UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

PEDRO FOLETTO PIMENTA

Solving the Kidney Exchange Problem using Graph Neural Networks Trained with No Supervision

Work presented in partial fulfillment of the requirements for the degree of Bachelor in Computer Science

Advisor: Prof. Dr. Luis C. Lamb
Coadvisor: Pedro H. C. Avelar

Porto Alegre
2023

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitor: Prof. Carlos André Bulhões Mendes
Vice-Reitora: Profª. Patricia Helena Lucas Pranke
Pró-Reitora de Graduação: Profª. Cíntia Inês Boll
Diretora do Instituto de Informática: Profª. Carla Maria Dal Sasso Freitas
Coordenador do Curso de Ciência de Computação: Prof. Marcelo Walter
Bibliotecário-chefe do Instituto de Informática: Alexsander Borges Ribeiro

“In a properly automated and educated world, then, machines may prove to be the true humanizing influence. It may be that machines will do the work that makes life possible and that human beings will do all the other things that make life pleasant and worthwhile.”

— Isaac Asimov

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my family: my mother, Ângela Foletto, my father, Marcelo Soares Pimenta, and my sister, Clara Foletto Pimenta. I love you. Thank you for encouraging me to pursue my passions, while still giving me space to be who I am and make my own decisions. I am forever grateful for the education you gave me.

I also thank my girlfriend, Teodora Foss, who made me a lot happier and filled my life with more meaning since the day we met. I love you.

I would also like to thank all my friends. I love you all too. This page is obviously too short to cite you all. However, I would like to give special thanks to two computer scientist friends who I admire a lot: José P. S. Martinez, who helped me directly in this work through long high level technical conversations, and João M. Flach, with whom I talked to a lot and traded advice while writing this document, thus helping me to better explain my ideas and describe the results.

I also wish to thank all my teachers throughout my life. I thank professor Paolo Rech for teaching me what is a bit, professor Rodrigo Machado for teaching me what is a graph, professor Bruno Castro da Silva for teaching me what is backpropagation, and professor Manuel Menezes de Oliveira Neto for teaching me what is the Fourier transform.

I also thank my colleagues for all the fun and sometimes even productive discussions. This was surely a really important part of my graduation.

Last but not least, I would like to thank my advisers Prof. Dr. Luis C. Lamb and Pedro H. C. Avelar. Thank you for the support and advice given throughout this project.

ABSTRACT

This work introduces a new machine learning-based approach for solving the Kidney Exchange Problem (KEP), an NP-hard problem on graphs. The problem consists of, given a pool of kidney donors and patients waiting for kidney donations, optimally selecting a set of kidney donations so as to optimize the quantity and quality of transplants performed, while respecting a set of constraints about the arrangement of these donations. The proposed technique consists of two main steps: the first one is a Graph Neural Network (GNN) trained without supervision; the second is a deterministic non-learned search heuristic that uses the output of the GNN to find a valid solution. To allow for comparisons, we have also developed and experimented with an exact solution method that uses integer programming, two greedy search heuristics without the machine learning module, and the GNN alone with no heuristic. Finally, we analyze and compare the methods accuracy and performance and conclude that the learning-based two-stage approach is the best one in terms of solution quality, as it outputs approximate solutions on average 1.1 times better than the ones from the deterministic heuristics alone.

Keywords: Kidney Exchange Problem. Graph Neural Networks. Machine Learning. Deep Learning. Optimization.

Resolvendo o Problema de Troca de Rins Usando Redes Grafo-Neurais Treinadas sem Supervisão

RESUMO

Esse trabalho introduz um método baseado em aprendizado de máquina para resolver aproximadamente o problema de troca de rins (*Kidney Exchange Problem*), um problema NP-difícil em grafos. A técnica proposta consiste em dois principais passos: o primeiro é uma rede grafo-neural treinada sem supervisão que prediz um score para cada aresta do grafo de entrada; o segundo é uma heurística de busca sem aprendizado que usa esses scores para construir uma solução válida. Para fins de comparação, também foi implementado um método de solução exata que usa programação inteira e duas heurísticas de busca usadas sem o módulo de aprendizado, assim como uma versão da GNN sem uma heurística. Os métodos são analisados e comparados entre si, e é concluido que o método de dois passos baseado em aprendizado de máquina atinge os melhores resultados em termos de qualidade, construindo soluções aproximadas em média 1.1 vezes melhores que as de uma heurística sozinha.

Palavras-chave: Problema de Troca de Rins. Redes Grafo-Neurais. Aprendizado de Máquina. Aprendizagem Profunda. Otimização..

LIST OF FIGURES

LIST OF ABBREVIATIONS AND ACRONYMS

KEP	Kidney Exchange Problem
ML	Machine Learning
GPU	Graphics Processing Unit
GNN	Graph Neural Network
PNA	Principal Neighborhood Aggregation
GAT	Graph Attention Network
GATv2	Graph Attention Network updated version with dynamic attention

CONTENTS

1 INTRODUCTION

This chapter is separated in three sections. On the first one, the motivations for this study will be explained. Then, in the second, the objective of this study will be presented. Finally, on the third section, the structure of the rest of this document will be describe.

1.1 Motivation

This study addresses the use of machine learning approaches for the approximate solving of the Kidney Exchange Problem (KEP), an NP-Hard problem on graphs. The problem consists of, given a pool of kidney donors and patients waiting for kidney donations, optimally selecting a set of donations so as to optimize the quantity and quality of transplants performed, while still respecting a set of constraints about the arrangement of these donations. Solving this problem has both a practical and a theoretical motivation, which are described below. KEP will be further explained in detail in Section 3.1.

1.1.1 Practical Motivation

There are thousands of patients on the waiting list for kidney donations registered in the health systems of each country around the globe. Although many transplants are successfully carried out each year (for instance, in 2021 there were 4,832 kidney donations in Brazil (SERIE..., 2022b)), it is common for a patient to wait several months or even years for a transplant, and unfortunately many people die before receiving a donation.

In Brazil, out of the 54,964 patients waiting for transplants, 31,764 (57.7%) were on the list for kidney donations, according to the official Brazilian health ministry data (SERIE..., 2022a). In France, there are 16,181 patients waiting for a kidney transplant on January of 2020 (DIVARD; GOUTAUDIER, 2021). Despite being a country with a very developed and financed universal health care system, and being pioneer in kidney transplantation, with the first living donor transplant performed in 1952, its waiting list still increases from year to year. In the United States of America as of January 2023, there are 104,398 candidates on the waiting list for organ transplants, according to the Organ Procurement and Transplantation Network, the organization that serves as the United

State's organ transplant system; 89,005 (85.2%) of these patients wait for kidney donations (OPTN. . . , 2023).

Considering the extremely high demand, a good way to optimize the kidney exchanges in order to maximize the donations is very important. Besides the absolute quantity of donations, other metric that needs to be taken into account is the quality of donations. Some donors-patient transplants have more compatibility than others, and a low compatibility may be associated with a higher chance that the transplant goes wrong somehow (e.g. the patient's organism may reject the transplanted organ). Other than that, a patient may have a higher priority due to them having been waiting more time, or due to the severity of their disease or the urgency of the treatment. In this scenario where a donation could be the difference that saves a life, any minor improvement is welcome.

In order to maximize the kidney donations, the scenario was formalized into a combinatorial optimization problem, which consists of finding the optimal selection of kidney donations considering a pool of kidney donors and patients waiting for kidney transplants (ROTH; SöNMEZ; UNVER, 2004). Most of the KEP instances that need to be solved in real world situations are small enough to be solved with operational research algorithms in a reasonable time, even though the computational cost scales exponentially with the size of the input, due to the problem being NP-Hard. One may argue that this makes any work in solving this problem with an heuristic an useless effort, because even if the heuristic method's solution is almost as good as the optimal solution, any slight improvement may still be extremely valuable, as we are dealing with human lives, thus justifying the extra computational time and resources used for the obtaining the latter. Nevertheless, in the future larger instances may appear more and more often, for one reason or another, which may lead to intractability. For instance, as health systems around the globe cooperate, the sets of patients waiting for donations in each country may merge. Another situation that could happen is that, as more people get access to health systems, more people may discover a kidney disease through a medical diagnostic. The appearance of larger instances would thus potentially bring about the necessity of the use of heuristics, which would be the only way to solve them in reasonable time. Anyhow, even if that is not the case, improving the solution of this problem could still contribute to future advancements and, at the long term, these advancements could potentially save lives, decrease the cost of the kidney exchange allocation systems or, at least, improve our understanding of the problem and its solutions.

1.1.2 Theoretical Motivation

To the best of our knowledge, no other machine learning solution has been presented for the Kidney Exchange problem. Several methods that involve learning have already been successfully applied to other NP hard optimization problems in graphs, as better described in section 3.2. Considering that, it would seem very probable that the Kidney Exchange problem could also be solved with similar methods. However, we cannot be sure of it without actually implementing and verifying it. It may be very valuable to evaluate how well machine learning methods would perform, what would be their advantages and disadvantages when compared to the exact solution and to non-learned heuristics. Additionally, studying the obstacles and limits of the approach may lead to useful insights, new techniques, and possibly new directions for research. These insights, techniques and research directions are potentially extendable to the application of ML methods on other graph optimization problems, as a lot of them share many similarities with KEP.

1.2 Objectives

The main objective of this work is, thus, to answer the following question: **Can the Kidney Exchange problem be better approximately solved with the help of machine learning?** If positive, we want to evaluate the feasibility of utilizing such an approach in terms of the quality of the solutions it provides. Further, we are also interested in assessing how viable would such a method be in terms of computational time. Additionally, it is hoped that, by trying to answer these questions, we may also better understand the limitations of the employed machine learning methods for this problem and the potential future research directions for solving not only the KEP but also other optimization problems in graphs.

1.3 Structure

The remainder of this document is organized in the following manner: In chapter 2, Theoretical Basis, some key concepts for the understanding of this study are explained and reviewed. Next, in chapter 3, Related Work, other relevant studies related to this one

are described. Then, the methods employed in this study will be presented and described in chapter 4, Methods. Chapter 5, Methodology explains the practicalities of the implementation and execution of the performed experiments. Chapter 6, Results, presents the results of the experiments described in the previous chapter. Chapter 7, Discussion, analyses and discusses the results presents in the previous chapter. Finally, chapter 8 reports the objectives described at section 1.2 with the results presented at 6, summarizing the study's contribution, and then suggests future research directions.

2 THEORETICAL BASIS

This chapter aims to provide an explanation and review of the fundamental concepts that are necessary to understand this study.

2.1 Graphs

In computer science, graphs are widely used as models in a number of applications. For instance, they can represent a vast range of diverse situations, and are used to model real world scenarios like computer networks, knowledge databases, social networks, protein interactions, the world wide web and the connections between its pages, and many others (CAI; ZHENG; CHANG, 2017; GOYAL; FERRARA, 2018). Once a real world context of a problem is mapped to a graph, we are able to create computer applications that solve this problem using its graph, like finding the shortest route between two points in a map, for example.

A graph is a mathematical discrete object $G = \{V, E\}$ composed of a set of nodes V and a set of edges E that connect these nodes. A graph whose edges are directed, i.e. each edge has an origin node and a destination node, is called a directed graph, or a digraph, and a graph whose edges are not directed is called an undirected graph. Each node and edge of a graph may have attributes and/or properties associated to it. A very common scenario is that the graph's edges have weights, which may represent various quantities, such as distance, cost, or strength; in that case, we call it a weighted graph. In this work, we deal mainly with weighted directed graphs, such as the one in figure 2.1.

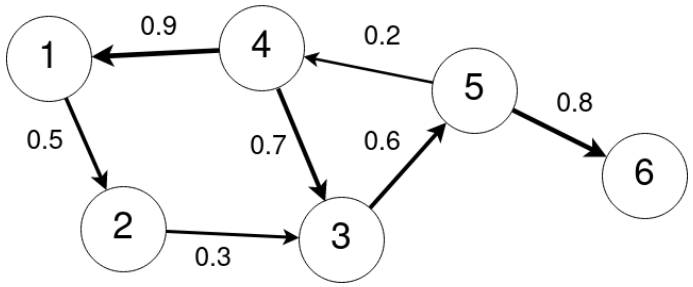

Figure 2.1 – Example of a simple small weighted directed graph with 6 nodes and 7 edges. The number in each node correspond to their ID; and the thickness of the edges represent their associated weight, which is written right beside or above it. Source: Author.

2.2 NP-Hard Problems

"If something is [NP-]hard to do, then it's not worth doing."
— HOMER SIMPSON

The class of NP-hard problems consists of computational problems that are notoriously difficult to solve. These problems are characterized by the fact that no known algorithm can solve them efficiently in the worst case. As the size of the problem instance, i.e. the input data for the algorithm, increases, the time required to solve it grows exponentially. Consequently, they are are intractable to solve optimally at large scales.

Many important optimization problems, such as the traveling salesman problem and the graph coloring problem, are NP-hard. These problems have significant implications for fields such as computer science, mathematics, and operations research, and have motivated the development of sophisticated algorithms and heuristics to approximate their solutions.

2.3 Heuristic Methods

"You can try the best you can
The best you can is good enough"
— RADIOHEAD

Heuristic methods are problem-solving strategies that prioritize efficiency over optimality. Unlike exact algorithms, which guarantee a globally optimal solution, heuristic methods provide a solution that is not necessarily the optimal solution, although sometimes they are likely to be very close to it. These methods are often used to tackle complex optimization problems that cannot be solved using exact algorithms, such as NP-Hard problems, either because the problem size is too large or because the solution space is too complex to be fully explored. Consequently, heuristic methods play an important role in tackling real-world optimization problems and are essential in many practical applications.

2.4 Artificial Neural Networks

Artificial Neural Networks (ANNs) are a class of machine learning algorithm originally inspired by the structure and function of the human brain. There are numerous different neural network-based models (also called deep learning models), and their architectures have been designed and optimized for specific applications and tasks, but overall all of them are based on the Feed Forward Network (FNN). FNNs consist of multiple layers of artificial neurons, each one receiving an input vector from the previous one, applying a learned non-learned linear transformation, and then passing the resulting vector to the next layer. Convolutional Neural Networks (CNNs), for instance, were developed specifically for image and signal processing tasks, and use a combination of convolutional layers and pooling layers to learn spatial or other signal-based features more efficiently. Recurrent neural networks, on the other hand, were designed to process sequential data, and thus are very efficient for tasks like time-series analysis and natural language processing. In this work, we deal with Graph Neural Networks, which were designed to operate on graph-structured data.

2.5 Graph Neural Networks

"As a net is made up of a series of ties, so everything in this world is connected by a series of ties. If anyone thinks that the mesh of a net is an independent, isolated thing, he is mistaken. It is called a net because it is made up of a series of interconnected meshes, and each mesh has its place and responsibility in relation to other meshes."

— BUDDHA

Graph Neural Networks (GNNs) are (deep) learning models that operates on graph-structured data. These opened up a number of applications in social networks, molecular structures, knowledge graphs, and now instances of the Kidney-Exchange Problem. GNNs incorporate graph structures into their architecture as an inductive bias, allowing them to capture the relationships between nodes in a graph. Graphs can vary in their structures and sizes, which does not conform to the fixed size vectors that FNNs and a great part of other ML methods expect as input. Another property of graphs is the *permutation invariance* of its nodes and edges, which means that they do not have an order or

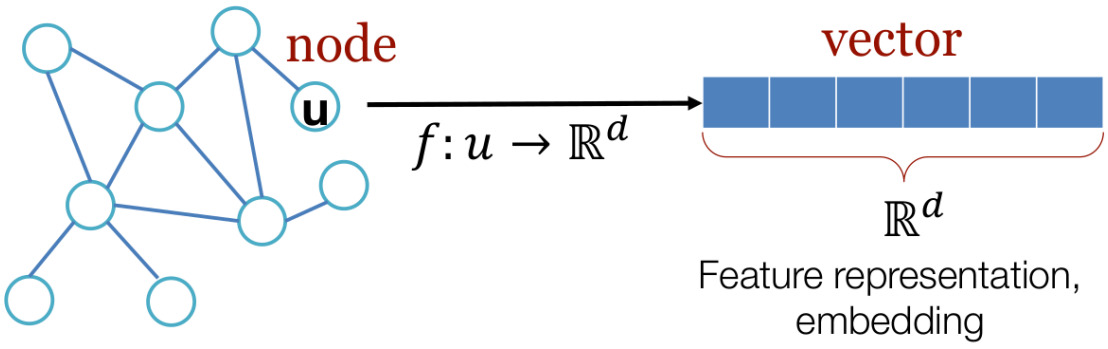

Figure 2.2 – Embedding of a graph node u into a d-dimensional vector. Source: (LESKOVEC, 2017)

a sequence; what matters it the relations between them. GNNs were designed to address particularities of graph-structured data such as these ones, and to learn to capture not only the raw information contained in the individual components of the graph, but also high level contextual information about the relations between them. The concept of GNNs were defined and introduced in 2005 (GORI; MONFARDINI; SCARSELLI, 2005), although further elaborated only in 2009 (SCARSELLI et al., 2009) with a definition that is closer to the current understanding. Since then they have been shown to achieve state-of-the-art performance on a wide range of graph-related tasks, such as node classification, graph classification, and link prediction.

Although there are many variations of GNNs, as a general rule they work by performing representation learning on graphs. Namely, they learn to map each node into a an embedding, i.e. an n-dimentional vector representation. Each node embedding contains relevant information about the node for the task, considering its position or role in the context of the graph, as well as information about the overall structure of the graph. There are also GNNs designed for edge-embedding, where it learns to map each edge of the graph into an embedding, or even graph-embedding, where it does the same but for the whole graph.

2.5.1 Message-Passing Layers

The key design element of GNNs is the use of message passing layers, which propagate information from a node's neighbors to the node itself, allowing the algorithm to learn about the node's properties and its position in the context of the graph, i.e. its relationship with other nodes in the graph. With each forward pass of a GNN message

passing layer, each node representation is updated incorporating information of its neighbor nodes, leading to an improved representation of each node of the graph, more useful for the task being solved. Consequently, stacking N message passing layers allows the GNN to compute representations from the N-neighborhood of each node. Not only that, but stacking more message passing layers also allows the GNN to build more refined node features, with more complex information about the relation between the node and its context in the graph. However, it comes with a cost: more layers also increase the inference and the training time, and has the chance of making the training harder, as deeper neural networks may suffer from *vanishing gradient*, which is when the partial derivative of the loss function approaches a value close to zero, and the partial derivative vanishes.

In the GNNs employed in this work, two different message passing layers were used: Principal Neighbourhood Aggregation (PNA) layers and Graph Attention Network (GAT) layers. Principal Neighbourhood Aggregation layers were presented in 2020 (CORSO et al., 2020), and use multiple aggregation functions with degree-scalers to better capture information from neighbor nodes. Graph Attention Network (GAT) layers were introduced in (VELIčKOVIć et al., 2017). This type of layer combines message passing layers with the attention mechanism that, although used at least since 1990 under names like multiplicative modules and sigma pi units (LECUN, 2020), were first introduced with this name to deep learning models in 2014, in Recurrent Neural Network models (BAHDANAU; CHO; BENGIO, 2014). This technique received then a surge of interest in the machine learning community after the 2017 article that introduced Transformer models (VASWANI et al., 2017). A way to think about the attention mechanism is that it aims to dynamically give "focus" on the most relevant part of the data, given a context (or *query*); in the case of GAT, each time before the message vectors of neighbor nodes of a given node are aggregated, these vectors are first weighted by attention coefficients, which are computed dynamically. Later, on 2021, (BRODY; ALON; YAHAV, 2021) an improved version of the GAT layers, called GATv2, was proposed. While GAT's attention mechanism assigns an attention score to each neighbor node unconditioned on the query node, GATv2 computes what the authors have called *dynamic attention* where, for every node, different attention coefficients are computed for each of their neighbor nodes.

2.6 Softmax

The *softmax* function is designed to convert an input vector y into a a probability distribution, and is defined by Formula 2.1. First, it applies the exponential function to each element y_i of y; then, it normalizes the resulting values by dividing each one by the sum of all the exponentials. Two important properties of the softmax operation are that it assures that all output values are between 0 and 1 and that their sum is equal to one, therefore serving as a kind of normalization.

$$\sigma(y_i) = \left(\frac{e^{y_i}}{\sum_j e^{y_j}} \right) j = 1, ..., n \tag{2.1}$$

3 MACHINE LEARNING AND THE KEP - A BRIEF REVIEW

"Scientific knowledge belongs to humanity."
— ALEXANDRA ELBAKYAN

This chapter aims to present a brief review of other works related to this study. It is separated in two parts: one about the Kidney-Exchange Problem and another about machine learning models for optimization problems in graphs. There have been work on both KEP and on ML methods applied to graph-based optimization problems, but to the best of our knowledge, this work is the first one that uses ML methods to solve KEP.

3.1 Kidney Exchange Problem

Kidney disease affects millions of people worldwide, and the two known treatment options for end-stage kidney disease are dialysis and kidney transplantation (DELORME et al., 2022). Transplantation is the preferred treatment for the most serious forms of kidney disease (ROTH; SöNMEZ; UNVER, 2004) due to it being cheaper and offering a better quality of life and better life expectancy (AXELROD et al., 2018). The source of the kidney can be either a cadaver or a live donor, as the human body has two kidneys and often only one suffices.

The compatibility of a transplant between a donor and a recipient is determined by a number of different factors, such as the blood-group compatibility, tissue-type compatibility, the ages and general health of the donor and the recipient, the size of the donor kidney, and many others (DELORME et al., 2022). The lower the compatibility between a donor and a patient, the lower is the chance of success of a kidney transplant between them.

In the last decades, there have started to be *paired kidney exchanges*, which are cycles involving donor-patient pairs such that each donor cannot give a kidney to their intended recipient because of some kind of incompatibility, but each patient can receive a kidney from a donor from another pair (ROTH; SöNMEZ; UNVER, 2004). These cycles were first performed with only two donor-patient pair nodes, but later longer cycles of kidney exchanges were performed. One small example of a kidney exchange cycle is illustrated on the left of Figure 3.1. Another possibility of exchange scheme is to create exchange chains, that begin with a donation of an altruistic or cadaveric kidney donor,

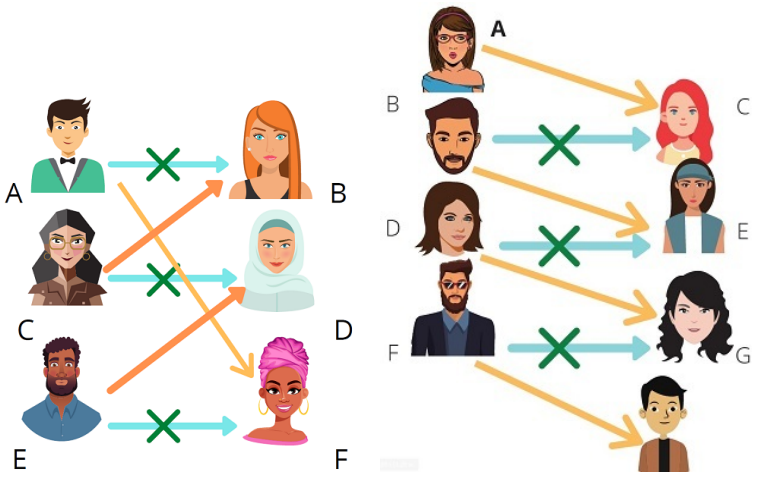

Figure 3.1 – On the left, a *kidney exchange cycle*, i.e. a cycle of donations of Patient Donor Pairs (PDPs). On the right, a *kidney exchange path*, i.e. a series of donations starting on an altruistic donor (NDD), following through with PDPs, and optionally ending on a patient without an associated donor. The green arrows with a cross represent kidney donation incompatibility between patients and their associated donor pair; yellow and orange arrows represent donation compatibility and together form the solution for each instance. Source: (INFORMATION..., 2022).

follows with chained donations of patient-donor pairs, and then finishes with a donation either to a patient with or without associated donor. In this study, the donation chains are also referred to as *paths*, a term often used for describing sequences of connected nodes in graph problems. One small example a kidney exchange path is illustrated on the right of Figure 3.1. To find the best possible allocation, i.e. the optimal solution to the problem, considering a set of donors, patients, and patient-donor pairs, a mix of both cycles and chains can be selected, as long as the cycles and paths do not intersect each other.

Ideally, these cycles and chains could have an unlimited size. In real life, however, there is a practical limit to the size of the paired kidney exchange cycles and chains: the kidney donation surgeries in a chain often must be done simultaneously, so as to ensure that every patient receives a kidney before her associated donor donates her kidney. However, organizing many simultaneous surgeries is logistically very complex, and sometimes impractical or even impossible. Even if they do not have to be done simultaneously, it is generally required at least that every patient-donor pair receive a kidney before they give a kidney. Furthermore, there are numerous other logistical difficulties that rise when dealing with longer cycles and chains, which makes it highly desirable or sometimes even necessary that these donation cycles and chains have limited size. The longest kidney transplant chain successfully performed had a size of 35, and happened between 6 January and 17 June 2015, in the USA (LONGEST..., 2020), although in most situations

the maximum reasonable size is considerably smaller.

3.1.1 Problem Definition

The Kidney Exchange Problem (KEP) was first mathematically formalized by A. Roth et al. in (ROTH; SöNMEZ; UNVER, 2004), and then slightly updated in various ways in subsequent works. A summary of the variations found in the literature and of models and techniques currently employed to solve them can be found at (BIRó et al., 2021).

In the formalization used in this study, each instance of the KEP is represented by a directed weighted graph $G = \{V, E\}$, and an optional nonnegative integer parameter k, which represents the maximum length allowed for the cycles and paths in the solution. Each patient, donor, and patient-donor pair is mapped to a graph node $v \in V$; they will be referred to as patient (P) nodes, non-directed (or altruistic) donor (NDD) nodes, and patient-donor pair (PDP) nodes. The set of nodes V is thus accordingly partitioned into sets P, NDD and PDP. The graph's edges $e \in E$ represent donation compatibility: an edge going from one node to another represents that a kidney donation in this direction is possible; the corresponding edge weight w_e encodes how compatible the donor is to the recipient.

Solving a KEP instance means to optimally select a set of cycles and chains to optimize the transplants performed. This includes maximizing not only the quantity but also the quality of the transplants, which is encoded in the edge weights. The solution must also respect a set of constraints. Each node may participate at most in one transplant as a donor and in another as a receiver. Also, PDP nodes can only donate a kidney if they receive one, although they can receive without donating. Nodes of type P can only receive donations, and NDD nodes can only donate. The problem could thus be put in a single phrase: "*Given a list of kidney needing patients, kidney donors, and patient-donor pairs, and a compatibility index between each possible donor and receiver, what is the best possible selection of donations that can be performed so that the total quantity of transplants, weighted by the compatibility indexes, is maximized, while still respecting a given size limit to the kidney exchange cycles and chains in the solution?*"

Our KEP formalization is represented in the set of equations below, inspired on the so-called Recursive Algorithm formulation described by (ANDERSON et al., 2015). We use a binary variable y_e for each edge $e \in E$ that indicates if the edge is part of the solution

or not, as well as auxiliary variables f_v^i and f_v^o for each node $v \in V$, which represent its *flow in* and *flow out* values, i.e. the node's number of incoming and outcoming edges in the solution, and are defined at Equations 3.9 and 3.10. $\mathcal{C}$ represents the set of existing cycles and paths in graph, where $C \in \mathcal{C}$ is a collection of edges, i.e. $C \subset E$. $\mathcal{C}_k$ is a subset of $\mathcal{C}$ (i.e. $\mathcal{C}_k \subset \mathcal{C}$) containing the cycles and paths that use k or fewer edges. The objective (defined at Expression 3.1) is to maximize the number of edges in the solution y, weighted by the associated edge weights w, while still respecting the KEP constraints (Equations 3.4, 3.5, 3.6, 3.7, and 3.8).

$$\max \sum_{e \in E} w_e y_e \tag{3.1}$$

$$\text{s. t.} \quad \sum_{e \in \mathcal{N}_{in}(v)} y_e = f_v^i \quad v \in V \tag{3.2}$$

$$\sum_{e \in \mathcal{N}_{in}(v)} y_e = f_v^o \quad v \in V \tag{3.3}$$

$$f_v^o \leq f_v^i \leq 1 \quad v \in PDP \tag{3.4}$$

$$f_v^o \leq 1 \quad v \in NDD \tag{3.5}$$

$$f_v^i \leq 1 \quad v \in P \tag{3.6}$$

$$\sum_{e \in C} y_e \leq |C| - 1 \quad C \in \mathcal{C} \setminus \mathcal{C}_k \tag{3.7}$$

$$y_e \in \{0, 1\} \quad e \in E \tag{3.8}$$

$$f_v^i = \sum_{e \in \mathcal{N}_{in}(v)} y_e \quad v \in V \tag{3.9}$$

$$f_v^o = \sum_{e \in \mathcal{N}_{out}(v)} y_e \quad v \in V \tag{3.10}$$

$$\mathcal{N}_{in}(v) = \{e \forall e \in E, e = (v', v)\}$$

$$\mathcal{N}_{out}(v) = \{e \forall e \in E, e = (v, v')\}$$

The constraints ensure the result is a valid solution for KEP: the first two (Eq. 3.2 and Eq. 3.3) are necessary for the use of the flow in and flow out variables, the third one (Eq. 3.4) controls the flow in and flow out of the PDP nodes, the fourth one (Eq. 3.5) does the same but for NDD nodes, the fourth one (Eq. 3.6) does the same but for P nodes, the

sixth one (Eq. 3.7) prohibits cycles or paths with length longer than a given limit k, and the seventh one (Eq. 3.8) defines the domain of the $\mathbf{y}$ variable.

It has been proven that this problem is NP-Hard (ABRAHAM; BLUM; SANDHOLM, 2007), although it can become polynomial-time solvable if some of the constraints are relaxed, such as limiting the exchange cycles and chains length to 2, or removing the length restriction entirely.

3.2 Machine Learning Methods for Optimization Problems in Graphs

In the last few years, many machine learning-based approaches that effectively solve several different optimization problems in graphs have been proposed, although none of them designed for solving KEP. Graph optimization problems already solved with the help of machine learning include the Set Covering Problem (YANG; RAJGOPAL, 2020), Graph Colouring (LEMOS et al., 2019; SANTOS; LAMB, 2020), Minimum Vertex Cover (SATO; YAMADA; KASHIMA, 2019; ABE et al., 2019), Maximum Cut (DAI et al., 2017), Graph Partitioning (NAZI et al., 2019), Maximum Independent Set (LI; CHEN; KOLTUN, 2018), Maximum Common Subgraph (BAI et al., 2020), and the Travelling Salesperson Problem (also called the travelling salesman problem or TSP), one of the most famous NP-hard problems, often used to represent the class, and some variants of it (JOSHI; LAURENT; BRESSON, 2019; JOSHI et al., 2021; PRATES et al., 2018; VINYALS; FORTUNATO; JAITLY, 2015; WU et al., 2019; KOOL; HOOF; WELLING, 2018).

In 2015, the authors of (VINYALS; FORTUNATO; JAITLY, 2015) presented a new type of neural network called Pointer Networks, designed to learn how to reorder the elements of an input sequence; they validated the method by using it to solve 3 problems, including the TSP. In 2016, researchers presented in (BELLO et al., 2016) a framework for combinatorial optimization problems using neural networks and reinforcement learning; the work focused on the TSP, which is a graph problem, but the approach was designed to work with any combinatorial optimization problem. In 2017, the authors of (DAI et al., 2017) proposed the utilization of a combination of graph representation learning and reinforcement learning to solve graph optimization problems; they showed that their proposed approach effectively learns to solve at least three of those problems: Minimum Vertex Cover, Maximum Cut, and TSP. In 2018, the decision variant of the TSP, called Decision Traveling Salesman Problem (DTSP), which is to decide if a given TSP instance

admits a Hamiltonian route with a cost no greater than a given threshold C, and is also NP-Hard, was solved in (PRATES et al., 2018) with a GNN.

In 2019, the authors of (JOSHI; LAURENT; BRESSON, 2019) tried to solve the TSP using a two stage technique that is very similar to the one presented in this study (described at section 4.3.7 and illustrated at Figure 4.1): firstly, a GNN processes the input graph and create scores for each edge of the graph; then, a non-learned search heuristic, which in this case was beam search, uses these scores to construct a solution. There have been other approaches that use similar techniques: in (PENG; CHOI; XU, 2021) the authors review graph learning methods for solving combinatorial optimization problems, with a focus on two-stage techniques, where the first is based on graph representation learning, which embeds the input graph into low-dimension vectors, and the second uses the embeddings learned in the first stage; (LAMB et al., 2020) surveys the use of GNNs as a model of neural-symbolic computing and their applications, which includes combinatorial optimization problems; (JOSHI et al., 2021) unifies and refines several of such two stage techniques for neural combinatorial optimization, and test it on the TSP.

4 METHODS

This chapter describes the methods used in this study for solving the Kidney Exchange Problem. We classified these methods in 3 categories: integer programming methods, non-learnable heuristic methods, and learnable heuristic methods. All of them use the same input information, which is a KEP instance, and return the solution in the same format, which is a binary label for each edge, indicating if it is in the solution or not. All of them are evaluated on the test dataset, described at Section 5.3, with the exception of the integer programming method, as explained in Section 5.5, but only the learned heuristics use the training dataset, during their training phase.

4.1 Integer Programming

To obtain the analytical solution, i.e. the optimal solution, the formulation presented in 3.1.1 was implemented with PyCSP3 (LECOUTRE; SZCZEPANSKI, 2020). There are other integer programming formulations for the KEP, including another one presented in the same article, as well as others in (ROTH; SöNMEZ; ÜNVER, 2007), (ABRAHAM; BLUM; SANDHOLM, 2007) and (CONSTANTINO et al., 2013). This formulation was chosen because it is the most straightforward one.

As PyCSP3, the solver used for implementing this problem, does not support float values in the objective function, the edge weights had to be adapted. Originally float values between 0 and 1, each of them was multiplied by 10000 and converted to integer, changing their domain to values between 0 and 10000. This transformation can be seen as having the same effect as limiting the digits of precision of the original decimal value representation to that of 4 decimal digits. Thus, its effect is similar to a small noise in the edge weight values, and the adaptation should not affect significantly the results.

4.2 Non-Learnable Heuristics

To evaluate the implemented methods that use machine learning, we decided to compare them to non-learnable heuristics, i.e. heuristic methods that do not use learning techniques. This section aims to describe these non-learnable heuristic methods. To the best of our knowledge, however, there are no canonical heuristics for the KEP. For this

reason, we implemented two search heuristics, which are described below.

4.2.1 Greedy Paths

This algorithm greedily selects paths that start on NDD nodes and goes through PDP or P nodes one by one until there is no more nodes to be selected, or until a P node is reached. It starts by selecting the edge with the highest weight considering only the subset of edges that have an NDD node as source. Then, considering only the edges that come from the previous node, it follows by selecting always the next edge with the highest weight, until there are no more available edges left that would continue the path. After a path has been added to the solution, the edges connected to nodes of this path are masked, and the Greedy Paths algorithm repeats the process until no more NDD nodes with valid outcoming edges are available. This algorithm is described at Algorithm 1.

Algorithm 1 Greedy-Paths

procedure GREEDY-PATHS($G = (N, E), k$)
    paths $\leftarrow$ []
    while $|\,\mathrm{GP}(G, k)| > 0$ **do**
        path $\leftarrow \mathrm{GP}(G, k)$
        paths $\leftarrow$ paths $\oplus$[path]
        $G \leftarrow (N \setminus \{n \,\forall\, n \in \text{paths}\}, E \setminus \{e \,\forall\, e \in E, src(e) = n \vee tgt(e) = n\})$
    return paths
procedure GP($G = (N, E), k$) ▷ *Gets one Greedy Path*
    $E_{NDD} \leftarrow \{e \,\forall\, e \in E, src(e) \in NDD\}$
    if $|E_{NDD}| = 0$ **then**
        return []
    $e_c \leftarrow \arg\max_{e \in E_{NDD}\}} w_e$
    path $\leftarrow [src(e_c)]$
    while $|\,\mathrm{OE}(tgt(e_c))| > 0 \wedge |\,\text{path}\,| < (k+1)$ **do**
        path $\leftarrow$ path $\oplus [tgt(e_c)]$
        $e_c \leftarrow \arg\max_{e \in \mathrm{OE}(tgt(e_c)) \setminus \bigcup_{n \in \text{path}} \mathrm{IE}(n)} w_e$
    return path
procedure OE($G = (N, E), n$) ▷ *Outgoing Edges*
    return $\{e \,\forall\, e \in E, src(e) = n\}$
procedure IE($G = (N, E), n$) ▷ *Incoming Edges*
    return $\{e \,\forall\, e \in E, tgt(e) = n\}$

4.2.2 Greedy Cycles

This algorithm greedily selects cycles of PDP nodes. It starts by selecting the edge with the highest weight considering only the subset of edges that have a PDP node as source and a PDP node as destination. Then, PDP nodes are greedily added to the solution in the same way as done by the Greedy Paths method until the cycle ends, or until it arrives at a node already added in the cycle, in which case the cycle is closed and the nodes before the node where it is closed are removed from the cycle. After a cycle is added to the solution, the Greedy Cycles algorithm applies a mask on the edges connected to nodes of this cycle, and then repeats the process until no more PDP nodes with valid outcoming edges are available. This algorithm is described at Algorithm 2.

Algorithm 2 Greedy-Cycles

procedure GREEDY-CYCLES($G = (N, E), k$)
  cycles $\leftarrow$ []
  while $|\,\mathrm{GC}(G, k)| > 0$ **do**
    cycle $\leftarrow \mathrm{GP}(G, k)$
    cycles $\leftarrow$ cycles $\oplus$[cycle]
    $G \leftarrow (N \setminus \{n \,\forall\, n \in \text{cycles}\}, E \setminus \{e \,\forall\, e \in E, src(e) = n \vee tgt(e) = n\})$
  return cycles
procedure GP($G = (N, E), k$) ▷ *Gets one Greedy Cycle*
  $E_{PDP} \leftarrow \{e \,\forall\, e \in E, src(e) \in PDP, dst(e) \in PDP\}$
  if $|E_{PDP}| = 0$ **then**
    return []
  $e_c \leftarrow \arg\max_{e \in E_{PDP}\}} w_e$
  cycle $\leftarrow [src(e_c)]$
  while $tgt(e_c) \notin$ cycle **do**
    if $|\,\mathrm{OE}(tgt(e_c))| \leq 0 \vee |\,\text{cycle}\,| \geq k$ **then**
      return [] ▷ *Unable to close cycle (dead end)*
    cycle $\leftarrow$ cycle $\oplus [tgt(e_c)]$
    $e_c \leftarrow \arg\max_{e \in \mathrm{OE}(tgt(e_c)) \setminus \bigcup_{n \in \text{cycle}} \mathrm{IE}(n)} w_e$
  return cycle
procedure OE($G = (N, E), n$) ▷ *Outgoing Edges*
  return $\{e \,\forall\, e \in E, src(e) = n\}$
procedure IE($G = (N, E), n$) ▷ *Incoming Edges*
  return $\{e \,\forall\, e \in E, tgt(e) = n\}$

4.3 Learnable Heuristics

This section aims to describe the learnable heuristics designed for KEP. Subsections 4.3.1, 4.3.2, 4.3.3, and 4.3.4 explain the most important modules and techniques, and subsections 4.3.5, 4.3.6, and 4.3.7 explain how they are used and combined to create methods that approximately solve KEP.

4.3.1 GNN Architecture

As KEP instances are graphs, using GNNs to extract more detailed and abstract information can help in constructing an approximate solution. Therefore, GNN models were chosen as the main learning module for the machine learning methods.

Before the execution of the GNN, the initial node features are set: the number of incoming edges, the number of outcoming edges, and the type of the node (NDD, PDP or P) represented with a one-hot vector.

The architecture of the GNN used in this work is the following: firstly, there is a message passing phase, where the node features are passed through a PNA layer, and then through two consecutive GATv2 layers; after each message passing layer, a ReLU activation function is applied, followed by a dropout regularization; next, for each node, the node features are passed through a fully connected feed forward neural network, followed by another ReLU activation function; then, the edge features are constructed by concatenating the original input edge features with the node features of the origin and destination nodes associated to each edge; these edge features are then passed, individually, through a fully connected feed forward neural network, which outputs a score for each edge; at this point, a skip connection adds the original edge weights to the edge scores; finally, a node-wise softmax operation, which is described at 4.3.4, is applied so as to normalize these scores in relation to the scores of other edges that share the same source node.

In order to make information flow not only in the original direction of the original edges, each message passing layer is accompanied by an associated layer, which we call *counter edge layer*, that is exactly similar, but with different learned weights, and with the difference that the information is propagated in the opposite direction, i.e. flowing from the destination node to the origin node. Each time message layers are executed, the output of the original and of the counter edge layers is concatenated before being passed to the next layers.

4.3.2 KEP Unsupervised Loss Function

This loss was designed to capture, without the need for the exact solution as a label or any other supervision, the essence of what we are trying to maximize: the sum of weights of edges that are in the predicted solution. It is defined as the log of the sum of weights of all edges of the input instance over the sum of weights of edges that are in the predicted solution, weighted by the scores predicted by the GNN. This loss function is presented in Formula 4.1, where w represents the vector of edge weights, $pred$ represents the vector of predicted classes (which indicates if each edge is contained in the solution or not), and s represents the vector of scores attributed by the GNN for each edge.

$$KEP\_Loss\,(w, pred, s) = \log \frac{\sum_{e \in E} w_e}{\sum_{e \in E} w_e pred_e s_e} \tag{4.1}$$

The multiplication by the sum of weights of all edges of the input instance serves as a normalization; this way, if an input instance were to have unusually high or unusually low values for its edge weights, this variation would not affect the loss function as much. The log function was applied at the end because the range of possible values coming out of dividing the two sums is extremely large, and this would certainly disrupt the training process.

4.3.3 Loss Constraint Regularization

In order to integrate information about the first three constraints of the Recursive Algorithm KEP formulation (described at section 4.1) into the learning of the model, a loss regularization function was developed. It aims to model the restriction that each node must have at maximum one single outcoming edge and one incoming edge that are part of the solution. It is defined as the log of the division between the total quantity of edges in the solution and the number of unique nodes that appear in the solution as a origin/destination node. This makes it so that the regularization term value is proportional to the number of invalid edges in the solution, i.e. the total quantity in the graph of extra edges for each source/destination node. This function can then be added to the unsupervised loss (described in the subsection above) by summing their output values, weighted by coefficients, which become new hyper-parameters of the training.

4.3.4 Node-wise Softmax

The node-wise softmax operation is the application of an independent softmax operation (described at section 2.6) for each group of edges that share the same source node. In this way, for each node we will have a probability for each outgoing edge; in KEP, these values may represent a probability distribution for the donation options of the donor for each source node. Although in this work the operation was used grouping edges by source node, with a simple change of a parameter it can group edges by groups of common destination node as well.

$$\text{node-wise-softmax}(s_{e_{(n_i,n_j)}}) = \frac{e^{-s_{e_{(n_i,n_j)}}}}{\sum_{n_k \in \mathcal{N}(n_i)} e^{-s_{e_{(n_i,n_k)}}}} \tag{4.2}$$

The best of our knowledge, such a function does not exist in Pytorch Geometric, nor in any other publicly available library. The implementation uses Pytorch operations in order to allow PyTorch's automatic differentiation engine *pytorch.autograd* to work, which is necessary for training Pytorch models. As a bonus, it also allows it to run it in a GPU with CuDA, as other Pytorch methods do, and thus makes it easily paralellizable.

This method is potentially useful for any edge classification task on graphs, specially when the problem involves constraints in which only one edge may be chosen per node, be it destination or origin node. These constraints are very common in optimization problems in graphs; this is the case for KEP, for example: each donor or patient-donor pair may donate at most one kidney, and each patient or patient-donor pair may receive at most one kidney.

4.3.5 Unconstrained GNN Model Trained Without Supervision

This method, referred to from now on as *Unsupervised GNN*, consists of a GNN model with the architecture described at 4.3.1, which receives a KEP instance and outputs, for each edge of the input instance, a score and a binary prediction, which indicates if the edge is part of the predicted solution or not. The binary prediction is made independently for each edge, and consists of a simple decision threshold. This GNN model is trained using the unsupervised loss described at subsection 4.3.2 with the loss regularization term described at subsection 4.3.3. Although there is no guarantee that the solutions given by this method will be valid, the loss regularization term is used with the goal of inducing it

to respect the problem constraints.

4.3.6 Unconstrained GNN Model Trained With Supervision

The goal of this method, referred to from now on as *Supervised GNN*, is to try to induce the model to learn to predict, for each edge of the input instance, if it belongs or not in the optimal solution. Just as the method described at subsection above (4.3.5), it also consists of a GNN model with the architecture described at 4.3.1. With a single prediction step, it outputs, for each edge of the input instance, a score and a binary prediction, which indicates if the edge is part of the predicted solution or not. Similarly to the unsupervised version (described at 4.3.5), the binary prediction is made independently for each edge, and consists of a simple decision threshold. However, instead of the loss function described at subsection 4.3.2, the function used for training this GNN is the binary cross entropy loss function, which is very popular for supervised training of binary classification models. The exact solution, computed with the integer programming method described at section 4.1, would be used to generate a binary label for each edge, indicating if it belongs or not in the optimal solution. Also similarly to the unsupervised version, there is no guarantee that this method will output valid solutions; nonetheless, it is hoped that the model could learn to choose edges so as to mimic all properties of the optimal solutions, including being valid.

4.3.7 Two Stage Method

Inspired by the approach used in (JOSHI; LAURENT; BRESSON, 2019) and (JOSHI et al., 2021), this method follows a two steps structure: firstly, the learnable step, which is a GNN, takes the KEP graph instance as an input and outputs a score for each edge; then, the non-learnable step, which is one of the heuristic methods described above in the 4.2 section is executed, but using the scores given by the GNN instead of the edge weights. This process is illustrated in the diagram on Figure 4.1. The intuition behind this idea is that the GNN model will learn to encode in the edge score contextual information that will change the decisions of the search heuristic so as to maximize the total score of the final output solution.

This method is trained without supervision using loss described at subsection

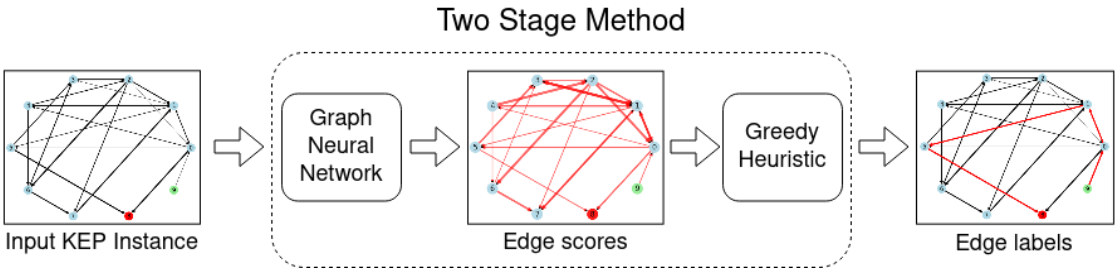

Figure 4.1 – Diagram representing an overview of the *two stage method*. The GNN takes the input KEP instance and computes a score for each edge of the graph; then, a greedy heuristic such as *GreedyCycles* or *GreedyPaths* uses these edge scores instead of the original edge weights to build an approximate solution, which is a binary label for each edge (*edge labels*), indicating if the edge is part of the approximate solution predicted or not. Source: Author.

4.3.2. There is no need to use the loss regularization term described at subsection 4.3.3 because the second step of the method ensures that the output will be a valid solution.

Two versions of the two stage method were implemented for the experiments. Both use the GNN described in subsection 4.3.1 for the first stage, but their second stage consist of different search heuristics. One uses the Greedy Paths search heuristic described at subsection 4.2.1 and is referred to later on as *GNN+GreedyPaths*. The other uses the Greedy Cycles search heuristic described at subsection 4.2.2 and is referred to later on as *GNN+GreedyCycles*.

5 METHODOLOGY

This chapter aims to explain in more detail the practical aspects of this work, such as how the experiments were implemented and conducted.

5.1 System configuration

The experiments described in this section, as well as the generation of the datasets and the executions of the solver that solves the problem, were run in a personal computer with the following specifications:

Processors: 4x Intel®Core™ i5-6600 CPU @ 330GHz.
Memory: 7,7Gb of RAM.
Graphics Processing Unit: NVIDIA GeForce GTX 1050 Ti/PCIe/SSE2.
Operating System: Ubuntu 22.04.1 LTS

5.2 Implementation

The code developed in this study was written with the objective of being clear, readable, reusable, and reproducible. It is available in a public repository in GitHub (KEP..., 2023), and pull requests with extensions, suggestions, or corrections are welcome and encouraged, as well any feedback. The code includes implementations for the generation of the dataset, the methods described in Chapter 4, the experiments and evaluations described at section 5.5, as well as scripts written to automate or facilitate the experiments, to visualize and investigate data, and to debug code.

The implementations were developed using the Python (PYTHON..., 2023) programming language (version 3.10.6). The libraries PyTorch (PYTORCH..., 2023) (version 1.13.1) and PyTorch Geometric (FEY; LENSSEN, 2019) (version 2.2.0) were extensively used for the implementation of all the methods used in this study, described at the section 4, as well as training and evaluation of the machine learning models, and creating, saving, loading, and manipulating data. Using PyTorch and PyTorch Geometric not only allowed for faster development, but also enables the training and inference of the models to be run in parallel in a GPU. For the generation of the dataset synthetic KEP instances,

the library NetworkX (NETWORKX. . . , 2023), which contains useful graph data structures and tools, was also used. The analytical integer programming solution described at 4.1 was implemented using the PyCSP3 library (LECOUTRE; SZCZEPANSKI, 2020), which allow us to model and solve combinatorial constraint satisfaction and optimization problems. Other popular Python libraries such as Pandas and Numpy were also utilized. The Matplotlib library was employed to generate the plots of comparisons between methods, the evolution of training loss, and so forth. With the purpose of maintaining a consistent programming style and high code quality, the Pre-Commit (PRE-COMMIT, 2023) framework was used in the project; it runs linters and code analysers before each commit, therefore automatically correcting and/or highlighting issues so as to alert the programmer of the need to adjust the code.

In order to improve the readability and comprehensibility of the code, some basic documentation was written. Most of the implemented functions, classes, and scripts have *docstrings* with descriptions of the functionality and instructions on how to use it. Other than that, some README files were added to the repository for more general high level information and instructions.

5.3 Dataset

Deep learning methods such as GNNs require lots of data; usually, many thousands of instances, or even millions, depending on the problem and the size of the model, are necessary for the models to converge to a good minima. With insufficient data, deep learning models are very prone to overfitting, or sometimes may not even learn anything at all. Medical data, however, is very scarce, and rarely available at this quantity. This happens for two main reasons. Firstly, a major problem with healthcare data is its sensitivity: as a lot of it is confidential information about the patients, it is usually highly protected and as a rule cannot be used without special consent, be it from the patients or at least from the health institute that owns the data. Furthermore, there are a limited number of medical cases of each given situation registered; although it would be useful to have a KEP dataset with millions of instances, this situation has not happened that many times, and not necessarily all of them have been registered digitally.

Due to the scarcity of the data, and considering that to train a machine learning model it usually takes at least tens of thousands of examples, the datasets were generated artificially. Three separate datasets were generated: the *train dataset*, with 10 thousand

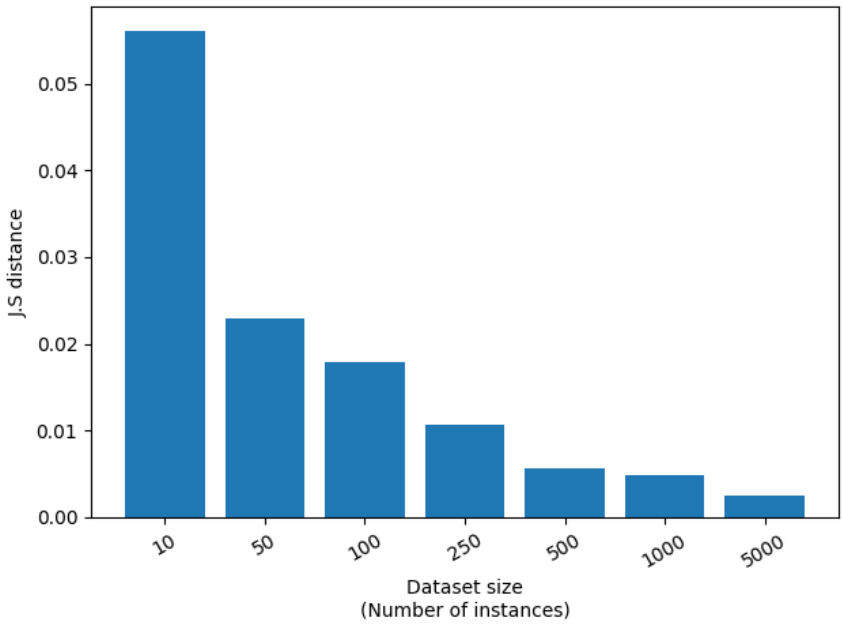

Figure 5.1 – Jenson-Shannon Distance (dissimilarity) between in-degree distributions in relation to the 10 thousand instances training dataset.

instances for the training of the model; the *validation dataset* with 100 instances, used for validation step during the training; and the *test dataset*, with 10 thousand instances, used to evaluate the performance of each one of the employed methods. This number of instances for the validation dataset was chosen because it was sufficiently big so that still kept roughly the same properties as the train and test datasets, but small enough that the validation does not slow down the training too much. To measure how different a randomly generated KEP dataset from a given size was from the 10 thousand instances train dataset, we measured the Jenson-Shannon Distance between the node's in-degree distributions of the two datasets. The results can be seen on Figure 5.1.

5.3.1 Artificial KEP Instance Generation

To generate each instance, first 300 separated nodes are created, and then 5500 edges are added sequentially linking random nodes, while still guaranteeing that no two edges connect the same two nodes in the same direction. The weight value of each edge is sampled from an uniform distribution of values between 0 and 1. To choose the number of nodes and edges of the KEP generated instances, the instances used for benchmarking in (ANDERSON et al., 2015) were used as reference. Considering the 25 instances presented in the Table S3, which they call "difficult" real-data instances, the average num-

ber of nodes on a KEP instance is 265.84, and the average number of edges is 5695.92. Thus, the values for the number of nodes and number of edges chosen were 300 and 5500, respectively. The proportion of the types of nodes was also chosen to be similar to the instances: roughly 90% of the nodes are PDP nodes, 5% are NDD nodes, and 5% are P nodes.

To keep track of the instances, an unique ID was assigned to each. For this, the Weisfeiler Lehman graph hash (SHERVASHIDZE et al., 2011) was used, which has strong guarantees that non-isomorphic graphs will get different hashes; it also considers node and edge features in its computation, thus differentiating even between graphs which have the same structure but different edge weight values.

5.4 Metrics

The objective of the Kidney Exchange Problem is to maximize the number of donations weighted by their compatibility index, i.e. their associated edge weights in the graph. Therefore, this was the main metric used to quantify the quality of each solution and to compare the different methods, and is also referred to as **score** in this study. As the operational performance of the methods was also to be compared, the total time that each method took to run per instance was also measured.

Ideally, the heuristics could be compared by using the optimality gap, which is defined as the distance between the heuristic solution score and the optimal solution score. However, as the randomly generated KEP instances of 300 nodes have shown to be intractable while using the PyCSP3 solver, i.e. impossible to solve optimally in a reasonable time, the optimality gap could not be measured.

For the trainable heuristics, the total training time was also measured and compared. To monitor and guide the training process, the evolution of the loss function value over the training time was also collected. Additionally, the mean score of the current model in the validation dataset was measured at each validation phase.

For a fair comparison between methods, all the measurements were made in the same test dataset, which is described in section 5.3, with the exception of the exact solution method, as explained in Section 5.5 below.

5.5 Experiments

The main goal of the experiments was to assess the machine learning methods, and compare them to the deterministic heuristics and to the exact solution, both in terms of the quality of the solutions as well as of their operational performance, i.e. the time it takes for a solution to be calculated.

Subsection 5.5.3 describes how the solution quality of each method described in Chapter 4 was measured and compared. The integer programming method, however, could not be measured in the same dataset because it took to long to run the solver. As a consequence, the *Supervised GNN* method, described in Subsection 4.3.6, could not be trained nor measured as well. To discover what instance sizes could be solved optimally in a reasonable time, an experiment was done to measure how much time it took to solve a KEP instance in relation to the input size; this experiment is described at Subsection 5.5.1. Subsection 5.5.2 describes the measurements and analysis done on the models training process.

5.5.1 Solver Execution Time Analysis

To measure how much time it takes to solve a KEP instance in relation to the input size, the following experiment was designed: first, we randomly generate 100 instances of each graph size (i.e. number of nodes), starting from 5 nodes and up to 300 nodes; then, each one is solved optimally using the integer programming method described at section 4.1. The elapsed time of each solver execution is collected for later analysis.

5.5.2 Training of the Machine Learning models

The training of the GNN models was separated in epochs. In each epoch, we iterate through the 10 thousand instances of the train dataset, predicting, calculating the loss, and updating the GNN weights according to it. At every 500 instances, a validation phase is run, where the model is evaluated on the validation dataset and a checkpoint is saved. The chosen batch size was 1, which means that the predictions were done on one instance at a time, because it empirically seemed to be the best for the learning process. There were two machine learning models that were trained in this study: the unconstrained un-

supervised GNN and the two stage method, which are described at subsections 4.3.6 and 4.3.7, respectively. For each training process, it was measured how the loss value evolved over time in the training and in the validation datasets.

5.5.3 Evaluation of KEP Solving Methods

In order to do a fair comparison between the different heuristic methods, all of them were evaluated by predicting the solutions of all 10 thousand instances of the test dataset. We did not set the cycle/path size limit (parameter k in constraint represented by Eq. 3.7), i.e. the cycles and paths could have any length, as long as they respect the KEP constraints. For each prediction, it was measured the solution score, the number of edges in the solution, and the relation between the solution score (which is the sum of the edge weights of the solution edges) and the total sum of edge weights of the graph. In addition, the validity of the predicted solution was evaluated; if invalid, we measure the number of invalid edges in the solution, i.e. the number of edges that disrespect the restrictions. Furthermore, we measured the time it took for each method to solve each instance using a CPU. Then, it was measured, for each model in relation to the whole test dataset, the mean, standard deviation, and distribution for the scores and the prediction elapsed times.

6 RESULTS

This chapter aims to present the results obtained from the experiments described in the Section 5.5 of the previous chapter.

6.1 Solver Time Measurements

Figure 6.1 shows a box plot of the time that the solver took to optimally solve KEP instances in relation to the instance size. For that, graphs of sizes 5 to 15 (i.e. number of nodes) were used; initially, graphs with up to 300 nodes were going to be included in the analysis, but as solving graphs with 16 nodes or more would take several days to compute, they were excluded. To solve a hundred instances with 15 nodes, for instance, it took 101.2 hours in total.

As we can see, the experiment results show a pattern of exponential growth of computational time in relation to the input size. The mean time it took when the graph node number was beneath 10 was always below 1.5 seconds. For instances with 15 nodes, the mean time measured was 3569.33 seconds, i.e. roughly one hour.

6.2 Training of the GNN Models

Figures 6.2, 6.3, and 6.4 show the evolution of the loss value on the training and validation datasets for the two stage methods described at Subsection 4.3.7, *GNN+GreedyPaths* and *GNN+GreedyCycles*, and for the *Unsupervised GNN* method described at Subsection 4.3.5, respectively. The training of *GNN+GreedyPaths* took, in total, 44046 seconds (i.e. 12 hours); for *GNN+GreedyCycles*, it took 15705 seconds (i.e. 4.3 hours); for *UnsupervisedGNN*, 13275 seconds (i.e. 3.6 hours). As the labels for the artificial datasets could not be generated, the *Supervised GNN* method (described in section 4.3.6) could not be trained.

Figures 6.5, 6.6, and 6.7 show the evolution of the the mean score value for *GNN+GreedyPaths*, *GNN+GreedyCycles*, and *Unsupervised GNN*, respectively. Figure 6.8 shows the evolution of the standard deviation of the scores predicted on the validation dataset.

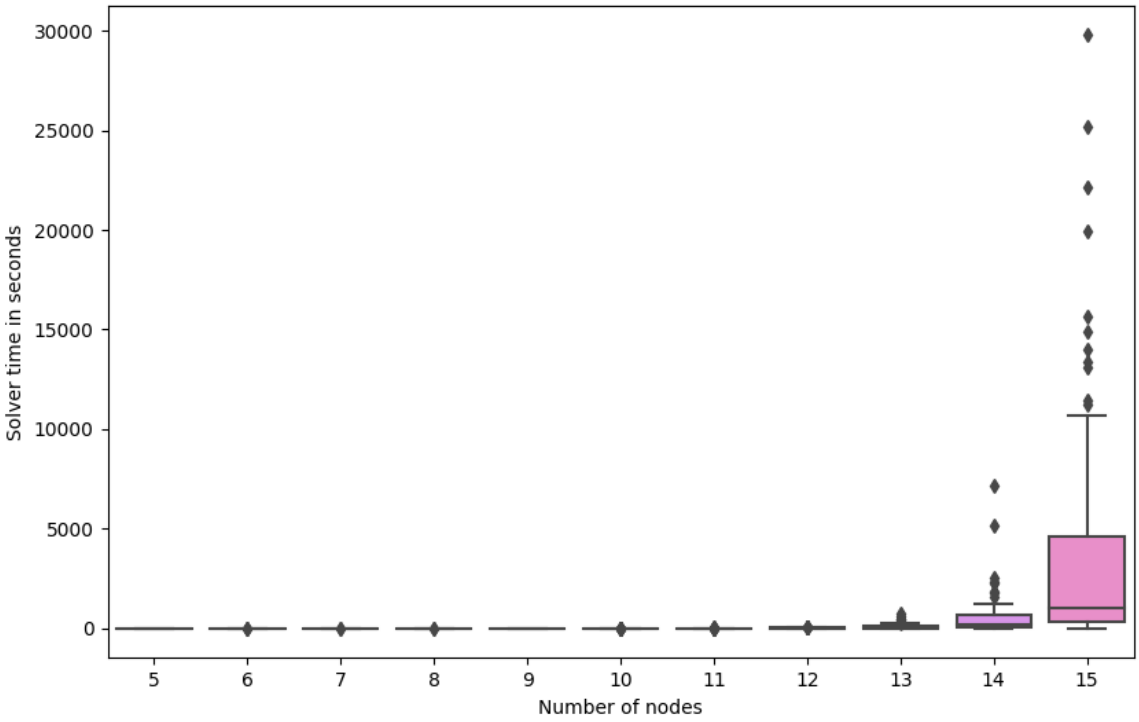

Figure 6.1 – Boxplot of the time it takes to run the solver on KEP instances of sizes 5 to 15 (i.e. number of nodes).

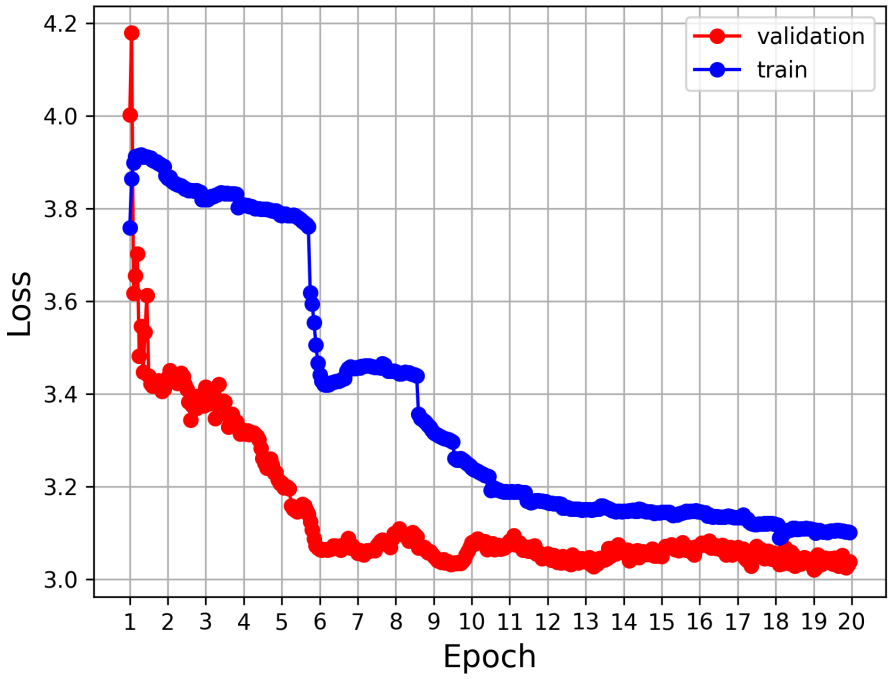

Figure 6.2 – Evolution of the training and validation loss for *GNN+GreedyPaths* method.

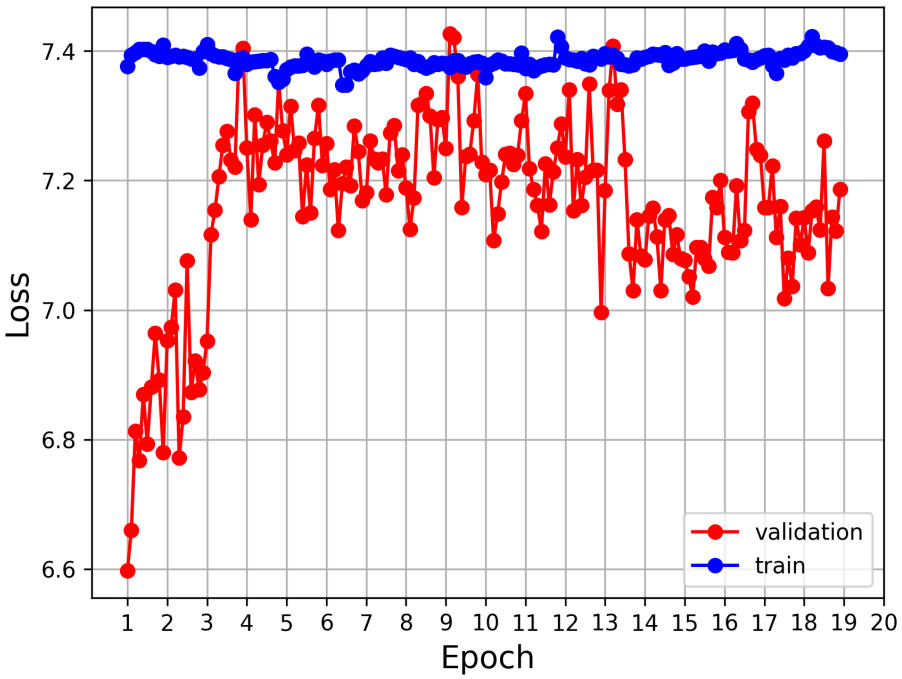

Figure 6.3 – Evolution of the training and validation loss for *GNN+GreedyCycles* method.

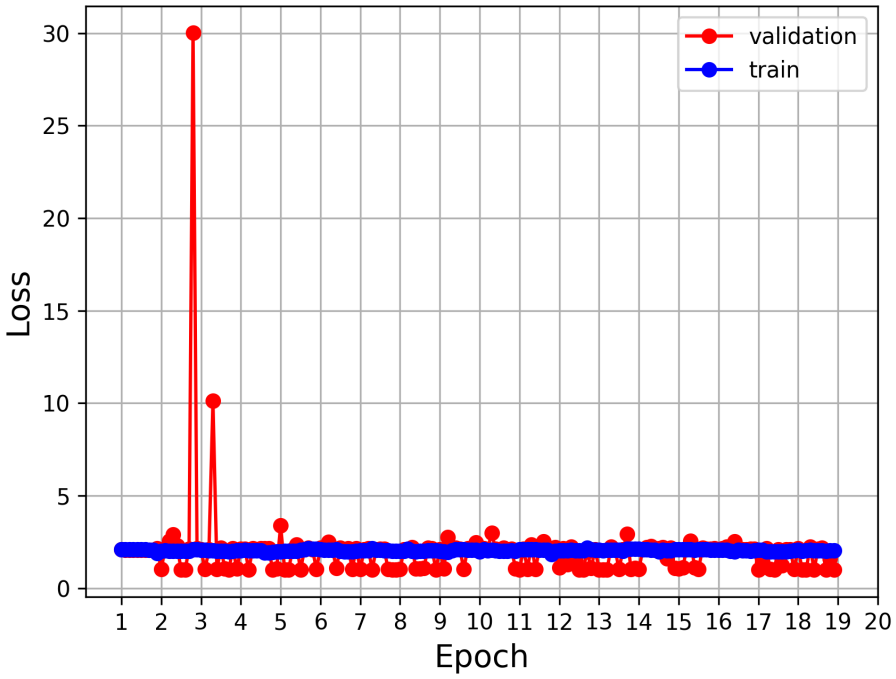

Figure 6.4 – Evolution of the training and validation loss for the *Unsupervised GNN* method.

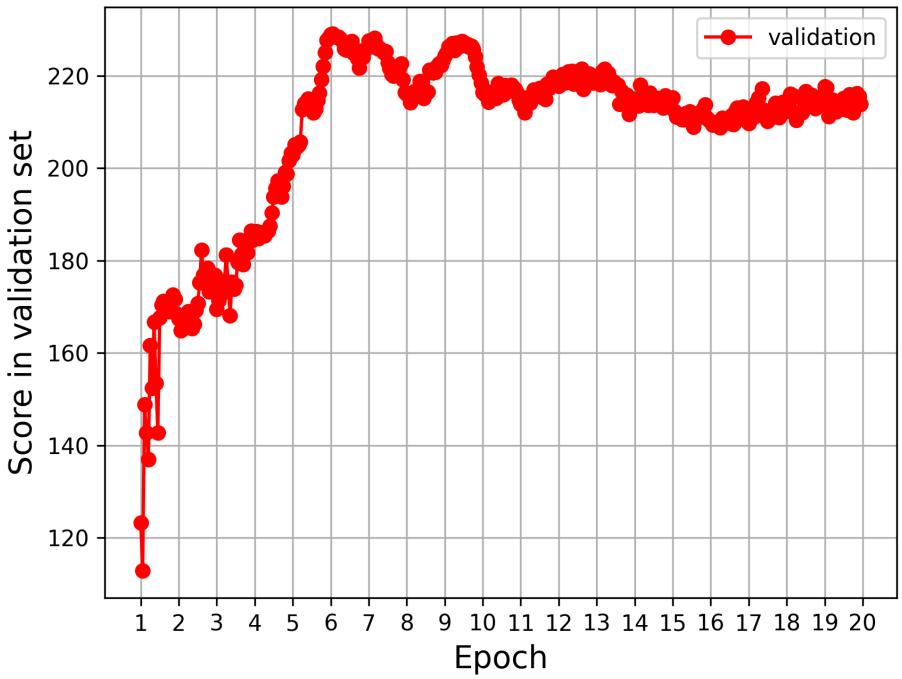

Figure 6.5 – Evolution of the score measured in the validation dataset for the *GNN+GreedyPaths* method.

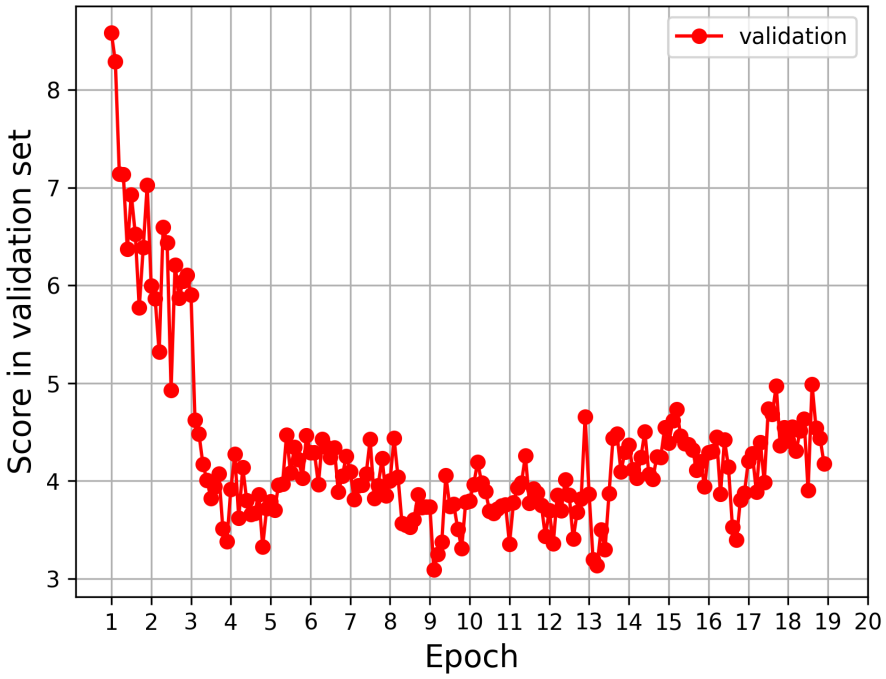

Figure 6.6 – Evolution of the score measured in the validation dataset for the *GNN+GreedyCycles* method.

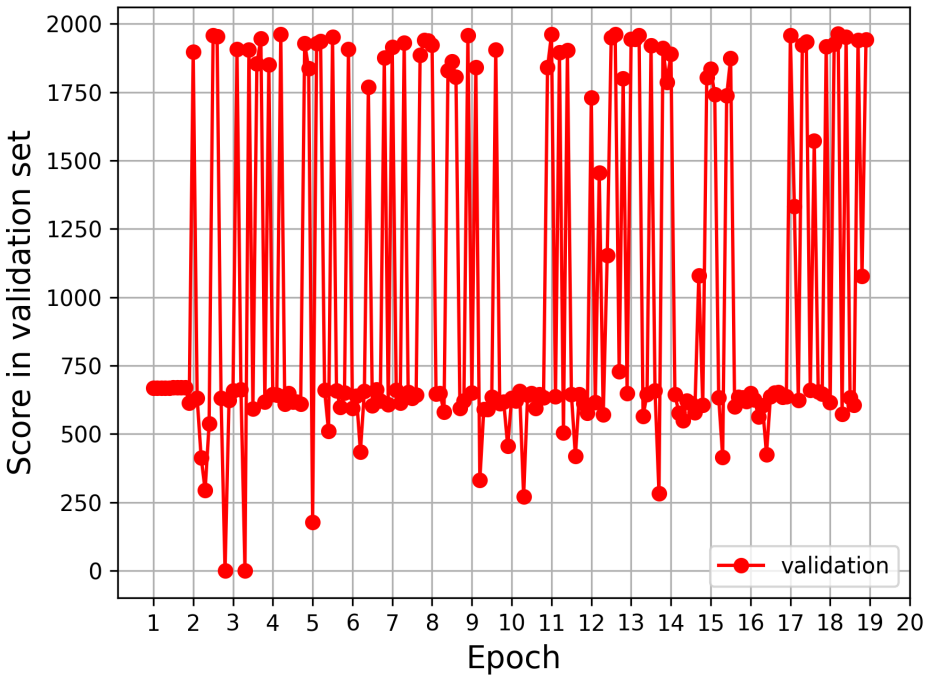

Figure 6.7 – Evolution of the score measured in the validation dataset for the *Unsupervised GNN* method.

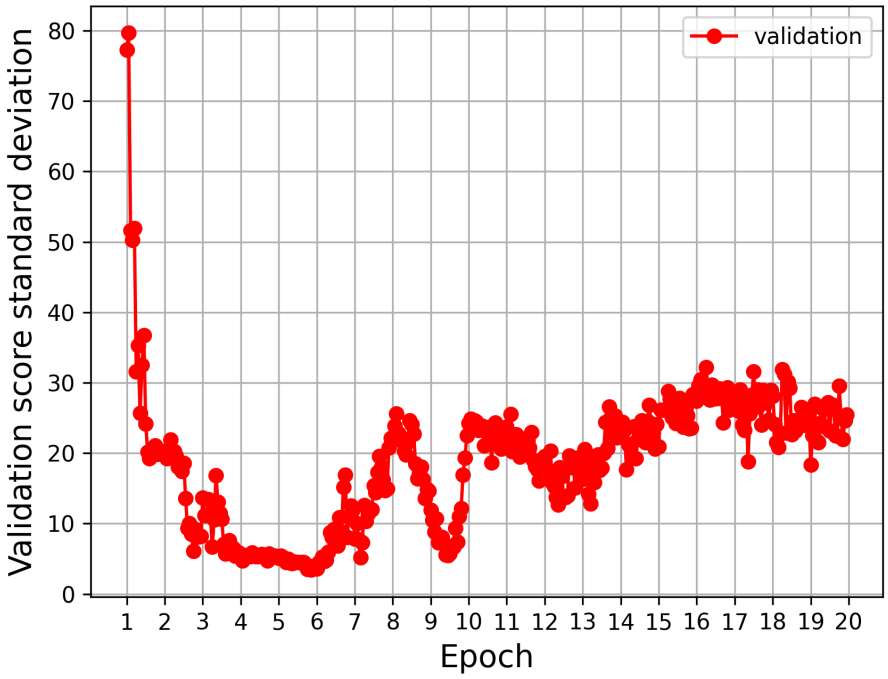

Figure 6.8 – Evolution of the standard deviation of score measured in the validation dataset for the *GNN+GreedyPaths* method.

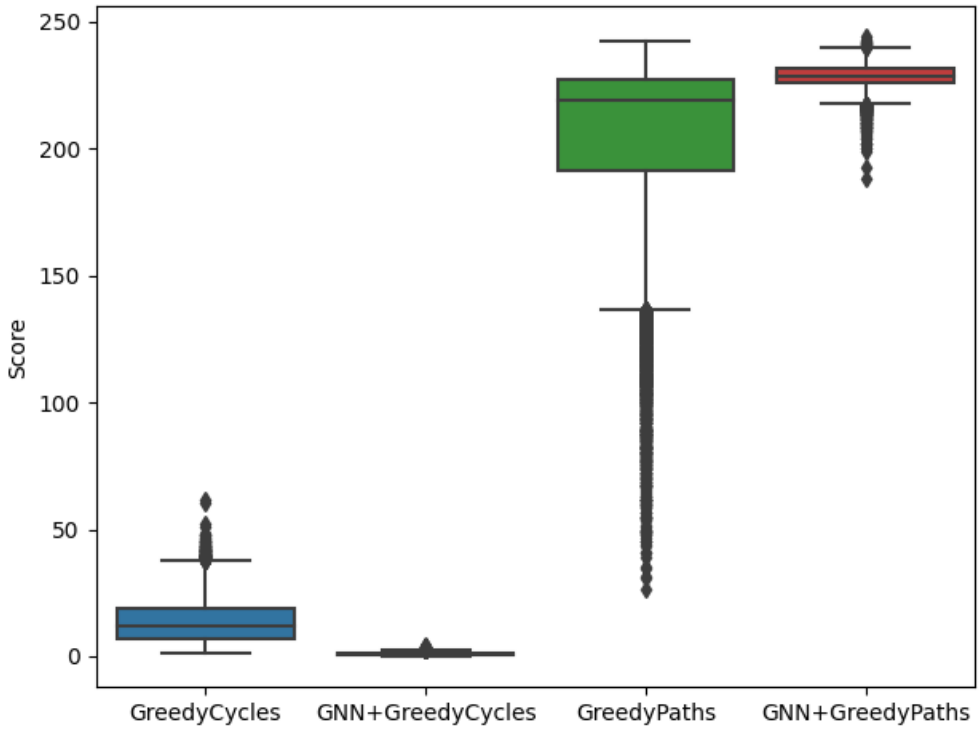

Figure 6.9 – Box plot comparing the approximate solution scores obtained when each of the evaluated methods was used in the test dataset. The evaluated methods were two non-learnt heuristics, *GreedyCycles* and *GreedyPaths*, and their 2 stage method versions, *GNN+GreedyCycles* and *GNN+GreedyPaths*.

6.3 Methods' Performances

Figure 6.9 shows a box plot of the approximate solution scores (i.e. the sum of the weights of the edges contained in the approximate solution) achieved by each method in the test dataset, with the exception of the *Unsupervised GNN*, *integer programming*, and *Supervised GNN* methods. As all the solutions found by the *Unsupervised GNN* method were invalid, there was no reason to evaluate their quality. The integer programming method is also absent from the plot, as it was not possible to run it in instances with 300 nodes. As a consequence, the *Supervised GNN* method also could not be evaluated, since it needed the exact solution from the integer programming method in order to be trained.

As we can see, the GreedyPaths heuristic method found much better solutions than GreedyCycles. The 2 stage method variations, *GNN+GreedyCycles* and *GNN+GreedyPaths*, obtained very different performances. Unfortunately, the GNN module in the *GNN+GreedyCycles* method did not manage to learn to output edge scores that help the heuristic; on the contrary, the quality of its approximate solutions are considerably worse than those from *GreedyCycles*. The *GNN+GreedyPaths* method, however, effec-

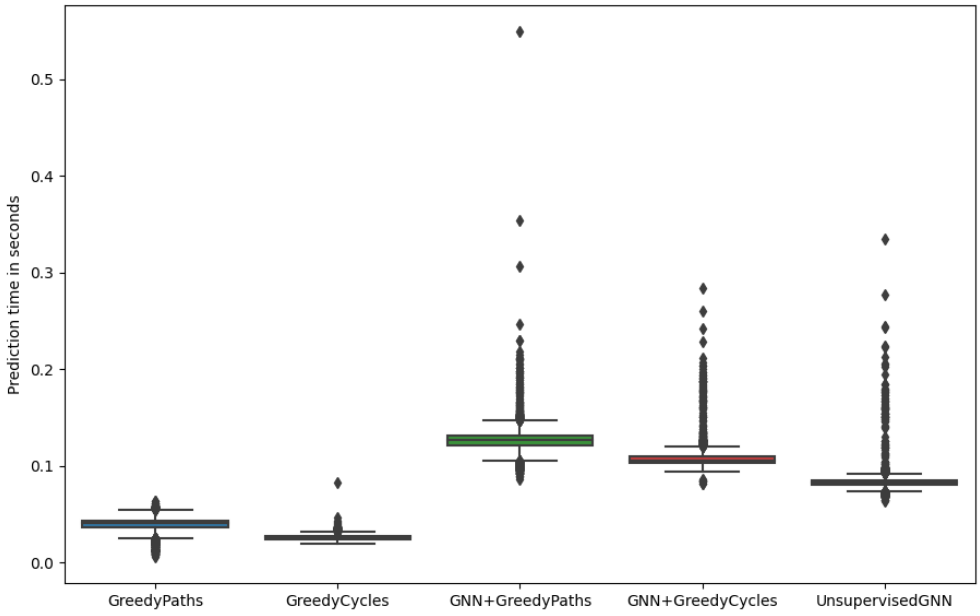

Figure 6.10 – Box plot comparing the time to compute a solution on each of the 10 thousand KEP instances of the test dataset, each one with 300 nodes. The evaluated methods were two non-learnt heuristics, *GreedyCycles* and *GreedyPaths*, their 2 stage method versions, *GNN+GreedyCycles* and *GNN+GreedyPaths*, and *UnsupervisedGNN*, which is a GNN trained and used without an heuristic.

tively learned to output better solutions than the non-learnt heuristic achieving a mean score of 228.40 on the test dataset, while *GreedyPaths* obtained 203.79; this shows an improvement of 12% of the mean solution score with the use of the GNN. We can also see that, although the best scores achieved by each of them are very similar, the score distribution is very different: while *GreedyPaths* outputs approximate solutions with very large range of scores, *GNN+GreedyPaths*'s approximate solutions have much more consistent scores, obtaining a decent performance throughout all instances of the dataset.

6.3.1 Methods' Computational Time

The box plot in Figure 6.10 shows the comparison between the time it took for each method to solve the KEP instances of the test dataset. As we can see, all of them took less then a second. Although the difference is not large, the two basic search heuristics took less time than the GNN based methods, and *GNN+GreedyPaths* took, on average, the most time.

7 ANALYSIS OF EXPERIMENTAL RESULTS

This chapter aims to present an analysis and interpretation of the results presented in the previous section, discussing the possible causes and consequences of what was observed.

7.1 Training of the GNN model

As we can see in Figure 6.2, the training of the *GNN+GreedyPaths* method was successful, seen as it managed to optimize the GNN by minimizing the loss function. The training of the *GNN+GreedyCycles* and *Unsupervised GNN* methods, however, were unsuccessful, as shown by the loss curves of Figures 6.3 and 6.4, which do not decrease over time.

Although at first glance at Figure 6.7 the *Unsupervised GNN* model seems to achieve great scores, unfortunately all its output solutions were invalid, i.e. they did not comply to the KEP constraints. It is clear that the loss constraint regularization (described at 4.3.3) added to the loss function was unsuccessful in helping the model learn to comply to the KEP constraints. This highlights the necessity of having a methods that guarantees, with total certainty, that all its output solutions are valid. However, even though there were many manual trials with different hyperparameter combinations, it is still possible that a variation of this technique could work with a different setting, i.e. another GNN architecture, other hyperparameters, and so on.

Because of the skip connection that sums the original edge weights to the predicted edge scores at the end of the GNN, the predicted solutions start off very similar to the ones made by GreedyPaths. Then, the changing of the GNN weights disrupts these scores, which increases the loss, but goes on to improve them, eventually arriving at a performance that is better than GreedyPaths. After some point (around epoch 6 in Figure 6.5), the learning converges to a solution, and after a while the performance starts to slowly worsen. The final model was chosen from the checkpoint with the highest score measured on the validation dataset, which was in epoch 6, step 3500. As we can see on Figure 6.8, at this point the model's scores on the validation also presented the lowest standard deviation, which indicates that the model's predictions were more consistent, maintaining a decent performance throughout all instances.

7.2 Solver Time Analysis

Because the Kidney Exchange problem is NP Hard, the time it takes to optimally solve each instance is expected to grow exponentially as the instance size grows. Regardless, considering that in (ANDERSON et al., 2015) real life KEP instances could be optimally solved in a reasonable time, it was expected that we would also be able to optimally solve the ones used in this study, since they have been constructed to have similar sizes to the ones used in the article. However, as described in Subsection 6.1, instances of size as small as 15 already took in average 1 hour to solve. Considering that we would want to optimally solve all 10 thousand instances of the test set in order to fairly compare to the other methods and to measure their optimality gap, this process would take an unreasonable time, estimated to be around 10 thousand hours, i.e. roughly 1.14 years. This estimate is only if the KEP instances on the test dataset had 15 nodes; for instances with 300 nodes, it would surely take an unreasonably enormous amount of time.

The number of nodes of the input instance is not, however, the sole factor that determines the time it takes to optimally solve it; in a set of instances with the same number of nodes, some are "harder" than others, i.e. take more time to solve. As the instance size increases, so does the variability of the time to solve it: the minimum and maximum times measured for instances with 15 nodes were 5.17 seconds and 29779.01 seconds (i.e. roughly 8 hours); for comparison, the minimum and maximum times for instances with 5 nodes were 0.8 and 1.3 seconds.

There can be several reasons why the authors of (ANDERSON et al., 2015) could compute the optimal solution of their KEP instances in much less time. First of all, they probably used a solver tool that is much more efficient than PyCSP3. In addition, the computer used may be much more powerful than the one used in this study. Also, it is important to remember that the NP-Hardness of KEP guarantees that the computational time it takes to solve the worst case scenario grows exponentially, but in practice real life instances may often have specific properties which may cause them to be either harder or easier to solve. Hence, another plausible reason is that their instances may be much easier to solve. Furthermore, the authors used a constrain relaxation technique that speeds up significantly the solving process. These set of reasons alone may not explain totally why they were able to optimally solve their KEP instances much faster than we did on our data; this may be further investigated in future work.

7.3 Methods' Performances

Ideally, we would want to evaluate and compare each method by measuring their optimality gap for each instance, i.e. how far the approximate solution is from the optimal one. However, as we do not have access to the optimal solution, this was not possible. We can nevertheless estimate it roughly by examining an upper bound: each instance has 300 nodes, and each node may donate and receive at most one kidney; thus, the solution with the most edges would contain cycles that together comprehend all nodes. As each edge weight is a value between 0 and 1, the maximum score possible is equal to 300, when all edges in the solution have a weight of 1. Hence, an upper bound for the score is the number of nodes, which in this case is 300. This is obviously extremely unrealistic, as it assumes that all nodes are PDPs, that there are a set of cycles that links all of them, and that the solution edge weights are equal to 1 (as the edge weights are values sampled from a uniform distribution between 0 and 1, their average value is 0.5). Considering the score upper bound of 300 as a very conservative estimate for the mean optimal solution value, we can estimate that the absolute and relative optimality gap for the GreedyPaths method would be 96.28 and 32%; for GreedyCycles, these values would be 286.29 and 95.4%; for GNN+GreedyPaths, 71.59 and 23.8%; for GNN+GreedyCycles, 299.15 and 99.7%. Hence, the improvement on the optimality gap of the GNN+GreedyPaths method in relation to GreedyPaths would be at least 34.4% (96.28 to 71.59), which is already a very substantial improvement.

It is clear that the two methods that searched for paths performed much better than the ones that searched for cycles. There are many possible explanations for this observed behaviour. Maybe the best cycles-only solution in a KEP instance is usually much worse than the best paths-only solution. This, however, can only be verified by making comparisons to the cycles-only and paths-only exact solutions, which are unavailable. Also, the GreedyCycles method is probably less efficient because it discards the path it is constructing if it does not end up closing a cycle. It also shortens the constructed cycle if it closes before the node it had begun on, which may also lead to worse performance overall. The GreedyPaths method does not have these issues, as it always keeps the edges it adds to each path it constructs.

We can also observe that the while the GNN module in the *GNN+GreedyPaths* improved the performance in relation to the basic non-learnable heuristic, in *GNN+GreedyCycles* it only worsened it. It is possible that its GNN module in *GNN+GreedyCycles* could not

learn the needed context to know if a given edge would lead to a longer and higher-valued cycle because it is too complex, and does not depend that much on the 3-neighborhood context, which is the limit of information gather in each node with the GNN architecture used, as it only has 3 message passing GNN layers. Another possible explanation is that it is way harder for a model to learn to compute edge scores that help the choices of GreedyCycles because it is inherently more complex than GreedyPaths, i.e. it is not just a sequence of simple decisions, as it also has to keep track of the rest of the nodes of the cycle being constructed, check if it closed a cycle, and remove from the solution in construction the edges added before the node where the cycle was closed. Put simply, the more complex the second step heuristic is, the harder it is for a machine learning model to learn to help it.

As explained at Section 6.3, *GNN+GreedyPaths* approximate solutions have much more consistent scores, which suggests that it probably handle much better "hard" instances. A plausible interpretation is that the GNN module helps the subsequent greedy heuristic to avoid choosing edges that are only locally good, but lead to worse paths overall. It is able to do this because it considers information of the neighborhood context.

7.3.1 Methods' Computational Time

The bottleneck of our GNN model is the message passing layers part that, considering the worst case of a complete graph, has a $\mathcal{O}(|V|^2)$ quadratic computational complexity, as for each node, it aggregates the messages from each of its neighbors. The next part of the GNN architecture, which makes computations on the edge features independently, is $\mathcal{O}(|E|)$, as it perform a fixed amount of computations per edge. At the end, the *node-wise softmax* operation (described at Susection 4.3.4) is equivalent to a softmax operation applied separately for the set of edges coming out of each node; its complexity is thus $\mathcal{O}(|V| * |E|)$. Thus, it can be said that the GNN module has a computation complexity of $\mathcal{O}(|V|^2 + |V| * |E| + |E|)$. It is also important to note that these operations are done mostly with matrix multiplication and thus are highly parallelizable. As for the heuristic methods *GreedyPaths* and *GreedyCycles*, their computational complexity is linear $\mathcal{O}(|V|)$, as in the worst case scenario one edge for each node will be added, one by one, into the solution; hence, it always performs a quantity of computational operations linearly proportional to the number of nodes of the input instance, at worst. The 2 stage method consists of a GNN module followed by one of the above-mentioned heuristics,

and has consequently a computational complexity of $\mathcal{O}(|V|^2 + |V| * |E| + |E| + |V|)$.

The results from Subsection 6.3.1 show that every method tested in this work took very little time to execute, with the exception of the *integer programming* method, which took so much time that applying it to 300 nodes instances became intractable. The *GreedyPaths* method took a bit more time than *GreedyCycles* probably because it found better solutions overall, and consequently took more computing steps to construct each solution. The same effect may also explain the prediction time difference between *GNN+GreedyPaths* and *GNN+GreedyCycles*. The *UnsupervisedGNN* method took a bit less time to execute than the two step methods, which was expected because it runs the same computations, but without the second step, which is the basic search heuristic.

8 CONCLUSION AND FUTURE WORK

This chapter summarizes this study, starting by trying to answer the questions presented in subsection 1.2 using the results from Chapter 6 and the analysis done in Chapter 7. Then, the main contributions will be reviewed. Finally, future research directions are suggested.

8.1 Conclusion

In this work, several heuristic methods with and without machine learning for approximately solving the Kidney Exchange Problem were proposed and investigated. They were tested on an artificial dataset and compared between each other and with an implementation of an exact solution method. Additionally, it was made an experiment for measuring the time it took for the exact solution method to solve an instance in relation to the instance size. The results of the evaluations and experiments were then analysed and discussed.

8.1.1 Answers to the Research Questions

As seen from the results in subsection 7.3, the main question presented at subsection 1.2 was answered: **yes, the Kidney Exchange problem can be better approximately solved with the help of machine learning**.

As for the feasibility of the ML methods, the *GNN+GreedyPaths* method surpassed all other evaluated heuristics in terms of the quality of the solutions it provides; among all evaluated methods, it remains the one that best approximately solves the dataset instances in a reasonable time. The other ML methods evaluated in the work (*UnsupervisedGNN* and *GNN+GreedyCycles*), however, did not achieve good results, as seen in Chapters 6 and 7.

Regarding the viability of such methods in terms of computational time, the GNN module adds an almost insignificant overhead when compared to the non-learnable heuristics. When compared to the solver running the exact solution, it is several orders of magnitude faster for instances with at least 15 nodes. The complexity of the two stage methods is linear, turning instances that were previously intractable due to their size into easily

approximately solvable in a reasonable time.

As for the limitations of the employed machine learning methods for this problem, it is clear that they are not simple to use, as they need to be properly trained, which is not easy to do. Although using the two stage method potentially improves considerably the performance in relation to the basic heuristics, as happened with the *GNN+GreedyPaths* method, it also introduces several new hyperparameters which need to be adequately set in order for the method to work. Applying supervised learning turned out to be unfeasible because of the need for the exact solution to be used as edge labels. Regarding the *UnsupervisedGNN* method, our results suggest that imposing the constraints through adding terms in a loss function is actually really hard; hence, the method never learns to output valid solutions, rendering it useless. Concerning future research directions, several of them were gathered throughout the research process, and are listed in Section 8.2.

8.1.2 Main Contributions

In the following list, a summary of each of the main contributions that this work provided is presented.

Learnable Heuristics for KEP Although the two stage approach was already used by past work (JOSHI; LAURENT; BRESSON, 2019; PENG; CHOI; XU, 2021; JOSHI et al., 2021), this was the first work to adapt it and apply it to KEP. Two variations of the approach were implemented: *GNN+GreedyPaths* and *GNN+GreedyCycles*, the first one having achieved satisfactory performance.

Non-Learnable Heuristics for KEP Two new deterministic heuristic methods for approximately solving the Kidney Exchange Problem were introduced: *GreedyPaths* and *GreedyCycles*. They were also evaluated in the test dataset, giving insight on their effectiveness.

Node-wise Softmax A variation of the softmax activation function designed to be applied to edge scores in graph problems was created and implemented. It showed to be useful for the GNN, empirically improving the its performance. The author plans to contribute to the PyTorch library with the implementation of this technique, thus making it available and easily usable by its future users.

KEP Unsupervised Loss A novel loss function (described at 4.3.2) designed for KEP was introduced. It optimizes the weighted sum of the edges in the predicted so-

lution without the need for supervision. It was validated in the training of the *GNN+GreedyPaths* method, as seen in Figure 6.2 in Section 6.2, where it lead the GNN to learn to effectively help the heuristic method construct better approximate solutions.

Loss Constraint Regularization Terms for KEP Although the proposed loss regularization terms (described at 4.3.3) showed no positive results, it can still be considered a first step in the direction The author believes that it can be tweaked and improved upon, and as a result become useful in the future.

8.2 Future Directions

"Todos os caminham levam à morte.
Perca-se."
— JORGE LUIS BORGES

In terms of possible directions of further research, we include:

Using real data Use data collected by countries' or hospital's healthcare system to evaluate how the presented methods would perform in real life situations. This would also allow us to compare our proposed methods with others that were already evaluated in the same data.

Using better artificial data There more sophisticated methods for generating artificial KEP instances, such as the ones presented at (SAIDMAN et al., 2006) and (DELORME et al., 2022). They are still far from sufficiently similar to real data so as to substitute evaluating it. However, it would still probably give an evaluation of KEP solving methods that is closer to that of real life situations. Furthermore, training the model with these instances could also potentially lead to better results. Another possibility of generating better artificial data would be to use a graph generator model that learns to create instances similar to the real data; the Graph Variational Auto-Encoder presented at (SIMONOVSKY; KOMODAKIS, 2018) and the MolGAN presented at (CAO; KIPF, 2018) are good examples of candidate methods to be adapted for that goal.

Considering compatible Patient-Donor Pairs The current formulation does not consider how compatible is a donor to a patient in a patient-donor pair. This could be represented by self-loops in these nodes.

Simulating KEP instances over time Another possible improvement in the artificial data is to simulate the evolution of a KEP instance through time. Starting with an arbitrary KEP instance, a solution would be chosen, and the remaining nodes of the instance (i.e. those which were not chosen as part of the solution) would form a new KEP instance. Then, some nodes might be added or deleted to account for changes in the donation pool not related to KEP donations executed, and the process would start again. It would be expected that these series of instances would better reflect real life situations, and possibly give further insight on the performance of each method. Furthermore, this would also allow us to evaluate a method's performance on the long term, i.e. how much it contributes to the quality of successive approximate solutions. Methods which perform good on the long term would therefore maintain good performance on a series of simulated successive KEP instances, thus contributing to the reduction of the donation pool in the long term.

Training the models with supervised learning Another direction is to train the *SupervisedGNN* method described in Subsection 4.3.6, evaluate it, and compare it to the other methods. For that, we would first have to obtain the exact solutions. This could be done either by improving a lot the integer programming method's speed and/or by training on much smaller instances (i.e. instances with less than 15 nodes). Another promissing variation of this idea is to use the N best solutions, and create soft labels where each edge would have a value between 0 and 1 that would indicate how often it appears in the best solutions, weighted by the quality of these solutions.

Measuring the method's generalization with different instance sizes Inspired by (JOSHI et al., 2021), a promising future experiment is to assess how much the ML methods generalize to bigger or smaller graphs than the ones seen in the training phase.

Improving the GNN model There are several promising directions of improvement for the GNN model used in this work. For instance, its architecture could be changed in many different ways: by using different layers, adding new ones, changing the size of each layer, and so on. As there is too many possibilities, an automated process like a neural architecture search could be used to facilitate and speed up the procedure. Techniques such as batch normalization and layer normalization could also be tested. Another possible improvement is to make the GNN learn to compute an embedding (i.e. a vector representation) for the whole graph, and then use it in the forward pass by concatenating it to edge and/or node features at

some point. Another one is to focus more on the edge features, as we want to learn information about the edges, and not about the nodes; the edge features could be better propagated through the network using more complex operations or utilized earlier, before the message passing stage.

Experimenting with other machine learning models The GNN model used for creating the edge embeddings for the greedy heuristic could be subtituted by or combined with other machine learning models such as Transformers (VASWANI et al., 2017).

Running a GNN before every step of the greedy heuristic Instead of running the GNN once and then passing the edge scores to the greedy heuristic method, new node embeddings could be generated before each step of the heuristic. Although significantly costlier in terms of inference, as the GNN is executed many times per instance, this has shown good results for other graph route optimization problems (KOOL; HOOF; WELLING, 2018). This approach is called *autoregressive decoding* and explored for solving the TSP by (JOSHI et al., 2021).

Improving the search heuristic The greedy heuristics presented in section 4.2 could be improved; one such way is to use the beam search technique (MEDRESS et al., 1977). Alternatively, new search heuristics could be implemented as well.

Comparing with other heuristics There has been a few other heuristic designed to solve KEP, such as the Top Trading Cycles and Chains algorithm presented in (ROTH; SöNMEZ; UNVER, 2004) and the 7-step metaheuristic presented in (DELORME et al., 2022). One could also implement new heuristics; one possibility would be to combine the GreedyPaths and GreedyCycles heuristics in a new heuristic that tries to find both cycles and chains. To compare the presented method with these other ones, however, their performance would need to be measured using the same test data.

REFERENCES

ABE, K. et al. **Solving NP-Hard Problems on Graphs with Extended AlphaGo Zero**. arXiv, 2019. Disponível em: <https://arxiv.org/abs/1905.11623>.

ABRAHAM, D. J.; BLUM, A.; SANDHOLM, T. Clearing algorithms for barter exchange markets: Enabling nationwide kidney exchanges. In: **Proceedings of the 8th ACM Conference on Electronic Commerce**. New York, NY, USA: Association for Computing Machinery, 2007. (EC '07), p. 295–304. ISBN 9781595936530. Disponível em: <https://doi.org/10.1145/1250910.1250954>.

ANDERSON, R. et al. Finding long chains in kidney exchange using the traveling salesman problem. **Proceedings of the National Academy of Sciences**, v. 112, n. 3, p. 663–668, 2015. Disponível em: <https://www.pnas.org/doi/abs/10.1073/pnas.1421853112>.

AXELROD, D. A. et al. An economic assessment of contemporary kidney transplant practice. **American Journal of Transplantation**, v. 18, n. 5, p. 1168–1176, 2018. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1111/ajt.14702>.

BAHDANAU, D.; CHO, K.; BENGIO, Y. **Neural Machine Translation by Jointly Learning to Align and Translate**. arXiv, 2014. Disponível em: <https://arxiv.org/abs/1409.0473>.

BAI, Y. et al. **GLSearch: Maximum Common Subgraph Detection via Learning to Search**. arXiv, 2020. Disponível em: <https://arxiv.org/abs/2002.03129>.

BELLO, I. et al. **Neural Combinatorial Optimization with Reinforcement Learning**. arXiv, 2016. Disponível em: <https://arxiv.org/abs/1611.09940>.

BIRó, P. et al. Modelling and optimisation in european kidney exchange programmes. **European Journal of Operational Research**, v. 291, n. 2, p. 447–456, 2021. ISSN 0377-2217. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0377221719307441>.

BRODY, S.; ALON, U.; YAHAV, E. **How Attentive are Graph Attention Networks?** arXiv, 2021. Disponível em: <https://arxiv.org/abs/2105.14491>.

CAI, H.; ZHENG, V. W.; CHANG, K. C.-C. **A Comprehensive Survey of Graph Embedding: Problems, Techniques and Applications**. arXiv, 2017. Disponível em: <https://arxiv.org/abs/1709.07604>.

CAO, N. D.; KIPF, T. Molgan: An implicit generative model for small molecular graphs. arXiv, 2018. Disponível em: <https://arxiv.org/abs/1805.11973>.

CONSTANTINO, M. et al. New insights on integer-programming models for the kidney exchange problem. **European Journal of Operational Research**, v. 231, n. 1, p. 57–68, 2013. ISSN 0377-2217. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0377221713004244>.

CORSO, G. et al. **Principal Neighbourhood Aggregation for Graph Nets**. arXiv, 2020. Disponível em: <https://arxiv.org/abs/2004.05718>.

DAI, H. et al. **Learning Combinatorial Optimization Algorithms over Graphs**. arXiv, 2017. Disponível em: <https://arxiv.org/abs/1704.01665>.

DELORME, M. et al. Improved instance generation for kidney exchange programmes. **Computers and Operations Research**, v. 141, p. 105707, 2022. ISSN 0305-0548. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0305054822000107>.

DIVARD, G.; GOUTAUDIER, V. Global perspective on kidney transplantation: France. **Kidney360**, Ovid Technologies (Wolters Kluwer Health), v. 2, n. 10, p. 1637–1640, out. 2021.

FEY, M.; LENSSEN, J. E. **Fast Graph Representation Learning with PyTorch Geometric**. 2019.

GORI, M.; MONFARDINI, G.; SCARSELLI, F. A new model for learning in graph domains. In: **Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.** [S.l.: s.n.], 2005. v. 2, p. 729–734 vol. 2.

GOYAL, P.; FERRARA, E. Graph embedding techniques, applications, and performance: A survey. **Knowledge-Based Systems**, Elsevier BV, v. 151, p. 78–94, jul 2018. Disponível em: <https://doi.org/10.1016%2Fj.knosys.2018.03.022>.

INFORMATION for patients - ANZKX program. 2022. <https://www.donatelife.gov.au/for-healthcare-workers/ANZKX/information-patients-anzkx-program>. Accessed: 2023-03-17.

JOSHI, C. K. et al. Learning tsp requires rethinking generalization. In: . Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. Disponível em: <https://drops.dagstuhl.de/opus/volltexte/2021/15324/>.

JOSHI, C. K.; LAURENT, T.; BRESSON, X. **An Efficient Graph Convolutional Network Technique for the Travelling Salesman Problem**. arXiv, 2019. Disponível em: <https://arxiv.org/abs/1906.01227>.

KEP GNN GitHub repository. 2023. <https://github.com/pfpimenta/kep_gnn>.

KOOL, W.; HOOF, H. van; WELLING, M. **Attention, Learn to Solve Routing Problems!** arXiv, 2018. Disponível em: <https://arxiv.org/abs/1803.08475>.

LAMB, L. C. et al. **Graph Neural Networks Meet Neural-Symbolic Computing: A Survey and Perspective**. arXiv, 2020. Disponível em: <https://arxiv.org/abs/2003.00330>.

LECOUTRE, C.; SZCZEPANSKI, N. PYCSP3: modeling combinatorial constrained problems in python. **CoRR**, abs/2009.00326, 2020. Disponível em: <https://arxiv.org/abs/2009.00326>.

LECUN, Y. **Week 6 – Lecture: CNN applications, RNN, and attention**. 2020. <https://www.youtube.com/watch?v=ycbMGyCPzvE#t=43m30s>.

LEMOS, H. et al. **Graph Colouring Meets Deep Learning: Effective Graph Neural Network Models for Combinatorial Problems**. arXiv, 2019. Disponível em: <https://arxiv.org/abs/1903.04598>.

LESKOVEC, J. **Graph Representation Learning**. 2017. <https://cci.drexel.edu/bigdata/bigdata2017/files/Keynote_Leskovec.pdf>.

LI, Z.; CHEN, Q.; KOLTUN, V. **Combinatorial Optimization with Graph Convolutional Networks and Guided Tree Search**. arXiv, 2018. Disponível em: <https://arxiv.org/abs/1810.10659>.

LONGEST Kidney Transplant Chain - Guinness World Record Organization Distinguishes the National Kidney Registry for World's Longest Kidney Transplant Chain. 2020. <https://transplantsurgery.ucsf.edu/news--events/ucsf-news/88223/UCSF-Part-of-Longest-Kidney-Transplant-Chain---Guinness-World-Record-Organization-Distinguishes-E2%80%99s-Longest-Kidney-Transplant-Chain#>. Accessed: 2023-03-01. Disponível em: <https://transplantsurgery.ucsf.edu/news--events/ucsf-news/88223/UCSF-Part-of-Longest-Kidney-Transplant-Chain---Guinness-World-Record-Organization-Distinguishes-%E2\%80\%99s-Longest-Kidney-Transplant-Chain#>.

MEDRESS, M. et al. Speech understanding systems: Report of a steering committee. **Artificial Intelligence**, v. 9, n. 3, p. 307–316, 1977. ISSN 0004-3702. Disponível em: <https://www.sciencedirect.com/science/article/pii/0004370277900261>.

NAZI, A. et al. **GAP: Generalizable Approximate Graph Partitioning Framework**. arXiv, 2019. Disponível em: <https://arxiv.org/abs/1903.00614>.

NETWORKX.ORG. 2023. <https://networkx.org/>. Accessed: 2023-02-17.

OPTN data on organ donation, waitlist and transplant activity. 2023. <https://optn.transplant.hrsa.gov/data/view-data-reports/national-data/#>. Accessed: 2023-01-25.

PENG, Y.; CHOI, B.; XU, J. Graph learning for combinatorial optimization: A survey of state-of-the-art. **Data Science and Engineering**, v. 6, n. 2, p. 119–141, Jun 2021. ISSN 2364-1541. Disponível em: <https://doi.org/10.1007/s41019-021-00155-3>.

PRATES, M. O. R. et al. **Learning to Solve NP-Complete Problems - A Graph Neural Network for Decision TSP**. arXiv, 2018. Disponível em: <https://arxiv.org/abs/1809.02721>.

PRE-COMMIT. 2023. <https://pre-commit.com/>. Accessed: 2023-02-17.

PYTHON.ORG. 2023. <https://www.python.org/>. Accessed: 2023-02-17.

PYTORCH.ORG. 2023. <https://pytorch.org/>. Accessed: 2023-02-17.

ROTH, A.; SöNMEZ, T.; UNVER, U. Kidney exchange. **The Quarterly Journal of Economics**, v. 119, n. 2, p. 457–488, 2004. Disponível em: <https://EconPapers.repec.org/RePEc:oup:qjecon:v:119:y:2004:i:2:p:457-488.>

ROTH, A. E.; SöNMEZ, T.; ÜNVER, M. U. Efficient kidney exchange: Coincidence of wants in markets with compatibility-based preferences. **American Economic Review**, v. 97, n. 3, p. 828–851, June 2007. Disponível em: <https://www.aeaweb.org/articles?id=10.1257/aer.97.3.828>.

SAIDMAN, S. L. et al. Increasing the opportunity of live kidney donation by matching for two- and three-way exchanges. **Transplantation**, v. 81, n. 5, 2006. ISSN 0041-1337. Disponível em: <https://journals.lww.com/transplantjournal/Fulltext/2006/03150/Increasing_the_Opportunity_of_Live_Kidney_Donation.20.aspx>.

SANTOS, H. L. d.; LAMB, L. d. C. O. **Solving the decision version of the Graph Coloring Problem : a neural-symbolic approach using graph neural networks.** [s.n.], 2020. Disponível em: <https://search.ebscohost.com/login.aspx?direct=true&AuthType=shib&db=cat07377a&AN=sabi.001114939&lang=pt-br&scope=site&authtype=guest,shib&custid=s5837110&groupid=main&profile=eds>.

SATO, R.; YAMADA, M.; KASHIMA, H. **Approximation Ratios of Graph Neural Networks for Combinatorial Problems**. arXiv, 2019. Disponível em: <https://arxiv.org/abs/1905.10261>.

SCARSELLI, F. et al. The graph neural network model. **IEEE Transactions on Neural Networks**, v. 20, n. 1, p. 61–80, 2009.

SERIE HISTORICA LISTA DE ESPERA - BRASIL. 2022. <https://www.gov.br/saude/pt-br/composicao/saes/snt/arquivos/serie-historica-lista-de-espera-brasil.pdf/view>. Accessed: 2023-01-25.

SERIE HISTORICA TRANSPLANTES - BRASIL. 2022. <https://www.gov.br/saude/pt-br/composicao/saes/snt/arquivos/serie-historica-transplantes-brasil.pdf/view>. Accessed: 2023-01-25.

SHERVASHIDZE, N. et al. Weisfeiler-lehman graph kernels. **Journal of Machine Learning Research**, v. 12, n. 77, p. 2539–2561, 2011. Disponível em: <http://jmlr.org/papers/v12/shervashidze11a.html>.

SIMONOVSKY, M.; KOMODAKIS, N. Graphvae: Towards generation of small graphs using variational autoencoders. **CoRR**, abs/1802.03480, 2018. Disponível em: <http://arxiv.org/abs/1802.03480>.

VASWANI, A. et al. **Attention Is All You Need**. arXiv, 2017. Disponível em: <https://arxiv.org/abs/1706.03762>.

VELIčKOVIć, P. et al. **Graph Attention Networks**. arXiv, 2017. Disponível em: <https://arxiv.org/abs/1710.10903>.

VINYALS, O.; FORTUNATO, M.; JAITLY, N. **Pointer Networks**. arXiv, 2015. Disponível em: <https://arxiv.org/abs/1506.03134>.

WU, Y. et al. **Learning Improvement Heuristics for Solving Routing Problems**. arXiv, 2019. Disponível em: <https://arxiv.org/abs/1912.05784>.

YANG, Y.; RAJGOPAL, J. **Learning Combined Set Covering and Traveling Salesman Problem**. arXiv, 2020. Disponível em: <https://arxiv.org/abs/2007.03203>.