlier, the central goal of an ANN model revolves around learning the parameter values. Achieving this goal generally involves two significant steps, which are repeated finitely many times until the learning process is completed. One is the forward message passing step, and the other is the backpropagation step. Initially, before the learning happens, the model parameters will be randomly initialised. As we have seen before, the hidden layer parameter values and output layer parameter value can be estimated based on the input values. The output thus obtained will be verified against the actual output, and the error (cost or loss) is estimated. This concludes the forward pass step.

Based on the error value, the model parameters will be updated sequentially in a layer-wise manner starting from the output layer to the input layer. This is the crucial idea behind the backpropagation step. Both the steps will be repeated a few times till the model starts predicting correctly (i.e. the error is minimized). We will now do an in-depth study of the backpropagation step.

1.4.1.1 Backpropagation algorithm

Before we proceed with our discussion on backpropagation algorithm, we will introduce a few notations. In the discussion that follows, I assume there is at least one hidden layer in our network, that is, our network is a *multi-layer perceptron*. We define a^l_j to be activation or output of node j in layer l. Also, we define w^l_{jk} to be the weight for the connection to node j of layer l from node k of layer $l-1$.

See Figure 1.4 to obtain better clarity on the notations we have used.

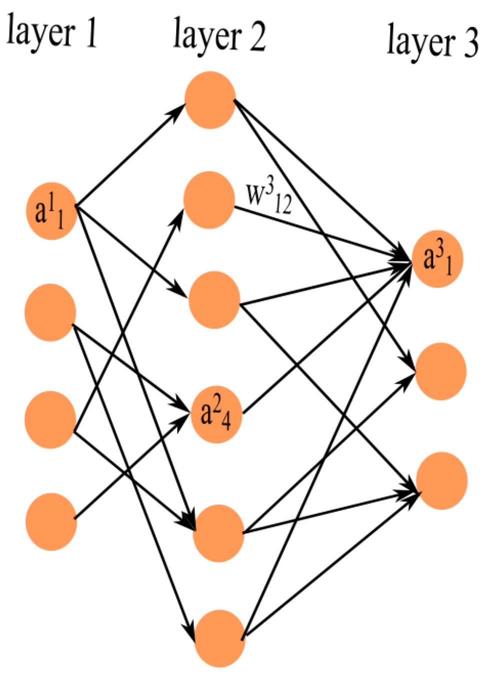

Fig. 1.4: Understanding the backpropagation algorithm.

From our previous discussions, we can define

$$a^l_j = \sigma\Big(\sum_k w^l_{jk} a^{l-1}_k\Big) = \sigma(z),$$

where z represents the quantity $\sum_k w^l_{jk} a^{l-1}_k$, and σ denotes the activation function. As said earlier, in order to reduce the cost C function, we need to update weights (model parameters). For this, we update the weight values using the technique of *gradient descent* which updates the weight value in the following fashion:

$$w^l_{jk} = w^l_{jk} - \eta \frac{\partial C}{\partial w^l_{jk}}. \tag{1.3}$$

That is, the weight values are changed (increased or decreased) in a direction opposite to the direction of the gradient of the cost function. Here, η denotes the learning rate which is usually decided by the programmer based on the requirements. Although gradient descent is a powerful optimization algorithm, it has its downside during implementation. It is computationally very difficult to perform the algorithm on large datasets as it involves the usage of whole training data to calculate gradients at each step. To overcome this problem and use algorithm effectively, we resort to choosing only a single random point or a few points from the dataset to calculate gradients without compromising the performance drastically. This method is employed in optimization techniques like *stochastic gradient descent (SGD)* and *mini-batch gradient descent*. If we observe equation (1.3), all the gradient descent algorithms generally involve the computation of partial derivative with respect to weight values. To do this computation for a large network with many parameters is computationally hard. In this scenario, we invoke the technique of backpropagation. Backpropagation using the *chain rule of calculus* helps to compute these derivatives in a sequential and layer-wise manner, starting with the output side and proceeding towards the input side. Derivatives associated with weights of one layer (say, for instance, l) will be used to compute the same for the previous layer (layer $l-1$). We will now see in detail how the backpropagation actually works.

By the chain rule of calculus, we have:

$$\begin{aligned} \frac{\partial C}{\partial w^l_{jk}} &= \frac{\partial C}{\partial z^l_j}\,\frac{\partial z^l_j}{\partial w^l_{jk}} \\ &= \frac{\partial C}{\partial z^l_j}\,a^{l-1}_k \\ &= \delta^l_j\,a^{l-1}_k, \end{aligned} \tag{1.4}$$

where δ^l_j denotes the expression $\frac{\partial C}{\partial z^l_j}$. Note that in each layer, we have multiple nodes. So, we can rewrite the expression for δ^l_j in the following manner using the chain rule

$$\begin{aligned} \delta^l_j &= \sum_k \frac{\partial C}{\partial z^{l+1}_k}\,\frac{\partial z^{l+1}_k}{\partial z^l_j} \\ &= \sum_k \delta^{l+1}_k\,\frac{\partial z^{l+1}_k}{\partial z^l_j}. \end{aligned} \tag{1.5}$$

By definition, we have

$$\begin{aligned} z_k^{l+1} &= \sum_m w_{km}^{l+1}\, a_m^l \\ &= \sum_m w_{km}^{l+1}\, \sigma(z_m^l). \end{aligned}$$

So, we have

$$\frac{\partial z_k^{l+1}}{\partial z_j^l} = w_{kj}^{l+1}\sigma'(z_j^l).$$

Substituting $\frac{\partial z_k^{l+1}}{\partial z_j^l}$ in equation (1.5), we get

$$\delta_j^l = \sum_k \delta_k^{l+1} w_{kj}^{l+1}\sigma'(z_j^l).$$

This shows that the δ values of a particular layer depend on the δ values of the next layer. Substitute the value of δ_j^l in equation (1.4) to obtain the final expression for $\frac{\partial C}{\partial w_{jk}^l}$. Hence, we get

$$\frac{\partial C}{\partial w_{jk}^l} = \sum_k \delta_k^{l+1} w_{kj}^{l+1}\sigma'(z_j^l) a_k^{l-1}.$$

Thus, we conclude our discussion on the working of artificial neural networks. Next, we will discuss in brief a special type of neural networks known as the *recurrent neural networks.*

1.4.2 Recurrent Neural Network (RNN)

In an ANN, a single input ultimately determines the activations of all the neurons through the remaining layers, and in particular, the network is static over time. On the other hand, a Recurrent Neural Network has a dynamic setting (time-varying behavior).

See Figure 1.5 for an illustration of the RNN architecture.

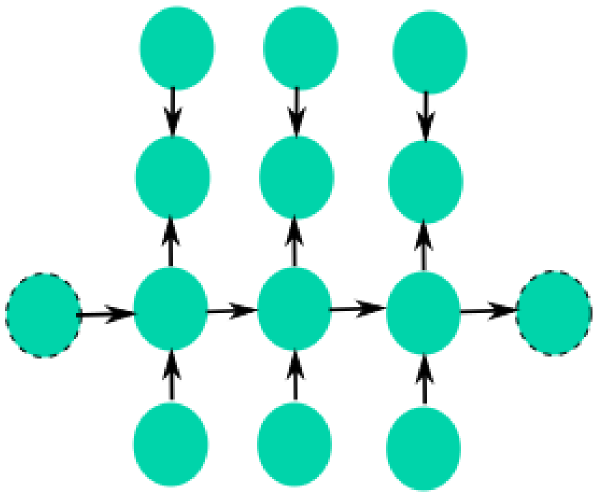

Fig. 1.5: A recurrent neural network architecture.

Hidden neurons of an RNN network might not just be determined by the activations in previous hidden layers, but also by the activations at earlier times. Indeed, a neuron's activation might be determined partly by its own activation at an earlier time. Sometimes, activations of hidden and output neurons will not be determined just by the current input to the network, but also by earlier inputs. So, RNNs are used in analysing data or processes that change over time. For instance, recurrent neural networks have their applications in fields like speech generation and stock price prediction, where current prediction has its dependence on previous outputs. Training an RNN model is more or less similar to an ANN, that is, by using backpropagation and gradient descent with small straightforward modifications. A drawback of an RNN is that it may sometimes suffer from the problem of *vanishing gradient*. Vanishing gradient refers to the problem that during the process of backpropagation, the gradient gets smaller and smaller as it is propagated back through the layers. This makes learning in early layers prolonged. The problem gets much worse in RNNs in comparison with ANNs, and since gradients are not just propagated backwards through layers, they are propagated backwards through time. Thus, if the network runs for an extended period of time, the gradient can become extremely unstable and hard to learn from. To overcome this problem, we generally use a modified RNN named *Long Short-Term Memory (LSTM)*. LSTMs are used in situations that demand the network to remember output from an instance in the past.

2. THE THEORY OF POLYHEDRA

We aim to understand the polytope structure associated with the optimization problems. To approach this goal, we must first get familiarised with some basic terminologies and results from polyhedral theory. The references used for this chapter are [3, 8].

2.1 Basic Introduction

Let us begin by exploring some basic definitions before we move on to understand the concepts of polyhedral theory.

Definition 2.1. For $x_1, \dots, x_k \in \mathbb{R}^n$ and $\lambda_1, \dots, \lambda_k \in \mathbb{R}$,

(a) The vector $\sum\limits_{i=1}^{k} \lambda_i x_i$ is called a *linear combination* of $x_1, \dots, x_k$.

(b) The vector $\sum\limits_{i=1}^{k} \lambda_i x_i$ is called a *conical combination* of $x_1, \dots, x_k$, if $\lambda_1, \dots, \lambda_k \geq 0$ holds.

(c) The vector $\sum\limits_{i=1}^{k} \lambda_i x_i$ is called an *affine combination* of $x_1, \dots, x_k$, if $\sum\limits_{i=1}^{k} \lambda_i = 1$ holds.

(d) The vector $\sum\limits_{i=1}^{k} \lambda_i x_i$ is called a *convex combination* of $x_1, \dots, x_k$, if both $\lambda_1, \dots, \lambda_k \geq 0$ and $\sum\limits_{i=1}^{k} \lambda_i = 1$ hold.

See Figure 2.1 for an illustration of the vectors in Definition 2.1.

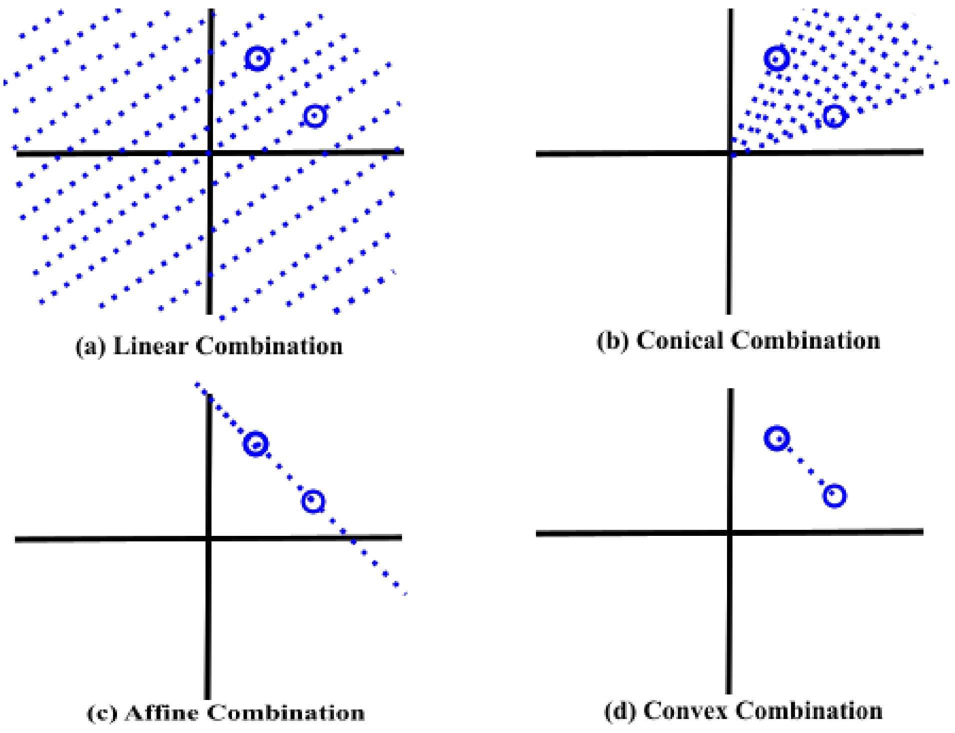

Fig. 2.1: Different combinations of vectors.

Definition 2.2. The set of all possible linear, conical, affine, or convex combinations of (finitely many) vectors in a set $S \subset \mathbb{R}^n$ is termed as the *linear hull*, *conical hull*, *affine hull*, or *convex hull*, respectively, of S.

We denote linear hull, conical hull, affine hull, or convex hull of set S by $lin(S)$, $cone(S)$, $aff(S)$, or $conv(S)$, respectively.

Example 2.3. Consider a set $S = \{(1,0,0),(0,1,0),(0,0,1)\} \subset \mathbb{R}^3$. Then:

(a) $lin(S)$: $\mathbb{R}^3$

(b) $cone(S)$: $\mathbb{R}^3_{\geq 0}$

(c) $aff(S)$: Plane passing through (1, 0, 0), (0, 1, 0), (0, 0, 1).

(d) $conv(S)$: Triangular region with vertices (1, 0, 0), (0, 1, 0), (0, 0, 1).

We have now laid the framework for introducing the notion of a polyhedron.

2.2 Polyhedron and its properties

We begin our discussion on polyhedral theory by defining the notion of hyperplanes and halfspaces.

Definition 2.4. Let $a \in \mathbb{R}^n$ with $a \neq 0$, and let $\alpha \in \mathbb{R}$. The set $\{x \in \mathbb{R}^n : ax = \alpha\}$ is called the *hyperplane* defined by a and α, and the set $\{x \in \mathbb{R}^n : ax \leq \alpha\}$ is called the *halfspace* defined by a and α.

The set $\{x \in \mathbb{R}^n : ax = \alpha\}$ is referred to as the *boundary hyperplane* of the halfspace $\{x \in \mathbb{R}^n : ax \leq \alpha\}$. See Figure 2.2 below for an illustration of a hyperplane.

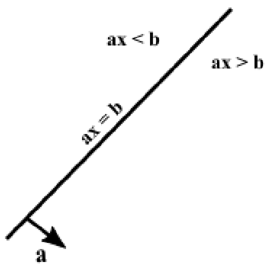

Fig. 2.2: A hyperplane and two half spaces.

Definition 2.5. A *polyhedron* is a set of the form $\{x \in \mathbb{R}^n : Ax \leq b\}$, where A is a $m \times n$ matrix and $b \in \mathbb{R}^m$.

Note that a polyhedron is an intersection of a finite number of half-spaces, as shown in Figure 2.3.

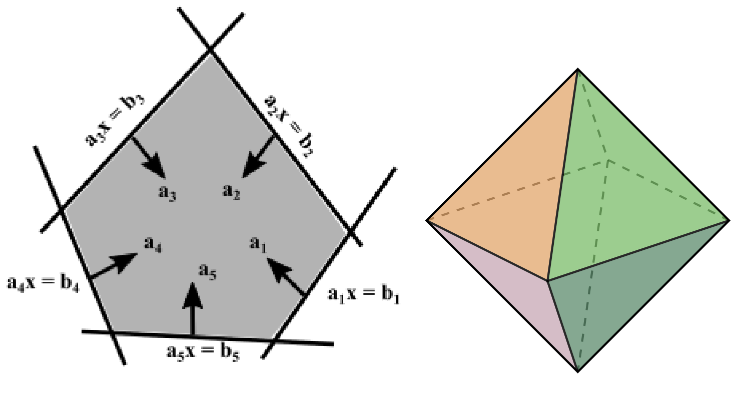

Fig. 2.3: Polyhedra in 2D and 3D formed as intersection of half spaces.

Definition 2.6. A set $L \subset \mathbb{R}^n$ is termed as a *bounded* set if there exists some constant k such that absolute value of every component of every element of L is less than or equal to k.

By Definition 2.6, a polyhedron can either extend to infinity, or can be confined to a finite region.

Definition 2.7. A set $S \subset \mathbb{R}^n$ is *convex* if for any $x_1, x_2 \in S$, and for any $\lambda \in$ [0,1], we have $\lambda x_1 + (1-\lambda)y_1 \in S$.

Theorem 2.8. *(a) The intersection of convex sets is a convex set.*

(b) Every polyhedron is a convex set.

(c) A convex combination of a finite number of elements of a convex set also belongs to the same set.

(d) The convex hull of a finite number of vectors is a convex set.

Proof. (a) Consider the convex sets S_i for i belonging to some index set J. Suppose elements x_1 and x_2 belongs to the intersection set $\cap_{i \in J} S_i$. Let $\lambda \in$ [0,1]. Since each S_i is a convex set containing x_1 and x_2, each S_i will also have the element $\lambda x_1 + (1-\lambda)x_2$. So, the element $\lambda x_1 + (1-\lambda)x_2$ also belongs to intersection of the sets. Hence, by definition, the set $\cap_{i \in J} S_i$ is a convex set.

(b) Let c be a vector, and let d be a scalar. Consider vectors x_1 and x_2 satisfying $cx_1 \geq d$ and $cx_2 \geq d$, respectively, that is, the vectors x_1 and x_2 belong to the same halfspace. Let $\lambda \in$ [0,1]. Then clearly,

$$c(\lambda x_1 + (1-\lambda)x_2) \geq \lambda d + (1-\lambda)d = d,$$

which means that $\lambda x_1 + (1-\lambda)x_2$ also belongs to the same halfspace. This implies that a halfspace is convex. Since a polyhedron is the intersection of a finite number of halfspaces, the required result directly follows from part (a).

(c) From the definition of convexity, a convex combination of two elements of a set belongs to that set. We prove this by induction on the number of elements in the set. Let us assume the induction hypothesis that a convex combination of k elements of a convex set belongs to that set. Now, consider $k + 1$ elements $x_1, \ldots, x_{k+1}$ of a convex set S, and let $\lambda_1, \ldots, \lambda_{k+1}$ be nonnegative scalars that sum to 1. Without loss of generality, we assume that $\lambda_{k+1} \neq 1$. Then,

$$\sum_{i=1}^{k+1} \lambda_i x_i = \lambda_{k+1} x_{k+1} + (1 - \lambda_{k+1}) \sum_{i=1}^{k} \frac{\lambda_i}{1 - \lambda_{k+1}} x^i.$$

Clearly, the coefficients $\lambda_i/(l - \lambda_{k+l})$ for $i = 1, \ldots,\ k$, are nonnegative and sum up to unity. By the induction hypothesis, $\sum\limits_{i=1}^{k} \lambda_i x_i/(1-\lambda_{k+l}) \in S$. Then, the fact that S is convex along with the above equation imply that $\sum\limits_{i=1}^{k+1} \lambda_i x_i \in S$, which completes the induction step.

(d) Let S denote the convex hull of the finetely many vectors $x_1, \ldots,\ x_k$ and let $y = \sum\limits_{i=1}^{k} \alpha_i x_i, z = \sum\limits_{i=1}^{k} \beta_i x_i$ be two elements of S, where $\alpha_i \geq 0$, $\beta_i \geq 0$, and $\sum\limits_{i=1}^{k} \alpha_i = \sum\limits_{i=1}^{k} \beta_i = 1$. Let $\lambda \in [0, 1]$. Then,

$$\lambda y + (1 - \lambda)z = \lambda \sum_{i=1}^{k} \alpha_i x_i + (1 - \lambda) \sum_{i=1}^{k} \beta_i x_i = \sum_{i=1}^{k} (\lambda \alpha_i + (1 - \lambda)\beta_i) x_i.$$

Clearly, the coefficients $\lambda\alpha_i + (1-\lambda)\alpha_i$ for $i = 1, \ldots,\ k$, are nonnegative and sum to unity. Hence, $\lambda y + (1 - \lambda)z$ is a convex combination of $x_1, \ldots,\ x_k$ and, therefore, belongs to S. Thus, S is a convex set.

□

We have seen the basic concepts of polyhedra. Before we move on to understand the polytopes, we will introduce the concepts of cones and we will also explore some important theorems of Minikowski-Weyl.

2.3 Minkowski–Weyl Theorem for Cones

Definition 2.9. A set $C \subseteq \mathbb{R}^n$ is called a *polyhedral cone* if $C = \{x \in \mathbb{R}^n : Ax \leq 0\}$ for some $m \times n$ matrix A.

Remark 2.10. A polyhedral cone is the intersection of a finite number of half-spaces having origin on their boundaries, as shown in Figure 2.4.

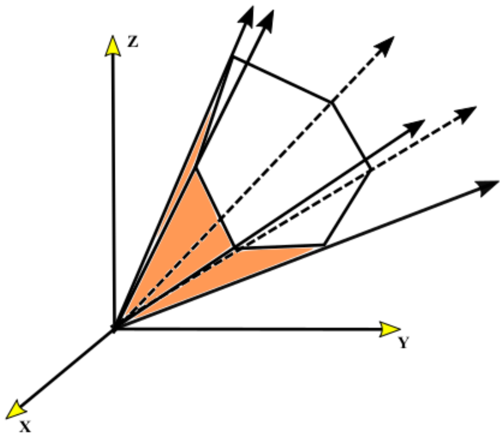

Fig. 2.4: A polyhedral cone.

Definition 2.11. A set $C \subseteq \mathbb{R}^n$ is called a *finitely generated cone* if C is a convex cone generated by some finite set of vectors $x_1, \ldots, x_n \in \mathbb{R}^n$, for $n \geq 1$.

In other words, $C = cone(x_1, \ldots, x_n)$.

Theorem 2.12 (Minkowski–Weyl Theorem for Cones)**.** *A subset of $\mathbb{R}^n$ is a polyhedral cone if and only if it is a finitely generated cone.*

We will skip the proof of this theorem as it involves some results from linear programming which are beyond the scope of this discussion. We will now introduce the concepts of polytope and how it differs from a polyhedron.

2.4 Minkowski–Weyl Theorem for Polytopes

Definition 2.13. A subset Q of $\mathbb{R}^n$ is called a *polytope* if it is the convex hull of some finite set of vectors in $\mathbb{R}^n$.

Definition 2.14. Given subsets A, B of $\mathbb{R}^n$, the *Minkowski sum* of A, B is defined as the set

$$A + B := \{x \in \mathbb{R}^n : \exists\, a \in A, b \in B \text{ such that } x = a + b\}.$$

Theorem 2.15 (Minkowski–Weyl Theorem)**.** *A subset P of $\mathbb{R}^n$ is a polyhedron if and only if P can be expressed as the Minkowski sum of the polytope $Q \subset \mathbb{R}^n$ and a finitely generated cone $C \subseteq \mathbb{R}^n$.*

Proof. Let $P \subseteq \mathbb{R}^n$. It suffices to show that the following two statements are equivalent.

(a) There exist a matrix A and a vector b such that $P = \{x \in \mathbb{R}^n : Ax \leq b\}$.

(b) There exist vectors $u_1, \dots, u_p \in \mathbb{R}^n$ and $v_1, \dots, v_q \in \mathbb{R}^n$ such that $P = conv(u_1, \dots, u_p) + cone(v_1, \dots, v_q)$.

We will first show that the statement (a) $\implies$ (b). Assume that (a) holds, and consider the polyhedral cone $C_P = \{(x, y) \in R^{n+1} : Ax - by \leq 0, y \geq 0\}$. Since C_P is a polyhedral cone, by Theorem 2.12, it is also a finitely generated cone. As $y \geq 0$ for every vector $(x, y) \in C_P$, the generators of C_P can be normalized in such a way that their $(n+1)^{th}$ component is either 0 or 1. Thus, there exist $u_1, \dots,\ u_p \in \mathbb{R}^n$ and $v_1, \dots,\ v_q \in \mathbb{R}^n$ such that

$$C_P = cone\left\{ \begin{bmatrix} u_1 \\ 1 \end{bmatrix}, \dots, \begin{bmatrix} u_p \\ 1 \end{bmatrix}, \begin{bmatrix} v_1 \\ 0 \end{bmatrix}, \dots, \begin{bmatrix} v_q \\ 0 \end{bmatrix} \right\}.$$

Since $P = \{x : (x, 1) \in C_P\}$, this implies that $P = \text{conv}\ (v_1, \dots,\ v_p) + \text{cone}\ (r_1, \dots, r_q)$, from which it follows that (a) $\implies$ (b).

For the converse, assume that (b) holds, and let $C_P \in \mathbb{R}^{n+1}$ be the finitely generated cone defined as above. Note that by definition, $P = \{x : (x, 1) \in C_P\}$. Since P is a finitely generated cone. It follows from Theorem 2.12 that C_P is also a polyhedral cone. Thus, there exists a matrix (A, b) such that $C_P = \{(x, y) \in \mathbb{R}^{n+1} : Ax - by \leq 0\}$. This implies that $P = \{x \in \mathbb{R}^n : Ax \leq b\}$. Thus we have shown that the statement (b) implies statement (a). $\square$

Theorem 2.15 is illustrated in Figure 2.5 below.

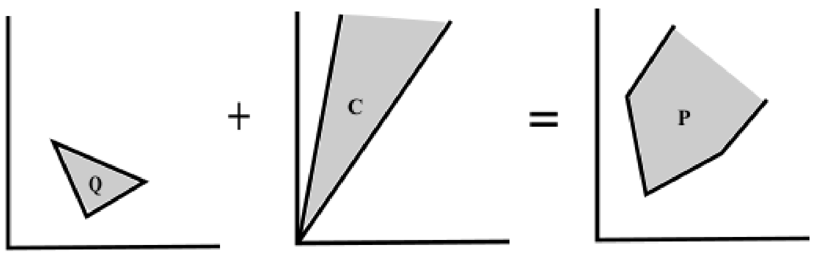

Fig. 2.5: An illustration of the Minkowski–Weyl Theorem.

The following is a direct consequence of Theorem 2.15.

Corollary 2.16 (Minkowski–Weyl Theorem for Polytopes)**.** A set $Q \subseteq \mathbb{R}^n$ is a polytope if and only if Q is a bounded polyhedron.

Next, we will discuss a very significant result relevant to polytopes.

Definition 2.17. Let P be a polyhedron. A vector $x \in P$ is an *extreme point* of P if x cannot be expressed as the convex combination of two distinct vectors $y, z \in P$.

Theorem 2.18. *A non-empty and bounded polyhedron (a polytope) is the convex hull of its extreme points.*

Figure 2.6 below gives an illustration of Theorem 2.18.

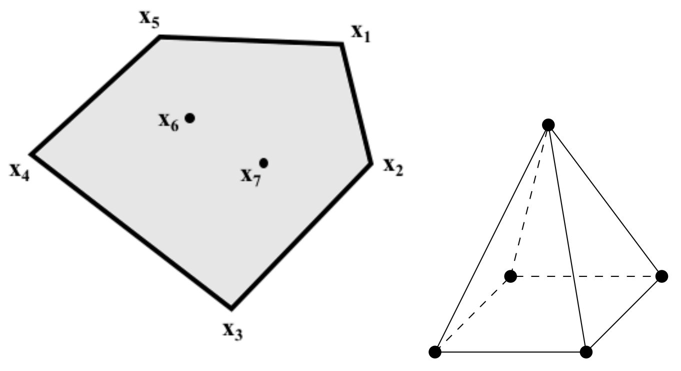

Fig. 2.6: Polytopes in 2D and 3D formed as convex hull of its extreme points.

As the proof of this theorem involves a few terminologies and results from future chapters, we will skip the proof for the moment. We have included the detailed proof of this theorem in Section 3.6.

Definition 2.19. A polytope in $\mathbb{R}^d$ is called a *0/1 polytope* if all its vertices are in $\{0, 1\}^d$.

A 0/1 polytope is the convex hull of a subset of the 2^d point set $\{0, 1\}^d$, for some $d \geq 0$. The polytopes we will discuss as part of this project will be 0/1 polytopes.

3. THE QUADRATIC ASSIGNMENT POLYTOPE

In this chapter, we will study the quadratic assignment polytope. First, we will introduce a graph-theoretic framework to introduce the quadratic assignment polytope. Following this, we will investigate its properties. This chapter is based on [14, 15, 18].

Before we develop the required framework to define the QAP polytope, it is necessary to introduce a reformulated (but equivalent) version of the QAP using permutation matrices. This reformulated version was introduced by Lawler in 1936 [18].

Definition 3.1. A *permutation matrix* is a 0/1 matrix obtained by permuting the rows of an $n \times n$ identity matrix according to some permutation of the integers 1 to n.

Remark 3.2. Every row and column of a permutation matrix contains precisely a single 1 with 0's everywhere else, and every permutation corresponds to a unique permutation matrix.

The Lawler version of quadratic assignment problem can be summarised as follows:
Let Π_n be the set of all $n \times n$ permutation matrices. The QAP task is to find:

$$\min \quad \Big(\sum_{\substack{i,k=1 \\ i<k}}^{n} \sum_{\substack{j,l=1 \\ j\neq l}}^{n} d_{ijkl} x_{ij} x_{kl} + \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \Big),$$

s.t $X = (x_{ij})_{i,j=1,\ldots,\,n} \in \Pi_n$,
where, $d_{ijkl} = a_{ik}b_{jl}$ ($1 \leq i, j, k, l \leq n$; $i \neq k$ or $j \neq l$).

Now, we build the graph theoretic framework required to establish a foundation for introducing the quadratic assignment polytope. Consider the graph $\mathcal{G}_n = (\mathcal{V}_n,\, \mathcal{E}_n)$ with the node set $\mathcal{V}_n = \{(i,j) : 1 \leq i, j \leq n\}$ and the edge set $\mathcal{E}_n = \{\{\text{(i,j),(k,l)}\} \in \binom{\mathcal{V}_n}{2} : i \neq k,\, j \neq l\}$, where $\binom{\mathcal{V}_n}{2}$ denotes the set of all cardinality 2 subsets of the node set $\mathcal{V}_n$. For ease of working, let us define the notation, $[i,j,k,l] := \{(i,j),(k,l)\}$ for all edges $\{(i,j),(k,l)\} \in \mathcal{E}_n$. See the illustration in Figure 3.1 below for the graph $\mathcal{G}_4$.

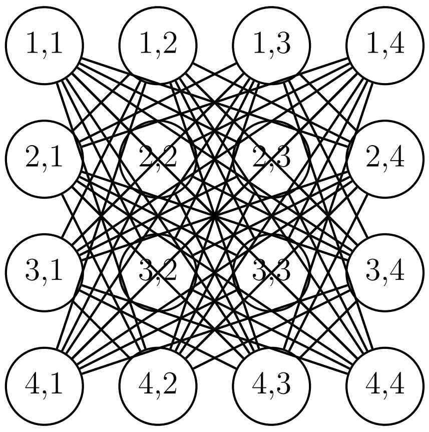

Fig. 3.1: The graph $\mathcal{G}_4$.

Note that, the graph $\mathcal{G}_n$ has the following two important properties which help in building a non-trivial connection between the above graph $\mathcal{G}_n$ and the QAP:

(a) The size of maximal clique of graph $\mathcal{G}_n$ is n.

(b) The n-cliques of $\mathcal{G}_n$ has one to one correspondance with the $n \times n$ permutation matrices.

See the illustration in Figure 3.2 for a maximal clique in $\mathcal{G}_n$ when $n = 4$.

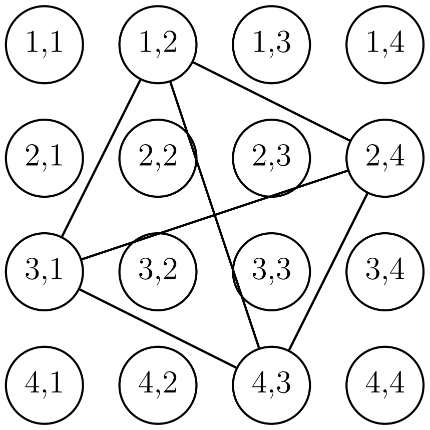

Fig. 3.2: A 4-clique in $\mathcal{G}_4$.

We will now introduce some notation that we would be using multiple times in our future discussions. Note that $x \in \mathbb{R}^{\mathcal{V}_n}$ is a vector whose components are associated with the nodes in $\mathcal{V}_n$. For instance, in our previous example, the graph $\mathcal{G}_4$ had 16 vertices. Hence, a vector $x \in \mathbb{R}^{\mathcal{V}_n}$ will have 16 components, each component associated to a single vertex in $\mathcal{V}_n$.

Earlier in this chapter, we have introduced the Lawler version of QAP. We would use the same here to formulate quadratic assignment problem as a graph theory problem. Doing this will help us in defining the QAP Polytope in our later discussions.

Suppose we are provided with an instance of Lawler QAP say, $(QAP)^{(n)}_{(c,d)}$,

$$\begin{aligned} &\min \quad \Big(\sum_{i,j,k,l=1}^{n} d_{ijkl} x_{ij} x_{kl} + \sum_{i,j=1}^{n} c_{ij} x_{ij} \Big), \\ &\text{s.t } X = (x_{ij})_{1 \le i,j \le n} \in \Pi_n, \end{aligned}$$

where $(c \in (\mathbb{R}^n)^2, d \in (\mathbb{R}^n)^4)$. To represent the given instance of Lawler QAP as a graph theory problem, we assign weights to nodes and edges of $\mathcal{G}_n$ by $(c', d') \in \mathbb{R}^{\mathcal{V}_n} \times \mathbb{R}^{\mathcal{E}_n}$ by setting $c'_{(i,j)} := c_{ij}$, for each node $(i,j) \in \mathcal{V}_n$ and $d'_{[i,j,k,l]} := d_{ijkl}$, for each edge $[i,j,k,l] \in \mathcal{E}_n$ (with $i < k$). So, solving the problem of $(QAP)^{(n)}_{(c,d)}$ is equivalent to finding the minimal node and edge-weighted n-clique in the graph $\mathcal{G}_n$ weighted by (c', d') as above.

We will introduce some new notations that will be of relevance in subsequent sections. Suppose $x \in \mathbb{R}^V$ is a vector whose components are associated with the nodes in V, and $U \subset V$ be a subset of the nodes. Then we denote $x(U)$ to be the sum of all components of x belonging to elements in U.

Let the set of (node sets of) n-cliques of $\mathcal{G}_n$ be denoted by

$$\mathcal{CL}_n := \{C \subseteq \mathcal{V}_n : C \text{ is a } n\text{-clique of } \mathcal{G}_n\}.$$

In a formal setting, solving QAP can be stated as follows:

$$\begin{array}{cc} \min & c'(C) + d'(\mathcal{E}_n(C)), \\ \text{s.t.} & C \in \mathcal{CL}_n. \end{array}$$

For any subset $U \subset \mathcal{V}_n$ of $\mathcal{G}_n$, we denote the characteristic vector or incidence vector of U by $x^U \in \mathbb{R}^{\mathcal{V}_n}$, that is, for any $v \in \mathcal{V}_n$, we set

$$x^W_v := \begin{cases} 1, & \text{if } v \in U, \text{ and} \\ 0, & \text{if } v \notin U. \end{cases}$$

Analogously, for any subset $F \subseteq \mathcal{E}_n$ of edges of $\mathcal{G}_n$, we denote by y^F, the characteristic vector of $F \in \mathbb{R}^{\mathcal{E}_n}$, that is, for $e \in \mathcal{E}_n$, we define

$$y^F_e := \begin{cases} 1, & \text{if } e \in F, \text{ and} \\ 0, & \text{if } e \notin F. \end{cases}$$

The *incidence vector* of a n-clique $C \subset \mathcal{V}_n$ in $\mathcal{G}_n$ is the 0/1 vector $(x^C, y^{\mathcal{E}_n(C)})$. This vector belongs to the space $\mathbb{R}^{\mathcal{V}_n} \times \mathbb{R}^{\mathcal{E}_n}$.

Definition 3.3. We define the *Quadratic assignment polytope* by

$$\mathcal{QAP}_n := conv\Big(\{(x^C, y^{\mathcal{E}_n(C)}) : C \in \mathcal{CL}_n\}\Big).$$

3.1 Symmetries of $\mathcal{QAP}_n$

To describe the symmetry of a polytope, we must formally introduce a few basic definitions concerning affine spaces and affine transformations.

Definition 3.4. A set $S \subset \mathbb{R}^n$ is *affine subspace* of $\mathbb{R}^n$ if there exists $p \in \mathbb{R}^n$ and a vector subspace V of $\mathbb{R}^n$ such that every $x \in S$ can be written as $p + v$ for some $v \in V$.

We write $s = p + V$. If $S \neq \mathbb{R}^n$, S is said to be a *proper affine space*. Moreover, the dimension of affine space is equal to dimension of vector space V. In Definition 3.4, p is not unique, but there is a unique choice for V. For instance, if $q = p + V$ for some $v \in V$, then $q + V$ and $p + V$ render the same affine subspace.

Example 3.5. Every vector subspace of $\mathbb{R}^n$ is an affine subspace. The converse need not be true since an affine subspace need not necessarily have the zero vector.

Example 3.6. A single point in $\mathbb{R}^n$ forms an affine subspace. In this case, $V = \{0\}$.

Example 3.7. Let $A \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$, where m and n are positive integers. If $S = \{x \in \mathbb{R}^n : Ax = b\}$ is non-empty, then S is an affine subspace. To see this, let x' be a solution of $Ax = b$. Then, $S = x' + N$, where N is null space of A.

In fact, the converse of the assertion in Example 3.7 also holds.

Theorem 3.8. *Every proper affine subspace of* $\mathbb{R}^n$ *is of form* $\{x \in \mathbb{R}^n : Ax = b\}$ *for some* $A \in \mathbb{R}^{m\times n}$ *and* $b \in \mathbb{R}^m$.

Proof. Consider an affine subspace given by $p + V$, where $p \in \mathbb{R}^n$ and V is proper vector subspace of $\mathbb{R}^n$. Then, there exists a matrix A such that V equals null space of A. Taking $b = Ap$, we have $p + V = \{x \in \mathbb{R}^n : Ax = b\}$. □

We will now discuss the definition of an affine map.

Definition 3.9. For any two vector spaces E and F, a function $f : E \to F$ is an *affine map* if for every affine combination $\sum_{i \in I} \lambda_i u_i$, we have

$$f\Big(\sum_{i \in I} \lambda_i u_i\Big) = \sum_{i \in I} \lambda_i f(u_i).$$

Equivalently, a function $f : E \to F$ is an *affine map* if there is some linear map $h : E \to F$ and some fixed vector $c \in F$ such that

$$f(x) = h(x) + c, \qquad \forall\, x \in E.$$

Now we will define an affine transformation and see its relevance in the context of understanding the symmetry of a polytope.

Definition 3.10. An *affine transformation* is a function that maps an affine space onto itself while preserving dimension of any affine subspaces.

Affine transformations need not necessarily preserve angles between lines, or the distances between points, though they preserve the ratio of the distance between points lying on a straight line. Formally, an affine transformation on $\mathbb{R}^n$ is a map $F : \mathbb{R}^n \to \mathbb{R}^n$ of the form $F(p) = Ap + q$, $\forall\, p \in \mathbb{R}^n$, where A is a fixed linear transformation.

Affine transformations preserve the dimension of affine subspaces, so to realise symmetries of QAP polytope, it is sufficient to look at the distance preserving affine transformations of the vector space $\mathbb{R}^{\mathcal{V}_n} \times \mathbb{R}^{\mathcal{E}_n}$ that maps the polytope $\mathcal{QAP}_n$ to itself. We observe that the graph $\mathcal{G}_n$ is invariant under permutation of rows (or columns) and transposition of nodes (mapping (i,j) to (j,i) $\forall$ $(i,j) \in \mathcal{V}_n$). So, the automorphisms of the graph $\mathcal{G}_n$ clearly induce injective maps from the set of the incidence vectors of n-cliques to itself. Thus, permutations of the rows or the columns or transposing the node-set $\mathcal{V}_n$ also induce the symmetries for $\mathcal{QAP}_n$. Hence, it suffices to prove results about $\mathcal{QAP}_n$ polytope up to permutations of the rows (and columns), and transpositions of $\mathcal{V}_n$.

3.2 Understanding the QAP polytope better

To better understand the QAP Polytope, we derive a linear description of a polytope $\mathcal{P}_n \in \mathbb{R}^{\mathcal{V}_n} \times \mathbb{R}^{\mathcal{E}_n}$ whose integer points can be shown to be vertices of $\mathcal{QAP}_n$, that is, the incidence vectors of n-cliques of $\mathcal{G}_n$.

For $1 \leq \alpha \leq n$, we denote $row_\alpha := \{(\alpha, j) : 1 \leq j \leq n\}$ to be the α-th row of $\mathcal{V}_n$, and we denote $col_\alpha := \{(i, \alpha) : 1 \leq i \leq n\}$ to be the α-th column of $\mathcal{V}_n$. Observe that, for any point $(x, y) \in \mathcal{QAP}_n$, the following two equations clearly hold:

$$x(row_i) = 1, \tag{3.1}$$

and

$$x(col_j) = 1, \tag{3.2}$$

where $1 \leq i, j \leq n$.

For two disjoint subsets $U_1, U_2 \subset V$ of nodes, we denote $(U_1 : U_2)$ to be the set of all edges having one node in U_1 and one in U_2. For the singleton set $\{v\}$, we often omit the brackets and write as follows $\delta(v)$.

Note that, for any point $(x, y) \in \mathcal{QAP}_n$, the following pair of equations hold:

$$y((i,\ j) : row_k) - x_{(i,\ j)} = 0, \tag{3.3}$$

$$y((i,\ j) : col_l) - x_{(i,\ j)} = 0, \tag{3.4}$$

where $1 \leq i, j, k, l \leq n$ and $i \neq k$, $j \neq l$.

So, from the two sets of equations we have discussed above, it is clear that for all $(x,y) \in \mathcal{QAP}_n$, the following equations hold:

$$\begin{aligned} x(row_i) &= 1, \\ x(col_j) &= 1, \\ y((i,\ j) : row_k) - x_{(i,\ j)} &= 0, \\ y((i,\ j) : col_l) - x_{(i,\ j)} &= 0. \end{aligned}$$

We denote the above system of equations by the system $A^{(n)}(x,y) = b^{(n)}$. Note that, this system consists of $2n + 2n^2(n-1)$ equations. We now define a polytope

$$\mathcal{P}_n := \{(x,y) \in \mathbb{R}^{\mathcal{V}_n} \times \mathbb{R}^{\mathcal{E}_n} : A^{(n)}(x,y) = b^{(n)}, y \geq 0\}.$$

Remark 3.11. From the system of equations, $A^{(n)}(x,y) = b^{(n)}$, it is clear that $(x,y) \in \mathcal{P}_n$ implies $y \leq 1$, $x \geq 0$, and $x \leq 1$.

The next theorem reveals that the already found constraints are sufficient to identify vertices of $\mathcal{QAP}_n$ (incidence vectors of n-cliques of $\mathcal{G}_n$) among the integer vectors of the polytope $\mathcal{P}_n$.

Theorem 3.12. *$\mathcal{QAP}_n = conv\ (\mathcal{P}_n \cap \mathbb{Z}^{\mathcal{V}_n} \times \mathbb{Z}^{\mathcal{E}_n})$.*

Proof. Clearly, $\mathcal{QAP}_n \subseteq conv\ (\mathcal{P}_n \cap \mathbb{Z}^{\mathcal{V}_n} \times \mathbb{Z}^{\mathcal{E}_n})$ holds. Now, we will show that $\mathcal{QAP}_n \supseteq conv\ (\mathcal{P}_n \cap \mathbb{Z}^{\mathcal{V}_n} \times \mathbb{Z}^{\mathcal{E}_n})$. Let $(x,y) \in (\mathcal{P}_n \cap \mathbb{Z}^{\mathcal{V}_n} \times \mathbb{Z}^{\mathcal{E}_n})$. We have already seen that equations of the type (3.1) and (3.2) hold for the $\{0,1\}$-vector (x,y). Also, x is the characteristic vector of an n-clique $C \subseteq \mathcal{V}_n$ of graph $\mathcal{G}_n$ and y is the characteristic vector of some subset $F \subseteq \mathcal{E}_n$ of the edges of $\mathcal{G}_n$. So, it suffices to prove that $F = \mathcal{E}_n(C)$. Consider an arbitrary edge $e = [i,j,k,l] \in F$. Since equations of the type (3.3) hold true, we have

$$\begin{aligned} y((i,\ j) : row_k) - x_{(i,\ j)} &= 0, \text{and} \\ y((k,\ l) : row_i) - x_{(k,\ l)} &= 0. \end{aligned}$$

So, we can conclude that $(i,j), (k,l) \in C$, and thus $F \subseteq \mathcal{E}_n(C)$. On the other hand, add up $(n-1)$ times all the equations of the type (3.1) and (3.2) with all the equations of type (3.3) and (3.4). Then divide the obtained result by 4 to obtain the equation

$$y(\mathcal{E}_n) = \frac{1}{2}n(n-1).$$

This implies $|F| = |\mathcal{E}_n(C)|$, and our assertion follows. □

Definition 3.13. Two polytopes $P \subseteq \mathbb{R}^n$ and $Q \subseteq \mathbb{R}^n$ are *affinely isomorphic* if there is an affine map $\phi : \mathbb{R}^n \to \mathbb{R}^m$ that induces a bijection between points of P and Q, or between the vertices of P and the vertices of Q.

Note that by Theorem 3.8, there exits an affine subspace $\mathcal{A}$ of $\mathbb{R}^{\mathcal{V}_n} \times \mathbb{R}^{\mathcal{E}_n}$ defined by $A^{(n)}(x,y) = b^{(n)}$. In the next section, we will show that with the aid of an orthogonal projection map, an isomorphism (Definition 3.13) can be defined between the $\mathcal{QAP}_n$ polytope and a polytope $\mathcal{QAP}^*_{n^*}$ belonging to a lower dimensional vector space. Further, we will study the polytope $\mathcal{QAP}^*_{n^*}$ to understand the required structural properties of the quadratic assignment polytope.

3.3 Another representation of $\mathcal{QAP}_n$

We define $W^* = row_n \cup col_n$ to be the collection of nodes from the n^{th} row and the n^{th} column of the graph $\mathcal{G}_n$. Let $F^* = \{e \in \mathcal{E}_n : e \cap W^* \neq \phi\}$, $U = \{(x,y) \in \mathbb{R}^{\mathcal{V}_n} \times \mathbb{R}^{\mathcal{E}_n} : x_{W^*} = 0, y_{F^*} = 0\}$, and let $\pi : \mathbb{R}^{\mathcal{V}_n} \times \mathbb{R}^{\mathcal{E}_n} \to U$ be orthogonal projection onto U.

Proposition 3.14. The projection map π restricted to an affine subspace $\mathcal{A}$ of $\mathbb{R}^{\mathcal{V}_n} \times \mathbb{R}^{\mathcal{E}_n}$ is injective.

Proof. First, we will show that we can express the components of points in A belonging to elements in $W^* \cup F^*$ as a linear combination of elements in $\mathcal{V}_n \setminus W^* \cup \mathcal{E}_n \setminus F^*$. By equations (3.1) and (3.2), this is apparent for elements in W^*. To prove this for F^*, it suffices to consider three possibilities for an edge $[i,j,k,l] \in F$. The first two cases are $i,j,k < n$, $l = n$ and $i,j,l < n$, $k = n$. We can use an appropriate equation from (3.3) (with i, j, and k in the first case) and an equation of type (3.4) (with i, j, and l in the second case), to achieve this.

Now the case that remains is, $i, j < n$, $k = n$, and $l = n$. Using equation (3.3) for i, j, and n, we can express $y_{[i,j,n,n]}$ like we have already expressed $y_{[i,j,n,l]}$ for $l < n$. Thus, we have shown that there is a linear function

$$\phi : \mathbb{R}^{\mathcal{V}_n \setminus W^*} \times \mathbb{R}^{\mathcal{E}_n \setminus F^*} \to \mathbb{R}^{W^*} \times \mathbb{R}^{F^*}$$

such that for all $(x, y) \in \mathcal{A}$, we have $(x_{W^*}, y_{F^*}) = \phi(x_{\mathcal{V}_n \setminus W^*}, y_{\mathcal{E}_n \setminus F^*})$. Hence, we can define

$$\Psi : \mathbb{R}^{\mathcal{V}_n} \times \mathbb{R}^{\mathcal{E}_n} \to \mathbb{R}^{\mathcal{V}_n} \times \mathbb{R}^{\mathcal{E}_n}$$

by $\Psi(x, y) = (x', y')$ with

$$\begin{aligned} (x'_{W^*}, y'_{F^*}) &= (x_{W^*}, y_{F^*}) - \phi(x_{\mathcal{V}_n \setminus W^*}, y_{\mathcal{E}_n \setminus F^*}), \text{ and} \\ (x'_{\mathcal{V}_n \setminus W^*}, y'_{\mathcal{E}_n \setminus F^*}) &= (x_{\mathcal{V}_n \setminus W^*}, y_{\mathcal{E}_n \setminus F^*}). \end{aligned}$$

Note that, we get a triangular matrix with the same main diagonal entries as the matrix corresponding to Φ. So, Ψ is an affine transformation of $\mathbb{R}^{\mathcal{V}_n} \times \mathbb{R}^{\mathcal{E}_n}$ that induces on $\mathcal{P}_n$ the orthogonal projection onto U. $\square$

We identify the space U with the space $\mathbb{R}^{\mathcal{V}_{n-1}} \times \mathbb{R}^{\mathcal{E}_{n-1}}$. Thus, for $n^* = n - 1$, $\mathcal{QAP}^*_{n^*} = \pi(\mathcal{QAP}_n) \subset \mathbb{R}^{\mathcal{V}_{n^*}} \times \mathbb{R}^{\mathcal{E}_{n^*}}$ is a polytope in $\mathbb{R}^{\mathcal{V}_{n^*}} \times \mathbb{R}^{\mathcal{E}_{n^*}}$ that is isomorphic to $\mathcal{QAP}_n$.

Remark 3.15. The vertices of the polytope $\mathcal{QAP}^*_{n^*}$ are the projections of the vertices of the original polytope on the removal of the last row and the last column of $\mathcal{G}_n$. They are the the incidence vectors of n^* - and $(n^* - 1)$ - cliques of $\mathcal{G}_{n^*}$, as illustrated in Figure 3.3.

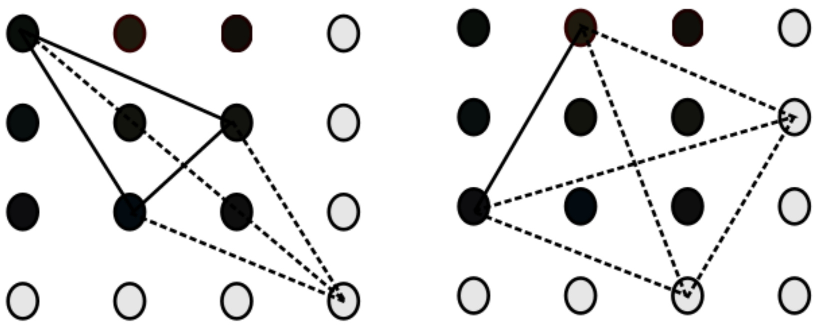

Fig. 3.3: Effect of projection π.

We can mimic the earlier notations for the incidence vectors to (n^*-1)-cliques of $\mathcal{G}_{n^*}$ and define the new polytope as follows.

Definition 3.16. We define

$$\mathcal{QAP}^*_{n^*} = conv\{(x^{C^*}, y^{C^*}) : C^* \text{ is an } n^*\text{- or an } (n^*-1)\text{-clique of } \mathcal{G}_{n^*}\}.$$

We denote by g, the one-to-one map that assigns to every n-clique $C \subset \mathcal{V}_n$ of $\mathcal{G}_n$, the n^*- or (n^* - 1) clique $C^* \subset \mathcal{V}_{n^*}$ of $\mathcal{G}_{n^*}$ that arises from the clique C by removing the node(s) in the n^{th} row and in the n^{th} column.

Remark 3.17. Suppose two faces of $\mathcal{QAP}_n$ and $\mathcal{QAP}^*_{n^*}$ correspond to each other with respect to isomorphism induced by the projection map π. Then the vertices of the polytopes (identified with the cliques) correspond to each other via the bijection g.

As in the case of polytope $\mathcal{QAP}_n$, permutations of the rows (and columns), and transpositions of the node set induce symmetries for polytope $\mathcal{QAP}^*_{n^*}$. Moreover, it is also possible to derive by similar methods, a linear description of a polytope whose integer points form the vertices of $\mathcal{QAP}^*_{n^*}$. We will explore this description in detail, and we will also see some results that will help us compute the dimension of QAP Polytope. Note that, from now on, we will consider the polytope $\mathcal{QAP}^*_n$ instead of $\mathcal{QAP}^*_{n^*}$.

3.4 The Polytope $\mathcal{QAP}_n^*$

We will derive a linearly described polytope $\mathcal{P}_n^* \subseteq \mathbb{R}^{\mathcal{V}_n} \times \mathbb{R}^{\mathcal{E}_n}$ whose integer points are the vertices of $\mathcal{QAP}_n^*$. Let $1 \leq \nu_1,\, \nu_2 \leq n$. Then, for all points $(x, y) \in \mathcal{QAP}_n^*$, we have:

$$x(row_{\nu_1}) + x(row_{\nu_2}) - y(row_{\nu_1} : row_{\nu_2}) = 1, \text{ and}$$
$$x(col_{\nu_1}) + x(col_{\nu_2}) - y(col_{\nu_1} : col_{\nu_2}) = 1.$$

Let us denote this system of equations by $A^{*(n)}(x, y) = b^{*(n)}$. Also, for all $v \in \mathcal{V}_n$, $i \in \{\{1, \ldots, n\} \setminus r(v)\}$, $j \in \{\{1, \ldots, n\} \setminus c(v)\}$, and $(x, y) \in \mathcal{QAP}_n^*$, we have

$$y(v : row_i) - x_v \leq 0, \text{ and}$$
$$y(v : col_j) - x_v \leq 0.$$

Thus, $\mathcal{QAP}_n^*$ is contained in the polytope

$$\mathcal{P}_n^* := \left\{ (x,y) \in \mathbb{R}^{\mathcal{V}_n} \times \mathbb{R}^{\mathcal{E}_n} \;\middle|\; \begin{array}{ll} A^{*(n)}(x,y) = b^{*(n)}, & \\ y(v : row_i) - x_v \leq 0, & (v \in \mathcal{V}_n, i \neq r(v)) \\ y(v : col_j) - x_v \leq 0, & (v \in \mathcal{V}_n, j \neq c(v)) \\ y \geq 0. & \end{array} \right\}.$$

Remark 3.18. For all $(x, y) \in \mathbb{R}^{\mathcal{V}_n} \times \mathbb{R}^{\mathcal{E}_n}$, $(x,\, y) \in \mathcal{P}_n^* \implies x \leq 1$, $x \geq 0$, and $y \leq 1$.

Lemma 3.19. Let $C \subseteq \mathcal{V}_n$ be an arbitrary clique of $\mathcal{G}_n$. If $A^{*(n)}(x^C,\, y^{\mathcal{E}_n(C)})$ $= b^{*(n)}$, then C is of size n or $n - 1$.

Proof. We know that $|C| \leq n$. If $|C| < n-1$, then there exists two rows row_1 and row_2 (assumed without loss of generality) such that $C \cap row_1 = C \cap row_2 = \phi$ contradicting the fact that $x^C(row_1) + x^C(row_2)$ - $y^{\mathcal{E}_n(C)}(row_1 : row_2) = 1$. $\square$

Theorem 3.20. *$\mathcal{QAP}_n^* = conv(\mathcal{P}_n^* \cap (\mathbb{Z}^{\mathcal{V}_n} \times \mathbb{Z}^{\mathcal{E}_n}))$.*

Proof. Clearly, $\mathcal{QAP}_n^* \subseteq conv(\mathcal{P}_n^* \cap (\mathbb{Z}^{\mathcal{V}_n} \times \mathbb{Z}^{\mathcal{E}_n}))$. We need to show that $\mathcal{QAP}_n^* \supseteq conv(\mathcal{P}_n^* \cap (\mathbb{Z}^{\mathcal{V}_n} \times \mathbb{Z}^{\mathcal{E}_n}))$. By Remark 3.18, we know that (x, y) is a $\{0, 1\}$-vector. So, $x \in \mathbb{R}^{\mathcal{V}_n}$ is a characteristic vector of some node set C, and $y \in \mathbb{R}^{\mathcal{E}_n}$ is a characteristic vector of some edge set F. Since $x(row_1)$ + $x(row_2)$ - $y(row_1 : row_2) = 1$ and $y(v : row_1) - x_v \leq 0$ for all $v \in row_2$, we can conclude that $x(row_1) \leq 1$. A similar argument holds for all other rows and columns, and thus, C must be a clique of $\mathcal{G}_n$. Hence, by Lemma 3.19, it suffices to show that $F = \mathcal{E}_n(C)$.

Let $\{v, w\} \in F$, then $y(v : row_{r(w)}) - x_v \leq 0$ implies that $v \in C$. Similarly, $y(w : row_{r(v)}) - x_w \leq 0$ implies that $w \in C$. This shows that $F \subseteq \mathcal{E}_n(C)$. Also, observe that for any $\{v, w\} \in \mathcal{E}_n(C)$, we have $x(row_{r(v)}) + x(row_{r(w)}) - y(row_{r(v)} : row_{r(w)}) = 1$, so there exists an edge in $F \cap (row_{r(v)} : row_{r(w)})$. This implies that $|\mathcal{E}_n(C)| \leq |F|$, and our assertion follows. □

Next, we will study the system $A^{*(n)}(x, y) = b^{*(n)}$ describing an affine space

$$\mathcal{A}^* := \{(x, y) \in \mathbb{R}^{\mathcal{V}_n} \times \mathbb{R}^{\mathcal{E}_n} | A^{*(n)}(x, y) = b^{*(n)}\}$$

that contains $\mathcal{QAP}_n^*$. The system $A^{*(n)}(x, y) = b^{*(n)}$ has equations of the form

$$x(row_i) + x(row_k) - y(row_i : row_k) = 1 \qquad (1 \leq i < k \leq n) \qquad (3.5)$$

and

$$x(col_j) + x(col_l) - y(col_j : col_l) = 1. \qquad (1 \leq j < l \leq n) \qquad (3.6)$$

There are in total $n(n - 1)$ equations in the system. We will now try to explore the rank of the matrix A^* as it would help us prove our main result concerning the dimension of the QAP polytope.

We will work with the "*y-part*" of the matrix A^*, which we will call $M := A^{*(n)}_{\circ,\mathcal{E}_n}$. Let us define a total ordering of edges $\mathcal{E}_n$ by requiring that each edge

$[i,j,k,l] \in \mathcal{E}_n$ with $i < k$, $j < l$ has the successor edge $[i,l,k,j]$. The edges are ordered lexicographically according to the 4-tuples (i,k,j,l). On permuting the columns of M with respect to the above ordering of edges, we get a $n(n-1) \times |\mathcal{E}_n|$ matrix. For example, for $n = 3$, we get the matrix

$$\left(\begin{array}{cccccccccccccccccc}
1&1&1&1&1&1&&&&&&&&&&&& \\
&&&&&&1&1&1&1&1&1&&&&&& \\
&&&&&&&&&&&&1&1&1&1&1&1 \\
1&1&&&&&1&1&&&&&1&1&&&& \\
&&1&1&&&&&1&1&&&&&1&1&& \\
&&&&1&1&&&&&1&1&&&&&1&1
\end{array}\right).$$

We will now focus on exploring the bases of the matrix M. By bases, we mean the maximal subsets of linearly independent columns of M. Note that as the columns corresponding to edges $[i,j,k,l]$ and $[i,l,k,j]$ are identical, we may identify them as one for our ease of working. The resulting $n(n-1)\times\frac{1}{2}|\mathcal{E}_n|$ matrix, we denote it by M'. The matrix M' resembles the node-edge incidence matrix of the complete bipartite graph $K_{\frac{n(n-1)}{2},\frac{n(n-1)}{2}} = (\mathcal{R}_n \cup \mathcal{C}_n, (\mathcal{R}_n : \mathcal{C}_n))$, where $\mathcal{R}_n \cap \mathcal{C}_n = \phi$. We may refer to the pair of edges in $\mathcal{E}_n$ of the form $\{[i,j,k,l],[i,l,k,j]\}$ as a *crossing pair*. We identify the "left shore" $\mathcal{R}_n$ of the complete bipartite graph $K_{\frac{n(n-1)}{2},\frac{n(n-1)}{2}}$ with the set of all possible pairs of rows of $\mathcal{V}_n$, and the "right shore" $\mathcal{C}_n$ with the set of all possible pairs of columns of $\mathcal{V}_n$. With this interpretation, the edges $(\mathcal{R}_n : \mathcal{C}_n)$ are in one-to-one correspondence with the set of crossing pairs of edge set $\mathcal{E}_n$.

Balinski and Russakoff [1], showed that the set of bases of the node-edge-incidence matrix of the complete bipartite graph $K_{r,r}$ is the set of node-edge-incidence matrices of spanning trees (connected acyclic subgraph having all the vertices of the orginal graph) of $K_{r,r}$. Thus, the following proposition concerning the bases of the matrix $A^{*(n)}$ follows as a consequence of our discussion.

Proposition 3.21. 1. Exactly one arbitrary equation from the set of equations $A^{*(n)}(x,y) = b^{*(n)}$ is redundant, that is $rank(A^*) = n(n -$

$1) - 1$.

2. A subset $B \subseteq \mathcal{E}_n$ of edges of $\mathcal{G}_n$ corresponds to a basis of the matrix $A^{*(n)}$ if and only if:

   (a) $|B| = n(n-1) - 1$,

   (b) there exists no crossing pair in B, and

   (c) there exists no sequence $(x_0, y_0, x_1, y_1, \ldots, \mathrm{x}_{k-1}, y_{k-1})$ $(k \geq 2)$ of edges in B such that x_i and y_i connect the same rows of $\mathcal{V}_n$ and x_i and $y_{(i+1) \ mod \ k}$ connect the same columns of $\mathcal{V}_n$ for all $0 \leq i \leq k-1$.

We will now shift our focus towards developing the main result of this chapter, which deals with the dimension of the quadratic assignment polytope.

3.5 Dimension of QAP Polytope

In this section, we will discuss the idea of defining the dimension of a polytope in general and we will also establish a result concerning the dimension of the QAP polytope.

Definition 3.22. A finite collection of vectors $x_1, \ldots, x_k \in \mathbb{R}^n$ is *affinely independent* if unique solution to the system $\sum\limits_{i=1}^{k} \lambda_i x_i = 0$, $\sum\limits_{i=1}^{k} \lambda_i = 0$ is $\lambda_1 = \ldots \lambda_k = 0$.

Note that a set of linear independent points is by definition affinely independent.

Definition 3.23. *Dimension* of a set $S \subseteq \mathbb{R}^n$ is one less than the maximum number of affinely independent points in S.

A line segment formed as the affine hull of two affinely independent points has dimension one. On a similar note, by convention, an empty set is said to have dimension -1.

Definition 3.24. For any polytope $P \subset \mathbb{R}^n$, the *dimension* of P is the dimension of its affine hull.

In essence, the dimension of a polytope is the dimension of the smallest euclidean space that contains it.

Remark 3.25. The dimension of an affine space is equal to the dimension of vector space V. Also, from our previous discussion of affine spaces (Example 3.7), a non-empty set $\{x \in \mathbb{R}^n \mid Ax = b\}$, where $A \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$ is an affine space having its dimension given by nullity of A. So, by rank-nullity theorem, dimension of this affine space is equal to $n - rank(A)$.

Theorem 3.26. *For $n \geq 3$,*

$$\dim(\mathcal{QAP}_n) = (n-1)^2 + \tfrac{(n-1)^2(n-2)^2}{2} - ((n-1)(n-2)-1).$$

The following theorem reveals the dimension of $\mathcal{QAP}_n^*$ polytope, which in turn reveals the dimension of the original quadratic assignment polytope $\mathcal{QAP}_n$.

Theorem 3.27. *For $n \geq 2$,*

$$\dim(\mathcal{QAP}_n^*) = n^2 + \tfrac{n^2(n-1)^2}{2} - (n(n-1)-1).$$

Proof. From the symmetries of polytope $\mathcal{QAP}_n^*$, it is clear that we can restrict our proof to the case $e = [n,\, n-1,\, n-1,\, n]$. So, we may consider the face

$$F := \mathcal{QAP}_n^* \cap \{(x,y) \in \mathbb{R}^{\mathcal{V}_n} \times \mathbb{R}^{\mathcal{E}_n} \mid y_{[n,\, n-1,\, n-1,\, n]} = 0\}$$

of $\mathcal{QAP}_n^*$.

Denoting

$$\mathcal{F} := \{(x,y) \in \mathbb{R}^{\mathcal{V}_n} \times \mathbb{R}^{\mathcal{E}_n} \mid A^{*(n)}(x,y) = b^{*(n)},\ y_{[n,\, n-1,\, n-1,\, n]} = 0\},$$

we establish the following.

Claim 1: It suffices to show that $\mathcal{F} = \text{aff }(F)$.
Proof of Claim 1. By Proposition 3.21, $\dim(\mathcal{QAP}_n^*) \leq n^2 + \frac{n^2(n-1)^2}{2} - (n(n-1)-1)$. As F is a proper face of the polytope $\mathcal{QAP}_n^*$, the system $A^{*(n)}(x,y) = b^{*(n)}$ does not imply $y_{[n,\ n-1,\ n-1,\ n]} = 0$. Thus, from Proposition 3.21, we get $\dim(\mathcal{F}) = n^2 + \frac{n^2(n-1)^2}{2} - n(n-1)$. So, if $\mathcal{F} = \text{aff }(F)$, then

$$\begin{aligned} n^2 + \frac{n^2(n-1)^2}{2} - n(n-1) &= \dim(F) \\ &< \dim(\mathcal{QAP}_n^*) \\ &\leq n^2 + \frac{n^2(n-1)^2}{2} - (n(n-1)-1), \end{aligned}$$

which proves the theorem.

Claim 2: For $n \geq 2$, we have $\mathcal{F} = \text{aff }(F)$.
Proof of Claim 2. For simplifying the proof, we will assume that $n \geq 5$. For the cases $n = 2-4$, the claim can be verified to hold true using direct computations. Clearly, $\mathcal{F} \supseteq \text{aff }(F)$. So, it suffices to show that $\dim(\mathcal{F}) \leq \dim(F)$. From the proof of Claim 1, we have $\dim(\mathcal{F}) = \dim(\mathbb{R}^{\mathcal{V}_n} \times \mathbb{R}^{\mathcal{E}_n}) - n(n-1)$. Let $\mathcal{L} := \{(x,y) - (x',y') \mid (x,y), (x',y') \in F\}$. Then, we have $\dim(lin(\mathcal{L})) = \dim(F)$. It remains to produce a set $\mathcal{B} \subseteq \mathbb{R}^{\mathcal{V}_n} \times \mathbb{R}^{\mathcal{E}_n}$ of $n(n-1)$ vectors that satisfy the equation

$$lin(\mathcal{L} \cup \mathcal{B}) = \mathbb{R}^{\mathcal{V}_n} \times \mathbb{R}^{\mathcal{E}_n}, \tag{3.7}$$

for which we will require the following series of technical lemmas. Figure 3.4 illustrates these vectors and a few notations used in the respective proofs.

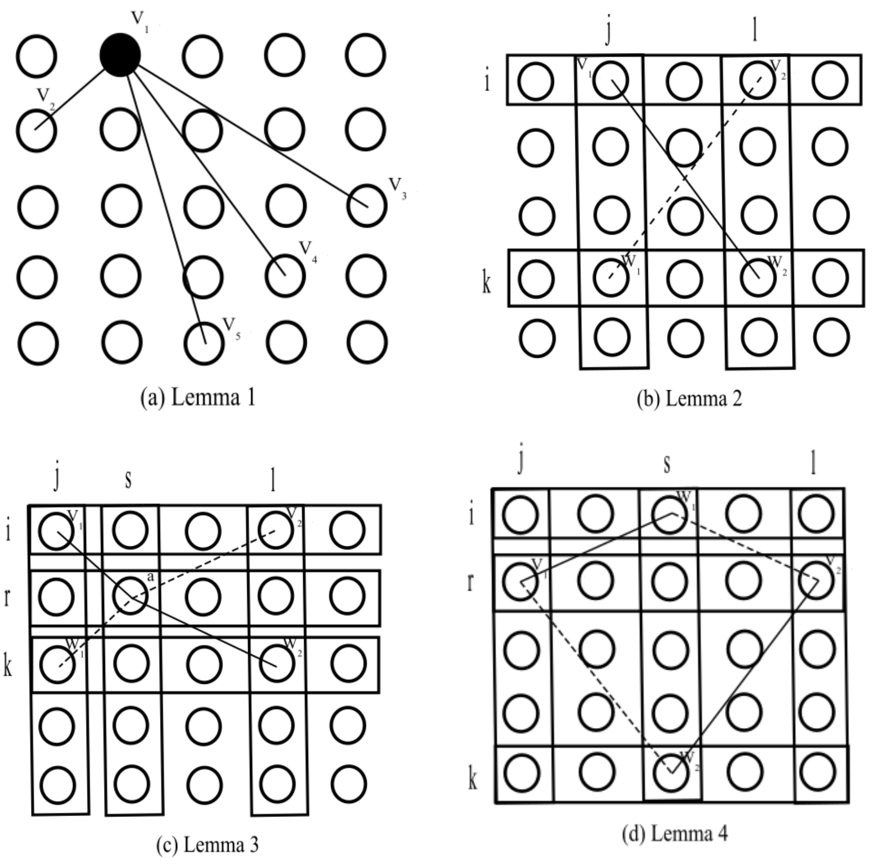

Fig. 3.4: The vectors in Lemmas 1 - 4.

Lemma 1. Suppose $n \geq 2$ and if $v_1, \ldots, v_n \in \mathcal{V}_n$ form an n-clique of $\mathcal{G}_n$ and $[n,\ n-1,\ n-1,\ n]$ is not an edge belonging to that n-clique, then

$$(x^{v_1}, 0) + \sum_{\alpha=2}^{n} (0, y^{\{v_1,\ v_\alpha\}}) \in \mathcal{L}.$$

Proof of Lemma 1. Let $C_1 := \{v_1, \ldots, v_n\}$ and $C_2 := \{v_2, \ldots, v_n\}$. As $(x^{C_1},\ y^{\mathcal{E}_n(C_1)}), (x^{C_2},\ y^{\mathcal{E}_n(C_2)}) \in F$, we can deduce that $(x^{v_1},\ 0)+\sum_{\alpha=2}^{n}(0,\ y^{\{v_1,v_\alpha\}}) = (x^{C_1},\ y^{\mathcal{E}_n(C_1)}) - (x^{C_2},\ y^{\mathcal{E}_n(C_2)}) \in F$.

Lemma 2. For $n \geq 4$, $[i,\ j,\ k,\ l] \in \mathcal{E}_n$ and $[n,\ n-1,\ n-1,\ n] \notin \{[i,\ j,\ k,\ l], [i,\ l,\ k,\ j]\}$, we have

$$(0,\ y^{[i,\ j,\ k,\ l]}) - (0,\ y^{[i,\ l,\ k,\ j]}) \in lin(\mathcal{L}).$$

Proof of Lemma 2. Consider the following vertices of the graph $\mathcal{G}_n$, $v_1 := (i,\ j)$, $v_2 := (i,\ l)$, $w_1 := (k,\ j)$ and $w_2 := (k,\ l)$. Since $n \geq 4$, there exists vertices $u_1, \ldots, u_{n-2} \in \mathcal{V}_n$ such that $C_1 := \{v_1,\ w_2,\ u_1, \ldots, u_{n-2}\}$ and $C_2 := \{v_2,\ w_1,\ u_1, \ldots, u_{n-2}\}$ form n-cliques of $\mathcal{G}_n$. Moreover, $[n,\ n-1,\ n-1,\ n] \notin \mathcal{E}_n(C_\alpha)$ for $\alpha = 1,\ 2$. We define $C_3 := C_1 \setminus \{v_1\}$, $C_4 := C_1 \setminus \{w_2\}$, $C_5 := C_2 \setminus \{v_2\}$, and $C_6 := C_2 \setminus \{w_1\}$. Then, $(x^{C_\alpha},\ y^{\mathcal{E}_n(C_\alpha)}) \in F$ for $1 \leq \alpha \leq 6$. Therefore, $(0,\ y^{[i,\ j,\ k,\ l]}) - (0,\ y^{[i,\ l,\ k,\ j]}) = (x^{C_1},\ y^{\mathcal{E}_n(C_1)}) - (x^{C_2},\ y^{\mathcal{E}_n(C_2)}) - (x^{C_3},\ y^{\mathcal{E}_n(C_3)}) - (x^{C_4},\ y^{\mathcal{E}_n(C_4)}) + (x^{C_5},\ y^{\mathcal{E}_n(C_5)}) + (x^{C_6},\ y^{\mathcal{E}_n(C_6)}) \in lin(\mathcal{L})$.

Lemma 3. Let $n \geq 5$, $1 \leq r,\ i,\ k \leq n$ and $1 \leq s,\ j,\ l \leq n$ be pairwise distinct indices and let $[n,\ n-1,\ n-1,\ n] \notin \{[r,\ s,\ i,\ j], [r,\ s,\ k,\ j], [r,\ s,\ k,\ l], [r,\ s,\ i,\ l]\}$. Then

$$(0,\ y^{[r,\ s,\ i,\ j]}) - (0,\ y^{[r,\ s,\ k,\ j]}) + (0,\ y^{[r,\ s,\ k,\ l]}) - (0,\ y^{[r,\ s,\ i,\ l]}) \in lin(\mathcal{L}).$$

Proof of Lemma 3. We define vertices $a := (r,\ s)$, $v_1 := (i,\ j)$, $v_2 := (i,\ l)$, $w_1 := (k,\ j)$ and $w_2 := (k,\ l)$. We have $n \geq 5$, so there exists vertices $u_1, \ldots, u_{n-3} \in \mathcal{V}_n$ such that $C_1 := \{a,\ v_1,\ w_2,\ u_1, \ldots, u_{n-3}\}$ and $C_2 := \{a,\ v_2,\ w_1,\ u_1, \ldots, u_{n-3}\}$ form n-cliques in $\mathcal{G}_n$. Also, note that $[n,\ n-1,\ n-1,\ n] \notin \mathcal{E}_n(C_\alpha)$ for $\alpha = 1,\ 2$. We define $C_3 := C_1 \setminus \{a\}$ and $C_4 := C_2 \setminus \{a\}$. Then, $(x^{C_\alpha},\ y^{\mathcal{E}_n(C_\alpha)}) \in F$ for $1 \leq \alpha \leq 4$. Therefore, $(0,\ y^{[r,\ s,\ i,\ j]}) - (0,\ y^{[r,\ s,\ k,\ j]}) + (0,\ y^{[r,\ s,\ k,\ l]}) - (0,\ y^{[r,\ s,\ i,\ l]}) = (x^{C_1},\ y^{\mathcal{E}_n(C_1)}) - (x^{C_2},\ y^{\mathcal{E}_n(C_2)}) - (x^{C_3},\ y^{\mathcal{E}_n(C_3)}) + (x^{C_4},\ y^{\mathcal{E}_n(C_4)}) \in lin(\mathcal{L})$.

Lemma 4. Let $n \geq 5$, $1 \leq r,\ i,\ k \leq n$ and $1 \leq s,\ j,\ l \leq n$ are pairwise distinct and $[n,\ n-1,\ n-1,\ n] \notin \{[i,\ s,\ r,\ j], [r,\ j,\ k,\ s], [k,\ s,\ r,\ l], [r,\ l,\ i,\ s]\}$

then

$$(0, y^{[i,\, s,\, r,\, j]}) - (0, y^{[r,\, j,\, k,\, s]}) + (0, y^{[k,\, s,\, r,\, l]}) - (0, y^{[r,\, l,\, i,\, s]}) \in lin(\mathcal{L}).$$

Proof of Lemma 4. Consider the following vertices of graph $\mathcal{G}_n$, $v_1 := (r,\, j)$, $v_2 := (r,\, l)$, $w_1 := (i,\, s)$ and $w_2 := (k,\, s)$. Since $n \geq 5$, there exists vertices $u_1, \ldots, u_{n-3} \in \mathcal{V}_n$, such that $C_1 := \{w_1,\, v_1,\, u_1, \ldots, u_{n-3}\}$, $C_2 := \{v_1,\, w_2,\, u_1, \ldots, u_{n-3}\}$, $C_3 := \{w_2,\, v_2,\, u_1, \ldots, u_{n-3}\}$, and
$C_4 := \{v_2,\, w_1,\, u_1, \ldots, u_{n-3}\}$ form n-cliques of $\mathcal{G}_n$. Also, note that $[n,\, n-1,\, n-1,\, n] \notin \mathcal{E}_n(C_\alpha)$, for $1 \leq \alpha \leq 4$. Therefore, $(0, y^{[i,\, s,\, r,\, j]}) - (0, y^{[r,\, j,\, k,\, s]}) + (0, y^{[k,\, s,\, r,\, l]}) - (0, y^{[r,\, l,\, i,\, s]}) = (x^{C_1},\, y^{\mathcal{E}_n(C_1)}) - (x^{C_2},\, y^{\mathcal{E}_n(C_2)}) + (x^{C_3},\, y^{\mathcal{E}_n(C_3)}) - (x^{C_4},\, y^{\mathcal{E}_n(C_4)}) \in lin(\mathcal{L})$.

Proof of claim 2 (contd.) Let $B := \{[1,\, j,\, 2,\, l] \mid 1 \leq j < l \leq n\} \cup \{[i,\, 1,\, k,\, 2] \mid 1 \leq i < k \leq n\}$ and denote by $\mathcal{B} := \{(0,\, y^e) \mid e \in B \cup [n,\, n-1,\, n-1,\, n]\}$. We will now prove that Equation (3.7) is valid by showing $(0,\, y^e) \in lin(\mathcal{L} \cup \mathcal{B})$ for all $e \in \mathcal{E}_n$ and $(x^v,\, 0) \in lin(\mathcal{L} \cup \mathcal{B})$ for all $v \in \mathcal{V}_n$. We will break this proof into five steps and the result from each step is depicted in Figure 3.5, while Figure 3.6 illustrates the notations used in the proofs of Steps 2-4. For brevity, we will fix the notation $R_\alpha := row_\alpha$ and $C_\alpha := col_\alpha$, for all $1 \leq \alpha \leq n$.

Step 1: $(0,\, y^e) \in lin(\mathcal{L} \cup \mathcal{B})$ for all $e \in (R_1 : R_2) \cup (C_1 : C_2)$.
Proof of Step 1. This step follows from Lemma 2.

Step 2: $(0,\, y^e) \in lin(\mathcal{L} \cup \mathcal{B})$ for all $e \in ((R_1 \cup R_2) : (C_1 \cup C_2))$.
Proof of Step 2. Let $e = [i,\, j,\, k,\, l] \in (R_i : C_l)$ where $i,\, l \in \{1,\, 2\}$. If j or $k \in \{1,\, 2\}$, then by Step 1, we are done. So, we assume $j,\, k \notin \{1,\, 2\}$, and we define $i' := \{1,\, 2\} \setminus \{i\}$ and $l' := \{1,\, 2\} \setminus \{l\}$. By Lemma 4, $(0,\, y^{[i,\, j,\, k,\, l]}) - (0,\, y^{[k,\, l,\, i,\, l']}) + (0,\, y^{[i,\, l',\, i',\, l]}) - (0,\, y^{[i',\, l,\, i,\, j]}) \in lin(\mathcal{L})$. Then, by Step 1, we have $(0,\, y^{[i,\, j,\, k,\, l]}) \in lin(\mathcal{L} \cup \mathcal{B})$.

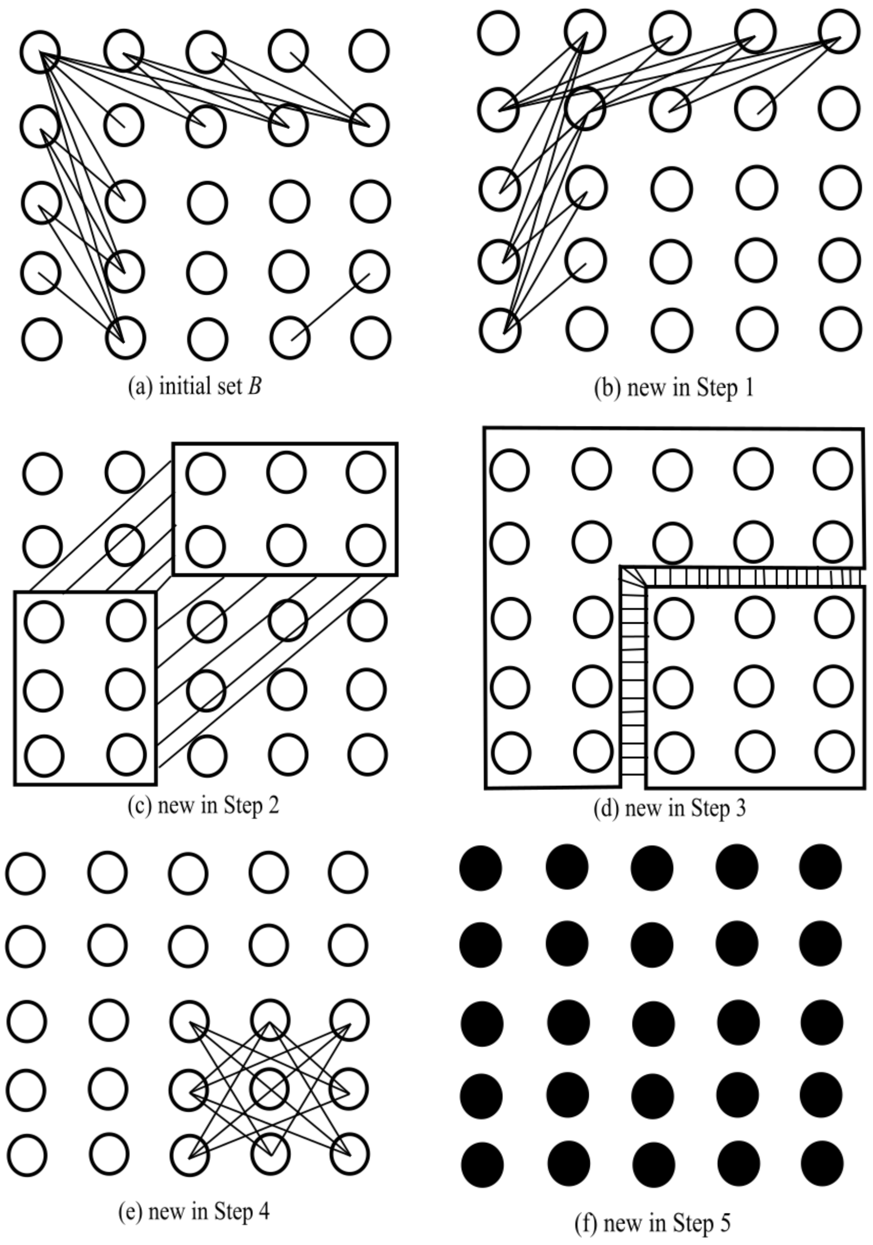

Fig. 3.5: The vectors $y^e \in \mathcal{B}$ and the ones newly shown to be in $\text{lin}(\mathcal{L} \cup \mathcal{B})$ for each of the steps in the proof.

Step 3: $(0,\, y^e) \in lin(\mathcal{L} \cup \mathcal{B})$, for all $e \in \delta(R_1 \cup R_2 \cup C_1 \cup C_2)$.

Proof of Step 3. Let $e = [i,\, j,\, k,\, l] \in \delta(R_1 \cup R_2 \cup C_1 \cup C_2)$ with $i \in \{1,\, 2\}$ and $k,\, l \notin \{1,\, 2\}$ (The case $j \in \{1,\, 2\}$ and $k,\, l \notin \{1,\, 2\}$ can be shown in a similar manner). We define $i' := \{1,\, 2\} \setminus \{i\}$ and $j' := \{1,\, 2\} \setminus \{j\}$. By

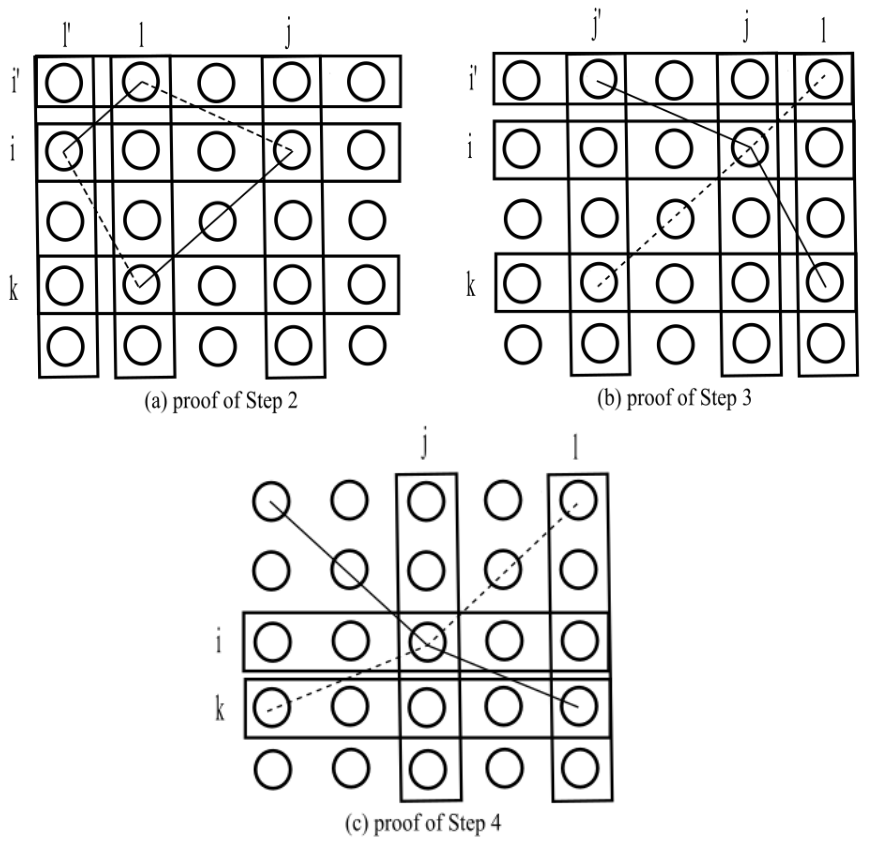

Fig. 3.6: The notations used in the proofs of Steps 2 - 4.

Lemma 3, $y^{[i,\, j,\, k,\, l]} - y^{[i,\, j,\, i',\, l]} + y^{[i,\, j,\, i',\, j']} - y^{[i,\, j,\, k,\, j']} \in lin(\mathcal{L})$. Then, by Steps 1 and 2, we have $y^{[i,\, j,\, k.\, l]} \in lin(\mathcal{L} \cup \mathcal{B})$.

Step 4: $(0,\, y^e) \in lin(\mathcal{L} \cup \mathcal{B})$ for all $e \in \mathcal{E}_n(\mathcal{V}_n \setminus (R_1 \cup R_2 \cup C_1 \cup C_2))$.

Proof of Step 4. Let $e = [i,\, j,\, k,\, l] \in \mathcal{E}_n(\mathcal{V}_n \setminus (R_1 \cup R_2 \cup C_1 \cup C_2))$. If $e = [n,\, n-1,\, n-1,\, n]$, then the claim is trivial. So, we assume $e \neq [n,\, n-1,\, n-1,\, n]$. By Lemma 3, $y^{[i,\, j,\, k,\, l]} - y^{[i,\, j,\, 1,\, l]} + y^{[i,\, j,\, 1,\, 1]} - y^{[i,\, j,\, k,\, 1]} \in lin(\mathcal{L})$. Then, by Step 3, we have $y^{[i,\, j,\, k.\, l]} \in lin(\mathcal{L} \cup \mathcal{B})$.

Step 5: $(x^v,\, 0) \in lin(\mathcal{L} \cup \mathcal{B})$, for all $v \in \mathcal{V}_n$.

Proof of Step 5. From the above four steps, it is clear that $(0,\, y^e) \in lin(\mathcal{L} \cup \mathcal{B})$ for all $e \in \mathcal{E}_n$. The proof for this step follows directly from Lemma 1.

Thus, the proof for claim 2 is complete and our assertion follows. □

3.6 Polytope and its extreme points

Before we conclude the chapter, we will use the concepts discussed in earlier sections of this chapter to provide a detailed proof of Theorem 2.18.

Proof (of Theorem 2.18). Since polyhedra are convex sets, every convex combination of extreme points belongs to the polyhedron. Thus, it suffices to prove the converse, that is, to show that every element of a bounded polyhedron can be represented as a convex combination of the extreme points. By definition, the dimension of P is the smallest integer k such that P is contained in some k-dimensional affine subspace of $\mathbb{R}^n$. We will use induction on the dimension of the polyhedron P.

Suppose that P has dimension equal to 0. Then it has just one point, namely the extreme point and hence the result holds trivially. Let us assume that the result holds true for all polyhedra of dimension less than k. Consider a non-empty and bounded k-dimensional polyhedron $P = \{x \in \mathbb{R}^n : a_i x \geq b_i, i = 1, \ldots, m\}$. Then by definition, P will be contained in a k-dimensional affine subspace S of $\mathbb{R}^n$. Without loss of generality, we can assume $S = \{x_0 + \lambda_1 x_1 + \ldots + \lambda_k x_k \mid \lambda_1, \ldots, \lambda_k \in \mathbb{R}\}$, where $x_1, \ldots, x_k \in \mathbb{R}^n$. Let $h_1, \ldots, h_{n-k}$ be $n-k$ linearly independent vectors that are orthogonal to vectors $x_1, \ldots, x_k$. Consider $f_i = h_i x_0$, for $1 \leq i \leq n-k$. Then, every element x of S satisfies

$$h_i x = f_i, \qquad\qquad i = 1, \ldots, n-k. \tag{3.8}$$

As $P \subset S$, this also holds true for all elements of P.

Consider an element p of P. If p is an extreme point of P, we are done. So, let $p \in P$ be a non-extremal point. Choose an arbitrary extreme point q of P and form a half-line consisting of all points of the form $p + \lambda(p - q)$, where λ is a non-negative scalar. As P is a bounded polyhedron, the half-line would eventually exit P and violate one of the constraints. Let us for instance, assume that constraint to be $a_j x \geq b_j$. By considering what happens when this constraint is just about to be violated, we find some $\lambda^* \geq 0$ and $v \in P$, such that $v = p + \lambda^*(p - q)$ and $a_j v = b_j$. Since the constraint $a_j x \geq b_j$ gets violated if $\lambda > \lambda^*$, it follows that $a_j(p - q) < 0$. Now, consider the polyhedron Q defined by

$$Q = \{x \in P \mid a_j x = b_j\} = \{x \in \mathbb{R}^n \mid a_i x \geq b_i,\ i = 1, \ldots,\ m,\ a_j x = b_j\}.$$

As $p, q \in P$, we have $h_i p = f_i h_i q$. This shows that $p - q$ is orthogonal to each vector h_i, for $i = 1, \ldots,\ n - k$. On the other hand, we have already shown that $a_j(p - q) < 0$, which shows that vector a_j is linearly independent from vectors h_i. Furthermore, we have equation (3.8) valid for all elements of P, and hence $Q \subset \{x \in \mathbb{R}^n \mid a_j x = b_j,\ h_i x = f_i,\ i = 1, \ldots,\ n - k\}$. The set on the right is defined by $n - k + 1$ linearly independent equality constraints. Hence, it is an affine subspace of dimension $k - 1$ and Q has dimension at most $k - 1$.

Applying the induction hypothesis to Q and v, we observe that v can be expressed as a convex combination

$$v = \sum_i \lambda_i u_i$$

of the extreme points u_i of Q, where λ_i are nonnegative scalars that sum up to one. Note that at any extreme point u of Q, the equation set $a_i u = b_i$ is valid for n linearly independent vectors a_i. So, u must also be an extreme point of P. From the definition of λ^*, we also have

$$p = \frac{v + \lambda^* q}{1 + \lambda^*}.$$

Therefore,

$$p = \frac{\lambda^* q}{1+\lambda^*} + \sum_i \frac{\lambda_i}{1+\lambda^*} u_i,$$

which shows that an arbitrarily chosen $p \in P$ can be expressed as the convex combination of extreme points of P. $\square$

4. SYMMETRIC TRAVELING SALESPERSON POLYTOPE

We will begin this chapter by introducing the symmetric traveling salesperson polytope. Following this, we will see the relationship between the earlier discussed quadratic assignment polytope and the symmetric traveling salesperson polytope. Finally, we would conclude our discussion on the symmetric traveling salesperson polytope by exploring its dimension. For this chapter, we have used the references [10, 13, 15].

Let H_n represent the set of all hamiltonian cycles of the complete graph $K_n = (V, E)$. For every hamiltonian cycle, $\Gamma \in H_n$, we associate a vector $x^{\Gamma} \in \mathbb{R}^E$ such that, $x_e^{\Gamma} = 1$ if $e \in \Gamma$ and 0 otherwise. This is the *incidence* or *characteristic vector*. Now, we can formally define the symmetric traveling salesperson polytope.

Definition 4.1. The symmetric TSP polytope, $\mathcal{Q}_n$ is defined as

$$\mathcal{Q}_n := conv\Big(\{x^{\Gamma} : \Gamma \in H_n\}\Big).$$

4.1 TSP Polytope as a projection of QAP Polytope

The necessary framework to identify polytopes of various combinatorial optimization problems like TSP as a projection of $\mathcal{QAP}_n$ will be introduced in this section. Consider a complete undirected graph $G = (V, E)$ on n nodes and let $A_0 \subseteq E$ be a subset of edges. Denote by Π_V, the set of all permutations of the node set V. Any permutation $\sigma \in \Pi_V$ of the nodes induces a map $\hat{\sigma} : E \to E$ by $\hat{\sigma}(\{v, w\}) = \{\sigma(v), \sigma(w)\}$.

Now, consider the combinatorial optimization problems with the set of feasible solutions $\Omega = \{\hat{\sigma}(A_0) : \sigma \in \Pi_V\}$. Denote the incidence vectors associated with the feasible solutions by

$$\Omega = \{\hat{\sigma}(A_0) : \sigma \in \Pi_V\},$$

and define the polytope

$$\mathcal{P}_\Omega = conv\Big(\{Y^A : A \in \Omega\}\Big).$$

Note that the symmetric TSP polytope $\mathcal{Q}_n$ by definition, is a polytope of the form $\mathcal{P}_\Omega$. Thus, the results we prove for the polytope $\mathcal{P}_\Omega$ applies automatically for the symmetric TSP polytope, $\mathcal{Q}_n$.

Theorem 4.2. *Every $\mathcal{P}_\Omega$ defined as above is a projection of the QAP Polytope ($\mathcal{QAP}_n$).*

Proof. In order to simplify the notations, assume without loss of generality, $V = \{1, \ldots, n\}$. Define a projection map $\pi_{A_0} : \mathbb{R}^{\mathcal{V}_n} \times \mathbb{R}^{\mathcal{E}_n} \to \mathbb{R}^E$ by assigning $\pi_{A_0}(x, y) = Y$ with

$$Y_{\{i,\, k\}} = \sum_{\{j,\, l\} \in A_0} (y_{[i,\, j,\, k,\, l]} + y_{[i,\, l,\, k,\, j]}), \qquad (a)$$

where $\{i,\, k\} \in E$ and this holds $\forall\, (x,\, y) \in \mathbb{R}^{\mathcal{V}_n} \times \mathbb{R}^{\mathcal{E}_n}$. It suffices to show that for any permutation σ of V and for vertex $(x,\, y)$ of $\mathcal{QAP}_n$ that belongs to the inverse permutation σ^{-1} of σ the equation

$$\pi_{A_0}(x, y) = Y^{\hat{\sigma}(A_0)} \qquad (b)$$

holds true. Thus, π_{A_0} induces a surjective map between the vertices of $\mathcal{QAP}_n$ and those of $\mathcal{P}_\Omega$. Consider any edge $\{i,\, k\} \in E$ of G. The vector $(x,\, y)$ is the incidence vector belonging to the n-clique that corresponds to σ^{-1}, and so we can conclude $y_{[i,\, \sigma^{-1}(i),\, k,\, \sigma^{-1}(k)]} = 1$. Note that $\{\sigma^{-1}(i),\, \sigma^{-1}(k)\} \in A_0$

if and only if $\{i,\, k\} \in \hat{\sigma}(A_0)$, $y_{[i,\, \sigma^{-1}(i),\, k,\, \sigma^{-1}(k)]}$ occurs as a summand in (a), which holds if and only if $i,\, k \in \hat{\sigma}(A_0)$. As $j \neq \sigma^{-1}(i)$ or $l \neq \sigma^{-1}(k)$, the term $y_{[i,\, j,\, k,\, l]} = 0$ for any edge $[i,\, j,\, k,\, l] \in \mathcal{E}_n$. Hence, the polytope $\mathcal{P}_\Omega$ is a projection of the QAP Polytope. □

Corollary 4.3. The Symmetric TSP polytope, $\mathcal{Q}_n$ is a projection of QAP Polytope, $\mathcal{QAP}_n$.

By now, we have seen that traveling salesperson polytope can be realised as projection of the quadratic assignment polytope. We will now explore the dimension of traveling salesperson polytope. For this purpose, we need to get familiarised with a few important terminologies and results from graph theory concerning the factorization of graphs.

4.2 Factorization of Graphs

A problem that occurs in many contexts is determining whether a given graph can be decomposed into finitely many line-disjoint spanning subgraphs having some prescribed property. Generally, we will be interested in the property of regularity of some specified degree. Before discussing this idea of factorization of graphs, let us see some basic definitions from graph theory.

Definition 4.4. A subgraph having all the vertices of the original graph is called a *spanning subgraph.*

Definition 4.5. A graph with empty edge set is called *null graph* or *totally disconnected graph.*

Now, we will see the definition for factor of a graph.

Definition 4.6. A spanning subgraph of graph G which is not totally disconnected is called a *factor* of graph G.

Definition 4.7. If the graph G is the line-disjoint union of its factors G_i, then G is termed as *sum* of its factors G_i and such a union is called a *factorization* of G.

Definition 4.8. A *n-factor* is a regular graph of degree n. If a graph G is a sum of n-factors, then their union is termed as *n-factorization.* Such a graph G can be called a *n-factorable* graph.

In Figure 4.1, the 1-factorization of the complete graph K_6 is depicted.

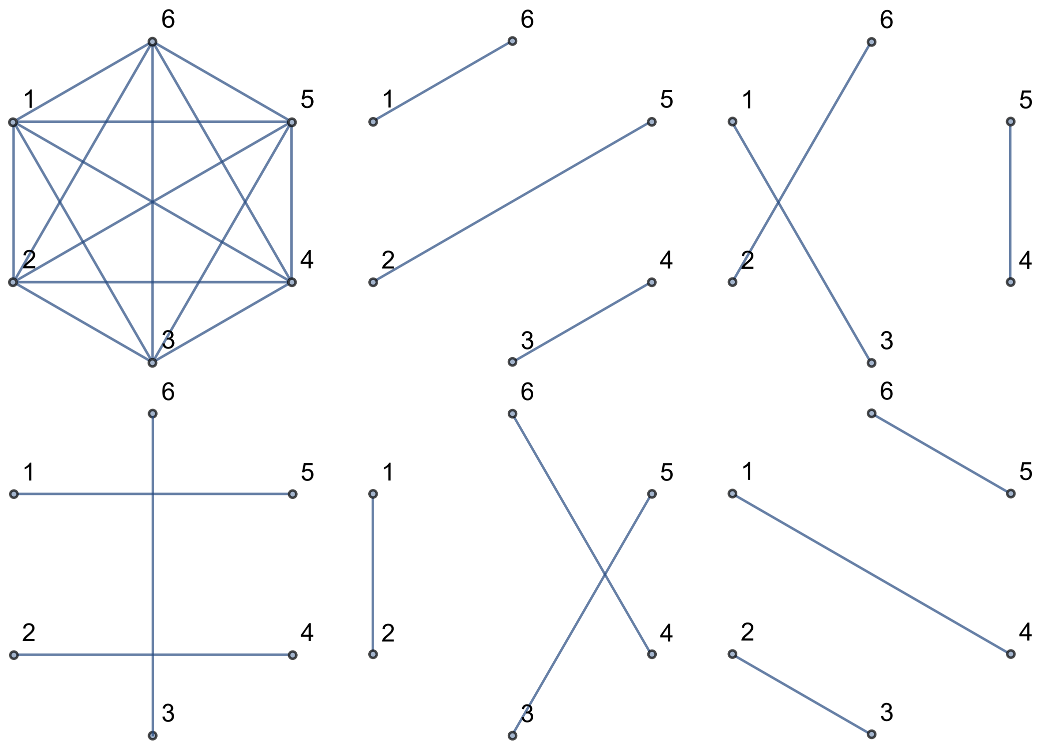

Fig. 4.1: A 1-factorization of K_6.

Note that we will use terms spanning cycle, hamiltonian cycle, and tour, interchangeably, in this chapter. All of these terminologies will refer to the same entity, unless specified otherwise.

Theorem 4.9. *The complete graph K_{2n+1} can be expressed as the sum of n spanning cycles.*

Proof. The aim is to construct n line-disjoint spanning cycles in the complete graph K_{2n+1}. First we label the vertices $v_1, \ldots, v_{2n+1}$ and construct n paths

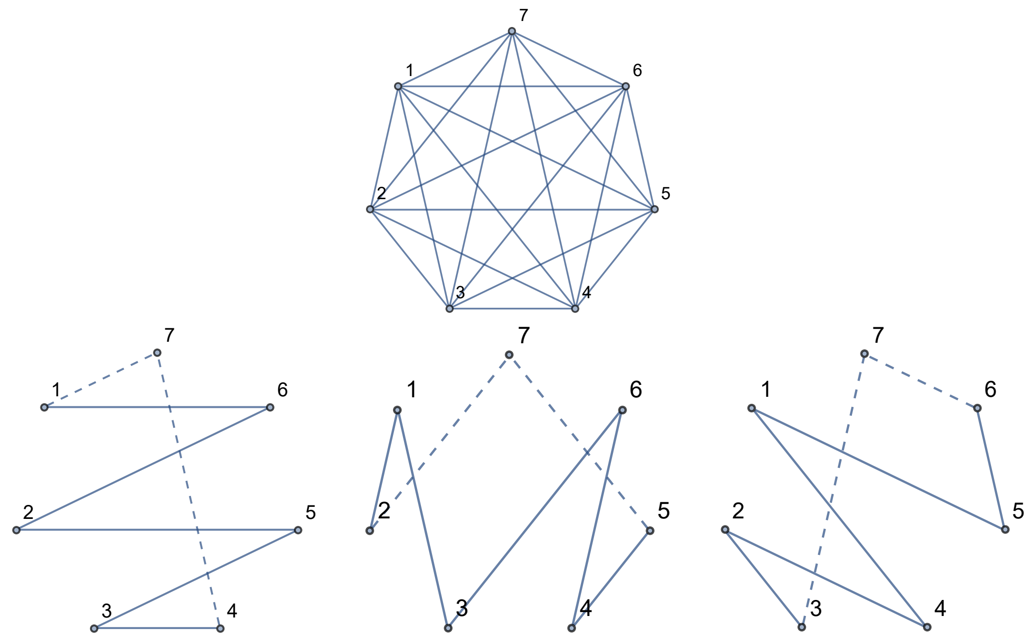

Fig. 4.2: K_7 as the sum of three spanning cycles Z_1, Z_2, and Z_3.

P_i through these vertices as follows:

$$P_i = v_i\, v_{i-1}\, v_{i+1}\, v_{i-2} \ldots v_{i+n-1}\, v_{i-n}.$$

Note that, the j^{th} vertex in the path P_i is v_k, where $k = i + (-1)^{j+1}[j/2]$, and all subscripts are taken as the integers $1, 2, \ldots, 2n \pmod{2n}$. Finally, construct the spanning cycle Z_i by joining the vertex v_{2n+1} to the endpoints of P_i. $\square$

The construction described in the proof of Theorem 4.9 is illustrated in Figure 4.2 for the graph K_7. The lines of the paths P_i are solid and the two added lines are dashed.

Theorem 4.10. *The complete graph K_{2n} can be expressed as the sum of a 1-factor and n-1 spanning cycles.*

An illustration of Theorem 4.10 is shown in Figure 4.3.

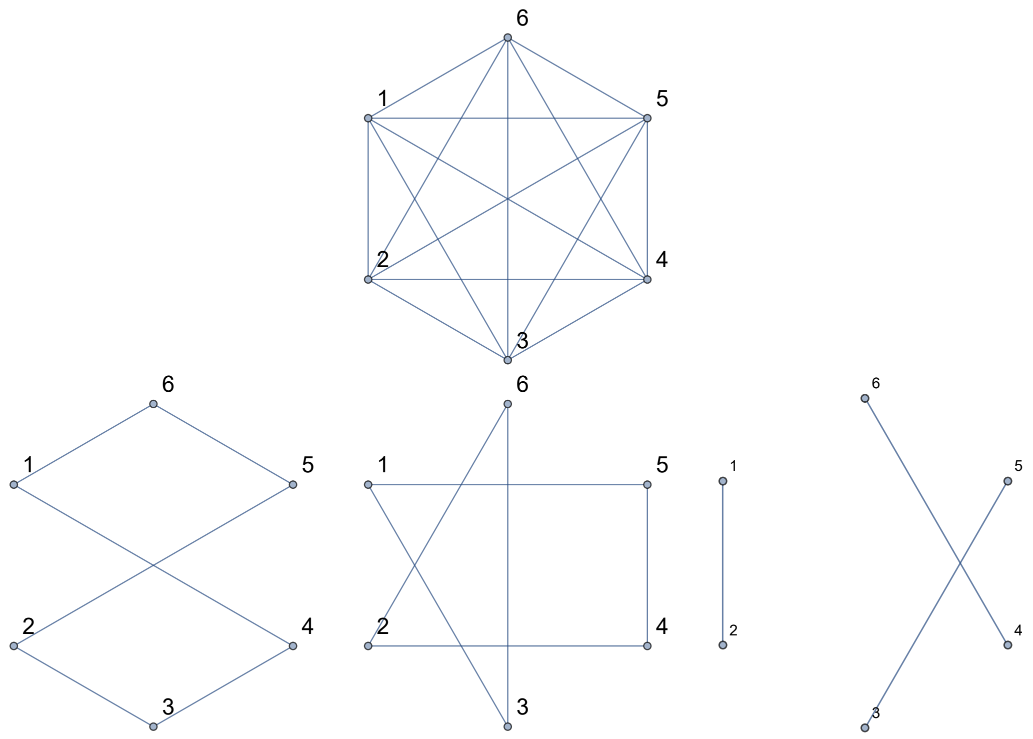

Fig. 4.3: K_6 as the sum of two spanning cycles Z_1, Z_2, and a 1-factor X_1.

4.3 Dimension of Symmetric Traveling Salesperson Polytope

Theorem 4.11. *The dimension of symmetric TSP polytope, $\mathcal{Q}_n$, equals $d_n = \frac{1}{2}n(n-3)$.*

Proof. It suffices to prove that $\mathcal{Q}_n$ has $(d_n + 1)$ affinely independent tours. For $n = 3$, this holds trivially. We will consider the cases when n is odd and n is even separately.

Suppose that $n = 2k + 2$. Then, since G is a complete graph, the induced subgraph of first $n-1$ nodes of G is also a complete graph. Thus, by Theorem 4.9, G can be expressed as the sum of k spanning cycles T_i of length $(n-1)$. From each T_i, construct new $(n-1)$ tours T_{ij} of length n by replacing one at a time, each edge $\{x, y\} \in T_i$ by the chain $[x, n, y]$. Thus, we get a total

of $k(n-1) = d_n + 1$ tours. Observe that the incidence matrix of the tours T_{ij} for $j = 1, \ldots, n-1$ (rows) versus the edges of K_n (columns) contains the submatrix $E - I$, where E is the $(n-1) \times (n-1)$ matrix of all ones and I is the $(n-1) \times (n-1)$ identity matrix. Also, note that the incidence matrix of $d_n + 1$ tours T_{ij} contains a $(d_n+1) \times (d_n+1)$ submatrix M, which is block-diagonal with its diagonal-blocks being equal to $E - I$ after some permutation of the rows and columns. Since, $E - I$ is nonsingular, it clearly follows that N is also a nonsingular matrix. So, there exist $(d_n + 1)$ linearly independent tours. Therefore, by definition, the dimension of the polytope Q_n is d_n, and hence the theorem holds when n is even.
Now, let us consider the case when $n = 2k + 1$ for $k \geq 2$. We proceed as in the first case and construct $(k-1)(n-1)$ linearly independent tours from the $k - 1$ tours of length $n - 1$. The 1-factor of K_{n-1} (by Theorem 4.10) can be completed arbitrarily to a $(n-1)$ tour of graph K_{n-1}, which in turn can be used to construct k tours of length n by replacing each edge $\{x, y\}$ in 1-factor by the chain $[x, n, y]$. Thus, we obtain a total of $d_n + 1$ tours whose incidence matrix contains a $(d_n + 1) \times (d_n + 1)$ block triangular matrix M with $k - 1$ blocks $E - I$ of size $(n-1) \times (n-1)$ and an additional block of similar type but of size $k \times k$. By similar arguments as in the first case, the assertion follows for this case. $\square$

5. USING THE METHOD OF GRAPH NEURAL NETWORKS TO SOLVE DECISION TSP

5.1 Introduction

The advent of neural networks has pushed research on pattern recognition and data mining. Many machine learning tasks such as face recognition, object detection, machine translation has been revolutionised by various end-to-end deep learning paradigms, e.g., convolutional neural networks (CNNs), recurrent neural networks (RNNs), etc. The success of deep learning in many domains can be attributed to the rapid progress in computational resources (e.g., GPU), the availability of big training data, and the effectiveness of the deep learning model to extract representations from various kinds of data like images, text, videos etc. A core assumption of these machine learning algorithms is that the instances are independent of each other. But, this assumption no longer holds for graph data because each instance (node) is related to others by links of various types. Let us discuss a few real-life instances where graph data predominates, thereby motivating the need to develop a neural network model that can handle graph data.

In e-commence, a graph-based learning system can exploit the interactions between users and products to make accurate recommendations. In chemistry, molecules can be modeled as graphs, and their bioactivity can be identified for drug discovery. In a citation network, papers are linked to each other via citationships, and the model can be trained to perform the task of categorising. Likewise, there are many scenarios where data represented in graph format is the most convenient and most appropriate mode of representation for learning tasks and problem-solving. This chapter is based on [2, 20, 21, 23].

5.2 Message Passing in Graphs

Consider a graph $G = (V, E)$ on which a certain computational experiment has to be performed upon. The basic idea of message passing in graphs can be thought of as a algorithm in which vertices of the graph communicate with one another to learn about their neighborhood. See the illustration in Figure 5.1 to get an intuitive understanding of message passing in graphs.

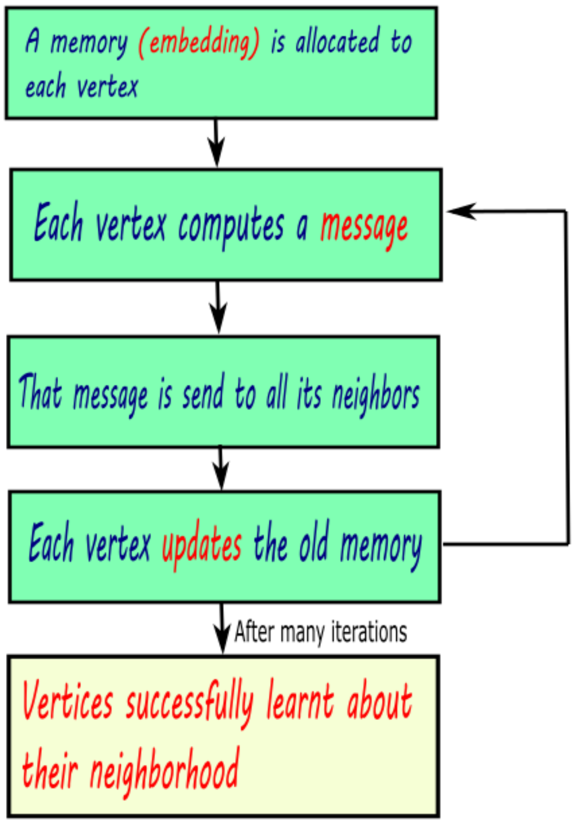
A memory (embedding) is allocated to each vertex
Each vertex computes a message
That message is send to all its neighbors
Each vertex updates the old memory
After many iterations
Vertices successfully learnt about their neighborhood

Fig. 5.1: Message passing in graphs.

To facilitate message passage, a “memory” (a vector embedding to a lower dimensional space) $x_i \in \mathbb{R}^d$ needs to be allocated for each node v_i (which can be initialized randomly or with zeros). In simple words, v_i can compute a “message” $msg(x_i)$ to send to each of its neighbors. Upon receiving a set of messages $X = \{msg(x_j) : v_j \in \mathcal{N}(v_i)\}$ from its neighbors v_j, each node v_i can update its previous memory through some “update” function, $x_i' \leftarrow update(x_i, X)$.

The motivation behind the GNN architecture is that suppose the functions $msg()$ and $update()$ can be well-trained, then after a finite number of iterations, each node will have in its memory, the information regarding its neighborhood. For this purpose, we model the function $msg()$: $\mathbb{R}^d \rightarrow \mathbb{R}^d$ as a Multi-Layer Perceptron (MLP) and the function $update()$: $\mathbb{R}^{2d} \rightarrow \mathbb{R}^d$ as an LSTM.

5.3 Graph Neural Network - A General Architecture

For a given graph $G = (V, E)$:

1. Assign a multidimensional vector (embedding) $x_i \in \mathbb{R}^d$ to each vertex $v_i \in V$ and collect all those vectors a matrix $X \in \mathbb{R}^{V \times d}$.

2. Compute a $(|V| \times d)$ matrix of messages $MSGS \leftarrow msg(X)$.

3. Compute the matrix product with G's adjacency matrix: $A \times MSGS$. This yields $(|V| \times d)$ dimensional matrix, where the i-th row is the vector sum of all the messages received by vertex v_i.

4. Now pass it as input for the update function: $update(X, A \times MSGS)$. This yields a $|V| \times d$ matrix where the i-th row is the updated memory vector of vertex v_i.

Thus, the core structure of GNN is a Recurrent Neural Network (RNN). More specifically, it is an LSTM applied over a matrix multiplication between an adjacency matrix and a function applied row-wise on a matrix of embeddings. Furthermore, the whole process iterated multiple times:

$$X^{(t+1)} \leftarrow update(X^{(t)}, A \times msg(X^{(t)})).$$

Remark 5.1. In order to evaluate a decision problem on graphs, we associate the matrix X to a scalar quantity, say mean(X) and perform the gradient descent on $loss = (\mathcal{Y} - \text{mean}(X))^2$. Then the GNN model learns to solve the decision problem on graphs, given enough training examples. The input for each problem is an adjacency matrix $A \in \mathbb{R}^{|V| \times |V|}$ and the output $\mathcal{Y}$ is a boolean value (0 or 1).

We will now focus on our main goal of solving the decision variant of TSP using GNN. Given a weighted graph $G = (V, E)$ and a number $C \in \mathbb{R}$, we saw that the decision TSP problem asks whether G admits a hamiltonian route with cost no larger than C. For this problem, in addition to the relational (graph) data as in the case of GNN architecture, we also need to assign weights (euclidean distance between the adjacent vertices) to the edges. So, in addition to the idea of vertex embedding, we need to introduce the idea of edge embeddings (a "memory" for edges).

5.4 Algorithm

The idea of the algorithm used for GNN model development can be summarised as follows:

Step 1: Vertices send messages to edges of which they are endpoints (i.e. vertex v sends messages to $\forall \{v_1, v_2\} : v = v_1 \vee v = v_2$).
Step 2: Each edge $\{v_1, v_2\}$ sends a message to vertices v_1 and v_2.
Step 3: Repeat Steps 1 - 2 for some finite number of message-passing iterations, until each edge has in its memory, the sufficient numerical and relational information relevant to the TSP problem.
Step 4: This information received by each of the edges will be associated to a scalar value and will be considered as that edge's "vote" (i.e. the probability with which it thinks that a route actually exists).

Step 5: Finally, the votes from all edges will be averaged to render the final prediction.

The GNN algorithm for decision TSP problem can be formally stated as follows:

Algorithm 1 Graph Neural Network TSP-Decide Solver

```
1:  procedure GNN–TSP (G = (V, E), C)
2:
3:      // Compute binary adjacency matrix from from edges to source and target vertices.
4:      $\mathbf{EV}[i,\ j] \leftarrow 1$ if and only if $(\exists\, v' \mid e_i = (v_j,\ v',\ w)) \mid \forall e_i \in E, v_j \in V$
5:
6:      // Compute initial edge embeddings.
7:      $\mathbf{E}^{(1)}[\mathrm{i}] \leftarrow E_{init}(w,\ C) \mid\ \forall e_i = (s, t, w) \in E$
8:
9:      // Run $t_{max}$ message-passing iterations.
10:     for t = $1, \ldots,\ t_{max}$ do
11:         // Refine each vertex embedding with messages received from edges in which it appears either as a source or a target vertex.
12:         $\mathbf{V}_h^{(t+1)},\ \mathbf{V}^{(t+1)} \leftarrow V_u(\mathbf{V}_h^{(t)},\ \mathbf{EV}^T \times E_{msg}(\mathbf{E}^{(t)}))$
13:         // Refine each edge embedding with messages received from its source and its target vertex.
14:         $\mathbf{E}_h^{(t+1)},\ \mathbf{E}^{(t+1)} \leftarrow E_u(\mathbf{E}_h^{(t)},\ \mathbf{EV} \times V_{msg}(\mathbf{V}^{(t)}))$
15:     // Translate edge embeddings into logit probabilities.
16:     $\mathbf{E}_{logits} \leftarrow E_{vote}(\mathbf{E}^{(t_{max})})$
17:     // Average logits and translate to probability (the operator <> indicates the arithmetic mean).
18:     prediction $\leftarrow$ sigmoid $(< \mathbf{E}_{logits} >)$
```

Upon training, the model learns:

1. A single $\mathbb{R}^d$ vector to be used as the initial vertex embedding for all vertices.
2. A function $E_{init} : \mathbb{R}^2 \to \mathbb{R}^d$ to compute an initial edge embedding for each edge given an edge weight w and a target cost C (MLP).
3. A function $V_{msg} : \mathbb{R}^d \to \mathbb{R}^d$ to compute messages to send from vertices to the edges of which they are endpoints (MLP).

4. A function $E_{msg} : \mathbb{R}^d \to \mathbb{R}^d$ to compute messages to send from each edge to the respective vertices connected to them (MLP).

5. A function $V_u : \mathbb{R}^{2d} \to \mathbb{R}^{2d}$ to compute an updated vertex embedding (plus an updated RNN hidden state) given the current RNN hidden state and a message (LSTM).

6. A function $E_u : \mathbb{R}^{2d} \to \mathbb{R}^{2d}$ to compute an updated edge embedding (plus an updated RNN hidden state) given the current RNN hidden state and a message (LSTM).

7. A function $E_{vote} : \mathbb{R}^d \to \mathbb{R}$ to compute a logit probability for predicting the existence of a route given an edge embedding (MLP).

5.5 Training the GNN model

The input for our GNN model comprises adjacency matrices from edge to source, target vertices (incidence matrices) $S, T \in \{0,1\}^{|E|\times|V|}$, the edge weight matrix D, and a target cost $C \in \mathbb{R}$. Since we are working on a decision problem, the actual output for any instance is **YES (1)** or **NO (0)**. Hence, it is a binary classification type problem. So, we will be using *binary-cross entropy function* as the cost function (error function). Before, we introduce the binary cross entropy function, let us understand the term *entropy.* Generally, we will use the term entropy to indicate disorder or uncertainty.

Definition 5.2. For a random variable X with probability distribution $p(X)$, *entropy* is defined as follows:

$$S := \begin{cases} \text{-}\int p(x)\ logp(x)\ dx, & \text{if } x \text{ is continuous, and} \\ \text{-}\sum_x p(x) logp(x) dx, & \text{if } x \text{ is discrete.} \end{cases}$$

Definition 5.3. The *binary cross entropy function* for output label y (y can take values 0 and 1) and predicted probability p is defined as follows:

$$L = -y\, log(p) - (1-y)\, log(1-p).$$

To minimize the binary-cross entropy error between model prediction and the ground truth, the model was trained with Stochastic Gradient Descent (SGD). Further, to speed up the training, SGD was performed on batches with multiple instances. For this purpose, a batch graph was generated by performing a disjoint union of all graphs in the batch. As subgraphs of the batch graph are disjoint, messages will not traverse between any pair of them, and there will be no change to the embedding refinement process when compared to a single run. A logit probability was computed for each edge in the batch graph, which was averaged among individual instances to compute a prediction for each one of them. Following this, the binary-cross entropy error was computed between model prediction and actual solution.

To train the model, training samples were created by sampling $n \sim \mathcal{U}(20, 40)$ random points on a $\frac{\sqrt{2}}{2} \times \frac{\sqrt{2}}{2}$ square and computing a distance matrix $D \in \mathbb{R}^{n \times n}$ with the euclidean distance computed between each pair of points. These distances, by construction, belong to $[0, 1]$. To obtain the accurate TSP optimal cost, we have taken the aid of Concorde TSP solver [11]. We have generated a total of 2^{20} graphs, from which we had sampled only 1024 graphs per epoch to ensure that the model had minimal chance encounter the same graph twice during training time. The key idea used for model training is as follows, we showed the model very similar instances with opposite answers. For each graph G with optimal tour cost C^* we produced two decision instances $X^+ = (G, 1.02C^*)$ and $X^- = (G, 0.98C^*)$ for which the answers are by construction **YES** and **NO**, respectively. As a result, we trained our model to predict the decision problem within a 2% positive or negative deviation from the optimal tour cost. Upon 2000 training epochs, the model achieved 80.16% accuracy averaged over the 2^{21} instances of the training set, and the model obtained 77% accuracy on a testing set of 2048 instances it had never seen before. The model used 64-dimensional embeddings for vertices as well as edges, and three-layered (64,64,64) MLPs were used with ReLU nonlinearities as the activations for all layers except for the last one, which had a linear activation. The edge embedding initialisation MLP was three-layered with layer sizes (8, 16, 32). We ran the GNN model for $t_{max} = 32$ time steps of message-passing.

5.6 Experiments and Discussions

5.6.1 Extracting Route Costs

Once we developed the required model for solving TSP-DECIDE, we tried to exploit the same model to predict TSP route cost within a reasonable margin from the optimal cost. Figure 5.2 shows how the model behaves when asked to solve the decision problem for varying target costs.

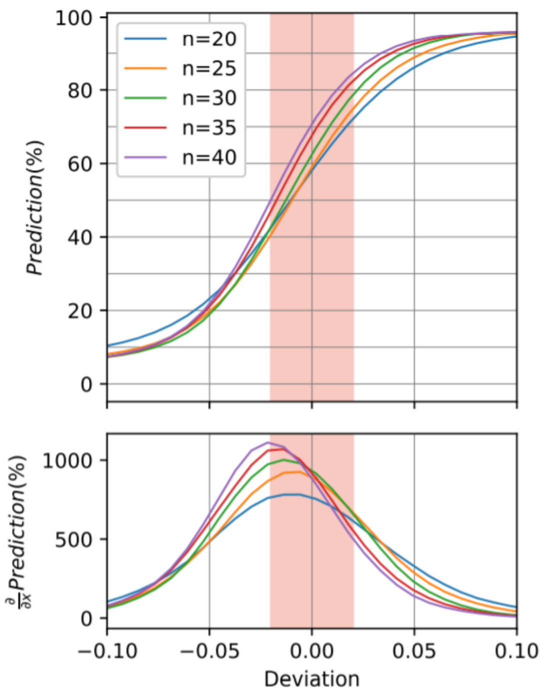

Fig. 5.2: Acceptance curves and its derivatives.
(Figure taken from the original work in [20] for illustration purpose).

The characteristic S shape reveals that the model feels confident that routes with very less target costs do not exist and also confident that routes with very large costs do exist. Between these two regimes, the prediction made by the model underwent phase transition, with the model becoming increasingly unsure as the target cost approach zero deviation from the optimal TSP cost. Note that the prediction for each deviation is averaged over 1024 instances.

Beardwood, Halton, and Hammersley [2] revealed a theoretical result that average TSP tour length for a set of n random (uniform) points on a plane is asymptotically proportional to $\sqrt{n}$. As a corollary, large instances allow for proportionally shorter routes than small instances. Observing Figure 5.2, we can see the theoretical result is also reflected in our experiment. For deviations close to zero, the model feels more confident that a route exists when the instance size is larger. Thus, the critical point (the deviation at which the model starts guessing **YES**) undergoes a left shift as the instance size increases, as seen in the curves' derivatives in Figure 5.2.

Algorithm 2 Binary Search to compute the optimal cost of TSP tour

```
1: procedure BINARY–SEARCH (G = (V, E), $p$, $\delta$)
2:
3:     // Choose an initial guess for the optimal route cost.
   $w^{n-}$ and $w^{n+}$ are the sets of the costs of the $n$ edges with smallest/largest
   costs respectively.
4:
5:     $C_{min} \leftarrow \sum w_i^{n-}$
6:     $C_{max} \leftarrow \sum w_i^{n+}$
7:     $C \sim \mathcal{U}(C_{min}, C_{max})$
8:     while $C_{min} < C(1-\delta) \vee C(1-\delta) < C_{max}$ do
9:         if GNN–TSP(G, C) < $p$ then
10:            $C_{min} \leftarrow C$
11:        else
12:            $C_{max} \leftarrow C$
13:        $C \leftarrow (C_{min} + C_{max})/2$
14:    return C
```

From our above discussion on S shaped acceptance curves, we get an intuitive idea that if we know nothing about the optimal cost, we can assume that we are closest to this value when the model's predictions are closest to 50%. That is, when model prediction is nearing 50%, the optimal cost of TSP lies within close range from the input target cost. Thus, we estimate an initial cost and perform a binary search on the x-axis of Figure 5.2. The detailed procedure is summarised in Algorithm 2.

For the algorithm, we took $\delta = 0.01$, $p = 0.5$ and used the weights after the model training and the single epoch of training was conducted for greater deviations. Algorithm 2 is able to predict TSP route costs with on average 1.5% absolute deviation from the ground truth value after running the algorithm for average 8-9 iterations on the test dataset (1024 n-city graphs with $n \sim \mathcal{U}(20, 40)$).

5.6.2 Model Performance on Larger Instances

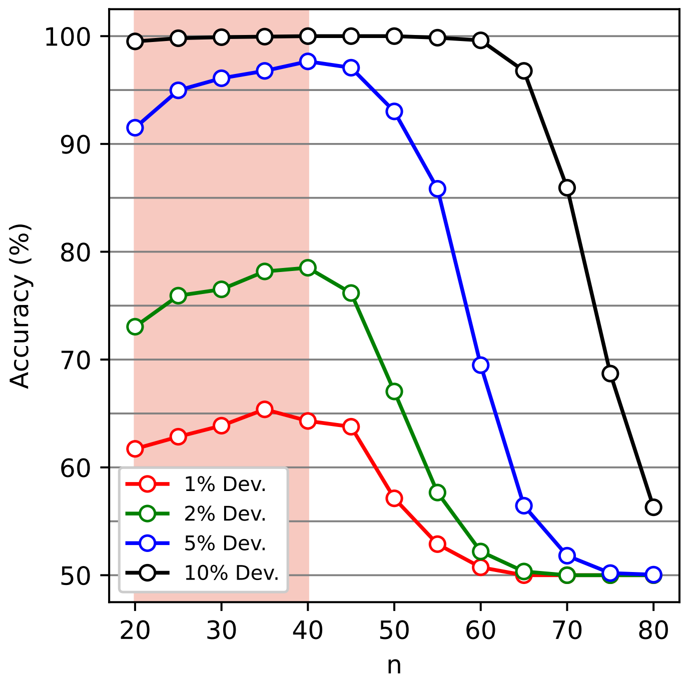

Fig. 5.3: Model performance on instances of varying sizes.

We had trained our model on instances with $20 \leq n \leq 40$ cities. The model learnt on smaller instance sizes, but we assessed its performance calibre on larger problem sizes. We averaged the trained model accuracy over test datasets of 1024 instances for varying values of n. We have displayed the results in Figure 5.3. As expected, the model was seen to perform better for larger deviations like 5% and 10%, and worse for smaller deviations like 1%. Note that a problem of double the size would require 2^n more time to compute by traditional algorithms, and thus such a rapid decay in accuracy seen in the figure was expected.

5.6.3 Model Performance on Larger Deviations

From the acceptance curves in Figure 5.2 and the accuracy curves in Figure 5.3, it is clear that the model generalises to larger deviations from the

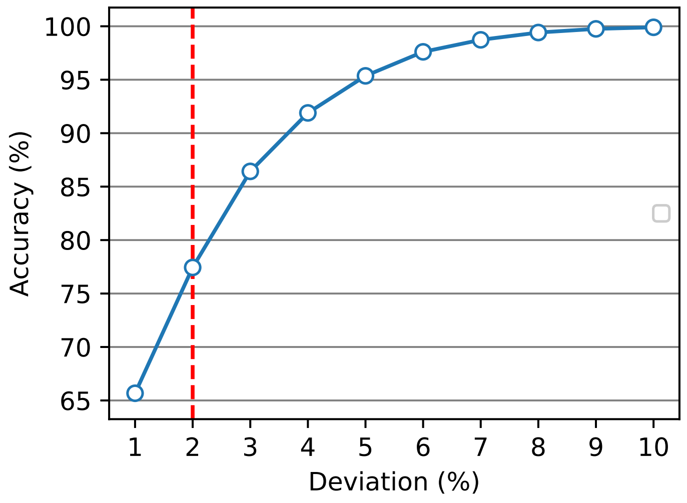

Fig. 5.4: Model performance on various values of deviation.

optimal tour cost than the 2% it was trained on. We checked the accuracy of the trained model on the same test dataset of 1024 n-city instances with $n \sim \mathcal{U}(20, 40)$ for varying deviations from the optimal tour cost. As we had expected, the model becomes more confident for larger deviations (see illustration in Figure 5.4), which is not surprising given that the corresponding decision instances were comparatively more relaxed than before. Figure 5.4 shows that accuracy increases until it plateaus near 100% for large deviations. Further, we observe that the model could obtain accuracies above the baseline (50%) for instances more constrained than those it was trained on, for instance, with 65% accuracy at 1%.

BIBLIOGRAPHY

[1] M. L. Balinski and Andrew Russakoff. On the assignment polytope. *SIAM Review*, 16(4):516–525, 1974.

[2] Jillian Beardwood, J. H. Halton, and J. M. Hammersley. The shortest path through many points. *Mathematical Proceedings of the Cambridge Philosophical Society*, 55(4):299–327, 1959.

[3] Dimitris Bertsimas and John Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, 1st edition, 1997.

[4] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[5] Rainer Burkard, Mauro Dell'Amico, and Silvano Martello. *Assignment problems*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2009.

[6] Rainer E. burkard. *Quadratic Assignment Problems*, pages 2741–2814. Springer New York, New York, NY, 2013.

[7] Eranda Çela. *The quadratic assignment problem*, volume 1 of *Combinatorial Optimization*. Kluwer Academic Publishers, Dordrecht, 1998. Theory and algorithms.

[8] Michele Conforti, Gerard Cornuejols, and Giacomo Zambelli. *Integer Programming*. Springer Publishing Company, Incorporated, 2014.

[9] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. The MIT Press, 2016.

[10] Martin Grötschel and Manfred Padberg. On the symmetric travelling salesman problem i: Inequalities. *Mathematical Programming*, 16:265–280, 12 1979.

[11] Michael Hahsler and Hornik Kurt. Tsp – infrastructure for the traveling salesperson problem. *Journal of Statistical Software*, 2:1–21, 12 2007.

[12] W. R. Hamilton. Account of the icosian calculus. *Proceedings of the Royal Irish Academy*, 6:415–416, 1858.

[13] F. Harary. *Graph Theory.* Addison-Wesley, Reading, MA, 1969.

[14] Michael Jünger and Volker Kaibel. A basic study of the qap-polytope. Technical report, Institut Für Informatik, Universität zu Köln, Pohligstrasse 1, D-50969, 1996.

[15] Volker Kaibel. *Polyhedral Combinatorics of the Quadratic Assignment Problem.* PhD dissertation, Institut Für Informatik, Universität zu Köln, Pohligstrasse 1, D-50969, 1997.

[16] T. P. Kirkman. On the representation of polyhedra. *Philosophical Transactions of the Royal Society, London*, 146:413–418, 1856.

[17] Tjalling C. Koopmans and Martin Beckmann. Assignment problems and the location of economic activities. *Econometrica*, 25:53–76, 1957.

[18] Eugene L. Lawler. The quadratic assignment problem. *Management Science*, 9(4):586–599, 1963.

[19] Michael Nielsen. Neural networks and deep learning, https://www.neuralnetworksanddeeplearning.com, 2021.

[20] Marcelo Prates, Pedro Avelar, Henrique Lemos, Luís Lamb, and Moshe Vardi. Learning to solve np-complete problems: A graph neural network for decision tsp. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33:4731–4738, 07 2019.

[21] Marcelo Prates, Pedro Avelar, Henrique Lemos, Luís Lamb, and Moshe Vardi. https://github.com/machine-reasoning-ufrgs/TSP-GNN.git, 07 2019.

[22] Kushal Shah. Evolutionary intelligence (youtube channel) https://www.youtube.com/channel/UCgzJ7IRb0k-349CVK6KqLqQ, 2021.

[23] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, Jan 2021.